
nVision User Guide

Release 2016.3

Peter Schregle, Impuls Imaging GmbH

Mar 29, 2018

Contents

1	Program Editions	5
1.1	nVision Designer	5
1.2	nVision Runtime	5
1.3	Module Matrix	6
2	Requirements	7
3	Licensing	9
4	Camera and Device Interfaces	11
5	Getting Started	15
5.1	Particle Analysis	15
6	Ribbon	23
6.1	File	24
6.2	Home	25
6.3	Locate	26
6.4	Measure	27
6.5	Verify	28
6.6	Identify	29
6.7	Process	30
6.8	Segmentation	36
6.9	Pipeline	37
6.10	Interactive Measurement	39
7	Tools	41
7.1	Location	41
7.2	Measurement	48
7.3	Verification	70
7.4	Identification	74
7.5	Miscellaneous	84
8	Dataflow	95
8.1	Linear Pipeline	97
8.2	Sub-Pipeline	98
9	Image Processing	103

9.1	Point Algorithms	103
9.2	Color	123
10	Image Analysis	129
10.1	Blob Analysis	129
10.2	Region Features	134
10.3	Pixel Based Features	158
10.4	Object Filtering	170
10.5	Gauging	170
10.6	Identification	173
10.7	Camera Calibration	176
11	Geometry	181
11.1	Geometry	181
11.2	Geometric Primitives	181
11.3	Geometric Transformations	184
11.4	Graphics	189
12	User Interface Creation	195
12.1	Controls	196
12.2	Widgets	217
13	Graphic Programming Language	221
13.1	Pipelines	221
13.2	Nodes	221
13.3	Pins	222
13.4	Subpipelines	222
13.5	Transform	223
13.6	Types	225
13.7	Conversions	225
14	Miscellaneous	227
14.1	Digital IO	227
14.2	CAD Drawing Support	238
14.3	Database Support	240
15	Reference	245
15.1	Ribbon	245
15.2	Tools	260
15.3	Nodes	276
16	Credits	695
16.1	Boost	695
16.2	zlib	695
16.3	bzip	696
16.4	FreeImage	697
16.5	DynamicExpresso	701
16.6	KBCsv	701
16.7	NetworkView	702
16.8	TaskDialog	705



User Guide

Machine Vision Development System Release 2017.1

*Copyright (c) 2017 Impuls Imaging GmbH
All rights reserved.*

Impuls Imaging GmbH

Schlingener Str. 4
86842 Türkheim

Germany/European Union

<http://www.impuls-imaging.com>

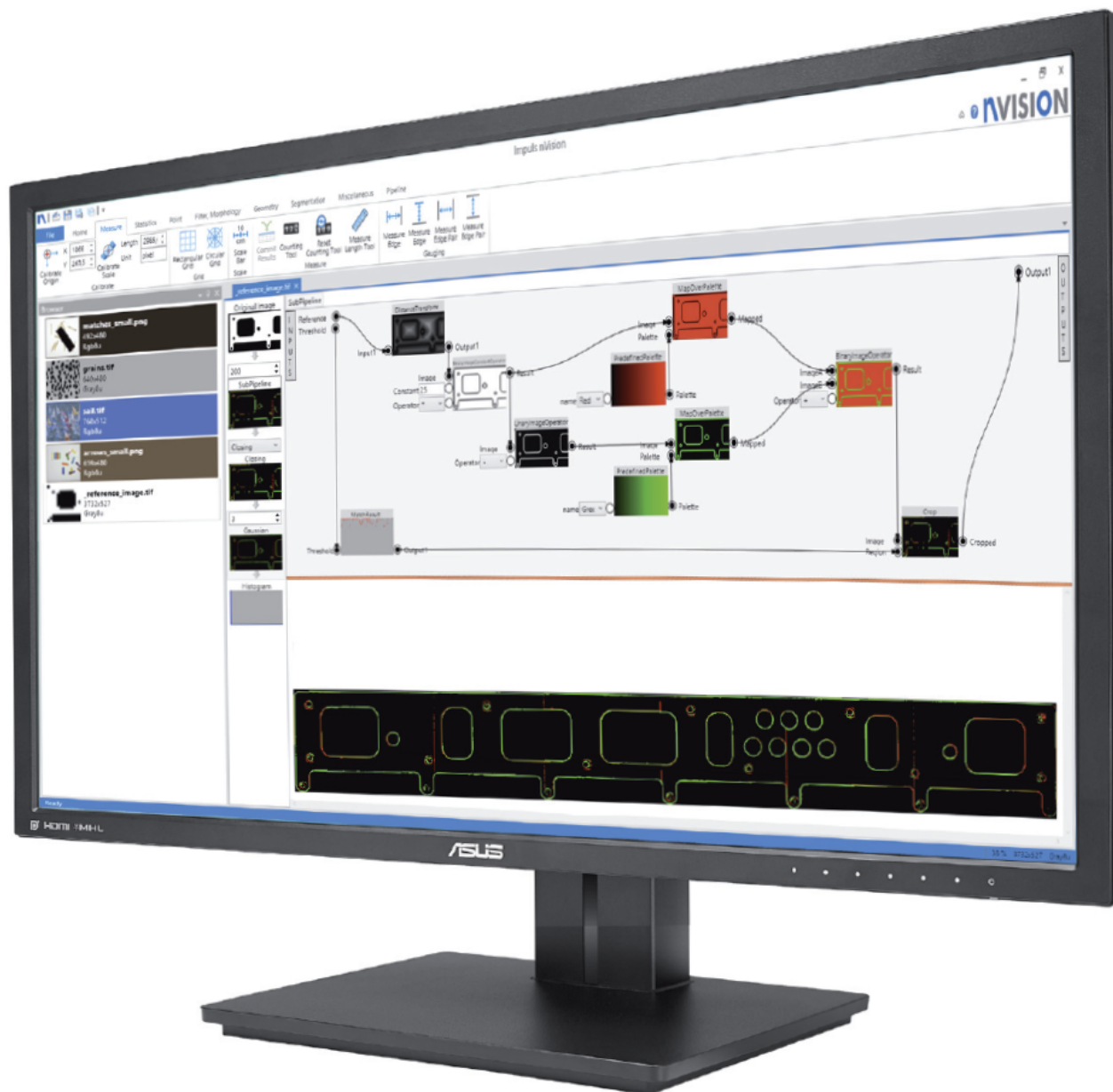
nVision is a computer vision development system. The main use is to create automated machine vision applications that are used in the industry. It uses a graphical approach to programming, which leads to much shorter development time, compared to traditional programming. In addition, the learning curve is shallow, which allows you to be immediately productive.

Since **nVision** has valuable and powerful tools for image inspection, it has proven to be usable in the scientific laboratory as well. It is the tool of choice in the lab, when a Photoshop-based approach is too laborious and when you need fast results. The graphical programming helps in the lab as well to quickly create the needed results in an automated way.

nVision helps you with a variety of vision based tasks needed in the industry:

- Check the position of a part to guide a handling systems or to put a vision tool in the right position.
- Identify a part based on marks, shape or other visual aspects.
- Verify a part to check if it has been built or assembled correctly.
- Measure a parts dimensions.
- Inspect a part for defects.

Make sure to read the **nVision Quick Start Guide**. It contains the most important information on a single page.



Made in Germany



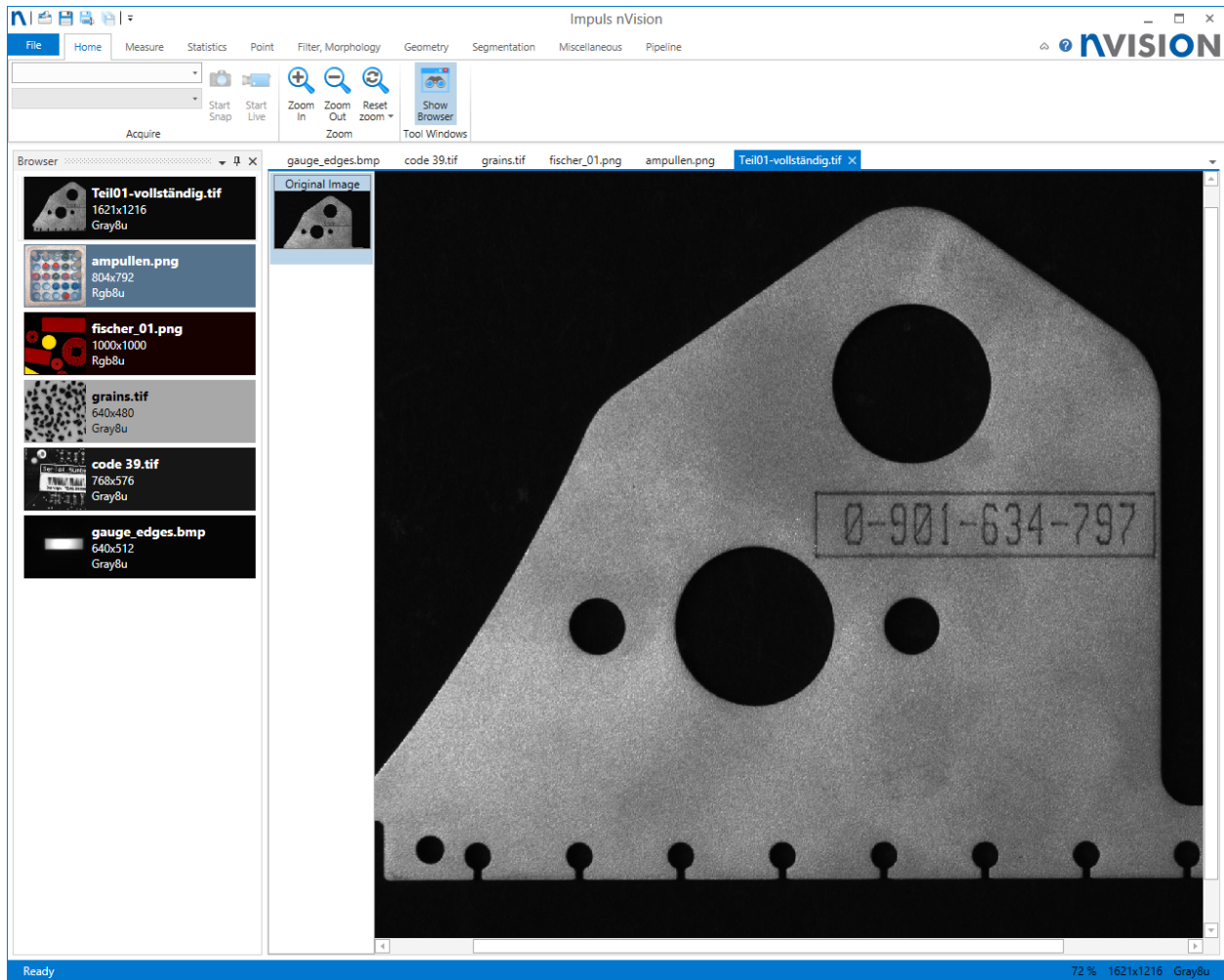


Fig. 1: The nVision window.



Quick Start Guide



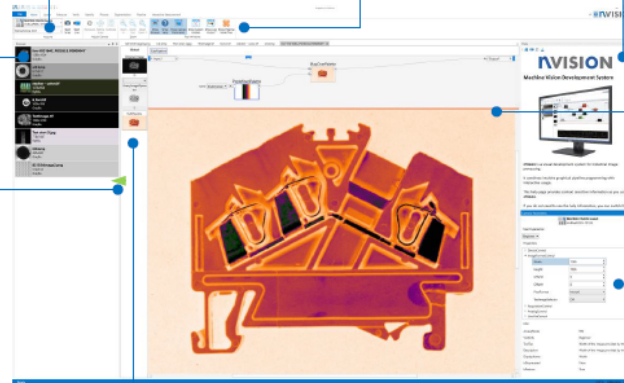
Show or hide the Browser window
Click File – Show Browser or F2 to show the Browser window. The Browser displays the loaded data (image files, data files or folder contents).

Chose from the available cameras
The supported cameras are listed and you can chose the camera and the mode (monochrome, color, bit-depth).

Explore commands on the ribbon
Each ribbon has tab has groups and each group has a set of related commands. Commands and tools are inserted into the linear pipeline.

Show or hide the Help window
Click File – Show Help or F1 to show the Help window. The contents of the Help window is sensitive to your actions.

Take snapshots of image data
At any time you can use the Commit command to copy the processed data to the Browser.



Drag down the orange splitter bar
of a sub-pipeline step to reveal the graphical program. Right-click into the pipeline pane to insert pipeline nodes and connect their inputs and outputs to build a graphical program.

Watch videos to learn more
<https://www.youtube.com/playlist?list=PLtSL1kEm4wVenIScAUbekxvlgzSFVVtr>.

Read and write camera parameters
The parameters of any GenICam compliant camera can be changed with the Camera Parameters window or within a graphical program (File – Show Camera Parameters or F3).

The linear pipeline
Shows the flow of commands or tools. Commands are single, atomar steps, while tools are implemented as sub-pipelines, usually with a customized GUI.

Explore Samples
Samples are listed in the Help window and can be loaded by a single click.

Read the nVision User Guide
Valuable information can be found at <http://nvision-user-guide.readthedocs.org>.

Pipelines are graphical programs.
They are stored alongside an image file (image.tif <-> image.tif.xpipe) or a folder containing image files (folder <-> folder.xpipe), or can be exported and imported separately.

nVision 2015.4

Impuls Imaging GmbH
Schlingener Str. 4
86842 Türkheim
www.impuls-imaging.com

nVision has two main components: the **nVision Designer** is the rapid application development system and the **nVision Runtime** is the runtime part. The **nVision Designer** provides a graphical user interface and a graphical programming system. The **nVision Runtime** provides the computer vision functionality in a modular way and a facility to execute the graphical programs. It also has the means to display a HMI (human-machine interface).

nVision Designer is what you use to create applications. When these applications are to be deployed to the machine on the factory floor, usually only the more cost-effective and modular **nVision Runtime** is used.

1.1 nVision Designer

nVision Designer comes with complete feature sets targeted to the intended audience.

Product	Description
nVision Designer MV	Targeted for machine vision development.
nVision Designer L	Targeted for laboratory use.
nVision Designer U	The ultimate version has everything.

1.2 nVision Runtime

The following runtime modules are available.

Module	Description
Image Processing	Contains basic image processing functionality, both monochrome and color, including camera support.
Analysis, Measurement	Contains image analysis functionality, blob analysis, camera calibration, gauging measurement.
Template Matching	Contains the matching and searching functionality, template matching, geometric pattern matching.
Barcode, Matrix-code	Contains the barcode reading functionality, both linear and two dimensional symbologies.
OCR, OCV	Contains the Optical Character Recognition and Optical Character Verification functionality.

1.3 Module Matrix

The table shows which modules are included in which designer edition.

Module	MV	L	U
Image Processing	*	*	*
Analysis, Measurement	*	*	*
Template Matching	*		*
Barcode, Matrixcode	*		*
OCR, OCV	*		*

CHAPTER 2

Requirements

nVision (Designer and Runtime) runs on Windows 7 or higher, 64 bit (recommended) or 32 bit (memory limitations apply). It supports many cameras of different vendors, either via the GigE Vision, USB3 Vision, and GenICam standards or directly via the manufacturers SDK.

CHAPTER 3

Licensing

nVision (Designer and Runtime) are licensed to the specific hardware. In order to run **nVision**, you need a license key.

When **nVision** is started for the first time, it needs to be activated. In addition to the license key, activation needs a hardware key, which is determined automatically. The activation process uses the hardware key, together with the license key and creates an activation key. It is the activation key - in combination with the two other keys - that actually unlocks the **nVision** software.

Usually, activation works through an internet connection and is simple and transparent. If you copy your license key to the clipboard before you start **nVision** for the very first time, you will hardly notice the process. Otherwise, a dialog such as the following will be displayed:

In order to continue, you should enter your license key and press the Return key. **nVision** will contact the licensing server and use the returned activation key to unlock. If the license server cannot be connected for some reason, you can phone (+49 8245 7749600) or write an email to info@impuls-imaging.com to tell us both your license key and the hardware key, and we will provide you with an activation key that you can enter manually.

Activation is perpetual for a specific hardware. If the hardware changes, this is considered a new hardware, which means that you have to activate **nVision** again. While **nVision** is licensed to one machine only, one license key permits three activations to cover the case of defective hardware.



Fig. 3.1: The **nVision** licensing dialog.

Camera and Device Interfaces

The primary interfaces to cameras used in **nVision** are the **GigEVision** and the **USB3 Vision** interfaces. These interfaces are standardized. Image data can be accessed via **GenTL** and camera properties can be read and written via **GenICam**.

For cameras interfaced via **GenTL** and **GenICam**, the full set of properties is accessible from within **nVision**.

The camera properties are accessible in the form of a properties window.

In addition, the camera properties can be read and written from within a graphical pipeline, i.e. the properties can be changed from within a program.

Secondary interfaces to cameras and devices exist via manufacturer SDKs for historical reasons. USB2 cameras are interfaced this way, since they were available before standardization, as well as interfaces to frame grabbers.

The following table lists tested devices:

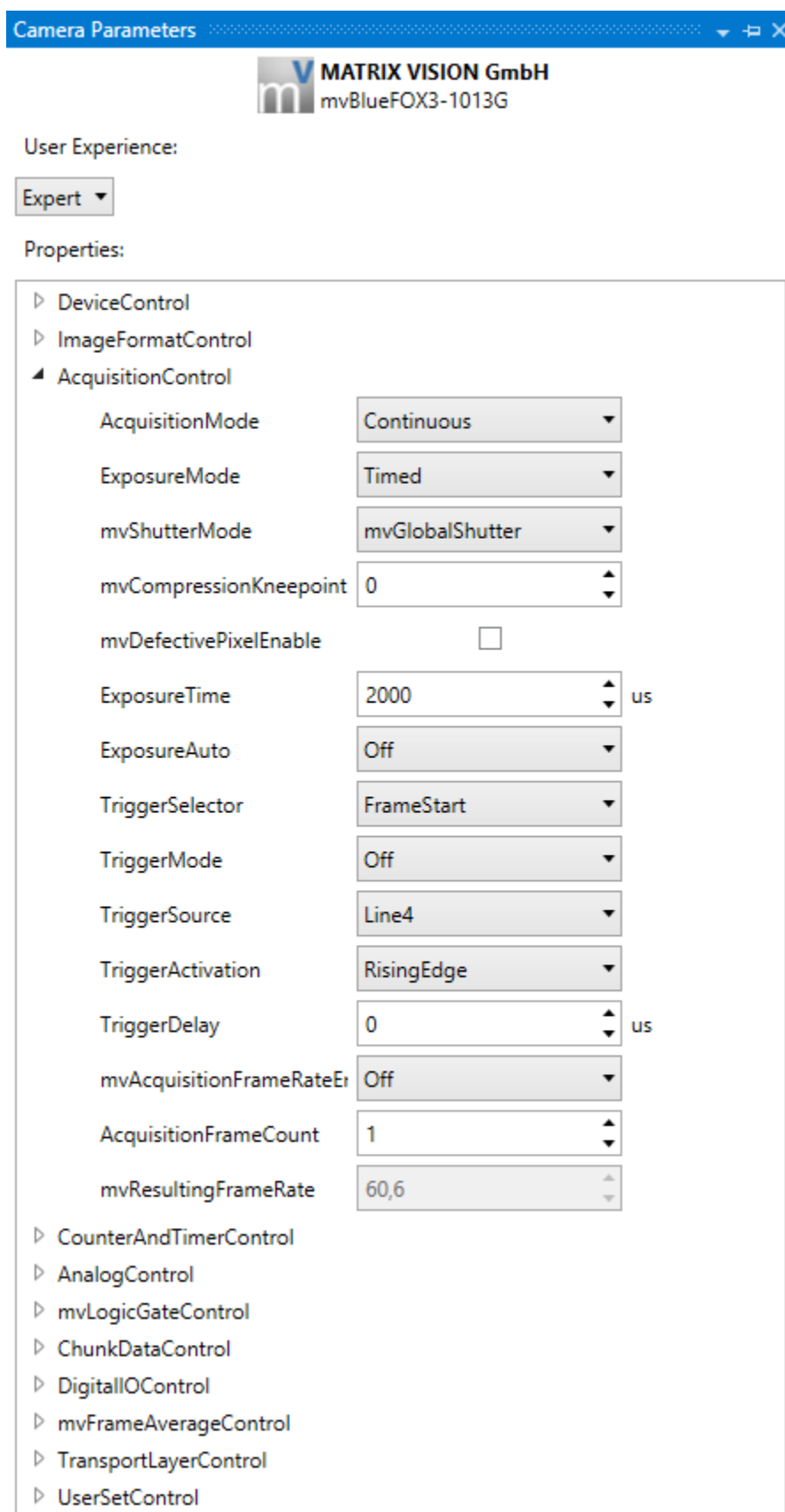


Fig. 4.1: The camera properties window.

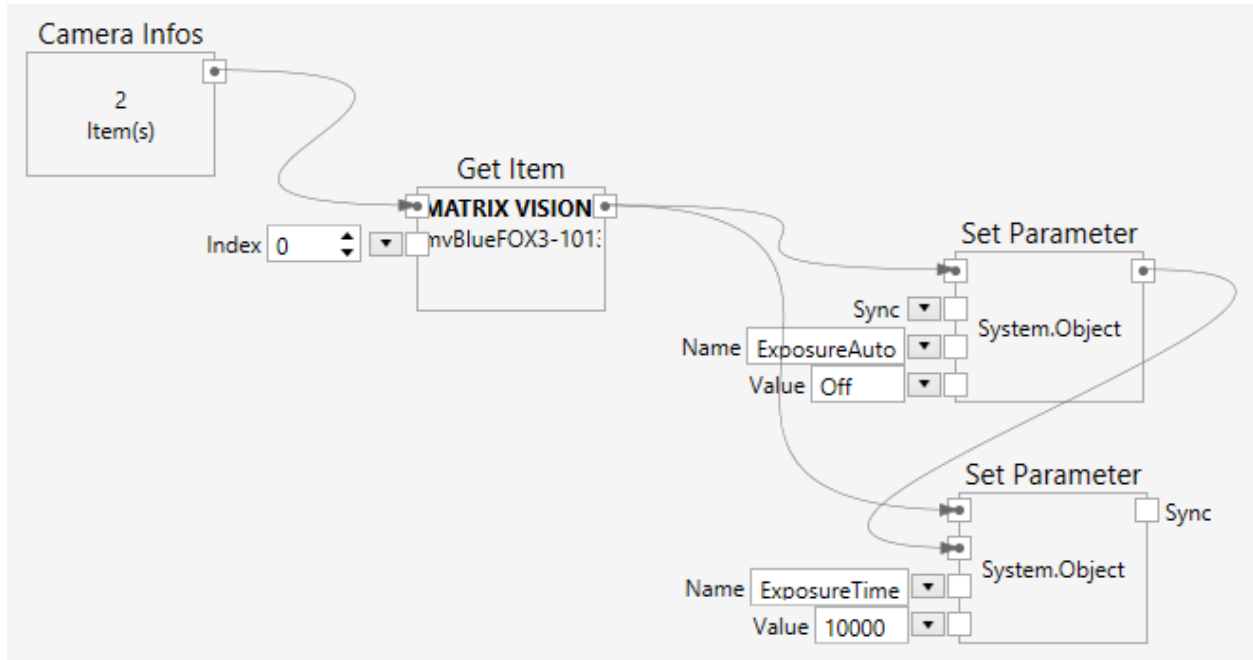














Fig. 4.2: Writing camera properties from within a pipeline.

Manufacturer		URL	Device	Interface
Allied Vision		www.alliedvision.com	GigE Cameras	GenTL / GenICam
Basler		www.baslerweb.com	GigE Cameras	GenTL / GenICam
Baumer		www.baumer.com	GigE / USB3 Cameras	GenTL / GenICam
Flir		www.flir.com	GigE Cameras (A315)	GenTL / GenICam
IDS		www.ids-imaging.com	USB Cameras	Direct (Vendor SDK)
IDS		www.ids-imaging.com	GigE Cameras	GenTL / GenICam
JAI		www.jai.com	GigE / USB3 Cameras	GenTL / GenICam
Leutron		www.leutron.com	USB Cameras	Direct (Vendor SDK)
Gardasoft		www.gardasoft.com	GigE Flash Controller	GenTL / GenICam
Matrix Vision		www.matrix-vision.com	Framegrabber	Direct (Vendor SDK)
Matrix Vision		www.matrix-vision.com	USB Cameras	Direct (Vendor SDK)
Matrix Vision		www.matrix-vision.com	GigE / USB3 Cameras	GenTL / GenICam

All other **GigE Vision** or **USB3 Vision** capable cameras should work with **nVision** right out of the box, but have not been tested in-house so far.

The screen shot shows the various parts of the **nVision** application.

At the top there is a ribbon with commands to control the application.

At the bottom, there is the work area, consisting of the browser and the workbench, where the main interaction takes place.

Before we explain **nVision** in detail, let us show a few simple examples of how to work with **nVision**.

5.1 Particle Analysis

Let us assume that the current task is to count and measure particles.

1. Open the image

The first step is to load an image. **nVision** can acquire images from digital cameras, but for now let us load an image from disk, by using the **File - Open** command:

In the dialog that opens, select the `cells.tif` file. The file is loaded and displayed both in the browser and in the work area.

2. Segment the image

Segmenting the image is next. The particles - in this case the cell nuclei - should be separated from the background. This can be done with the **Threshold** command from the ribbon's **Segmentation** tab.

A threshold of 100 works nicely for this image.

Of course you can change the thresholding mode as well as the threshold itself with the controls on top of the threshold node.

As a result of the thresholding, the appearance of the picture has changed, because the segmented cell nuclei are displayed in a red overlay color (partly transparent) on top of the original image. In addition, the threshold command has been appended to the linear processing pipeline, which now shows two steps.

3. Find connected components

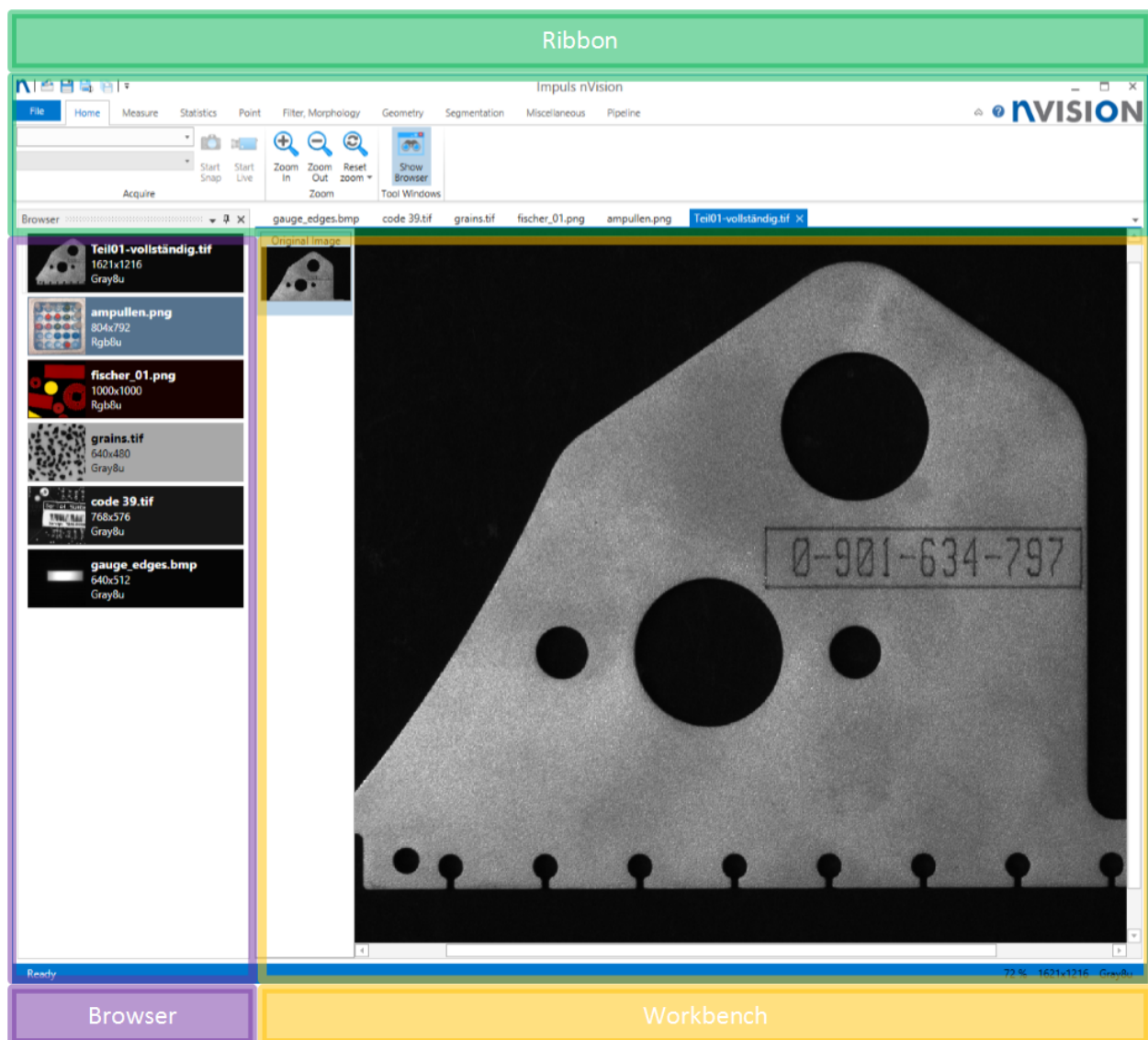


Fig. 5.1: The various parts of the nVision application.

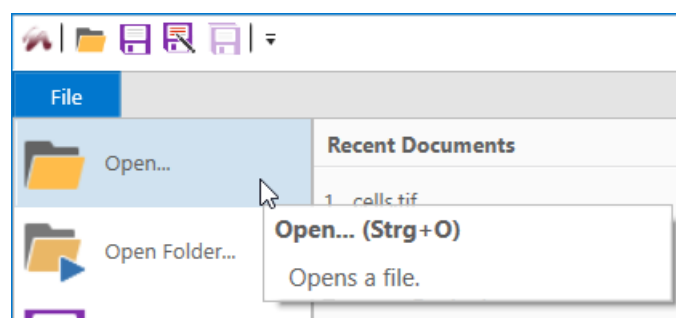


Fig. 5.2: Load an image from disk.

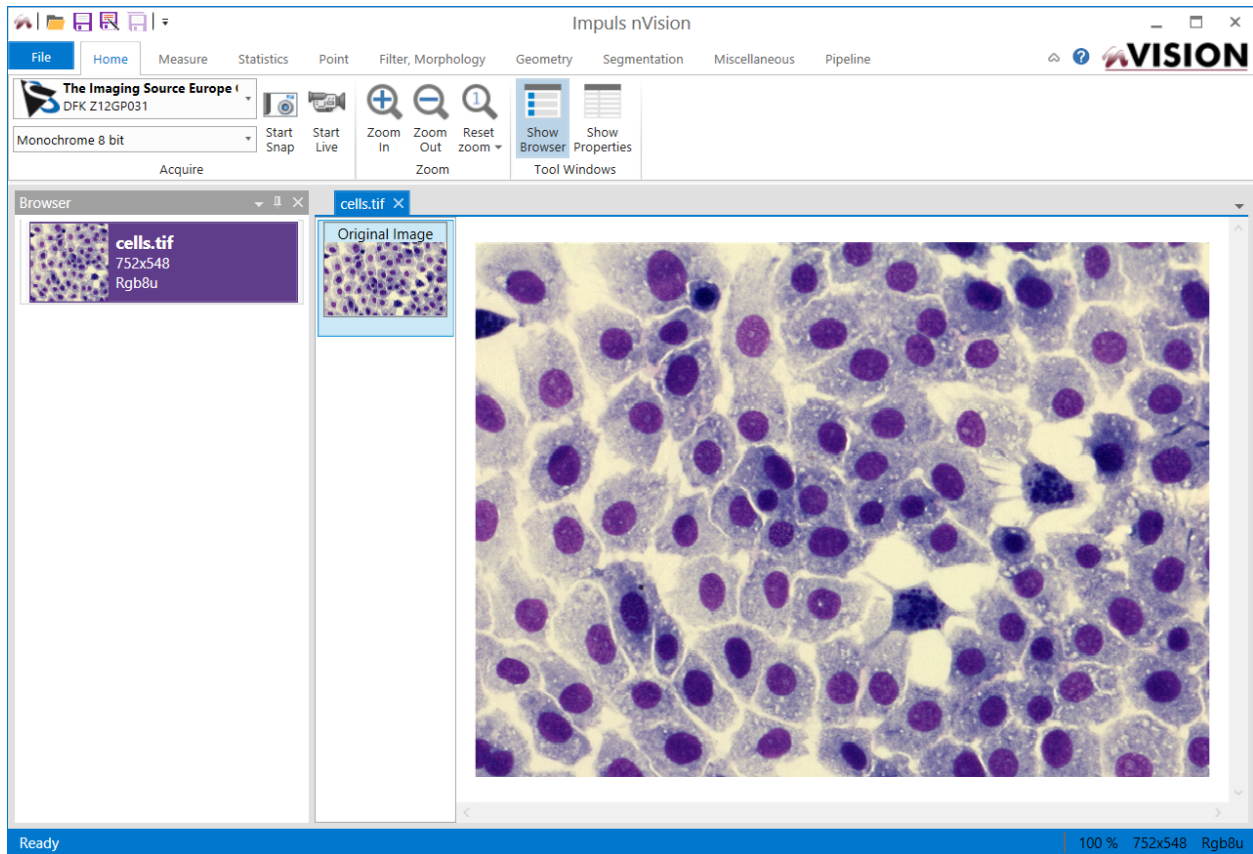


Fig. 5.3: nVision with a loaded image.

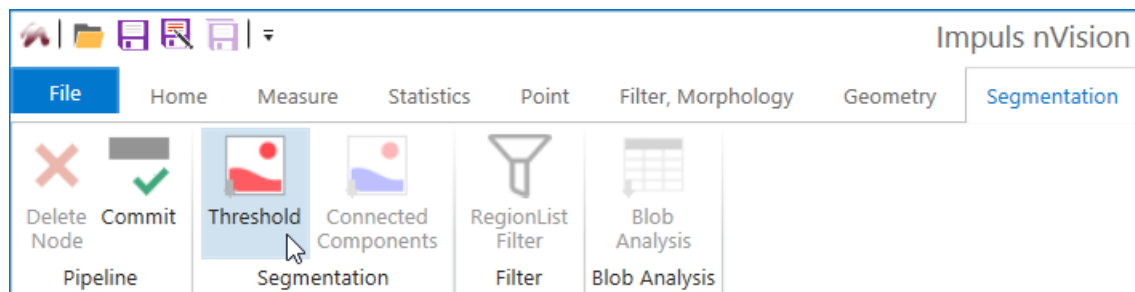


Fig. 5.4: Segmenting an image with the threshold command.

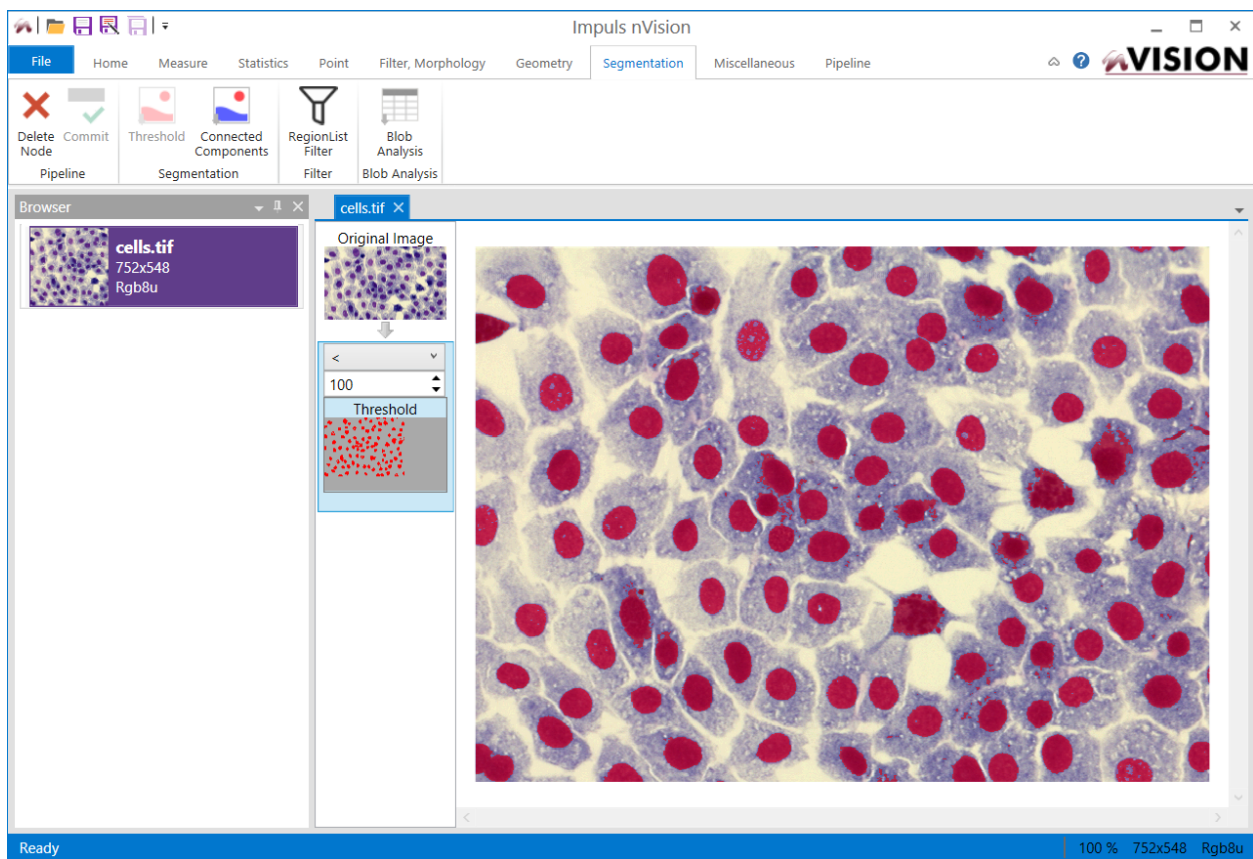


Fig. 5.5: The result of the segmentation is overlaid in red on top of the image.

From the **Segmentation** tab, use the **Connected Components** command.

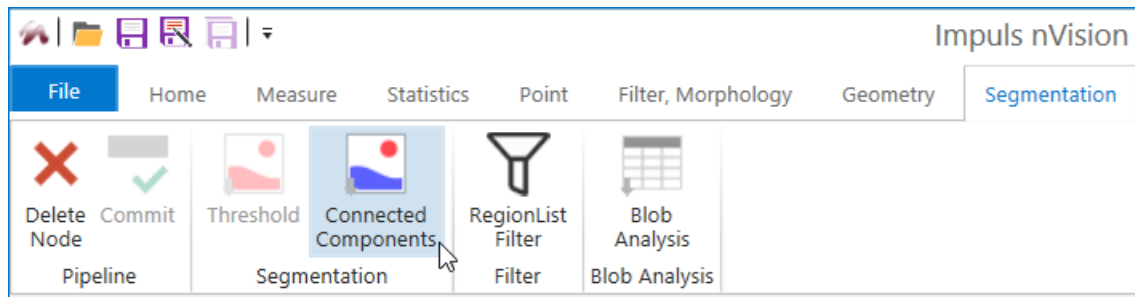


Fig. 5.6: Splitting a region into connected components.

This performs connected components analysis and splits the result from the previous segmentation step into connected objects.

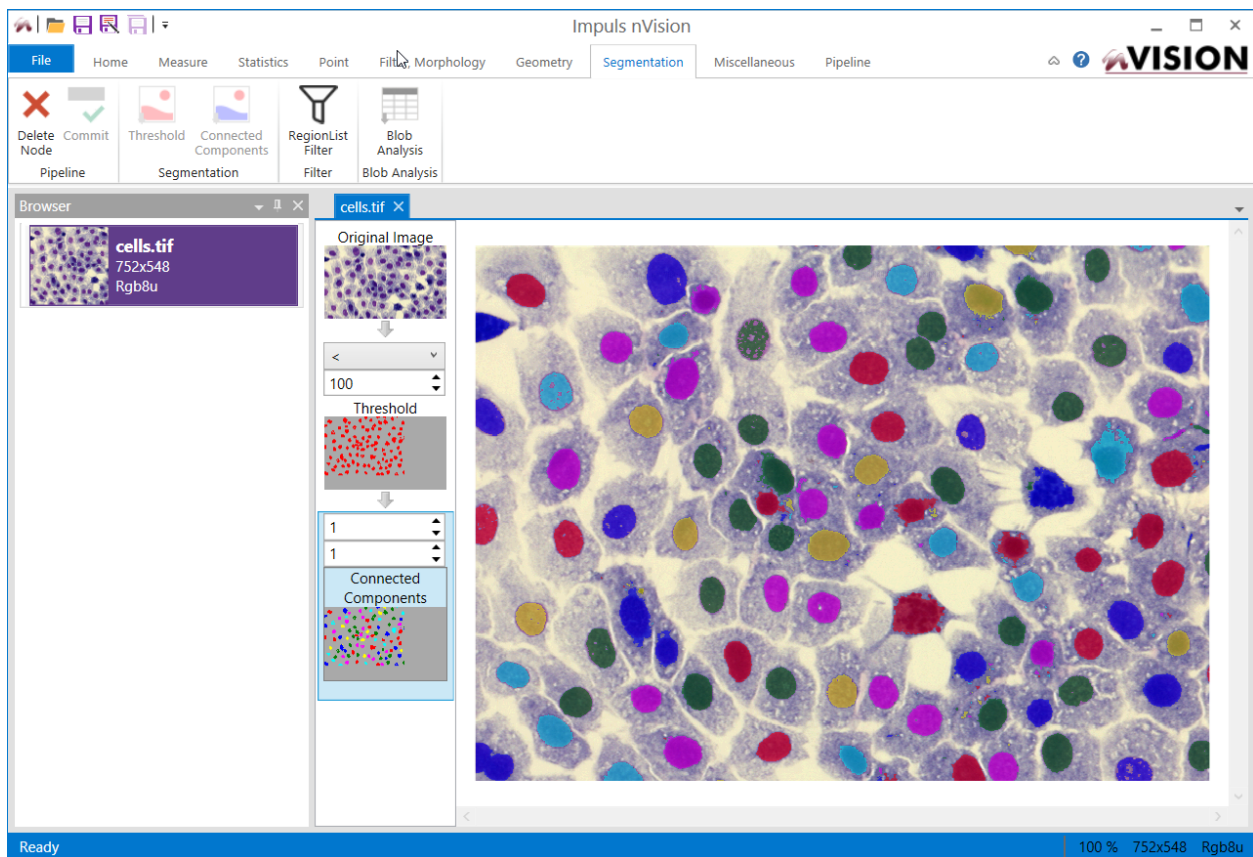


Fig. 5.7: The result of the component splitting is overlaid on top of the image in color.

The appearance of the picture has changed again: now the different cell nuclei are displayed in different overlay colors. Also, the connected components command has been appended to the linear processing pipeline, which now shows three steps.

4. Analyze particles

The final step in our short introductory example is to analyze the particles.

From the **Segmentation** tab, use the **Blob Analysis** command.

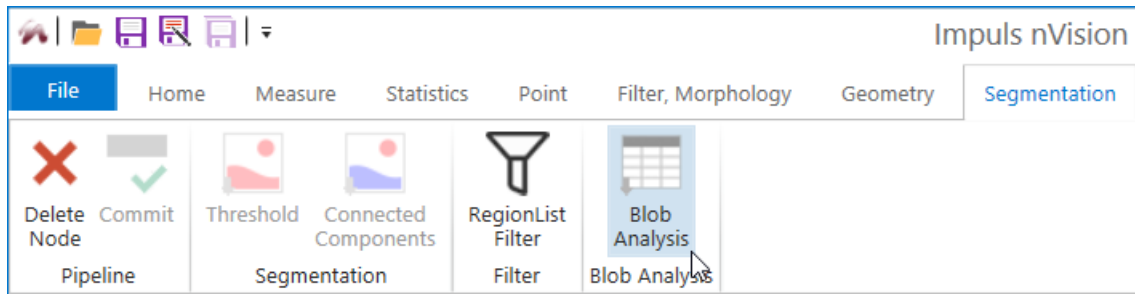


Fig. 5.8: Analyzing the blobs.

Now another step is appended to the pipeline displaying the total number of blobs, and a grid filled with numbers is displayed below the image.

For each particle in the image, there is a row in the grid showing particle measurements (also sometimes called features). Between the image and the grid is a line with some checkboxes: **Bounds** and **EquivalentEllipse**. These are features that can be graphically visualized.

Make sure that you have either **Bounds** or **EquivalentEllipse** checked and click the **Show** checkbox in the first column of any grid row. This will show either the bounding box or the equivalent ellipse of said object in a graphical fashion.

You can also click the colored objects in the graphical view, and the grid scrolls to the line of the selected object.

In addition you can click the **Commit** button on the ribbon to commit the measurement results to the browser window. From there, the data can be exported to a CSV (comma separated values) file, which can be loaded into Excel or any other programs for further manipulation.

nVision has built a pipeline of commands as you have issued the commands. This pipeline can be re-used and applied to other images. In fact, you have built a little program, but you have done so without any programming. If you close **nVision**, this pipeline will be saved along the image (if the image is called cells.tif, the associated pipeline will be called cells.tif.xpipe).

We hope that this little tutorial has whetted your appetite for more and that you continue reading and working with **nVision**. Enjoy!

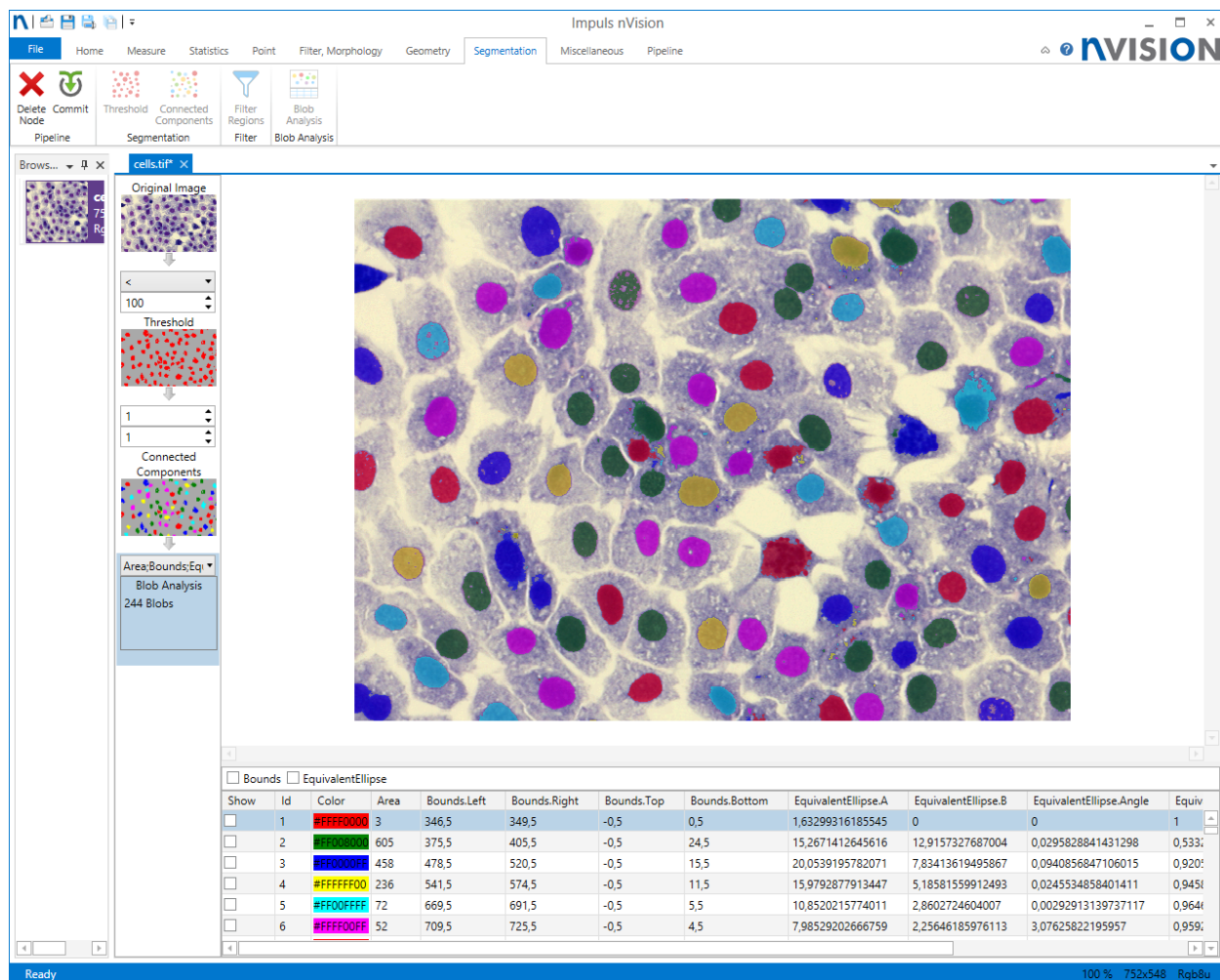


Fig. 5.9: The results of the blob analysis.

CHAPTER 6

Ribbon

The ribbon shows commands used to control **nVision**. The commands are grouped into tabs and then further into groups.

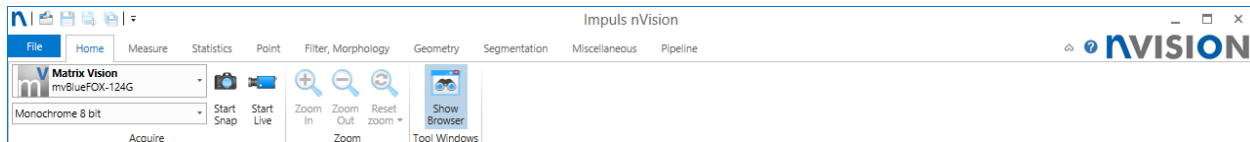









Fig. 6.1: The ribbon.

At the top of the ribbon there are some quick access commands.

	Shortcut	Command	Description
	CTRL-O		Opens a file.
			Saves to a file.
			Saves to a file under a new name.
			Saves all files.
			Deletes the selected node.
			Commits the results of the selected node as a document.

At the right of the ribbon, besides the **nVision** logo, there is the Help button. If you click on this button, the help window is shown. The help window provides context sensitive help as well as internet links to educational videos and this manual.

	Shortcut	Command	Description
	F1	Help	Displays documentation in a browser window.

Besides, still at the right, there is a button with a little upwards pointing arrow. Click on this button to minimize the ribbon.

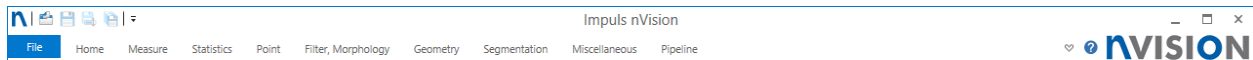


Fig. 6.2: The ribbon in its minimized state.

If the ribbon is in minimized mode, you can still access the commands by first clicking on the respective tab.

6.1 File

The file tab at the very left shows commands to open and save files, as well as to import and export pipelines.

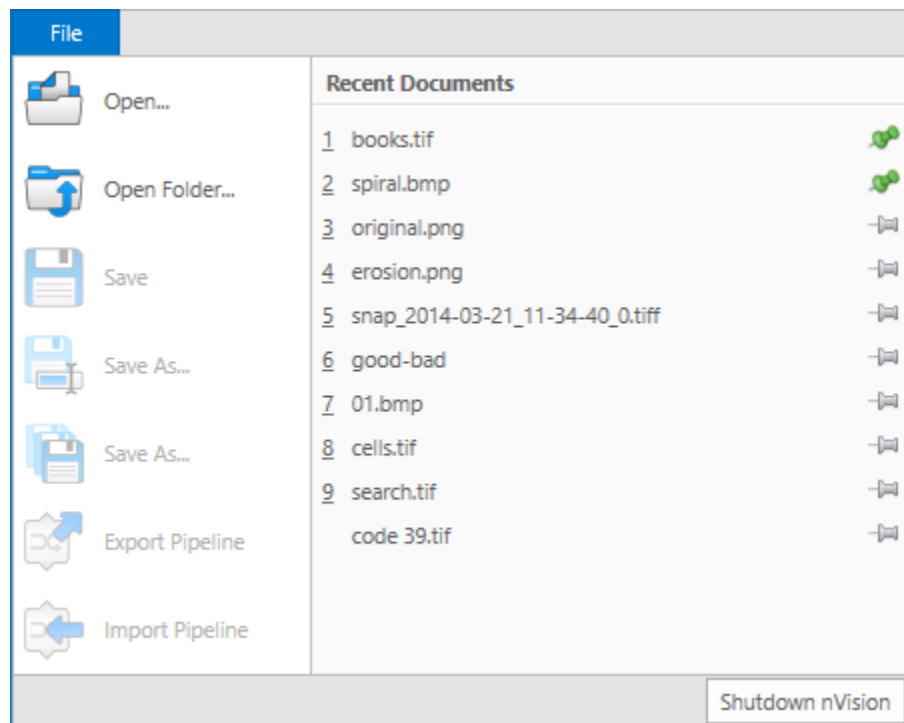


Fig. 6.3: The file menu with the list of recently opened files.

In addition, it shows a list of recently loaded documents. Documents on the recently loaded list can be pinned, so that they are kept at the top of the list.

At the bottom, there is also a button to shut down **nVision**.

6.2 Home

The **Home** tab shows the most common commands.

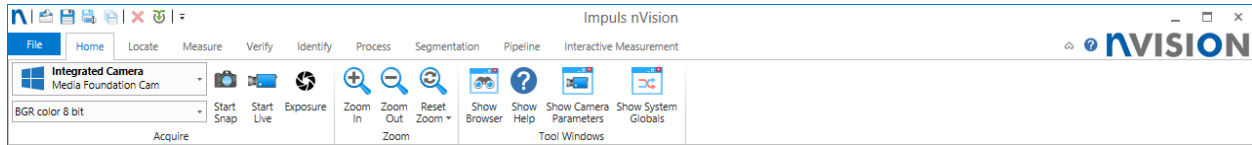







Fig. 6.4: The home tab.




At the left – in the **Acquire** group - there are selection boxes for cameras and the color mode of the camera (these are filled with entries, once a supported camera is properly installed) as well as two buttons that allow you to take a snapshot (Start Snap) from a camera or put the camera into live acquisition (Start Live/Stop Live).

	Shortcut	Command	Description
		Start/Stop Snap	Snaps a picture.
		Start/Stop Live	Starts or stops live acquisition.




Then - in the **Adjust Camera** - there are commands to adjust exposure and calibration of the camera.

	Shortcut	Command	Description
		Exposure	Inserts the exposure control tool.
		Exposure	Inserts the define scale tool.
		Exposure	Inserts the calibration tool.





Then – in the **Zoom** group – there are commands to set the zoom factor. You can **Zoom In**, **Zoom Out**, or reset the zoom (**Reset Zoom**). The **Reset Zoom** button has a drop-down menu with additional zoom commands: **Zoom to Fit Width**, **Zoom to Fit Height** and **Zoom to Fit**. These commands allow you to zoom an image to fit the window width or height or both.

	Shortcut	Command	Description
	CTRL+ +	Zoom In	Zoom in to show more details.
	CTRL+ -	Zoom Out	Zoom out to show more surrounding.
	CTRL+ALT+0	Reset Zoom	Resets the zoom factor.

Below the **Reset Zoom** command there is a submenu with additional related commands.

	Shortcut	Command	Description
		Zoom Fit	Set the zoom so that both width and height fit.
		Zoom to Fit Width	Set the zoom so that the width fits.
		Zoom to Fit Height	Set the zoom so that the height fits.

Then – in the **Tool Windows** group – there are buttons that allow you to show or hide various additional windows.

	Shortcut	Command	Description
	F11	Show Browser	Show/hide the browser.
	F1	Show Help	Show/hide the help window.
		Show Camera Parameters	Show/hide the camera parameters window.
		Show System Globals	Show/hide the system globals window.

6.3 Locate

The **Locate** tab shows commands that insert tools for part location. Part location very often is the first step in an inspection task.

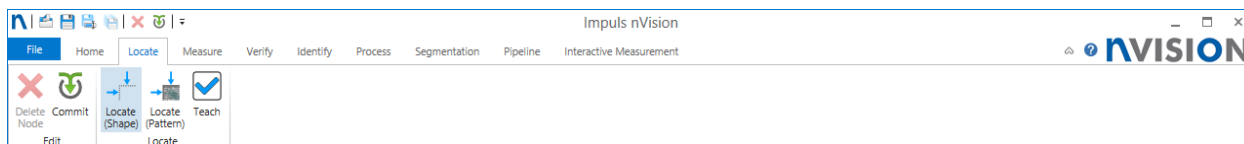







Fig. 6.5: The locate tab.

The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.

The **Locate** group contains the commands for localization.

	Shortcut	Command	Description
		Locate (Shape)	Locates a part using shapes (geometrical shape matching).
		Locate (Pattern)	Locates a part using pattern matching (normalized correlation).
		Teach	Teaches a part for subsequent location.

6.4 Measure

The **Measure** tab shows commands that insert tools for part measurement. Measurement of intensity or color, of geometric dimensions, as well as counting of features are often steps in an inspection task.

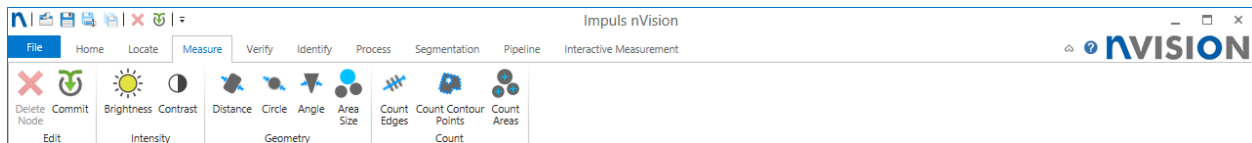






Fig. 6.6: The measure tab.





The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.




The **Intensity** Group contains the commands for intensity based measurement.

	Shortcut	Command	Description
		Brightness	Inspects the brightness inside a region of interest.
		Contrast	Inspects the contrast inside a region of interest.

The **Geometry** Group contains the commands for geometry based measurement.

	Shortcut	Command	Description
		Distance	Measures a length.
		Circle	Measures a circle.
		Angle	Measures an angle.
		Area Size	Measures an area.

The **Count** Group contains the commands for feature counting.

	Shortcut	Command	Description
		Count Edges	Counts the number of edges along a line.
		Count Contour Points	Counts the number of contour points.
		Count Areas	Counts the number of distinct blobs.

6.5 Verify

The **Verify** tab shows commands that insert tools for part verification.

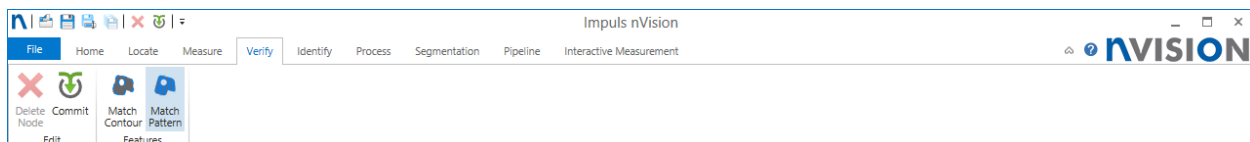






Fig. 6.7: The verify tab.

The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.

The **Features** group contains the commands for pattern matching.

	Shortcut	Command	Description
		Match Contour	Matches a region of a part using geometrical shape matching.
		Match Pattern	Matches a region of a part using normalized correlation.

6.6 Identify

The **Identify** tab shows commands that insert tools for part identification, such as decoding barcodes and matrix codes, as well as reading texts.

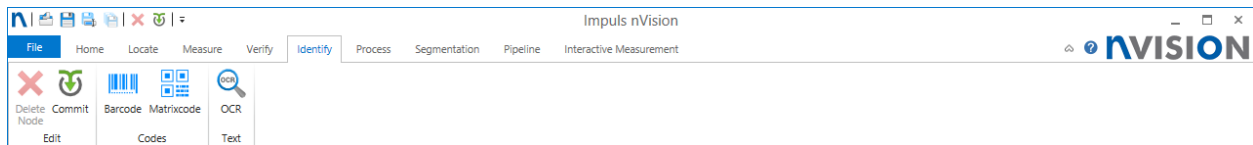


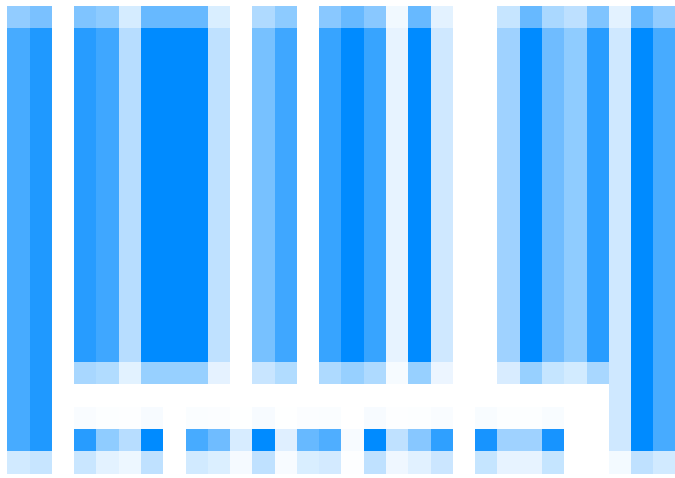



Fig. 6.8: The identify tab.


The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.

The **Codes** group contains the commands for barcode and matrix code decoding.

	Short-cut	Com-mand	Description
		Bar-code	Decodes a barcode in a re-gion of interest.
		Ma-trix-code	Decodes a matrix code in a region of interest.

The **Text** group contains the commands for optical character recognition.

	Shortcut	Command	Description
		OCR	Reads text in a region of interest using optical character recognition (OCR).

6.7 Process

The **Process** tab has commands for basic image processing. The commands are grouped into statistic operations, point processing (pixel-wise), color transformation, filtering, morphology and geometric transformations.

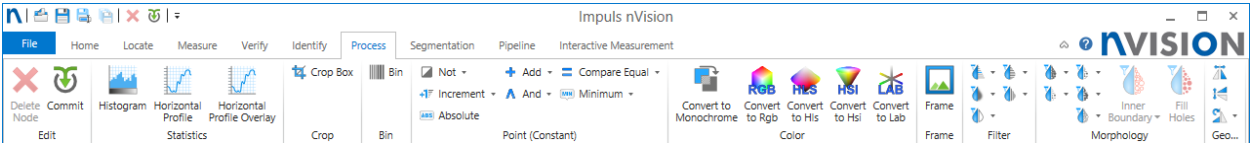







Fig. 6.9: The process tab.


The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.


The **Statistics** group contains commands for statistical processing.

	Shortcut	Command	Description
		Histogram	Calculates a histogram.
		Horizontal Profile	Calculates a horizontal profile.
		Horizontal Profile Overlay	Calculates a horizontal profile and overlays it on the image.

The **Crop** group contains the **Crop Box** command.

	Shortcut	Command	Description
		Crop Box	Crops a rectangular portion of an image.

The **Bin** group contains the **Bin** command.






	Shortcut	Command	Description
		Bin	Resize an image using binning.

The **Point** group contains point operations between an image and a constant. These operations look at single pixels only and ignore their neighborhood.


	Short-cut	Command	Description
		Not	Inverts every pixel of an image (using 1's complement).
		Negate	Inverts every pixel of an image (using 2's complement).
		Increment	Increments every pixel of an image by 1.
		Decrement	Decrements every pixel of an image by 1.
		Absolute	Takes the absolute value of every pixel of an image.
		Add	Adds a constant value to every pixel of an image.
		Subtract	Subtracts a constant value from every pixel of an image.
		Difference	Takes the absolute difference between every pixel of an image and a constant.
		Multiply	Multiplies every pixel of an image by a constant value.
		Divide	Divides every pixel of an image through a constant value.
		Logical And	And of every pixel of an image and a constant value.
		Logical Or	Or of every pixel of an image and a constant value.
		Logical Xor	Xor of every pixel of an image and a constant value.
		Equal	Compares every pixel of an image with a constant.
		Bigger	Compares every pixel of an image with a constant.
		Bigger or Equal	Compares every pixel of an image with a constant.
		Smaller	Compares every pixel of an image with a constant.
		Smaller or Equal	Compares every pixel of an image with a constant.

		Minimum	Calculates the minimum of every pixel of an image and a constant.














The **Color** group contains commands for colorspace conversions.

	Short-cut	Command	Description
		Convert to Monochrome	Converts every pixel of a color image to monochrome.
		Convert to Rgb	Converts every pixel of a color image to the RGB (red, green, blue) color space.
		Convert to Hls	Converts every pixel of a color image to the HLS (hue, luminance, saturation) color space.
		Convert to Hsi	Converts every pixel of a color image to the HSI (hue, saturation, intensity) color space.
		Convert to Lab	Converts every pixel of a color image to the LAB* color space.

The **Frame** group contains the **Frame** command.

	Shortcut	Command	Description
		Frame	Puts a frame around an image.






The **Filter** group contains linear filters. Linear filters produce their results by taking their neighbor pixels into consideration. Various filters use different filter kernels to create various filter characteristics.

	Shortcut	Command	Description
		Median	Calculates a median filter.
		Hybrid Median	Calculates a hybrid median which better preserves edges.
		Gaussian	Blurs an image with a gaussian averaging filter of varying size.
		Gauss	Blurs an image with a fixed size Gaussian kernel.
		Lowpass	Blurs an image with a lowpass filter.
		Hipass	Sharpens an image with a sharpening filter.
		Laplace	Calculates a laplace filter.
		Horizontal Sobel	Emphasizes edges with a horizontal Sobel filter.
		Horizontal Prewitt	Emphasizes edges with a horizontal Prewitt filter.
		Horizontal Scharr	Emphasizes edges with a horizontal Scharr filter.
		Vertical Sobel	Emphasizes edges with a vertical Sobel filter.
		Vertical Prewitt	Emphasizes edges with a vertical Prewitt filter.
		Vertical Scharr	Emphasizes edges with a vertical Scharr filter.

The **Morphology** group contains morphological operations, i.e. operations that change the shape. Some of these operations can be applied to images and to regions. Region versions are binary in nature and extremely fast because of their run-length implementation.

	Short-cut	Command	Description
		Dilate	Performs a dilation on image pixels. A dilation makes bright areas bigger and dark areas smaller.
		Dilate (Regions)	Performs a dilation on a region. A dilation makes a region bigger.
		Erode	Performs an erosion on image pixels. An erosion makes bright areas smaller and dark areas bigger.
		Erode (Regions)	Performs an erosion on a region. An erosion makes a region smaller.
		Open	Performs an opening on an image. Opening is an erosion followed by a dilation.
		Open (Regions)	Performs an opening on a region. Opening is an erosion followed by a dilation.
		Close	Performs a closing on an image. Closing is a dilation followed by an erosion.
		Close (Regions)	Performs a closing on a region. Closing is a dilation followed by an erosion.
		Gradient	Performs a morphological gradient on an image. Closing is a dilation minus an erosion.
		Gradient (Regions)	Performs a morphological gradient on a region. Closing is a dilation minus an erosion.
		Inner Boundary	Calculates the inner boundary of a region. The inner boundary consists of the region pixels that touch the background.
		Outer Boundary	Calculates the outer boundary of a region. The outer boundary consists of the background pixels that touch the region.
		Fill Holes	Fills holes in a region.

The **Geometry** tab contains commands for geometric transformations.

	Shortcut	Command	Description
		Mirror	Reverses right and left of an image.
		Flip	Reverses top and bottom of an image.
		Rotate Counter Clockwise	Rotate an image by 90 degrees counter clockwise.
		Rotate 180 Degrees	Rotates an image by 180 degrees.
		Rotate Clockwise	Rotates an image by 90 degrees clockwise.

6.8 Segmentation

The **Segmentation** tab groups commands related to segmentation and blob analysis.

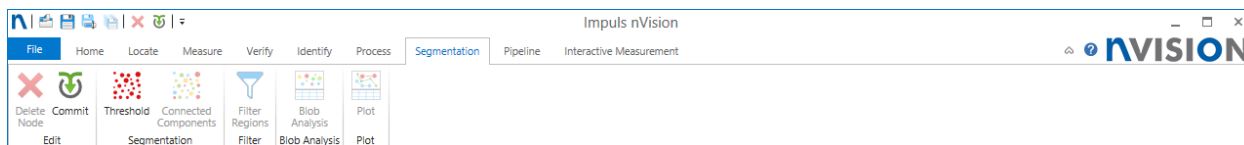







Fig. 6.10: The segmentation tab.



The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.

The **Segmentation** group contains the commands for thresholding, connected components separation and region filtering.

	Short-cut	Command	Description
		Threshold	Thresholds an image and returns a region.
		Connected Components	Splits a region into its connected components, that is a list of separate regions.
		Filter Regions	Filters regions according to a criterium.

The **Blob Analysis** group contains the commands for blob analysis and graphical plotting.

	Shortcut	Command	Description
		Blob Analysis	Performs blob analysis on a list of regions.
		Plot	Plots the results of a blob analysis.

6.9 Pipeline

The **Pipeline** tab groups pipeline related commands.

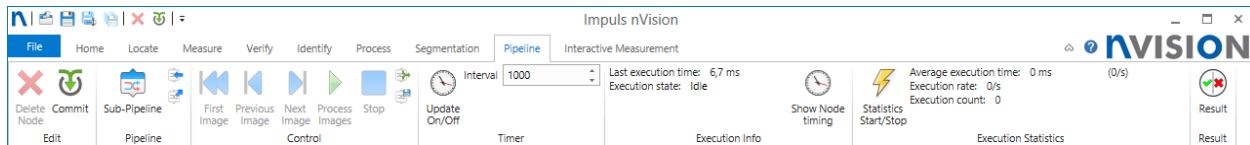







Fig. 6.11: The pipeline tab.








The **Edit** group contains the commands for pipeline editing, which are repeated on most ribbon tabs for good accessibility.

	Shortcut	Command	Description
		Delete Node	Deletes the selected node.
		Commit	Commits the results of the selected node as a document.


The **Pipeline** group contains the commands for sub-pipeline creation as well as import and export of pipelines.

	Shortcut	Command	Description
		Sub-Pipeline	Creates a sub-pipeline.
		Import Pipeline	Imports a pipeline from a file.
		Export Pipeline	Exports a pipeline to a file.


The **Control** group contains the commands for batch processing of images in folders.

	Shortcut	Command	Description
		First Image	Jumps to the first image in an image folder.
		Previous Image	Jumps to the previous image in an image folder.
		Next Image	Jumps to the next image in an image folder.
		Process Images	Processes all images in an image folder.
		Stop	Stops processing of images in an image folder.
		Export On/Off	Turns export nodes on or off.
		Export Once	Causes all export nodes to export once.


The **Timer** group allows to set the polling interval for polling nodes.

	Shortcut	Command	Description
		Update On/Off	Switch polling on/off.

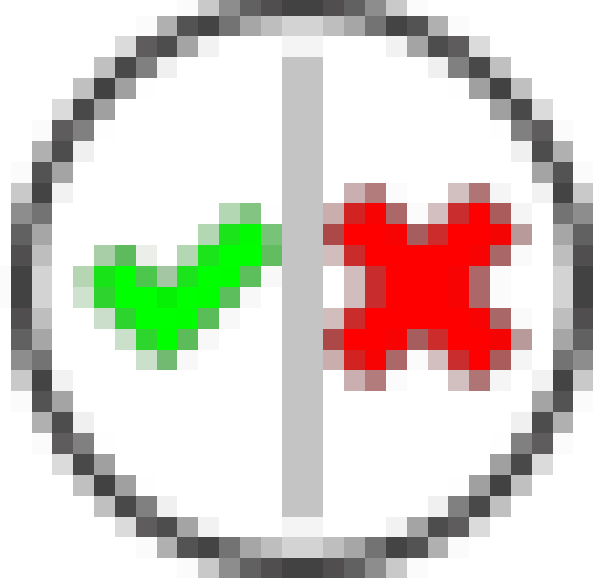
The **Execution Info** group provides information about program execution. It shows the last execution time of the program as well as the execution state.

	Shortcut	Command	Description
		Show Node Timing	Shows timing information of every node in a sub-pipeline.

The **Execution Statistics** group provides statistic information about program execution. It shows the average execution time as well as the achieved framerate.

	Shortcut	Command	Description
		Statistics Start/Stop	Switch statistics accumulation on/off.

The **Result** group contains the **Result** tool, which supports simple communication with a machine.

	Short-cut	Com-mand	Description
		Re-sult	This tool provides simple communication with a machine.

6.10 Interactive Measurement

The **Measure** tab shows commands that are related to interactive measurement.

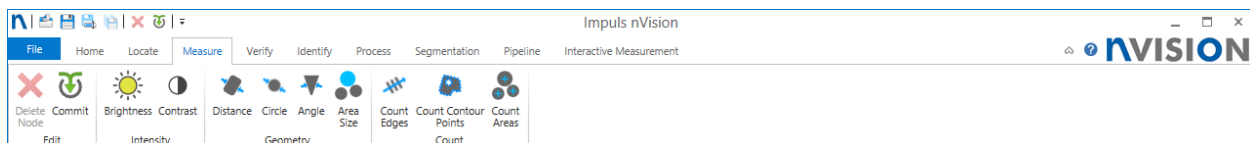







Fig. 6.12: The measure tab.


The **Calibrate** group contains the commands for interactive calibration.

	Shortcut	Command	Description
		Calibrate Origin	Toggles the origin calibration tool on or off.
		Calibrate Scale	Toggles the scale calibration tool on or off.





The **Overlay** group contains commands that switch graphical overlays on or off.

	Shortcut	Command	Description
		Rectangular Grid	Toggles the rectangular grid overlay on or off.
		Circular Grid	Toggles the circular grid overlay on or off.
		Scale Bar	Toggles the scale bar overlay on or off.


The **Picker** group contains the pixel value picker.

	Shortcut	Command	Description
		Picker	Interactively inspect pixel values.

The **Measure** group contains commands for interactive measurement.

	Shortcut	Command	Description
		Commit Results	Commits the results of the selected tool.
		Toggle the counting tool on or off.	Toggle the counting tool on or off.
		Reset Counting Tool	Resets the counting tool count.
		Measure Length Tool	Toggle the measure length tool on or off.

The **Gauging** group contains the edge inspector.

	Shortcut	Command	Description
		Horizontal Edge Inspector	This tool helps finding parameters for edge inspection.

nVision has a set of tools that are meant to make most machine vision tasks simple.

Many tasks in machine vision have similar steps, and the set of high level tools in **nVision** aims to make execution of these steps easy. The steps can be grouped into location, measuring, verification and identification.

The **Locate** menu groups the commands for part location, where location can be performed by pattern matching or shape matching.

The **Measure** menu groups various measurement tools, that can be used for intensity or dimensional measurement.

The **Verify** menu groups commands for pattern or shape verification.

The **Identify** menu groups the command for barcode and matrix code decoding, as well as for optical character recognition (OCR and OCV).

The tools are image oriented and display graphical elements on top of the image to select regions and other parameters.

In addition there are a few miscellaneous tools that do not fit into the above groups. They are documented below under the heading **Miscellaneous**.

7.1 Location

The location of a part is often a step that is executed at the beginning of a machine vision task.

7.1.1 Locate

The tools in this group provide location of parts based on features of these parts. These commands are often the first step in a sequence, because all subsequent tools respect the position and angle that these tools determine.

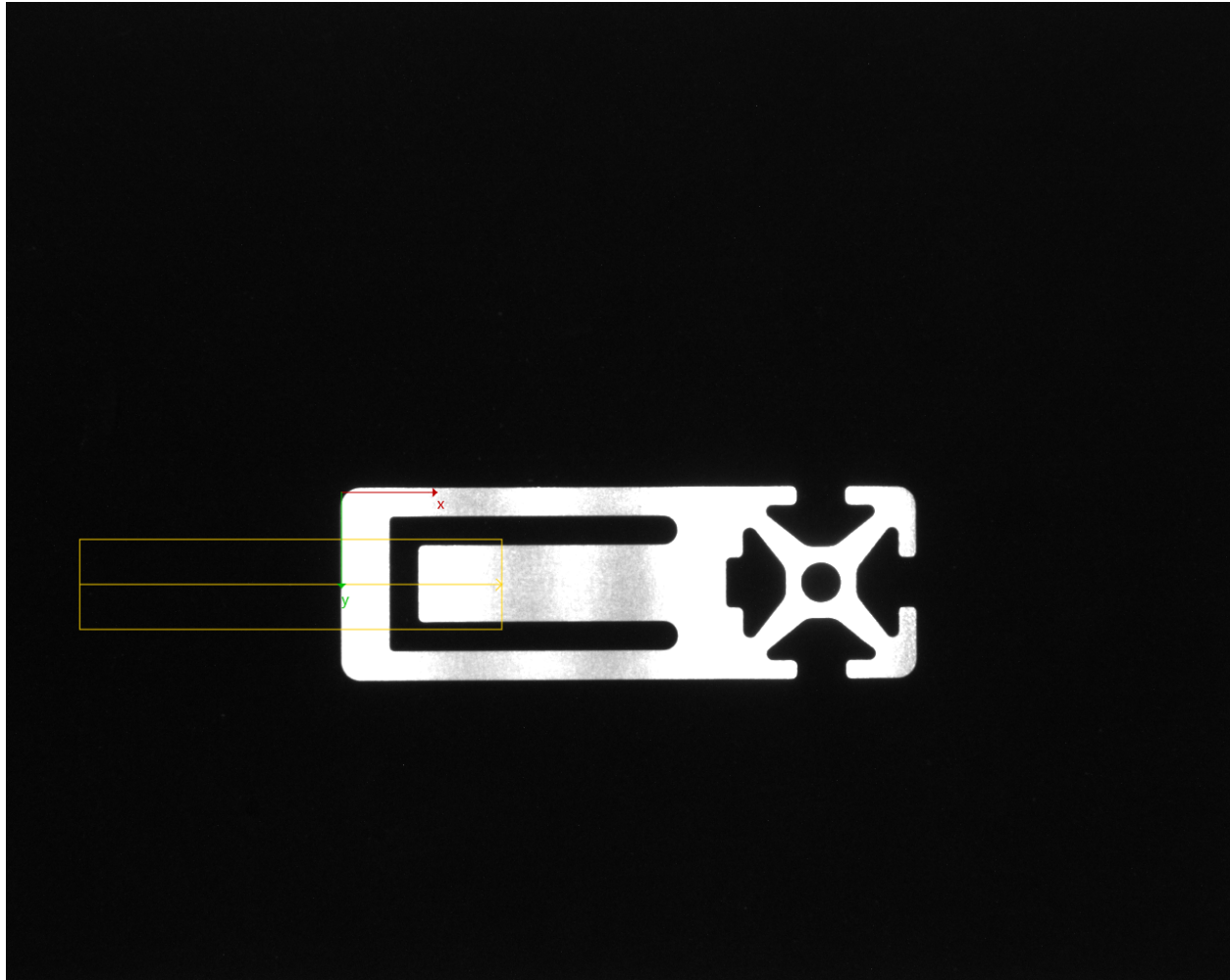
The locate tools determine a pose (a position as well as an angle) that can be used to adjust other tools regions of interest. If no location tool is used, the default pose is assumed to be in the middle of the picture.

Horizontal Shift

The **Horizontal Shift** tool uses horizontal edge detection to find a horizontal location. The tool is used by clicking the **Horizontal Shift** command button.



The tool displays a region of interest that is used to detect the shift. The region of interest can be moved by the user.



The **Horizontal Shift** tool has a configuration panel, which can be used to set parameters.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Horizontal Shift

Edge Detection Parameters

Polarity:

Both

Smoothing:

2,7

Strength:

15

Vertical Shift

The **Vertical Shift** tool uses horizontal edge detection to find a horizontal location. The tool is used by clicking the **Vertical Shift** command button.



The tool displays a region of interest that is used to detect the shift. The region of interest can be moved by the user.

The **Vertical Shift** tool has a configuration panel, which can be used to set parameters.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

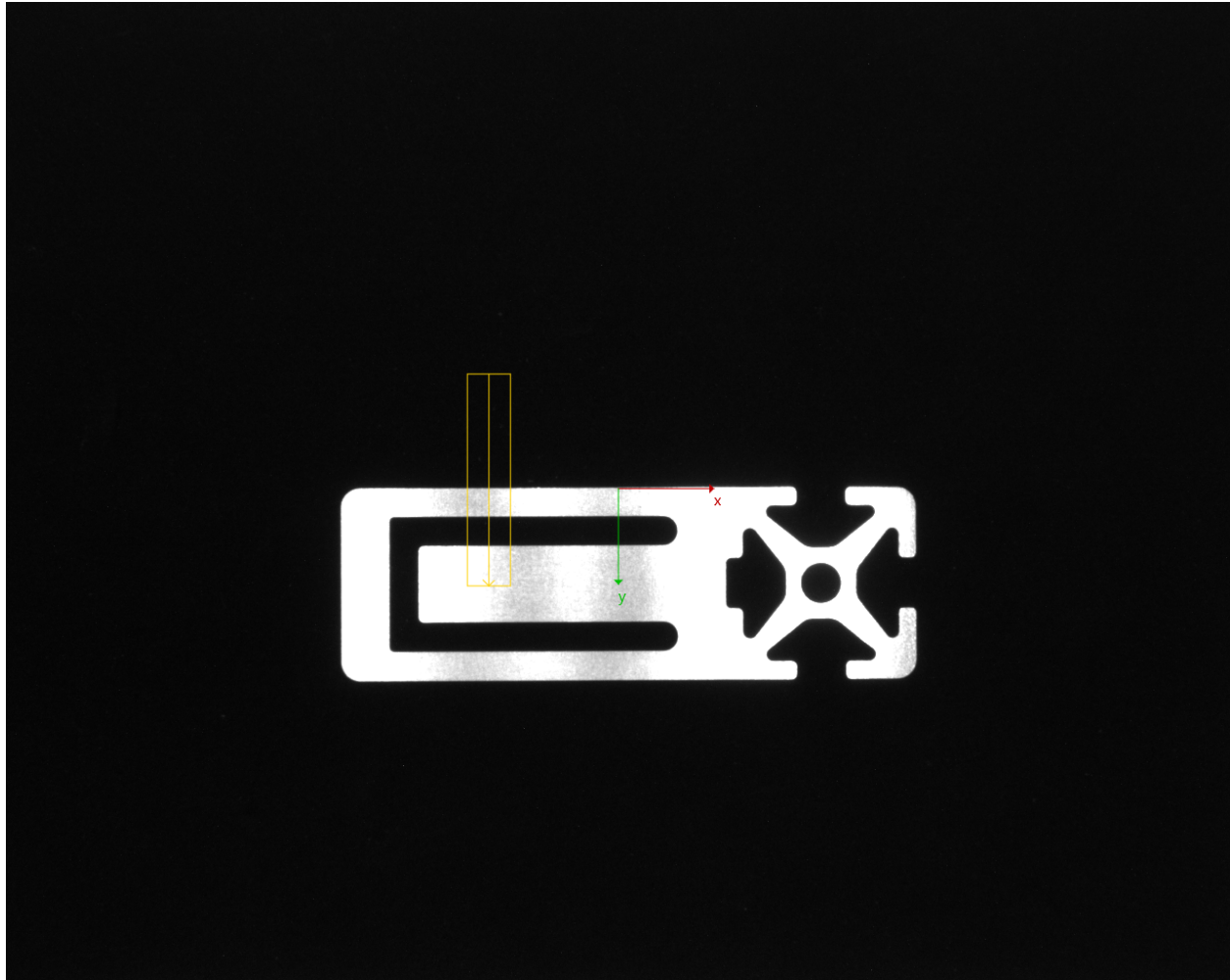
Locate (Shape)

The locate shape tool locates a part using geometric contour matching. The tool is used by clicking the **Locate (Shape)** command button.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).


Make sure that you select a characteristic portion as the pattern.



Vertical Shift

Edge Detection Parameters


Polarity:

Both 

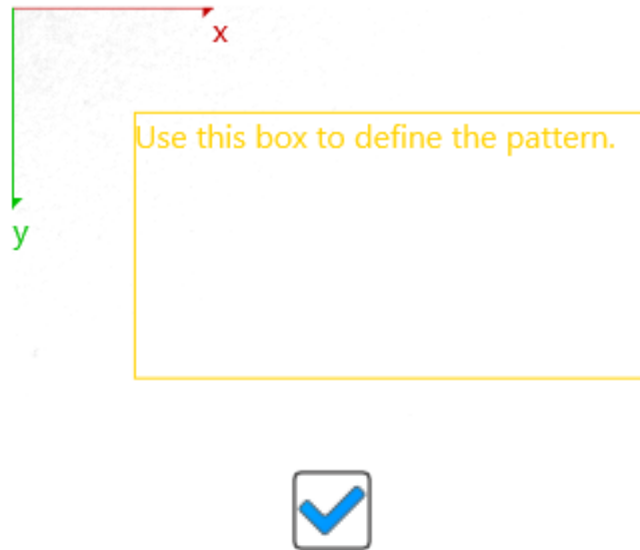
Smoothing:

2,7 

Strength:

15 





Once you have moved the region of interest to a suitable position for pattern definition, you click the **Teach** button to teach the pattern.

If the tool finds the pattern, it displays a green box where it found the pattern and also moves the coordinate axes to the middle of the box to visualize the new origin and rotation (the pose).

Subsequent tools use the pose to position their ROIs, such that the ROIs are always in the correct position, even if the part moves considerably.

The locate shape tool has a configuration panel, which can be used to set parameters.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.6). Matches below this score are not considered.

Locate (Pattern)

The locate pattern tool locates a part using pattern matching. The tool is used by clicking the **Locate (Pattern)** command button.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point or just a bit outside).

Make sure that you select a characteristic portion as the pattern.

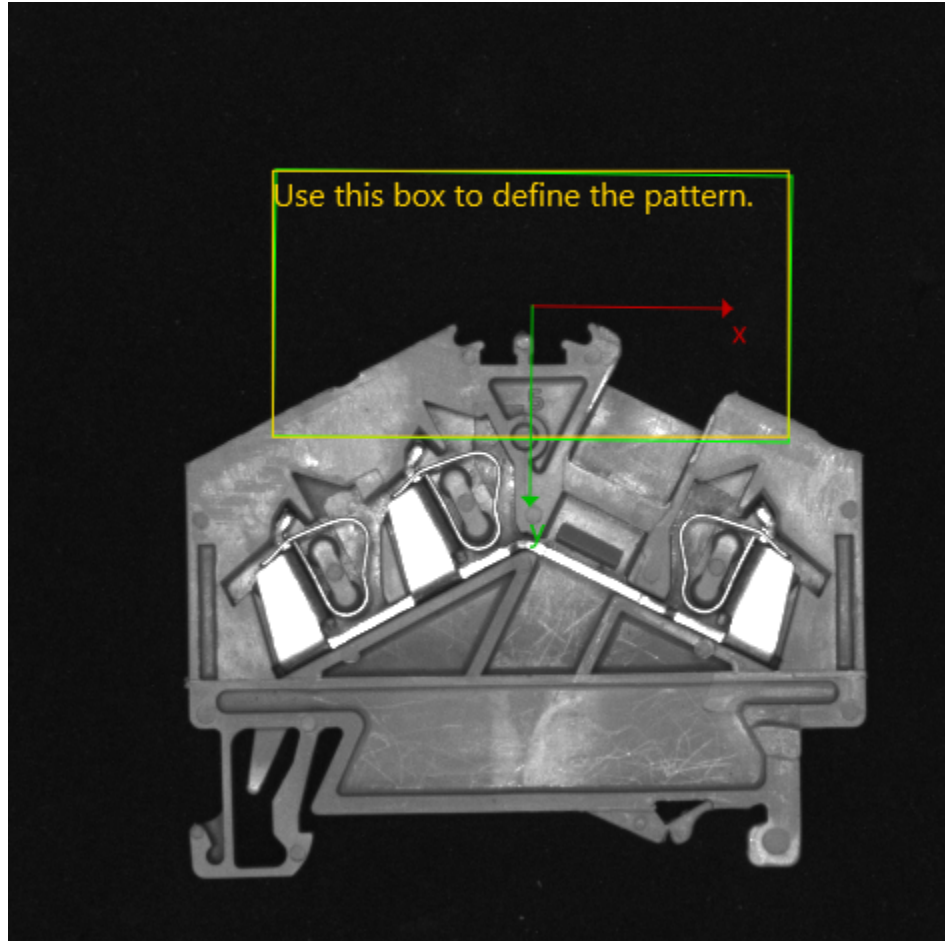
Once you have moved the region of interest to a suitable position for pattern definition, you click the **Teach** button to teach the pattern.


If the tool finds the pattern, it displays a green box where it found the pattern and also moves the coordinate axes to the middle of the box to visualize the new origin and rotation (the pose).

Subsequent tools use the pose to position their ROIs, such that the ROIs are always in the correct position, even if the part moves considerably.

The locate pattern tool has a configuration panel, which can be used to set parameters.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.8). Matches below this score are not considered.

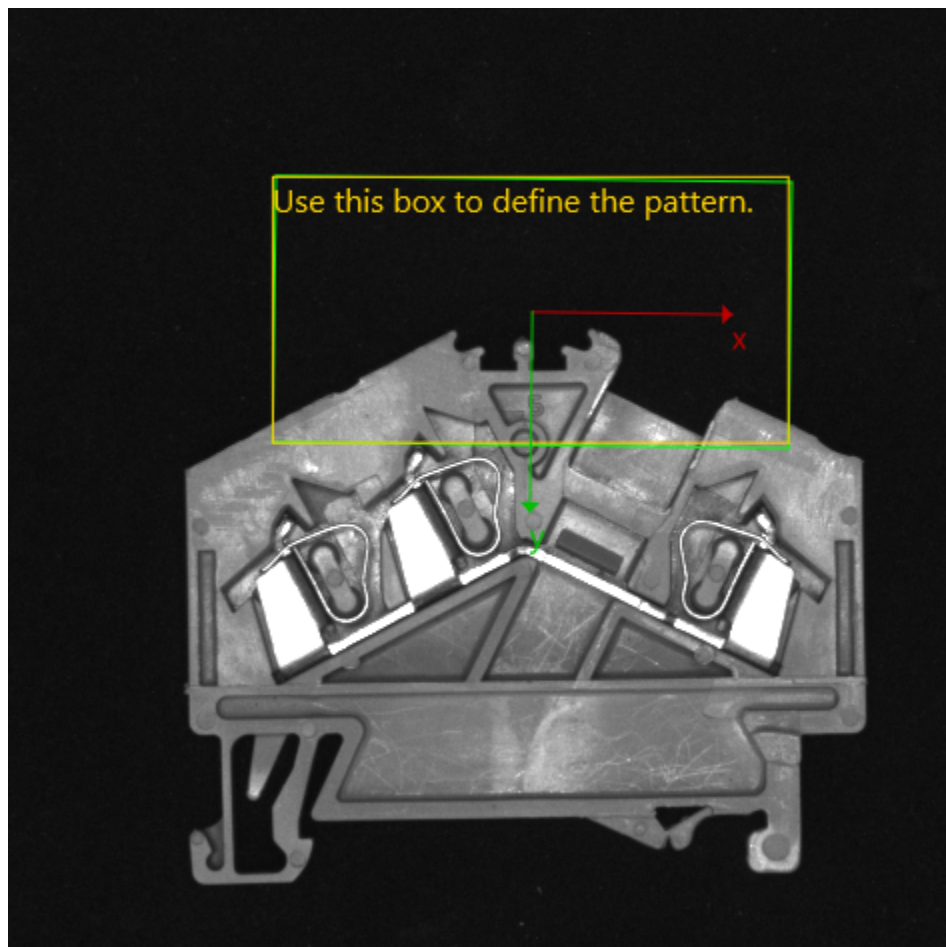
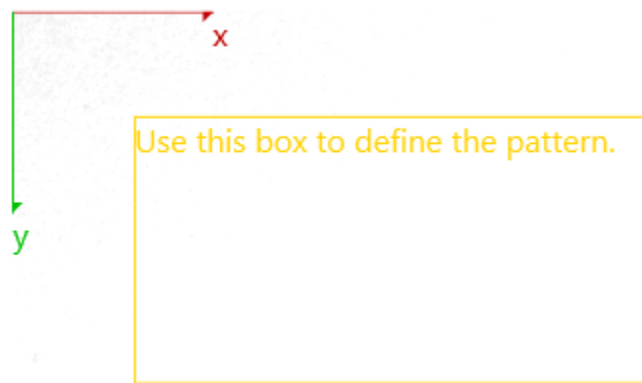


 **Locate (Shape): 0,81**

Min Score:

0,6





Locate (Pattern): 0,99

Min Score:

0,8

7.2 Measurement

7.2.1 Intensity

Intensity based tools measure features based on the greyvalues, such as brightness or contrast.

Brightness

The brightness tool measures the average brightness in a region of interest. The tool is used by clicking the **Brightness** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point or just a bit outside).

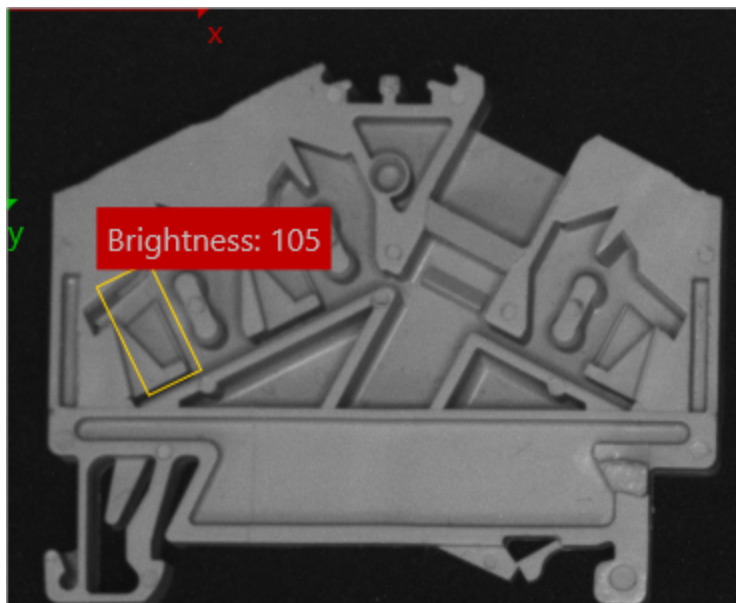
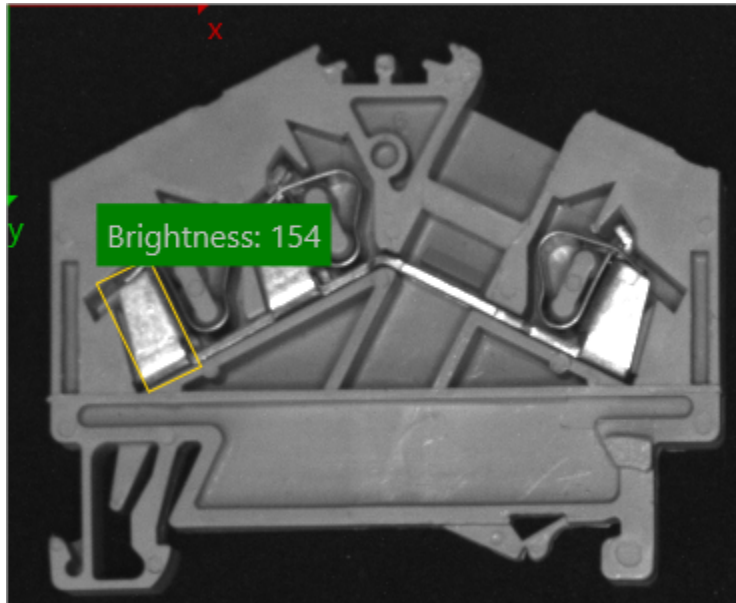


The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values (between 127 and 255 in the following example). Because of the high reflection of the metal part, the average brightness inside the region of interest is 154, which is in between the expected bounds.

The next picture shows the tool result, if the measured angle is not within the expected boundary values (between 127 and 255 in the following example). Because of the low reflection of the plastic part, the average brightness inside the region of interest is 105, which is outside the expected bounds.

The brightness tool has a configuration panel, which can be used to set parameters.

The **Min Brightness** parameter is used to set the minimal acceptable brightness (default = 127), the **Max Brightness** parameter is used to set the maximal acceptable brightness (default = 255).



Brightness: 255

Min Brightness:

Max Brightness:

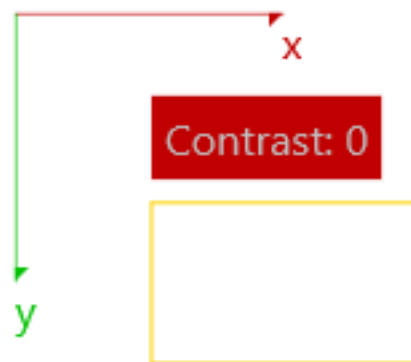
Contrast

The contrast tool measures the contrast in a region of interest. The tool is used by clicking the **Contrast** command button. Actually, the standard deviation of the greyvalues is used as a measure of the contrast.

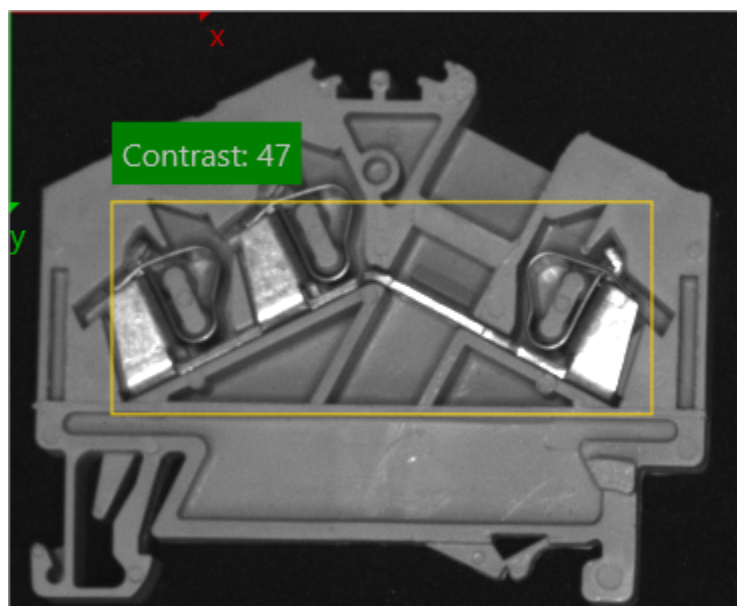


The tool displays graphical elements to specify the region of interest as well as to display the results.

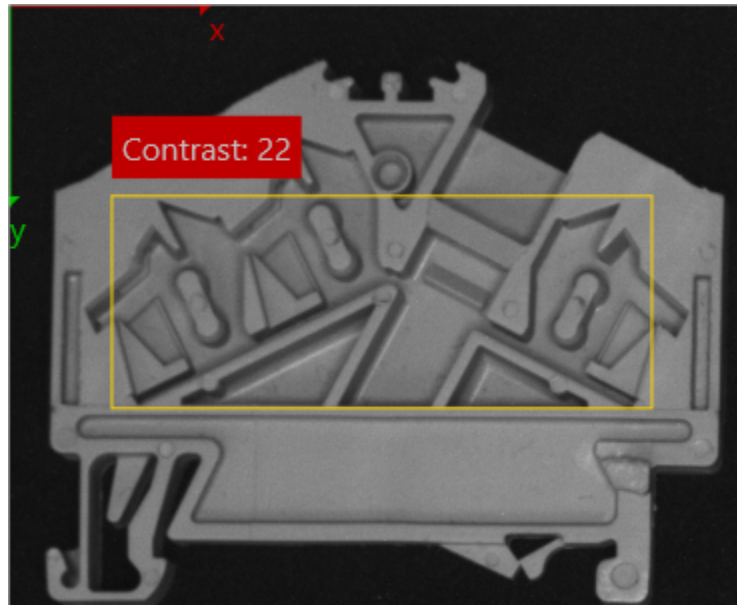
It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging it inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point or just a bit outside).




The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values (between 127 and 255 in the following example). Because of the high reflection of the metal part, the contrast inside the region of interest is 154, which is in between the expected bounds.



The next picture shows the tool result, if the measured angle is not within the expected boundary values (between 127 and 255 in the following example). Because of the low reflection of the plastic part, the contrast inside the region of interest is 105, which is outside the expected bounds.



The contrast tool has a configuration panel, which can be used to set parameters.


Contrast: 0

Min Contrast:

Max Contrast:

The **Min Contrast** parameter is used to set the minimal acceptable contrast (default = 30), the **Max Contrast** parameter is used to set the maximal acceptable contrast (default = 255).

7.2.2 Geometry

The tools in this group measure geometric features.

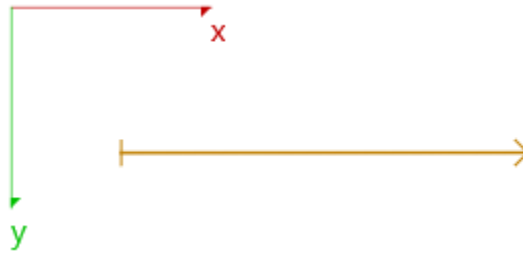
Distance

The **Distance** tool measures the distance between edges along a line. The tool is used by clicking the **Distance** command button.

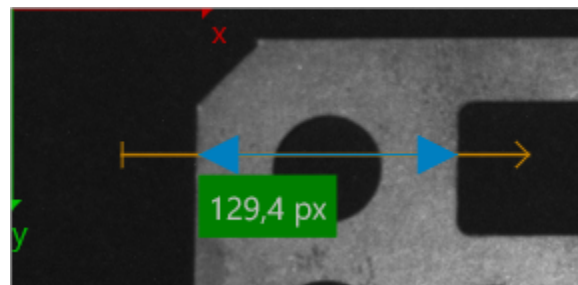


The tool displays graphical elements to specify the region of interest as well as to display the results.

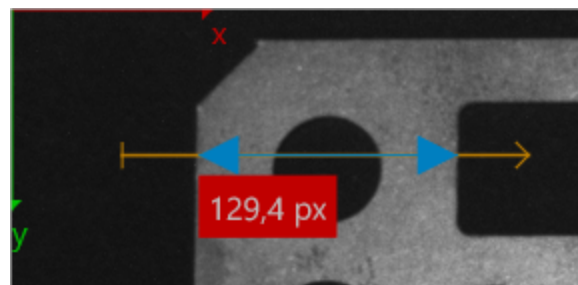
It displays a yellow line that specifies the search line. You can move the line around on the image by dragging it. You can also pick either of the line endpoints to move just this endpoint.



Once the search line has been positioned over two edges in the image it displays the calculated distance, which is visualized graphically and its numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values (between 129 px and 130 px in the following example):



The next picture shows the tool result, if the measured distance is not within the expected boundary values (between 49 px and 51 px):



This makes it easy to see the tool's outcome.

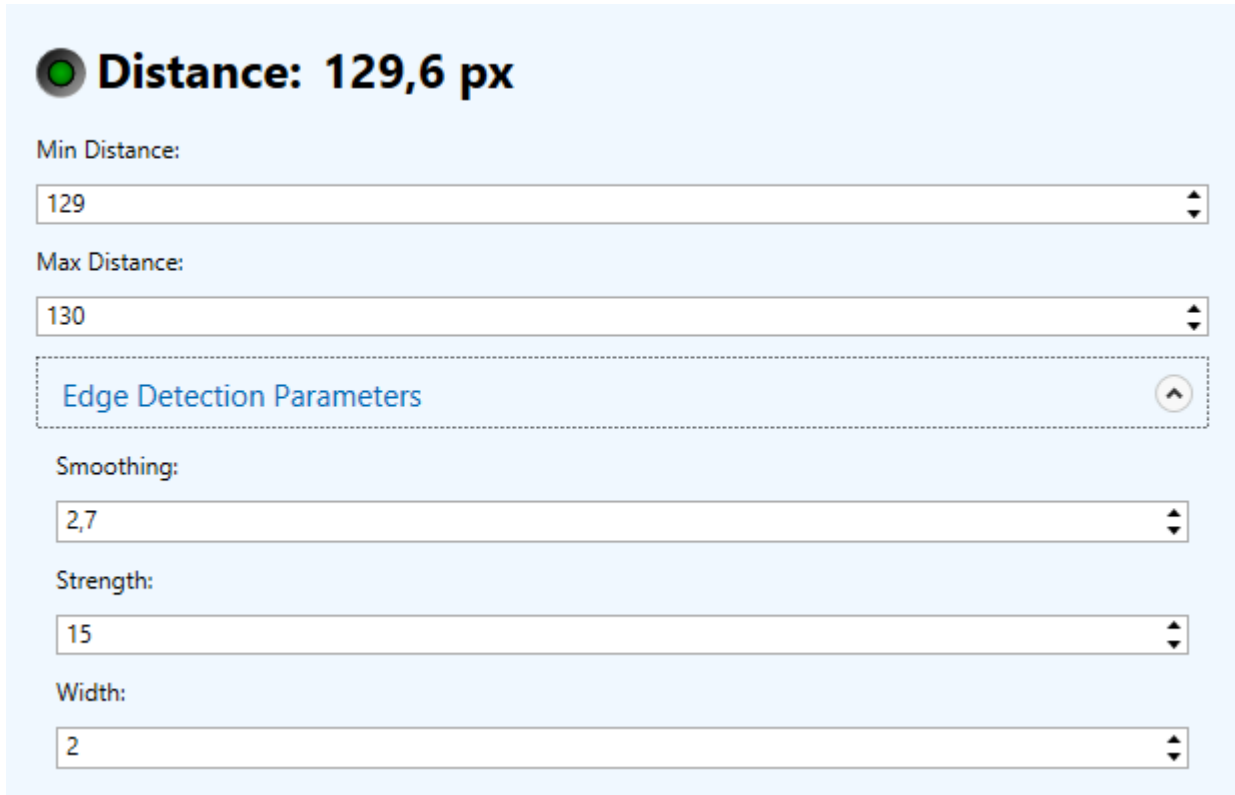
The distance tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum distance.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Width is the width of the stripe along the line region.



Distance: 129,6 px

Min Distance:

129

Max Distance:

130

Edge Detection Parameters

Smoothing:

2,7

Strength:

15

Width:

2

Distance (Two Points)

The **Distance (Two Points)** measures the distance between two points. The tool is used by clicking the **Distance (Two Points)** button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays two yellow lines labelled A and B that specify the search lines. You can move the lines around on the image by dragging them. You can also pick either of the line endpoints to prolong the line or move just this endpoint (move the mouse a little bit away from the endpoint, until the cursor changes).

Once the search lines have been positioned over two edges in the image the calculated distance, which is visualized graphically and its numerical value are displayed.

The **Distance (Two Points)** tool has a configuration panel, which can be used to set parameters.

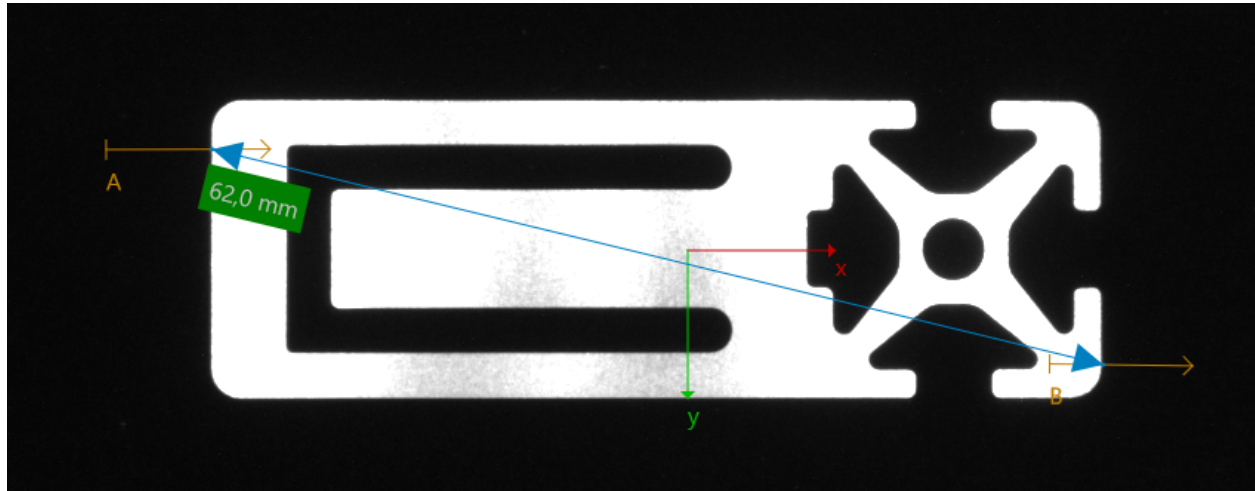
The **Min** and **Max** parameters are used to set the expected minimum and maximum distance.

The distance type between the two points can be the straight, horizontal or vertical distances.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Width is the width of the stripe along the line region.



Distance (Two Points): 62,0 mm

Min Distance:

50

Max Distance:

200

Select distance type

- ☒ Direct
- ☐ Horizontal
- ☐ Vertical

Edge Detection Parameters

Smoothing:

2,7

Strength:

11

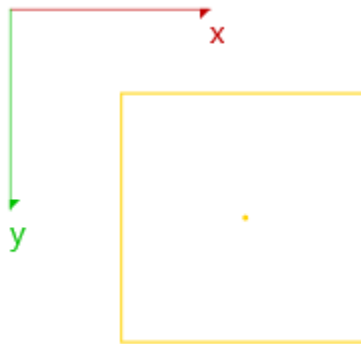
Circle

The circle tool measures a circle in a region of interest. The tool is used by clicking the **Circle** command button.

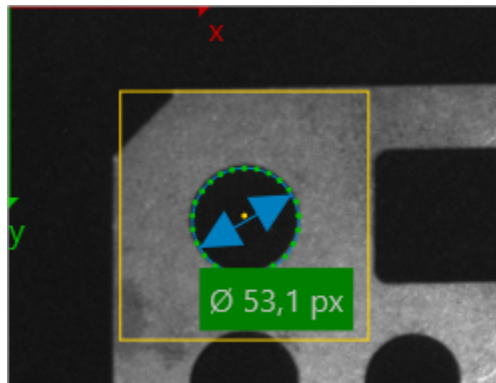


The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a yellow box that specifies the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside). Inside the box is a little yellow point that marks the center of the box.



Once the box has been positioned over a circle (or part of a circle) in the image it displays the calculated circle, which is visualized graphically and its numerical diameter is displayed in green, if it is in between the tool's minimum and maximum expected values (between 53 px and 54 px in the following example):



The next picture shows the tool result, if the measured distance is not within the expected boundary values (between 50 px and 51 px):

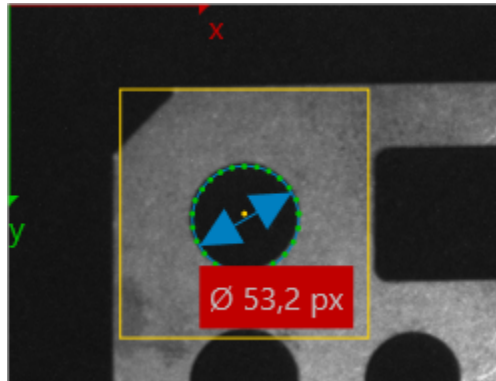
This makes it easy to see the tool's outcome.

The circle tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum diameter.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.



Circle: Ø 53,2 px

Min Ø:

53

Max Ø:

51

Edge Detection Parameters



Smoothing:

2,7

Strength:

15

Orientation:

InsideToOutside

Polarity

Rising

Spacing (°):

15

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Spacing is the angular spacing in degrees of the search lines.

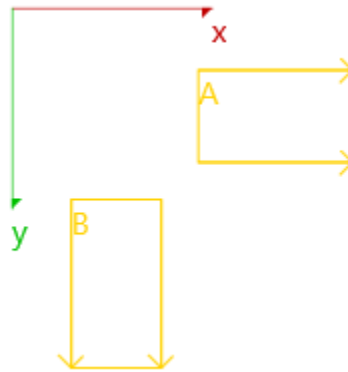
Angle

The angle tool measures the angle between two lines. The tool is used by clicking the **Angle** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays two boxes that are used to select regions, the boxes are labeled A and B. You can move the boxes around on the image by dragging their inside. You can also resize the boxes by picking them at their boundary lines (on the boundary line or just a bit outside) or at their corner points (on the corner point of just a bit outside). The arrows at the boxes visualize the search direction of the line detection (box A has horizontal arrows and should be used for vertical lines, box B has vertical arrows and should be used for horizontal lines).



Once the search box has been positioned over an edge in the image it displays the calculated edge points in green, and a resulting line in blue that it calculates by fitting a line through the points.

The tool has outlier detection as you can see in the following picture. The outlier points are marked in red and they are not used for the line fit.

The search boxes move with the pose, that is their position is relative to the part position that has been determined by a location tool upstream.

If the tool finds both lines it calculates the angle between those lines. The angle is visualized graphically and its numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values (between 89,7 ° and 90,3 ° in the following example):

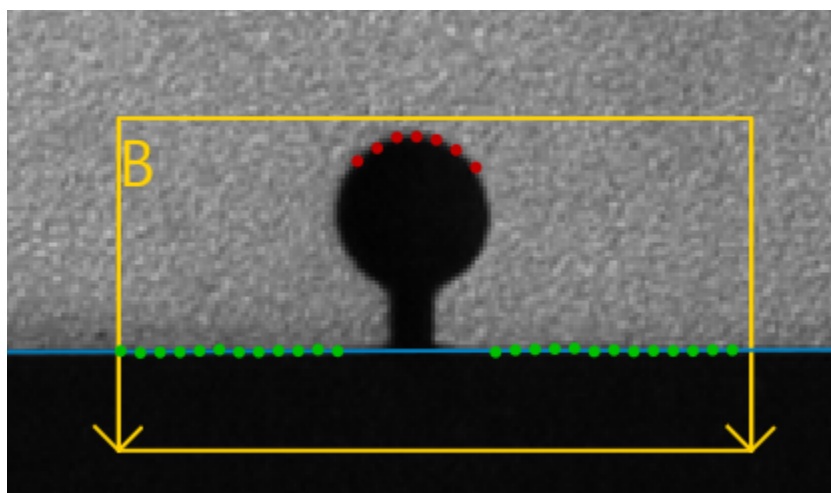
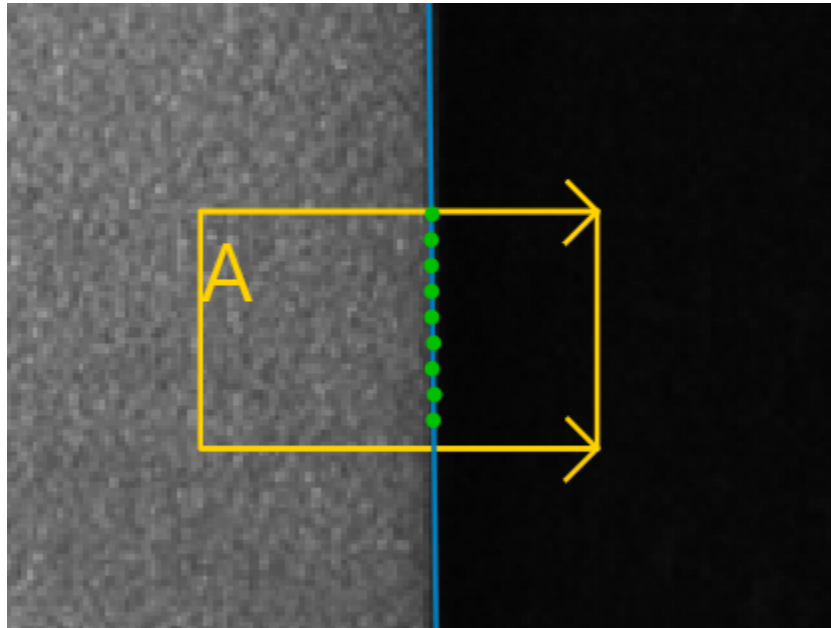
The next picture shows the tool result, if the measured angle is not within the expected boundary values (between 89,9 ° and 90,1 °):

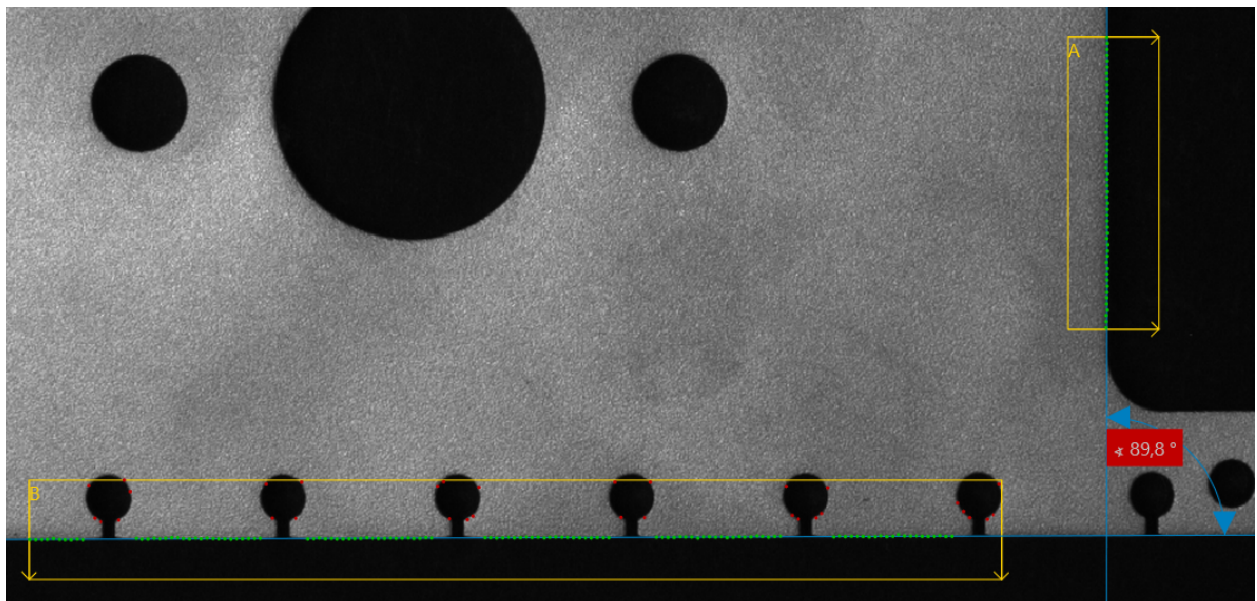
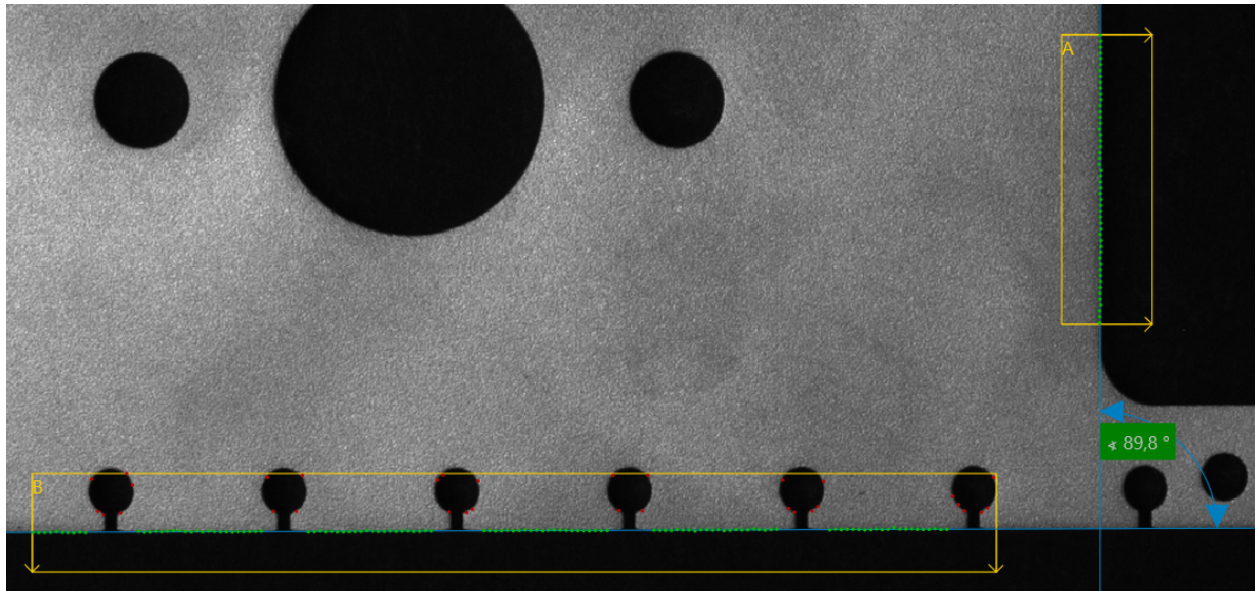
This makes it easy to see the tool's outcome.


The angle tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum angles.

The parameters in the **Edge Detection Parameters** affect the edge detection.





 **Angle:** °

Min:

0

Max:

360

Edge Detection Parameters



Smoothing:

2,7

Strength:

15

Polarity:

Both

Spacing (px):

5

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Spacing is the distance between the search lines of the edge detection (default = 5). The effect of the spacing can be seen by the density of the visualized edge points.

Area Size

The area size tool measures the object area in a region of interest. The tool is used by clicking the **Area Size** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The tool is meant to be used on regions where you have contrasting objects, either dark or bright. It automatically detects the objects. In a region without much contrast, using the tool is mostly meaningless because of noise.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values.

The next picture shows the tool result, if the measured area is not within the expected boundary values.

The area size tool has a configuration panel, which can be used to set parameters.

The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Polarity** can be set to **Dark Objects** or **Bright Objects**.

7.2.3 Count

The tools in this group count various things, and compare the number to the expected number or range.

Count Edges

The count tool counts the number of edges along a linear region of interest. The tool is used by clicking the **Count Edges** command button.

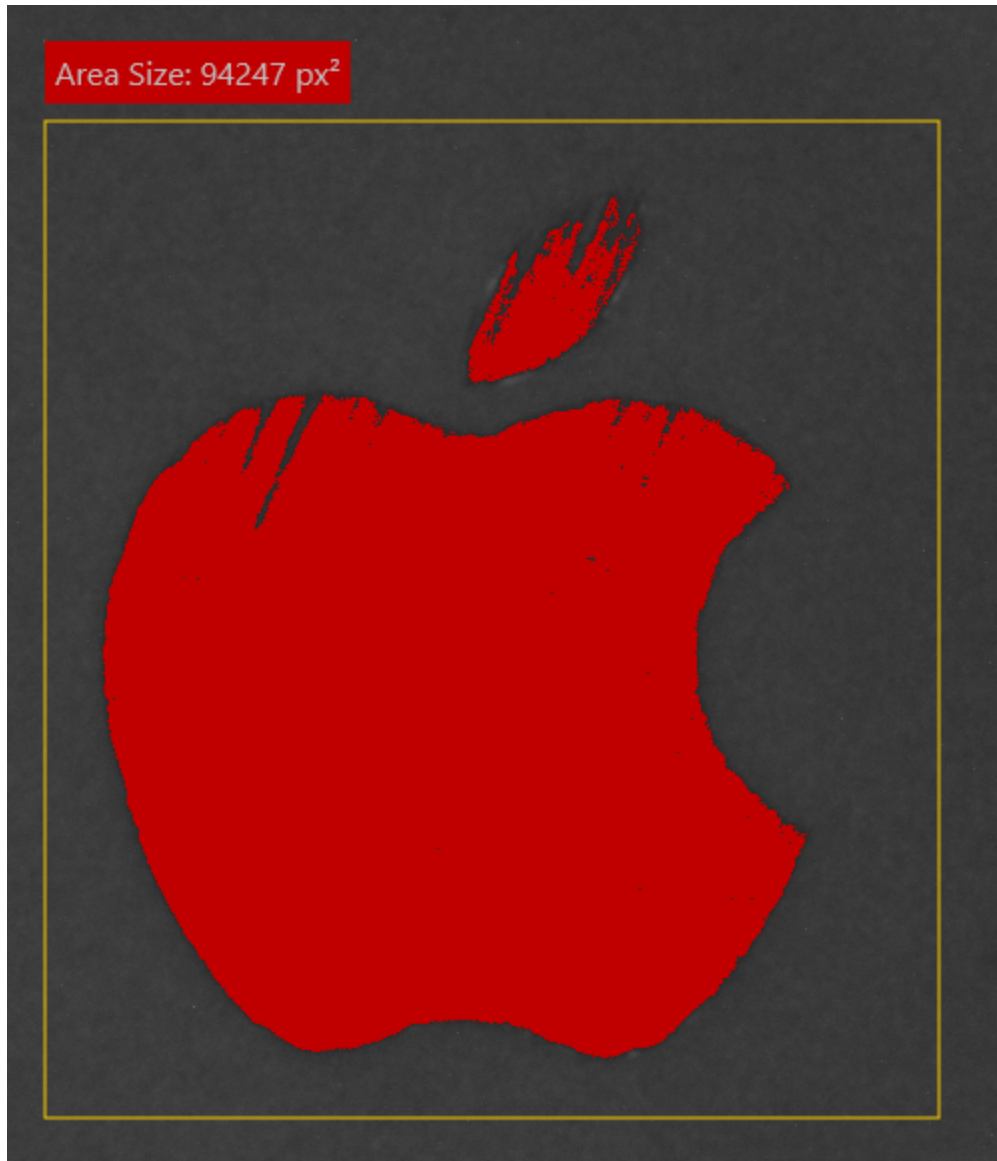
The tool displays graphical elements to specify the region of interest as well as to display the results.


It displays a yellow line that specifies the search line. You can move the line around on the image by dragging it. You can also pick either of the line endpoints to move just this endpoint.

Once the line has been positioned and finds edges, it visualizes them as blue points and it outputs their number. If the number is within expected bounds, the tool color is green.

The next picture shows the tool result, if the number of edges is not within the expected boundary values:






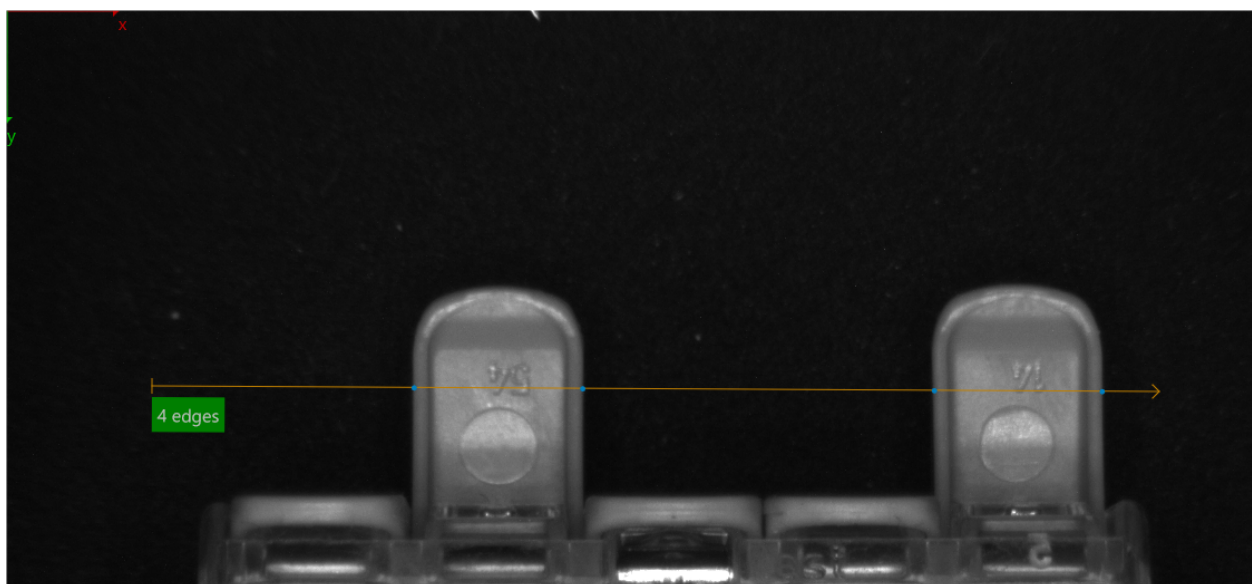
 **Area Size: 94230 px²**

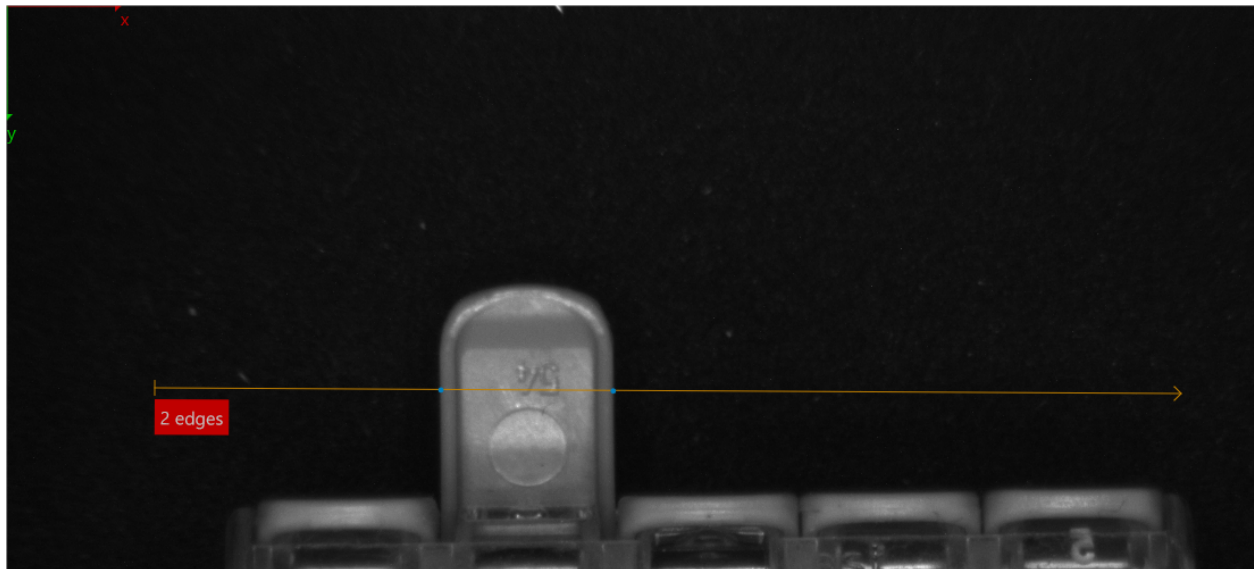
Min:

Max:

Parameters 


Polarity:






This makes it easy to see the tool's outcome.

The tool has a configuration panel, which can be used to set parameters.

 **Edges: 0**

Min:

Max:

Edge Detection Parameters 

Smoothing:

Strength:

The **Min** and **Max** parameters are used to set the expected minimum and maximum diameter.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Count Contour Points

The count contour points tool counts the number of high-contrasting pixels within a region of interest. The tool is used by clicking the **Count Contour Points** command button.

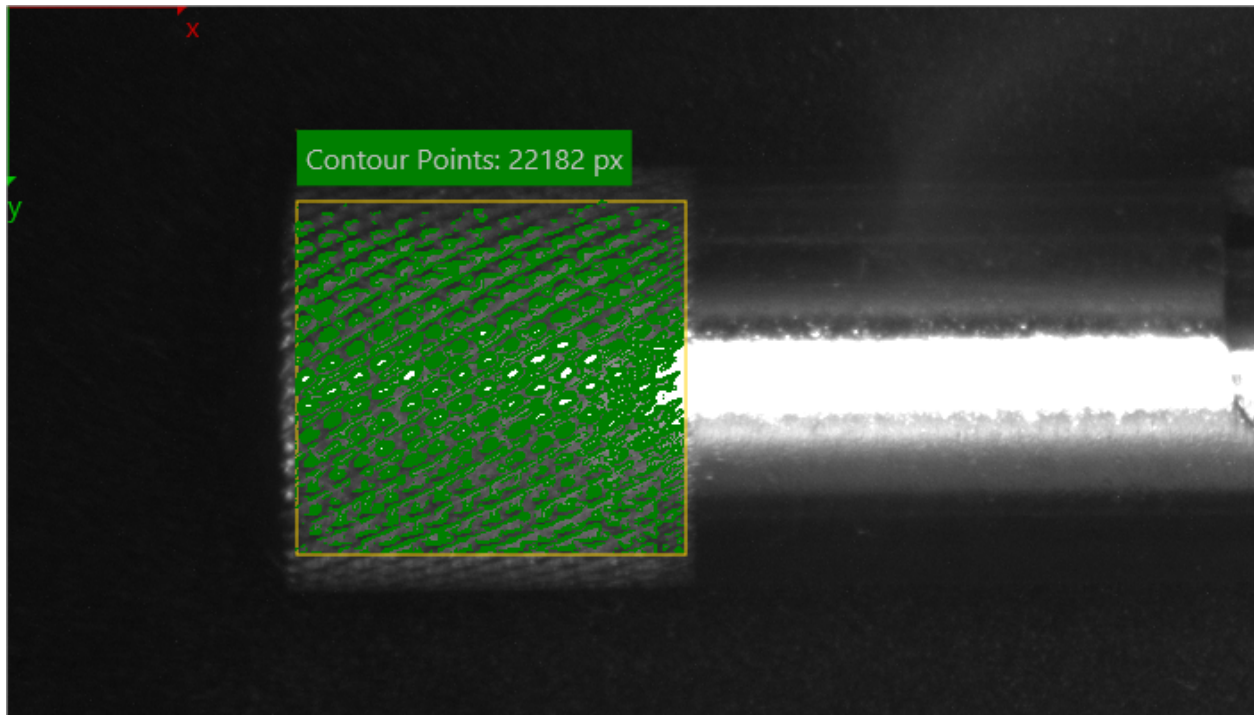


The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point or just a bit outside).

One use of the tool is to check for raffle or embossing on metal parts.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values.



The next picture shows the tool result, if the number of contour points is not within the expected boundary values.

The count contour points tool has a configuration panel, which can be used to set parameters.

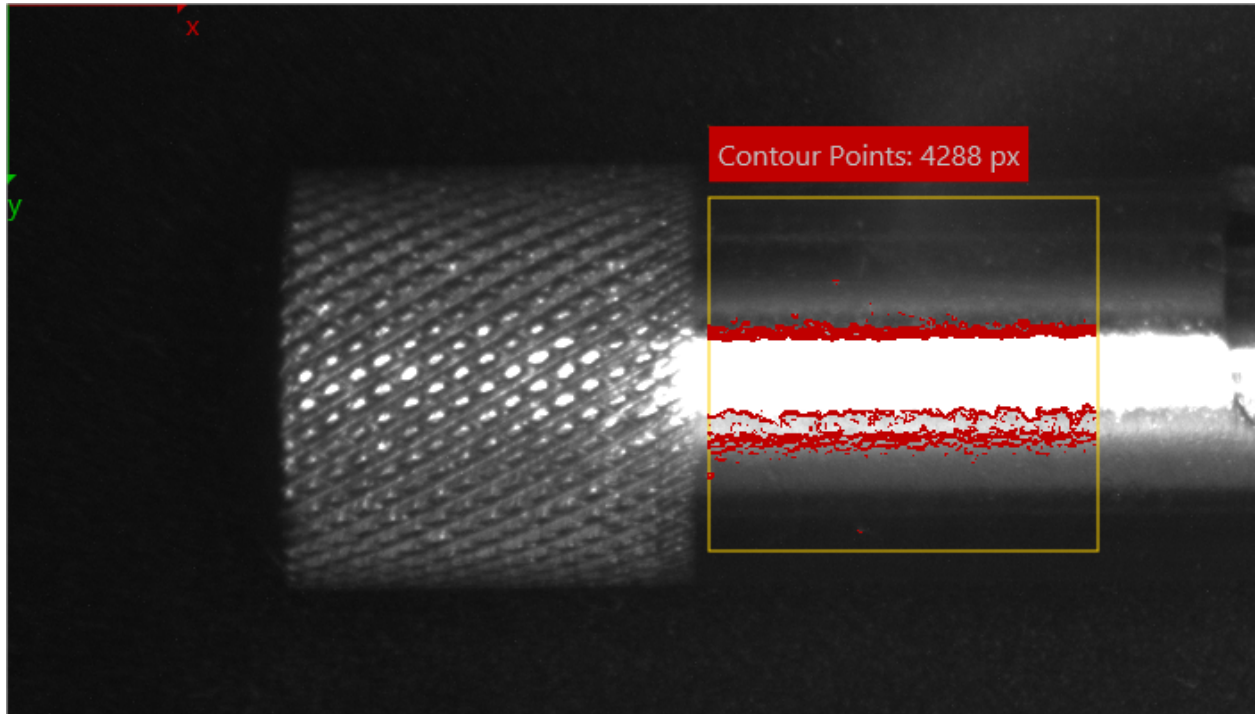
The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Sensitivity** can be set to a value between 0 (highly sensitive) and 255 (not sensitive).

Count Areas

The count areas tool counts the number of objects in a region of interest. The tool is used by clicking the **Count Areas** command button.

The tool displays graphical elements to specify the region of interest as well as to display the results.



Contour Points: 22425

Min Contour Points:

10000

Max Contour Points:

50000

Sensitivity:

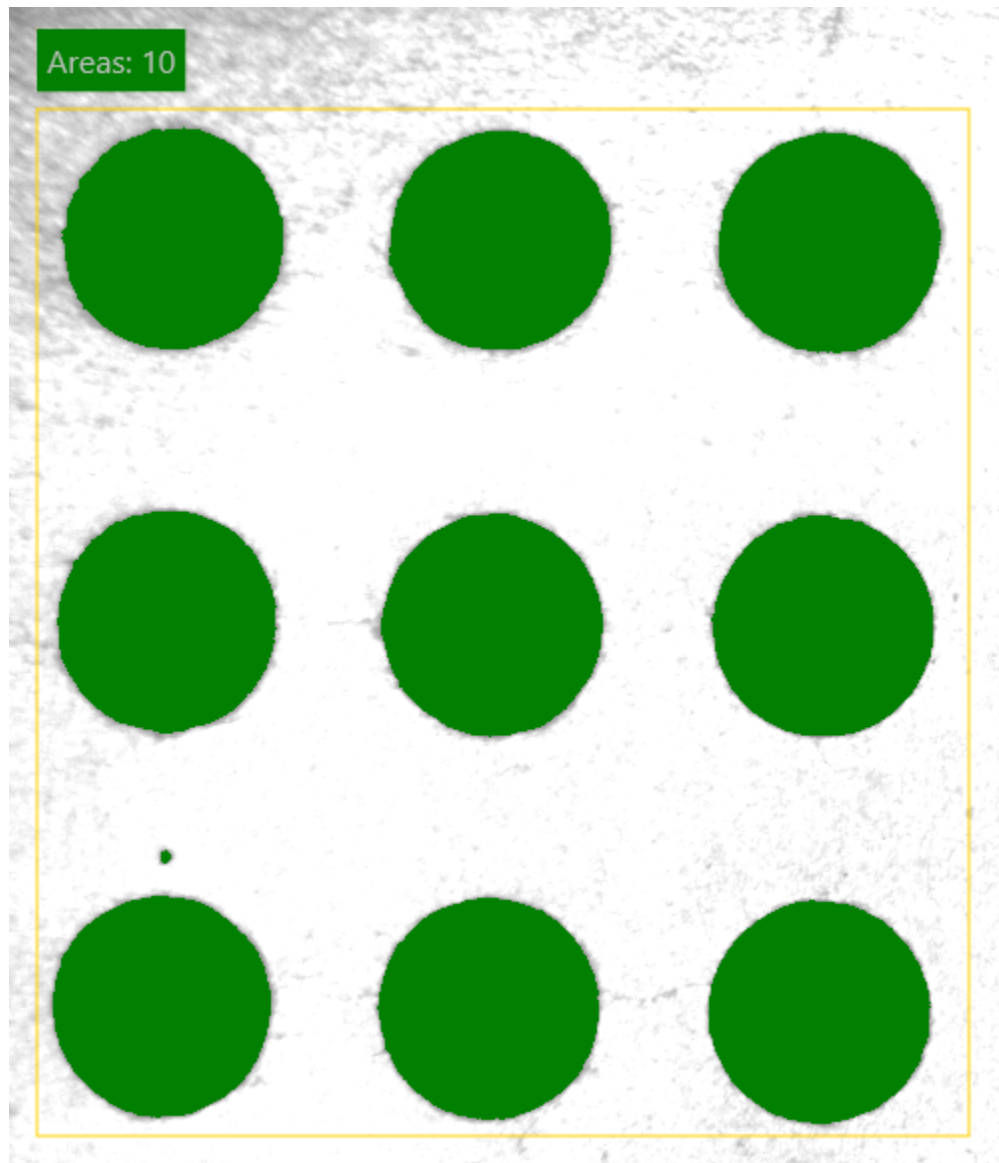
015



It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging it inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The tool is meant to be used on regions where you have contrasting objects, either dark or bright. It automatically detects the objects. In a region without much contrast, using the tool is mostly meaningless because of noise.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values.

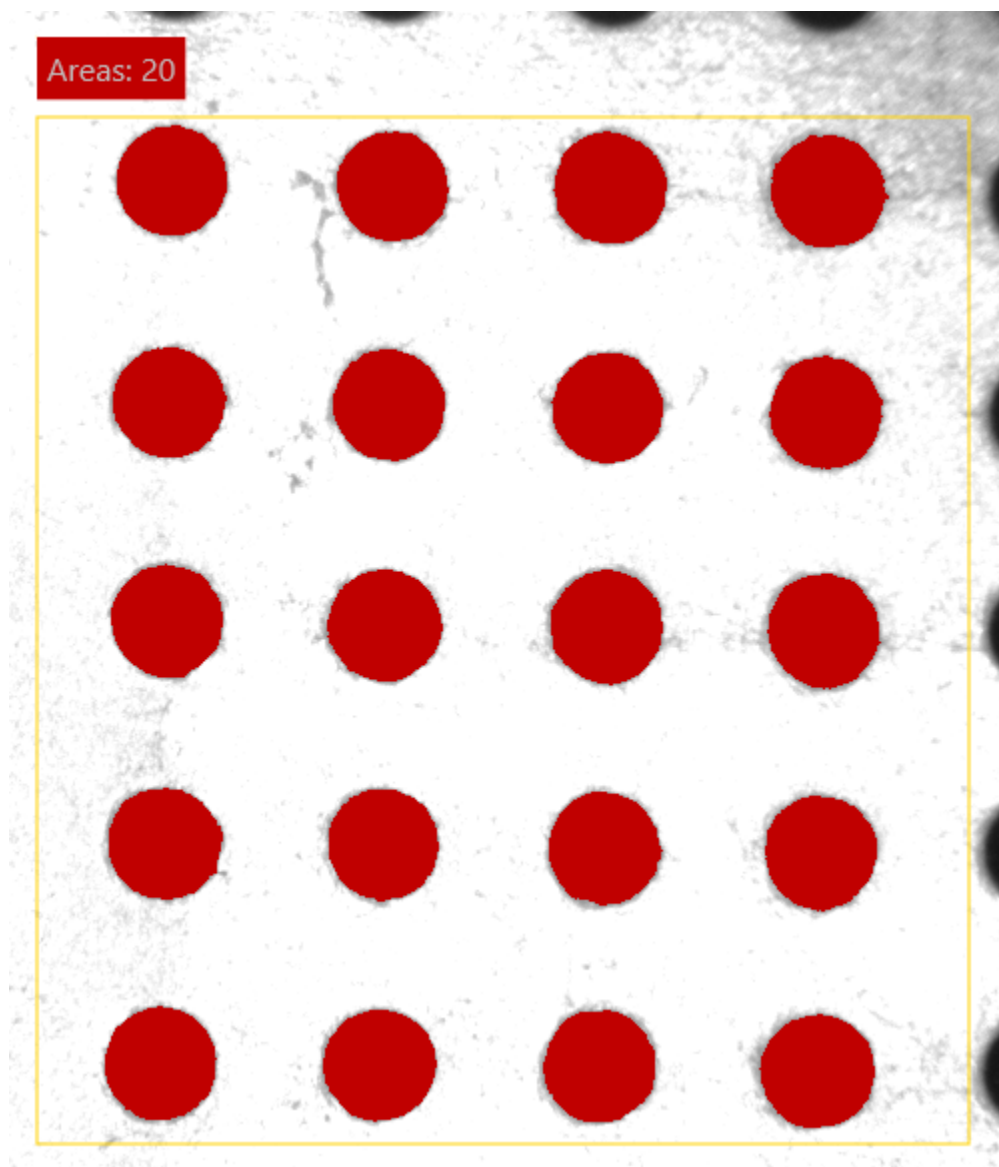


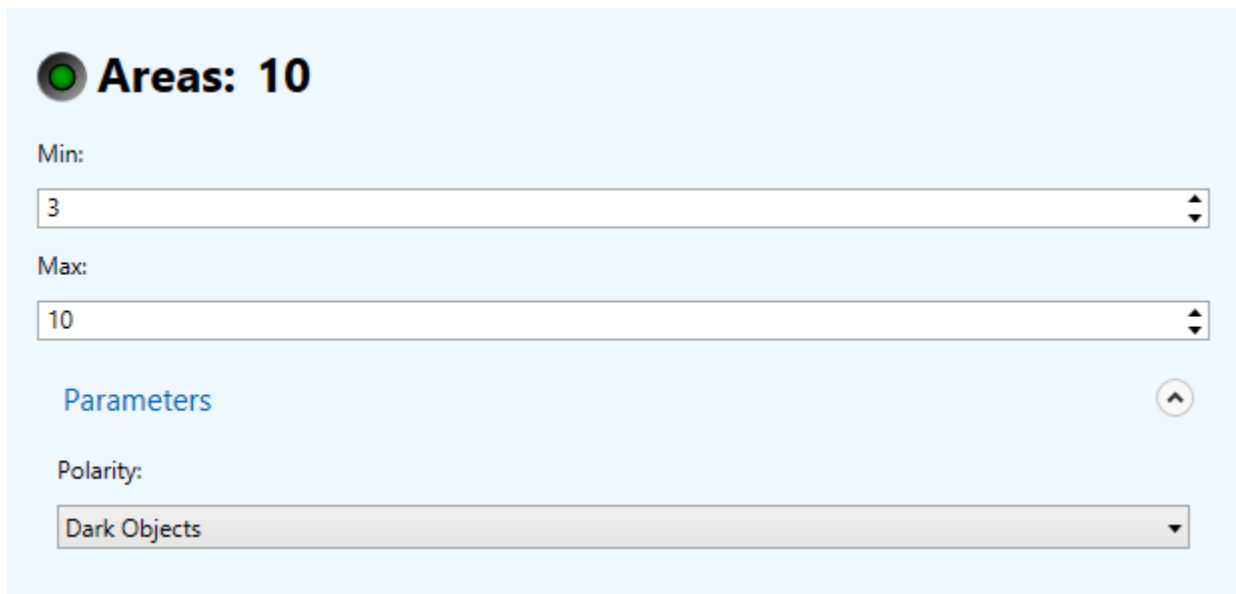
The next picture shows the tool result, if the measured area is not within the expected boundary values.

The count areas tool has a configuration panel, which can be used to set parameters.

The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Polarity** can be set to **Dark Objects** or **Bright Objects**.





The screenshot shows a configuration panel for the Match Contour tool. At the top, there is a green circular icon followed by the text "Areas: 10". Below this, there are two input fields: "Min:" with the value "3" and "Max:" with the value "10". Both fields have up and down arrow buttons on the right. Underneath these fields is a blue button labeled "Parameters" with an upward arrow icon to its right. At the bottom, there is a "Polarity:" label followed by a dropdown menu currently set to "Dark Objects".

7.3 Verification

7.3.1 Features

The tools in this group provide matching of parts based on features of these parts.

Match Contour

The match contour tool locates a part using geometric pattern matching. The tool is used by clicking the **Match Contour** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).

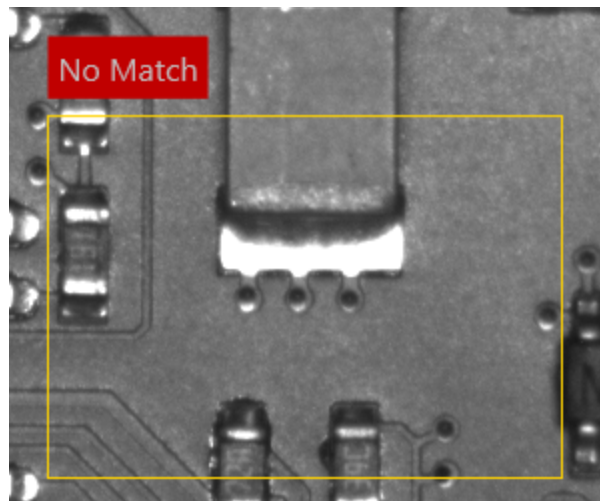
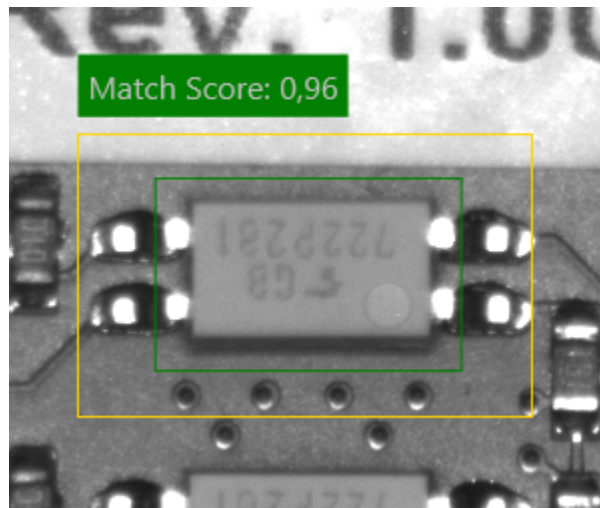
Usually, the box is put tightly around an area where you expect a specific pattern, which is loaded from a file somewhere.


If the tool finds the pattern, it displays a green box where it found the pattern and displays the match score (between 0 and 1, where 1 is a perfect match). Scores below 0.3 are mostly meaningless. Please note that the characteristic of the score is different to the one used in the Match Pattern command.

If the tool cannot find the pattern at all, or if the score is too low, it displays a red box if possible and outputs the text "No Match" in red.

The match pattern tool has a configuration panel, which can be used to set parameters.

The **Template** parameter specifies the match template, which is loaded from disk.



 **Match Score: 0,94**

Template:

pattern.png

Min Score:

0,7

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.7). Matches below this score are not considered.

Match Pattern

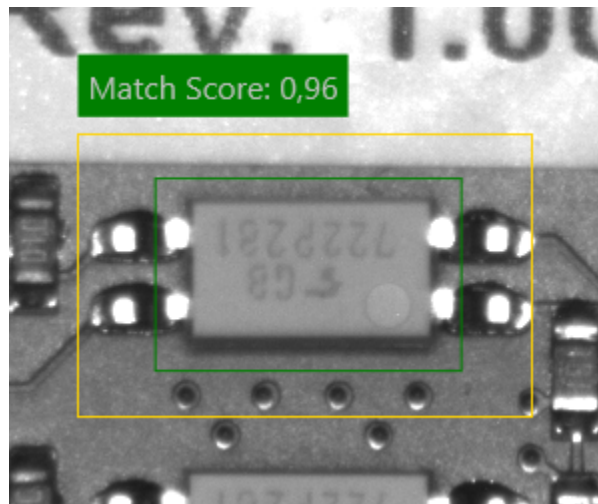
The match pattern tool locates a part using normalized correlation matching. The tool is used by clicking the **Match Pattern** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).

Usually, the box is put tightly around an area where you expect a specific pattern, which is loaded from a file somewhere.



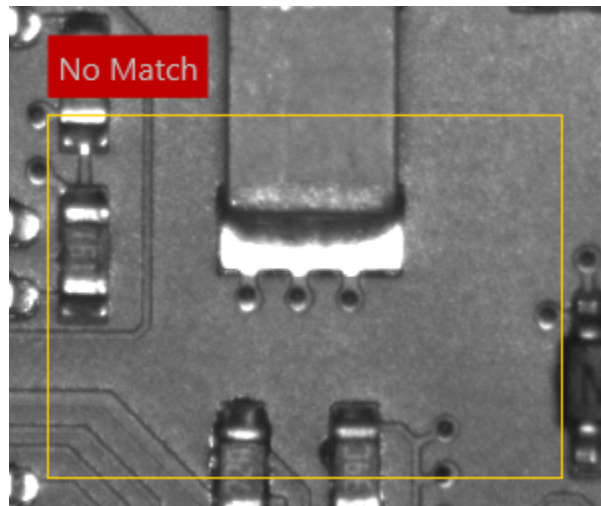
If the tool finds the pattern, it displays a green box where it found the pattern and displays the match score (between 0 and 1, where 1 is a perfect match). Scores below 0.5 are mostly meaningless. Please note that the characteristic of the score is different to the one used in the Match Contour command.


If the tool cannot find the pattern at all, or if the score is too low, it displays a red box if possible and outputs the text "No Match" in red.

The match pattern tool has a configuration panel, which can be used to set parameters.

The **Template** parameter specifies the match template, which is loaded from disk.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.7). Matches below this score are not considered.



 **Match Score: 0,94**

Template:

pattern.png

Min Score:

0,7

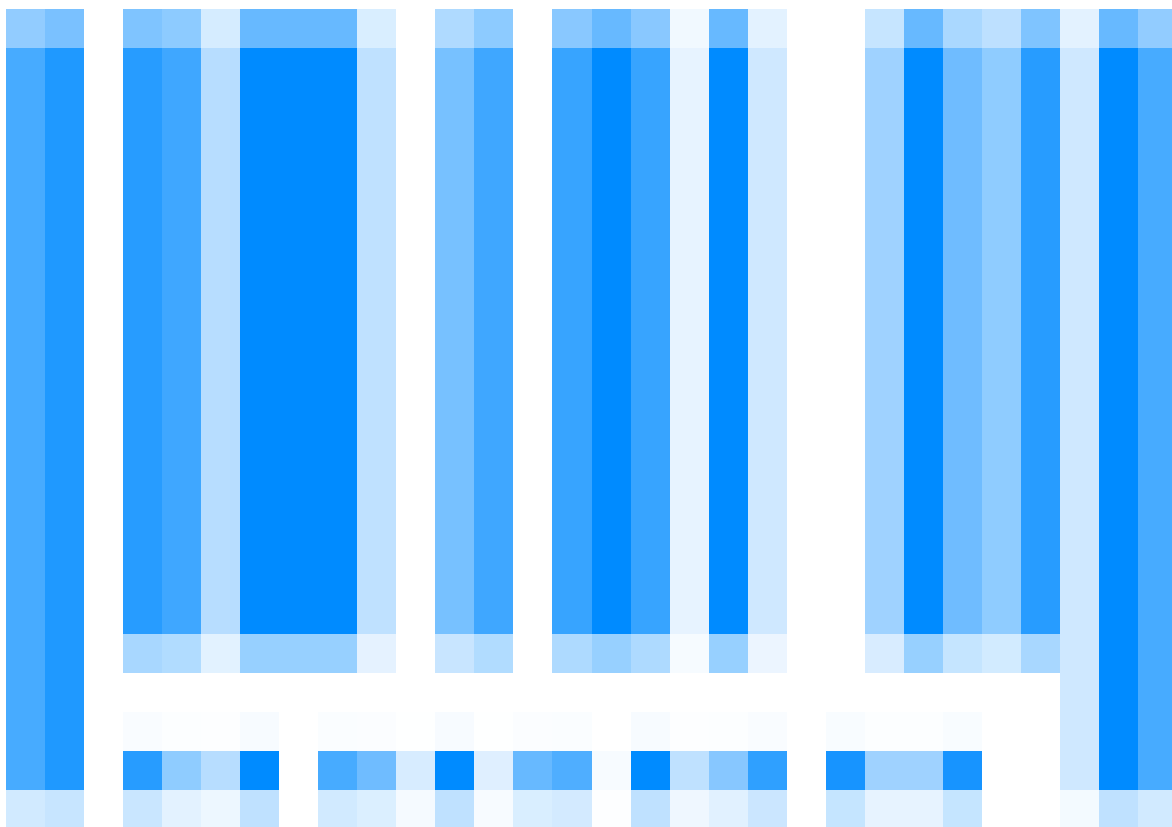
7.4 Identification

7.4.1 Codes

The tools in this group perform barcode and matrix code decoding. Various symbologies are supported. In addition, the tools can match the decoded text to an expected pattern.

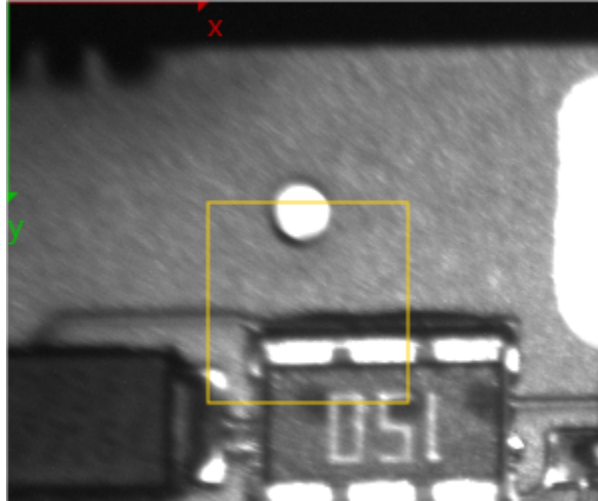
Barcode

The barcode tool decodes a barcode inside a region of interest. The tool is used by clicking the **Barcode** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a box that is used to select the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside).



Once the search box has been positioned over a code in the image it tries to decode the code. Its position and the decoded text is displayed in green, if it matches the expected text (anything in this case):

The next picture shows the tool result, if the decoded text does not match the expectation:

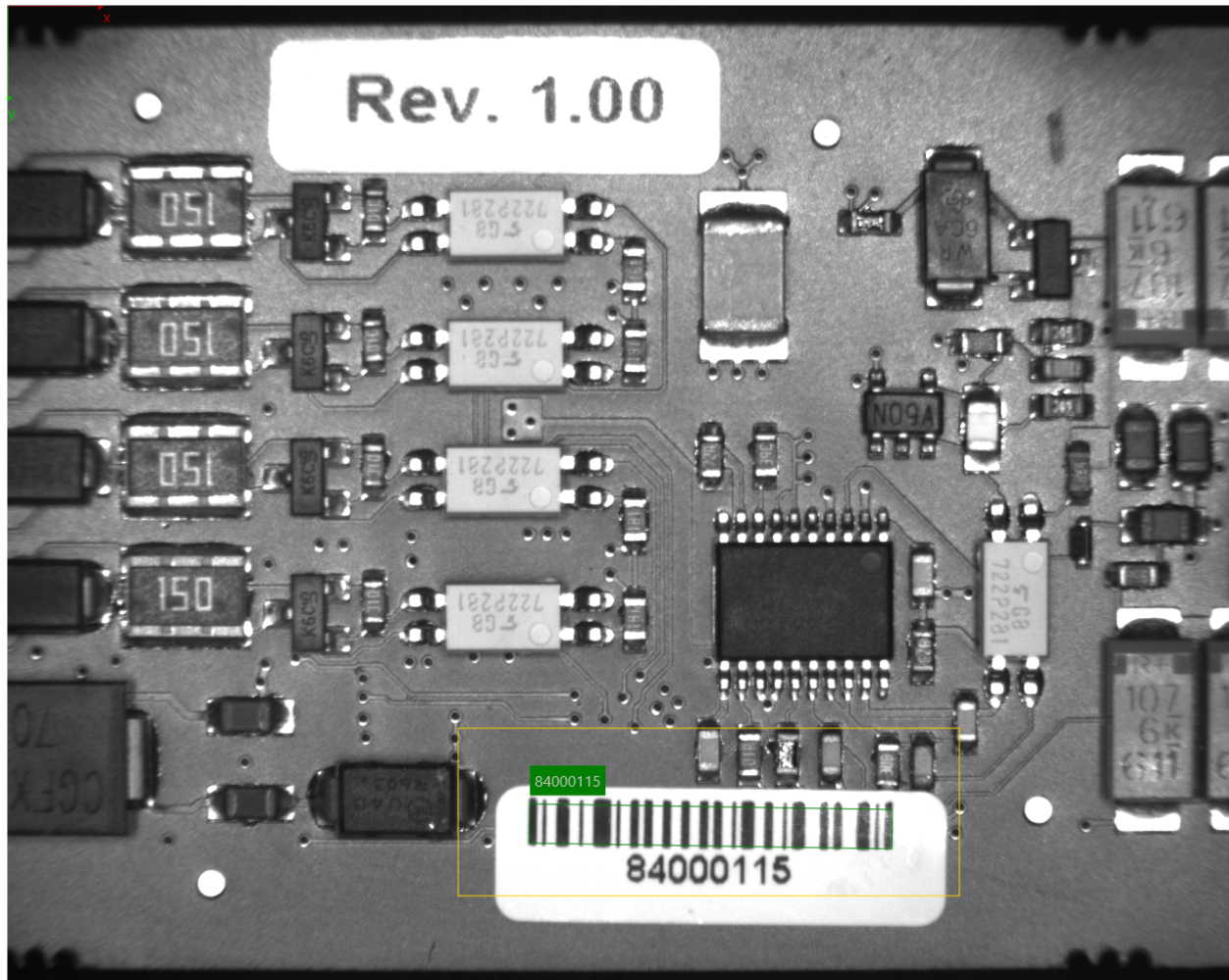
This makes it easy to see the tool's outcome.

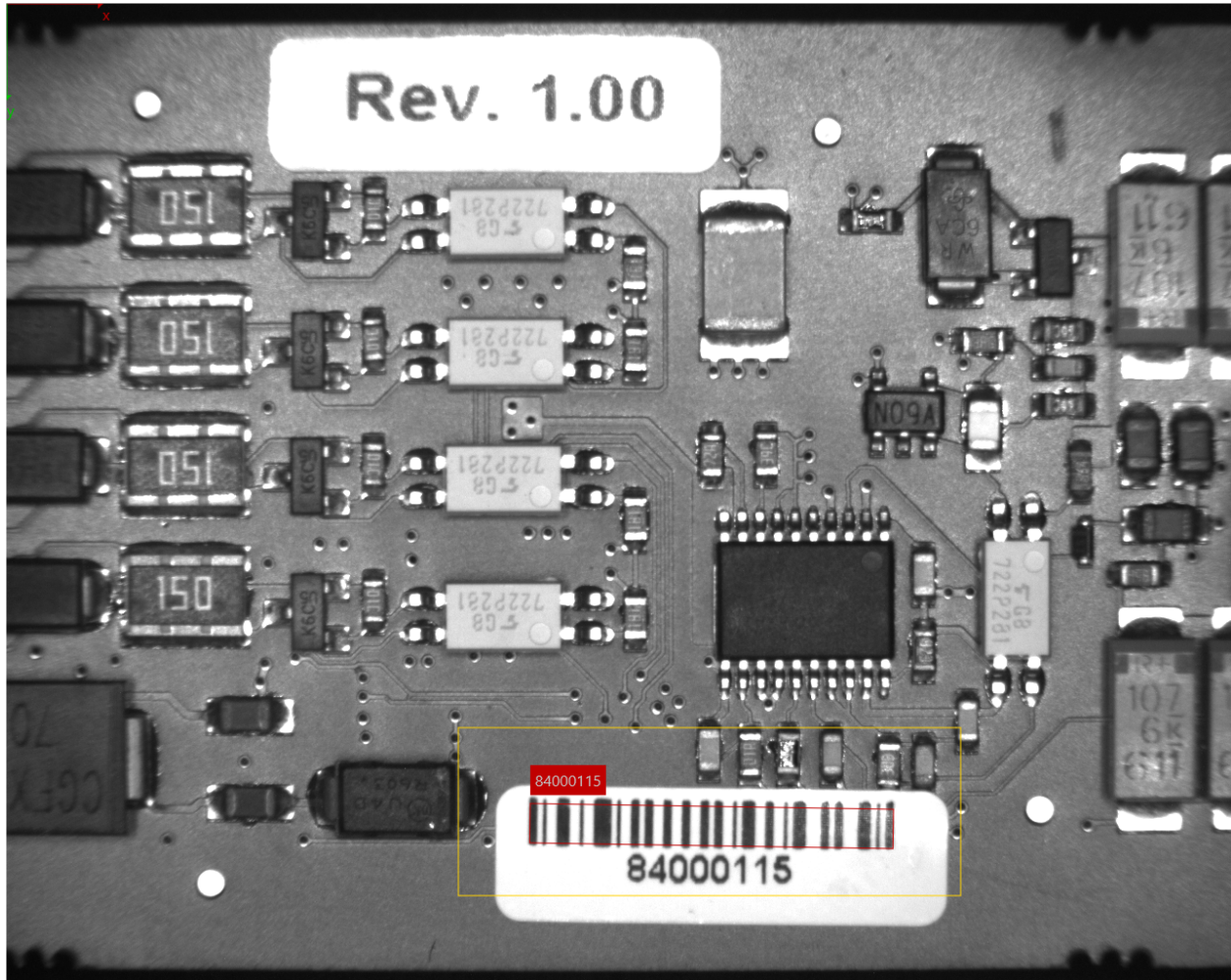
The barcode tool has a configuration panel, which can be used to set parameters.

The controls in **Pattern Matching** allow you to specify a pattern that the decoded result must follow in order to be correct.

Regular Expression: Defines the pattern that the decoded string should match. The pattern follows the rule of a regular expression, specifically the .NET variant of regular expressions.

The following table explains common cases. For full information, look at the [original documentation](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx) ([https://msdn.microsoft.com/en-us/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx)).





Barcode

Decoder Result

84000115

Symbology: Code 128

Checksum: valid

FNC1: no

Reader Programming: no

Append: no

Pattern Matching

Regular Expression:

|

☒ Ignore Case

Enabled Symbologies

☒ Code 39

☒ Code 39 Extended

☒ Code 93

☒ EAN 128

☒ EAN 8

☒ EAN 13

☒ UPC A

☒ UPC E

☒ Databar

☒ Databar Limited

Decoder Parameters

Number of Scanning Directions:

4

Scanning Density:

5

Noise:

40

☒ Check Quiet Zone

Pat-tern	Description		
.	Any character (except newline).	a.c	abc, aac, acc, ...
^	Start of a string.	^abc	abc, abcdefg, abc123,...
\$	End of a string.	abc\$	abc, endsinabc, 123abc,...
|	Alternation.	bill|ted	bill,ted
[. . .]	Explicit set of characters to match.	a[bB]c	abc, aBc
*	0 or more of previous expression.	ab*c	ac, abc, abbc, ...
+	1 or more of previous expression.	ab+c	abc, abbc, abbbc, ...
?	0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.	ab?c	ac, abc
{...}	Explicit quantifier notation.	ab{2}c	abbc
(...)	Logical grouping of part of an expression.	(abc){2}	abcabc
\\	Preceding one of the above, it makes it a literal instead of a special character.	a\\.b	a.b

Ignore Case: If checked the case of the characters is ignored, otherwise the case must match exactly.

The controls in **Enabled Symbolologies** allow you to enable or disable specific symbolologies. Supported symbolologies are Code 39, Code 30 Extended, Code 93, EAN 128, EAN 8, EAN 13, UPC A, UPC E, Databar and Databar Limited.

The controls in **Decoder Parameters** allow you to manipulate locator and decoder behavior.

Number of Scanning Directions: 0: vertical, 1: horizontal, 2: both, 3 and more: oblique, every 180/N degrees. The default setting of 4 means the locator scans every 45 degrees.

Scanning Density: The spacing between scanning-lines, in pixels; small values favor the decoding rate and increase the running time. 5 is the default.

Noise: The noise threshold. Threshold level to get rid of spurious bars caused by noise. 40 is the default.

Check Quiet Zone: Check the quiet zone. When checked, a quiet zone as large as the standard requires must be present; when set to false, it is advisable to disable other symbolologies to avoid false matches. The default is checked.

Minimum Bars: The minimum number of bars for a successful decode.

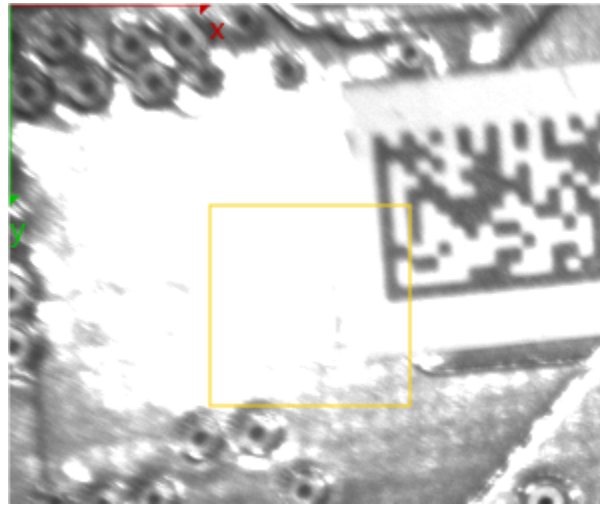
Matrixcode

The matrixcode tool decodes a two-dimensional barcode inside a region of interest. The tool is used by clicking the **Matrixcode** command button.



The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a box that is used to select the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside).



Once the search box has been positioned over a code in the image it tries to decode the code. Its position and the decoded text is displayed in green, if it matches the expected text (anything in this case):

The next picture shows the tool result, if the decoded text does not match the expectation:

This makes it easy to see the tool's outcome.

The matrixcode tool has a configuration panel, which can be used to set parameters.

The controls in **Pattern Matching** allow you to specify a pattern that the decoded result must follow in order to be correct.

Regular Expression: Defines the pattern that the decoded string should match. The pattern follows the rule of a regular expression, specifically the .NET variant of regular expressions.

The following table explains common cases. For full information, look at the [original documentation](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx) ([https://msdn.microsoft.com/en-us/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx)).





Matrixcode

Decoder Result

002401003C4661211931

Symbology: QR Code
FNC1: no
Reader Programming: no
Append: BlackOnWhite
View: Straight

Pattern Matching

Regular Expression:

☒ Ignore Case


Enabled Symbologies

☒ Data Matrix


☒ QR Code

Decoder Parameters

Polarities:

Auto 

Views:

Auto 

Pat-tern	Description		
.	Any character (except newline).	a.c	abc, aac, acc, ...
^	Start of a string.	^abc	abc, abcdefg, abc123,...
\$	End of a string.	abc\$	abc, endsinabc, 123abc,...
|	Alternation.	bill|ted	bill, ted
[. . .]	Explicit set of characters to match.	a[bB]c	abc, aBc
*	0 or more of previous expression.	ab*c	ac, abc, abbc, ...
+	1 or more of previous expression.	ab+c	abc, abbc, abbbc, ...
?	0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.	ab?c	ac, abc
{...}	Explicit quantifier notation.	ab{2}c	abbc
(...)	Logical grouping of part of an expression.	(abc){2}	abcabc
\\	Preceding one of the above, it makes it a literal instead of a special character.	a\\.b	a.b

Ignore Case: If checked the case of the characters is ignored, otherwise the case must match exactly.

The controls in **Enabled Symbolologies** allow you to enable or disable specific symbolologies. Supported symbolologies are Data Matrix and QR Code.

The controls in **Decoder Parameters** allow you to manipulate locator and decoder behavior.

Polarities: Selects the possible contrast of the marking. **Auto** (default), **Black on White** or **White on Black**.

Views: Selects the possible orientation of the marking. **Auto** (default), **Straight** or **Mirrored**.

7.5 Miscellaneous

7.5.1 Adjust Camera

The Adjust Camera tab on the Home menu contains the Exposure tool and various calibration tools, which can be used to set the exposure time of a camera and establish a world to image coordinate calibration.

Exposure

The **Exposure** tool is used to set the exposure time of a camera. It works with GenICam cameras that have the **exposure time** parameter. The tool is used by clicking the **Exposure** command button (Home - Acquire).



The tool displays the live image and shows over-exposed parts in yellow and underexposed parts in blue.

The exposure tool has a configuration panel, which can be used to set parameters.

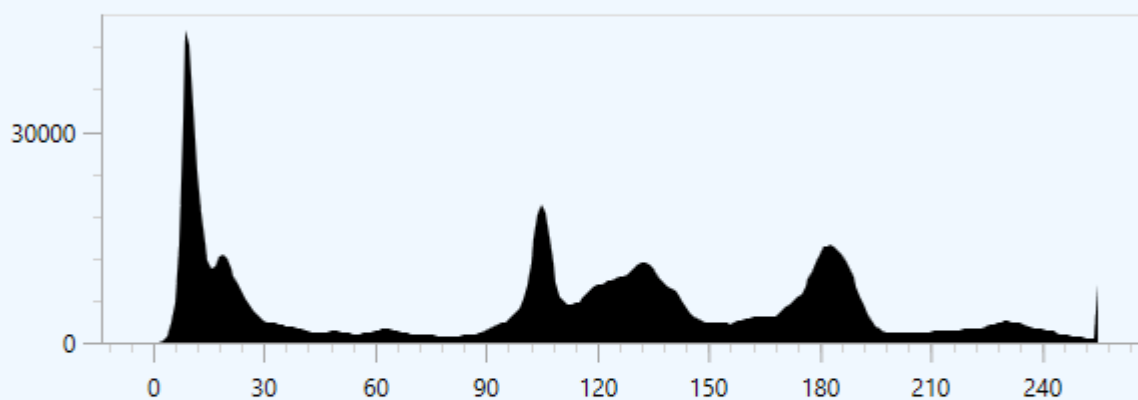


Exposure Control

Exposure Time:

■ Underflow: 0,1 %

■ Overflow: 0,8 %



Parameters

Underflow Threshold:

Overflow Threshold:

GenICam Parameter Name:

The **Exposure Time** is the exposure time in ms (milliseconds). It uses GenICam to write the exposure time to the camera, which has to be connected to the global **Camera** output port in the system globals.

Underflow and **Overflow** show the respective percentages of the underflow and overflow pixels.

At the bottom, a graphic of the brightness histogram is shown.

When the exposure time is changed, the blue and yellow image overlays, the percentage values and the histogram graphic are all affected and change. The graphic and numeric outputs help you to properly set exposure time interactively.

Define Scale

The **Define Scale** tool is used to define an overall scale in the image. You can use it if your optical axis is orthogonal to the measurement plane, and if the desired measurement accuracy is not too high.



The tool calculates a calibration factor that is used to convert pixel measurements to world units, such as μm , mm, cm or m (or any other spatial unit that is needed in your application). Once a calibration has been established, other length measurement tools respect the calibration and output their measurements in world unit (as opposed to image units in pixels).

The **Define Scale** tool can use a line to establish a scale.

Alternatively, it can use a circle to establish a scale.

The **Define Scale** tool has a configuration panel where you can select whether you want to use a line or a circle for setting the scale.

Once you have aligned the line or circle with a known distance in your image, type the distance as well as the unit and press the **Teach** button.

Calibrate

The **Calibrate** tool is used to establish a camera calibration. The calibration can correct a perspective distortion and still measure proper distances in the calibrated plane.

The tool detects the positions of dots on a calibration target and uses those to detect the perspective distortion. In addition to the detected dot positions the tool needs to know the distance between two dots in world units as well as the world unit.

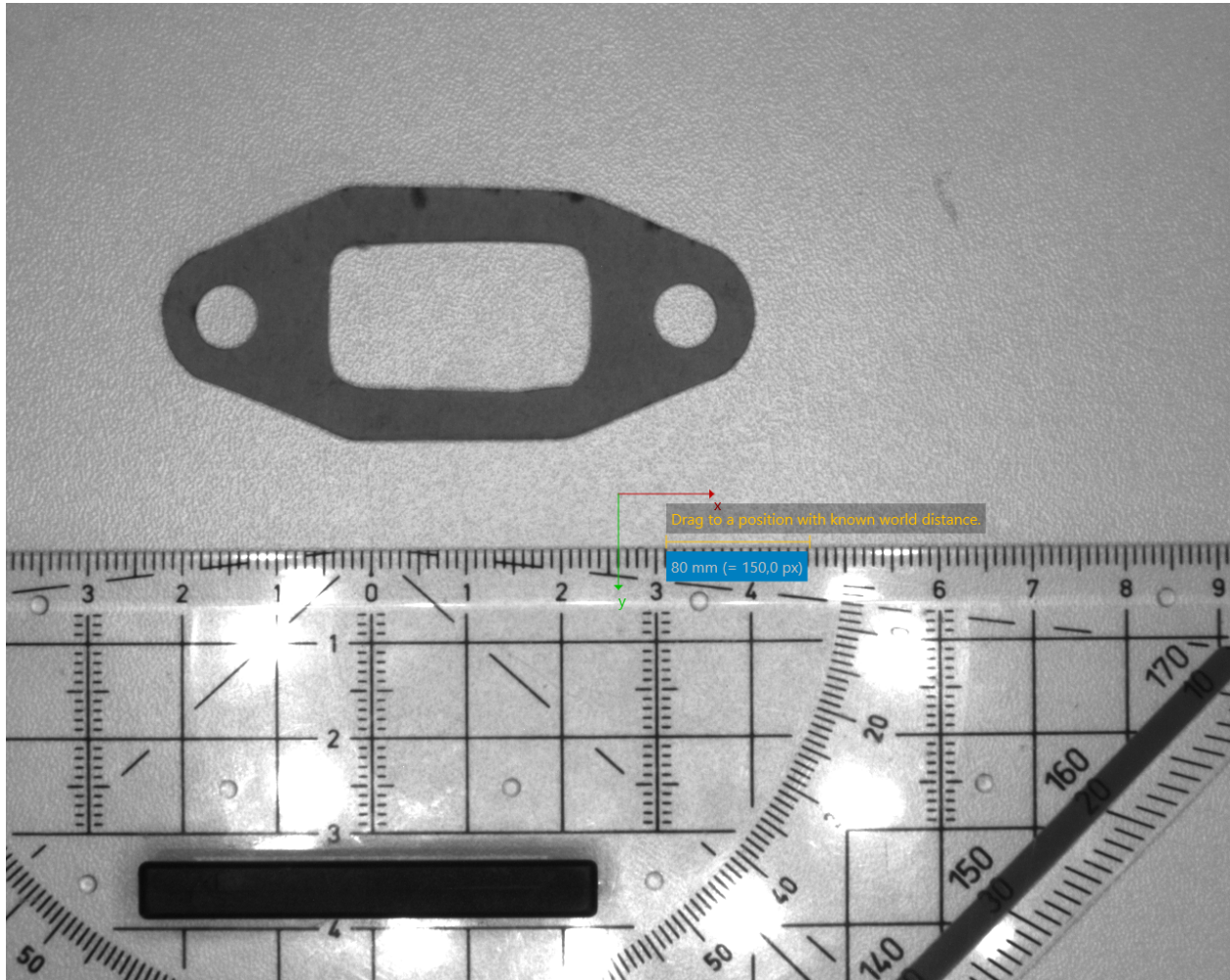
Only the dots within the region of interest turn blue and are used for calibration.

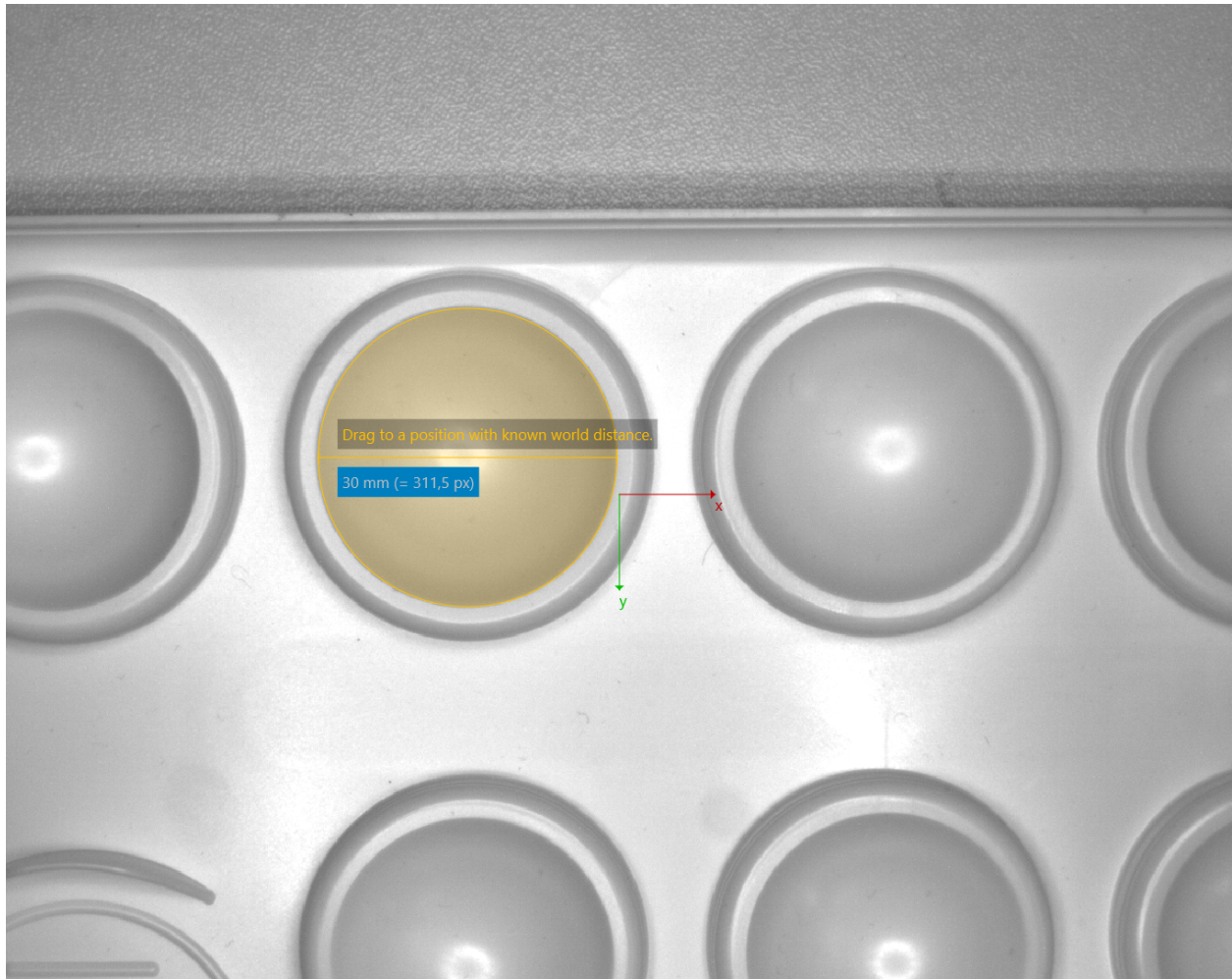
The **Calibrate** tool has a configuration panel where you can enter the distance between two dots as well as the world units.

Once you have a proper image of the calibration target and entered the distance between the dots as well as the unit, press the **Teach** button.

7.5.2 Pipeline

The Result tab on the Pipeline menu contains the result tool, which provides a simple means of communication.





Define Scale

1. Drag the ROI over the area inside the image.

Select ROI type

☒ Line Segment

☐ Circle

2. Define a scale.

80

3. Define a unit.

mm

4. Press the Teach button when ready.

Teach



Result

The result tool is used to communicate the result of an inspection task. It works with an Adam module from Advantec. The tool is used by clicking the **Result** command button (Pipeline - Result).

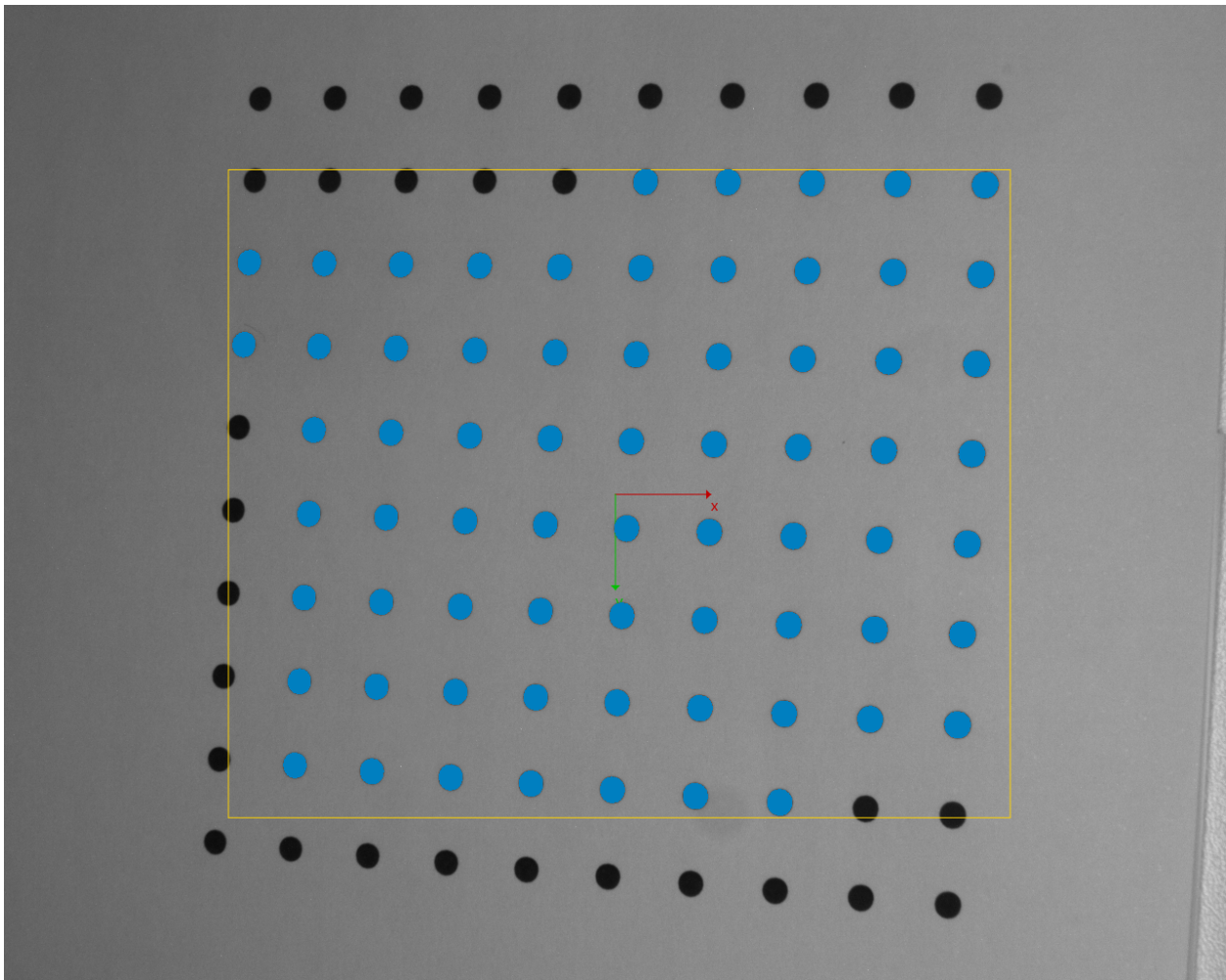
The result tool writes the result in the form of an electrical signal to the Adam module.

The result tool has a configuration panel, which can be used to set parameters.

The parameters in **Configure IO** specify the electrical lines that are switched on.

OK specifies the number of the output of the Adam module which is used to signal the **OK** state.

Not OK specifies the number of the output of the Adam module which is used to signal the **not OK** state.



Calibrate

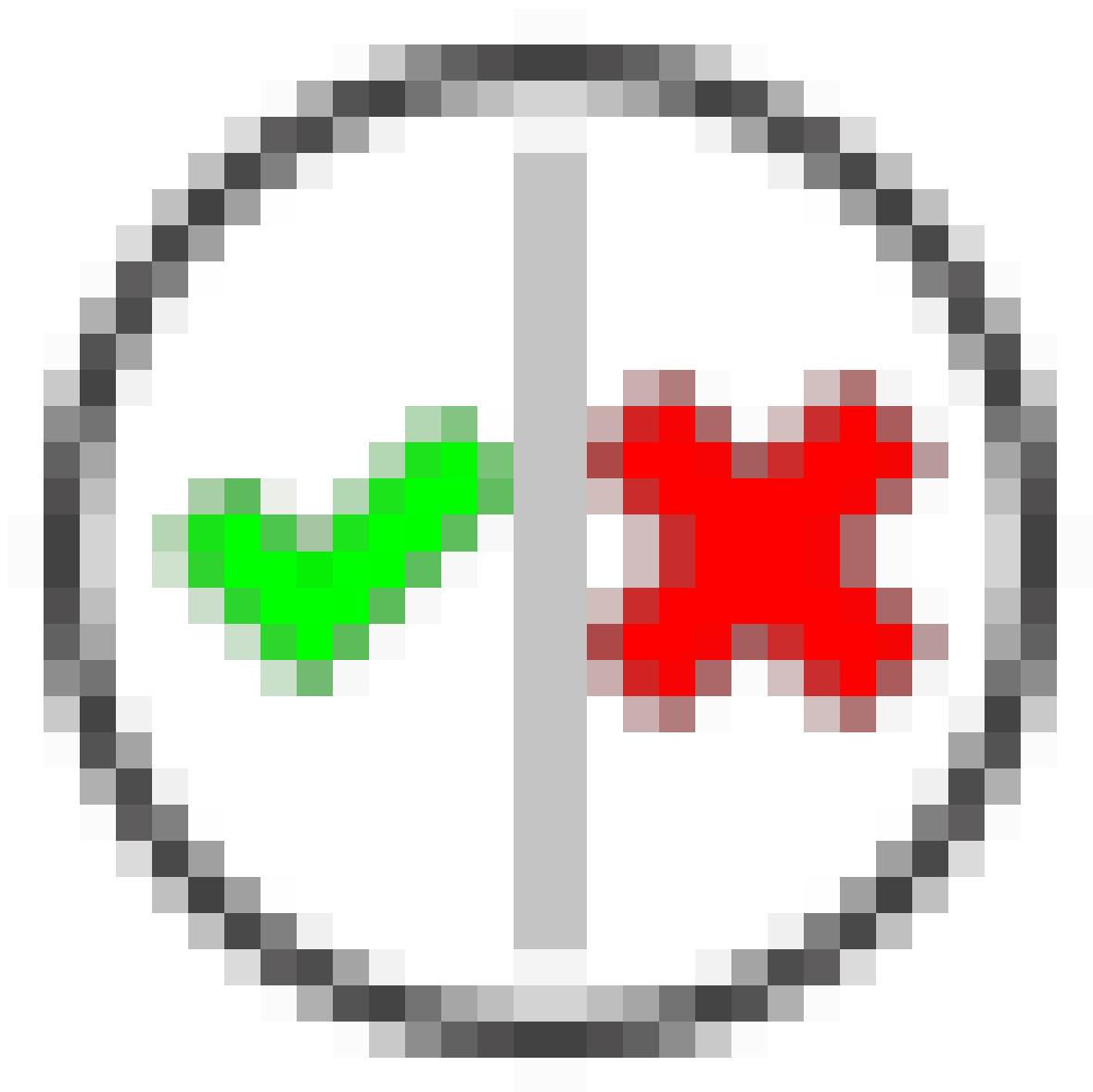
1. Put a calibration target under the camera or make sure you have loaded an image of a calibration target.

2. Define the distance between the marks:

3. Define a unit:

4. Press the Teach button when ready:

Teach



Result

Configure IO 

Ok:

1 

Not Ok:






2 

nVision can be programmed graphically using the concept of a dataflow pipeline.

While the user is executing image processing commands, **nVision** builds a pipeline of those commands. Such a pipeline is essentially a program that can then be executed over and over again, for potentially many images. The image data flows through the pipeline much like water flows through a system of pipes.

Here is an example of such a pipeline, where the data flows from top to bottom.

It is particularly easy to create this pipeline, just by issuing the following commands, one after the other:

	Command	Description
	File - Open	Opens a file (cells.tif from samples).
	Segmentation - Threshold	Binary thresholding (> 100).
	Segmentation - Connected Components	Split into connected components.
	Segmentation - Blob Analysis	Calculate features for objects.

There are some parameters on top of some pipeline steps that can be changed and that affect the outcome of the pipeline. If you want to know what these parameters are, move the mouse on top of the parameters editor and look at the tooltip that appears. Whenever a parameter is changed by the user, the pipeline reruns and the results are re-calculated.

What you have built this way is a linear sequence of steps - or pipeline nodes - and this is essentially a very simple program.

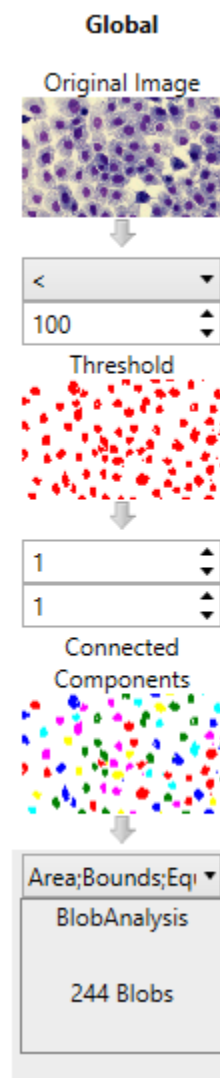


Fig. 8.1: A linear pipeline, where the data flows from top to bottom, following the arrows.

8.1 Linear Pipeline

A linear pipeline is the easiest way to create a program in **nVision**.

A linear pipeline is displayed at the left of the workbench. Each image can have its own pipeline and the pipeline is tightly coupled with the image. When an image is saved, the pipeline is saved along the image under the same name but with `.xpipe` added. If you save an image under the name `image.tif`, the associated pipeline will be save at the same location under the name `image.tif.xpipe`.

A pipeline can be explicitly exported to a file, in case you want to reuse it for a different image. This is done with the **Export Pipeline** command from the **File** menu. To actually re-use a pipeline for a different image, use the **Import Pipeline** command from the **File** menu.

A pipeline can also be used for a batch of files in a folder. In this case, you would first load the set of file using the **Open Folder** command from the **File** menu, and then load a pipeline using the **Import Pipeline** command from the **File** menu. On the **Pipeline** menu are commands that can then be used to run the pipeline on all images, or to single step forward or backward through the sequence of files.

The steps of a linear pipeline are called pipeline nodes or simply nodes.

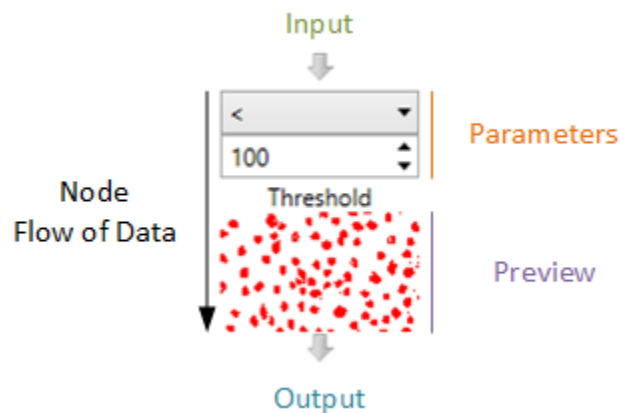


Fig. 8.2: A node in the linear pipeline. Data flows in from the top and out at the bottom. Parameters can be changed by the user using the controls at the top of the node. Output data is previewed in a little thumbnail presentation in the node.



Behind the scenes, building a linear pipeline is a complex process. When a command is executed from the menu, the associated node is appended to the existing pipeline. At the same time, it is connected to the pipeline, so that the data can flow to the new end. Also, the visualization in the workbench is changed, so that the result of the last pipeline node is shown.

The data that can flow through the pipeline is not restricted to images only. There are many more types of data, such as histograms, profiles, regions, blob analysis results or even numbers. Commands however may make sense for a limited set of types only. An image filter needs an image on input. As a consequence, when a linear pipeline is built, some commands on the menu may be grayed out, if the result of executing them would be an inconsistent linear pipeline.

Usually, the last node in a linear pipeline is selected, and new nodes are appended at the end. However, you can also select a node in the middle of the pipeline. In this case, a node would be inserted after the selected node. In such cases even more commands may be grayed out, since not only the input to the node must fit the available type, but also the output of the node.

Another feature of the linear pipeline are the previews that are displayed within the nodes. We have tried to make these previews as helpful as possible, so that it is easy for you to understand how a specific pipeline works. The previews are live, which you will appreciate if you have a camera connected.

Finally, the nodes in the linear pipeline have a context menu, which is shown when you move the mouse on top of the node:

	Command
	Show the node result in the workbench window.
	Toggle acquisition from a camera.

We encourage you to play with the commands and build pipelines as you wish. **nVision** is made to be explored by the user and we have tried to make this exploration phase as easy as possible. Once you know a few basics, **nVision** is very learnable, and a lot of its features can be found out by simple exploration, without the need of reading a lengthy manual.

8.2 Sub-Pipeline

Surprisingly many applications can be carried out using a linear pipeline, but at some time sooner or later, you will hit a task that cannot be solved this way. For this reason, we have created the possibility to work with branched pipelines. A branched pipeline can have multiple branches and is much more powerful and flexible than its linear variant. Here is an example of a very simple branched pipeline:

The ingredients of a branched pipeline are essentially the same as for the linear pipeline: nodes and connections between the nodes. With the increased flexibility comes a little bit more work for the user: the connections are no longer automatic, but must be done manually.

Although they are displayed slightly different, nodes in the linear and in the branched pipeline are equivalent. You can also see that the distinction between input and parameters is somewhat arbitrary. In fact, parameters are just additional inputs.

Inputs are either mandatory or optional. Mandatory inputs need to be connected to some other node upstream, otherwise the node will not be able to execute. If a node does not execute, it either displays a standard icon or nothing at all in its preview area. Optional inputs do not need to be connected and they show a little control element that allows to input a value. However, they can be connected to some other node upstream, and in this case the control element is hidden and the data is taken from the upstream node.

A sub-pipeline is created by executing the **Subpipeline** command from the **Pipeline** menu. This adds a sub-pipeline node to the linear pipeline. Since the sub-pipeline is empty at this time, the workbench shows an empty window, but at the top you see a tab with the name SubPipeline and an orange splitter. The splitter can be dragged down with the mouse, and if you do so, the upper portion of the window will display the branched pipeline area and the lower portion of the window will display the result. Since at this time the subpipeline is still empty, both areas will be mostly empty as well. In order to change the name of the sub-pipeline, click its title on top of the preview.

In the sub-pipeline, the commands are available via a context menu, which you can access by clicking with the right mouse button into the editor area. There are more commands available as in the linear case.

The commands in the context menu are organized in groups. Most commands add a node to the pipeline editor canvas, which you can then drag around with the mouse by clicking somewhere inside its box. You can also connect the ports of the nodes to other ports in order to define the flow of data. Arrows can only be drawn between compatible ports, and icons guide you and help you to determine which ports are compatible.

At the top is a search box where you can type your command. For example, if you search a command related to histograms, just start typing `hi`, and the menu show only thos commands that contain the letter `hi` in their name (or group).

It is helpful to have the help window open while you learn programming. When you browse the menu, the help window will show help for the command you are browsing.

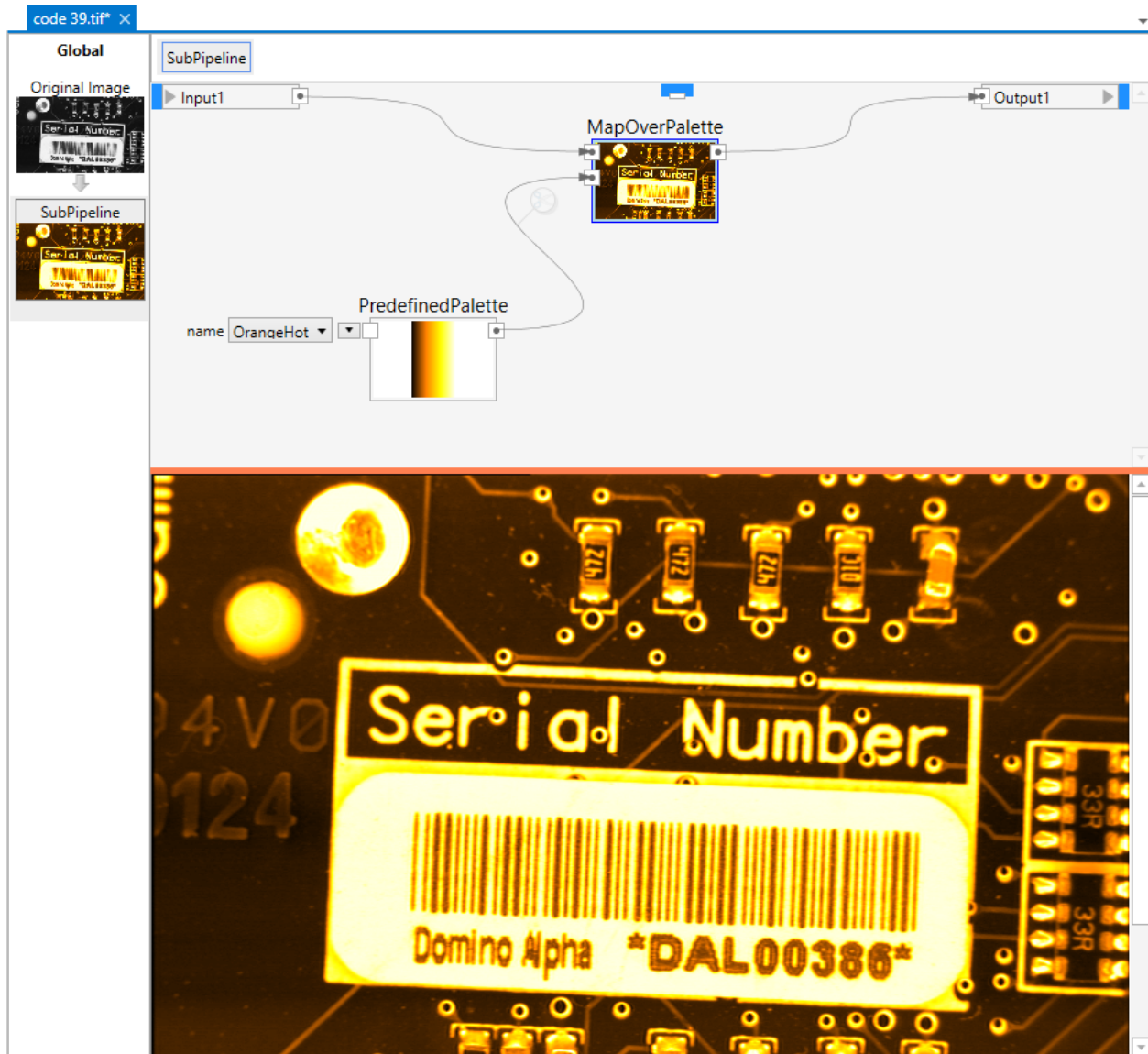


Fig. 8.3: A very simple branched pipeline. It takes an image on its single input, maps it over a palette and outputs the result. Another node creates the palette by choosing between a set of predefined entries. Below the pipeline you see the result, which in this case is a colored image.

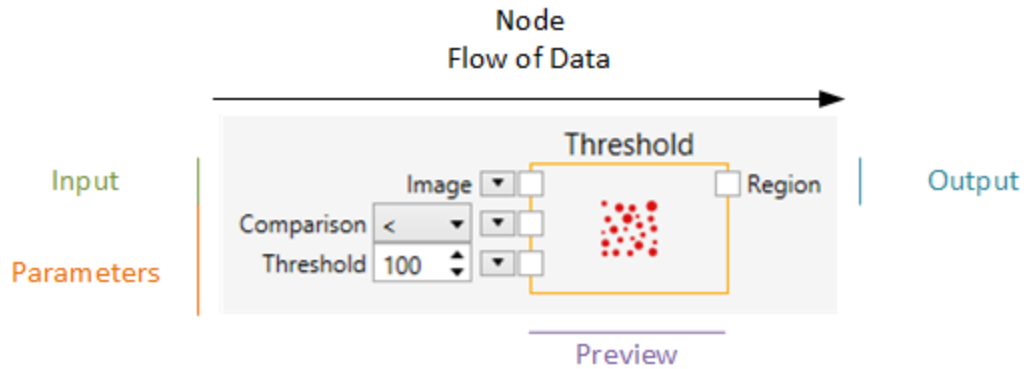


Fig. 8.4: A node in the branched pipeline. Data flows in from the left and out at the right. Parameters can be changed by the user using the controls at the left of the node. Output data is previewed in a little thumbnail presentation in the node.

The Pipeline group contains a few commands that are related to pipeline handling. The Imaging group contains commands that are related to image processing. The Vision group contains commands related to particle analysis, gauging, template matching, barcode and matrixcode reading as well as OCR. The Graphics group contains commands to create graphical overlays and editors, and the Support group contains any other commands that support you in creating efficient pipelines.

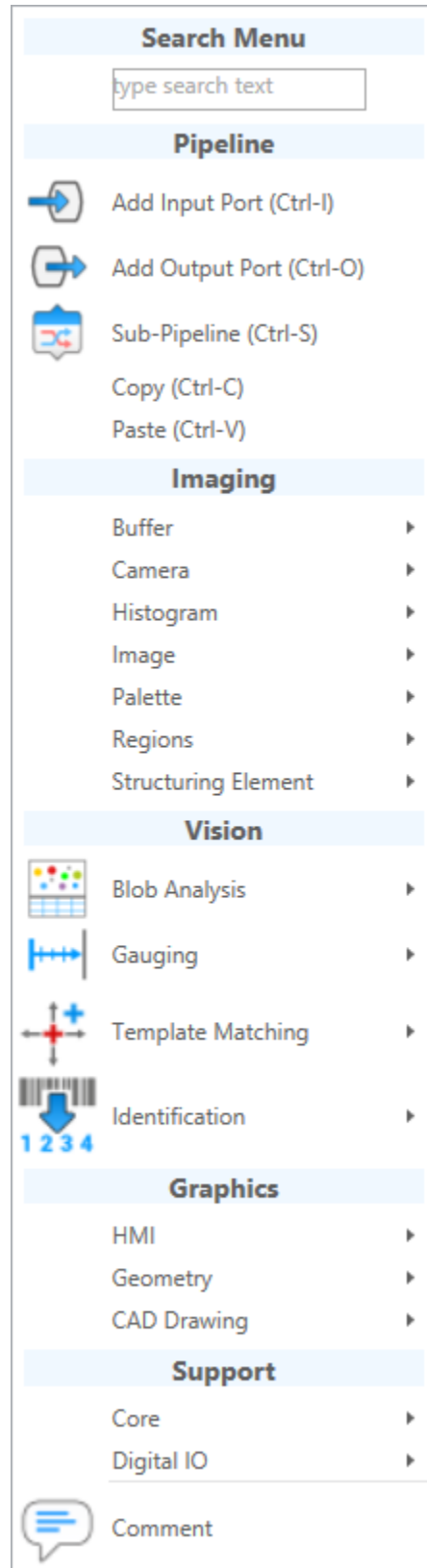
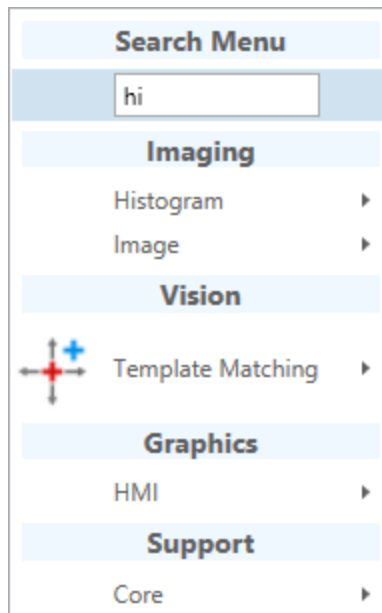


Fig. 8.5: The context menu of the sub-pipeline.



9.1 Point Algorithms

Point operations are the simplest image processing transformations. An output pixel value depends on the input pixel value only. Examples of such transformations are tone adjustments for brightness and contrast, thresholding, color space transformation, image compositing, image arithmetic or logic, etc.

Mathematically, a point operator can be written as

$$h(x, y) = g(f(x, y))$$

or

$$h = g \circ f$$

9.1.1 Classification of Point Operations

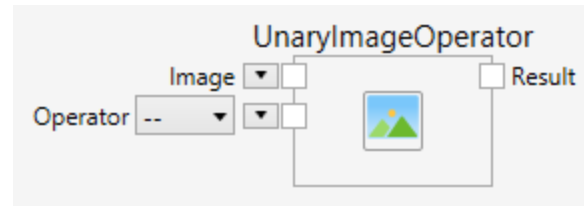
There are various ways to classify point operations. One way to classify them is by the number of arguments. Unary point operations take one argument, binary point operations take two arguments and tertiary point arguments take three arguments. Another way is to separate them into arithmetic, logical, and other operations. The table shows this classification for the operations implemented within **nVision**.

Arithmetic Logic Other	————— ————— ————— —————
Unary Absolute, Decrement, Increment, Negate, Square Root Not Colorspace transformation Binary Add, Difference, Subtract And, Or, Xor Tertiary	

9.1.2 Unary Image Operator

Applies a unary point operation to an image. Unary point operations are applied pixel by pixel, the same way for every pixel.

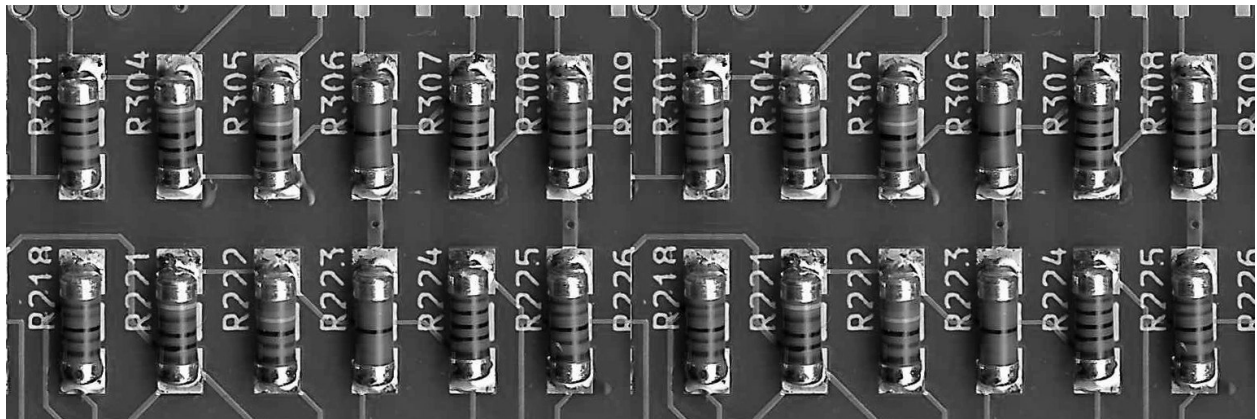
The following operators are available:



- ABS** Absolute
- 1** Decrement
- +1** Increment
- Negate
- ◻** Not
- √** Square Root

Absolute

Absolute.



$\text{abs}(\text{Image}) \rightarrow \text{Result}$

Decrement

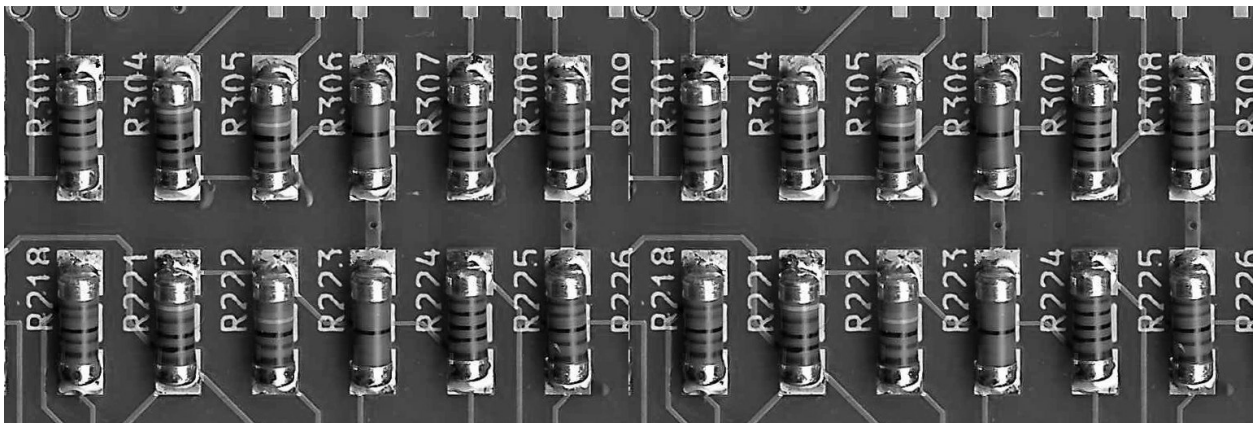
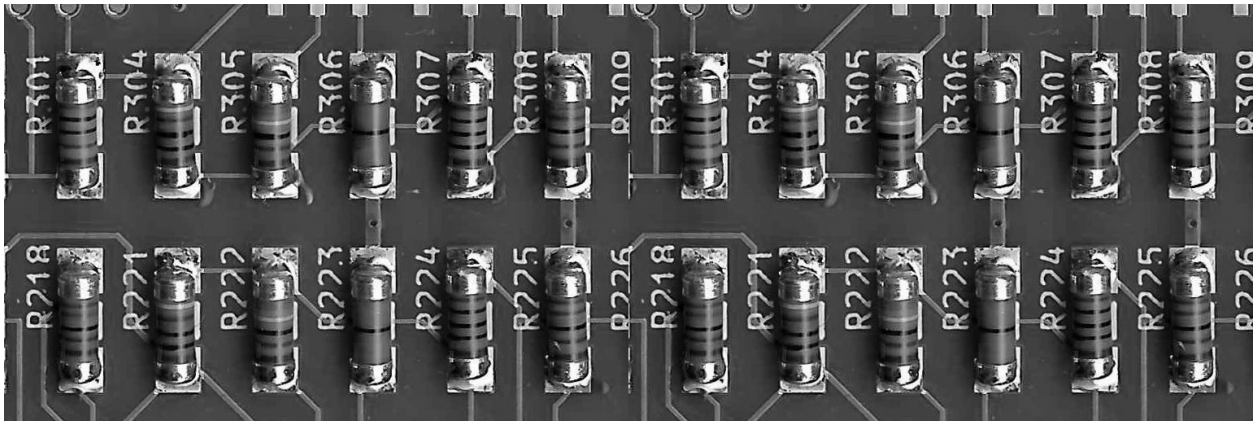
Decrement by 1.

$\text{Image} - 1 \rightarrow \text{Result}$

Increment

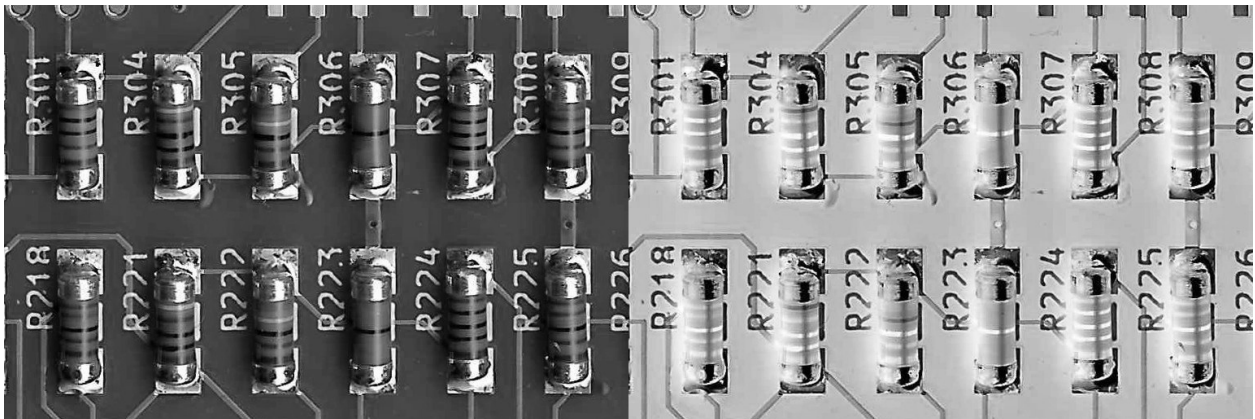
Increment by 1.

$\text{Image} + 1 \rightarrow \text{Result}$



Negate

Negate (2's complement).

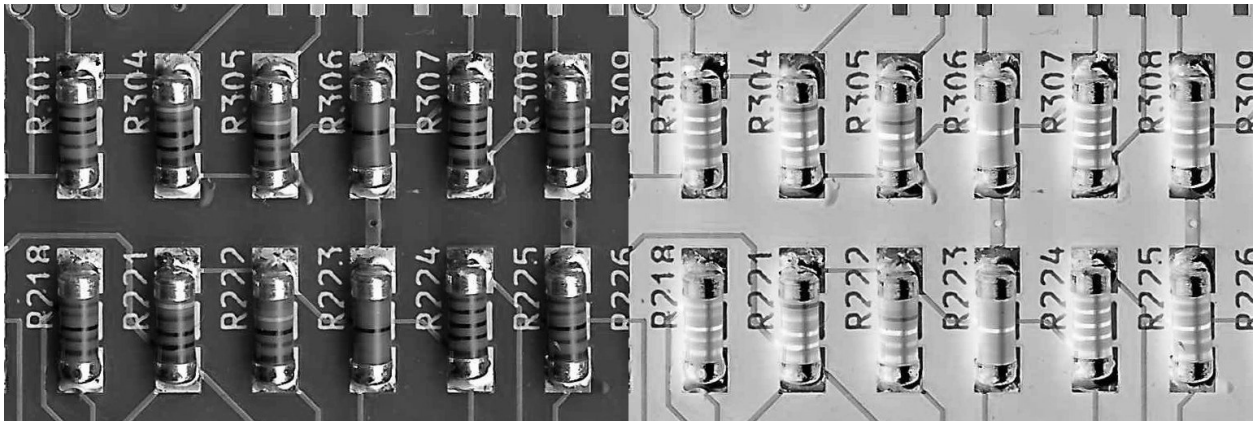


- Image -> Result

Not

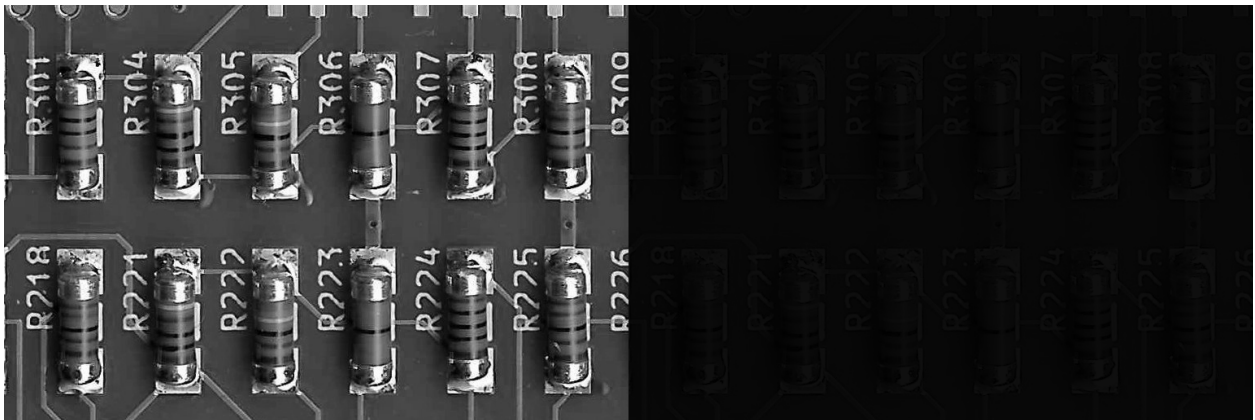
Not (1's complement).

~ Image -> Result



Square Root

Square root.



`sqrt(Image) -> Result`

Inputs

Image (Type: Image)

The input image.

Operator (Type: string)

Specifies the operator.

operator	operation
++	increment
--	decrement
-	negate (2's complement)
!	not (1's complement)
abs	absolute
sqrt	square root

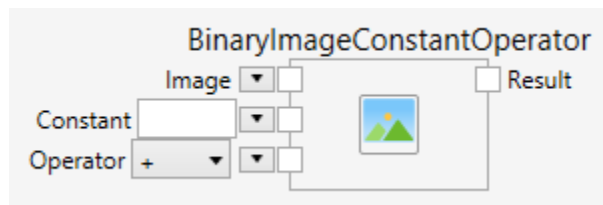
Outputs

Result (Type: Image)

The result image.








9.1.3 BinaryImageConstantOperator

Applies a binary point operation between an image and a constant. Possible operations can be grouped into arithmetic, logic and comparison fields. Binary point operations are applied pixel by pixel, the same way for every pixel.



Arithmetic Operators

The following operators are available:

-  Add
-  Subtract
-  Difference
-  Multiply
-  Divide
-  Multiply (Blend)
-  Divide (Blend)

Add

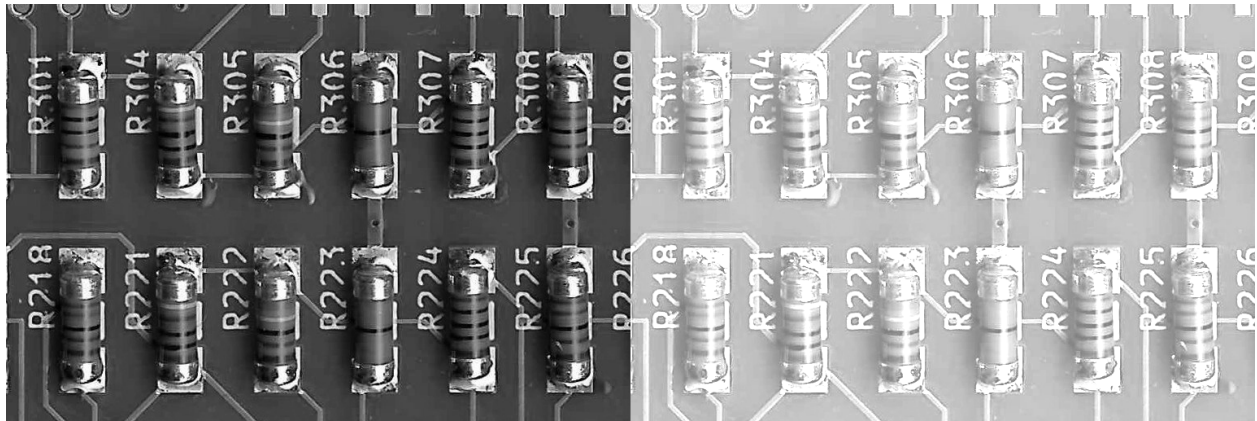
Add with saturation.

Image + Constant -> Result

Subtract

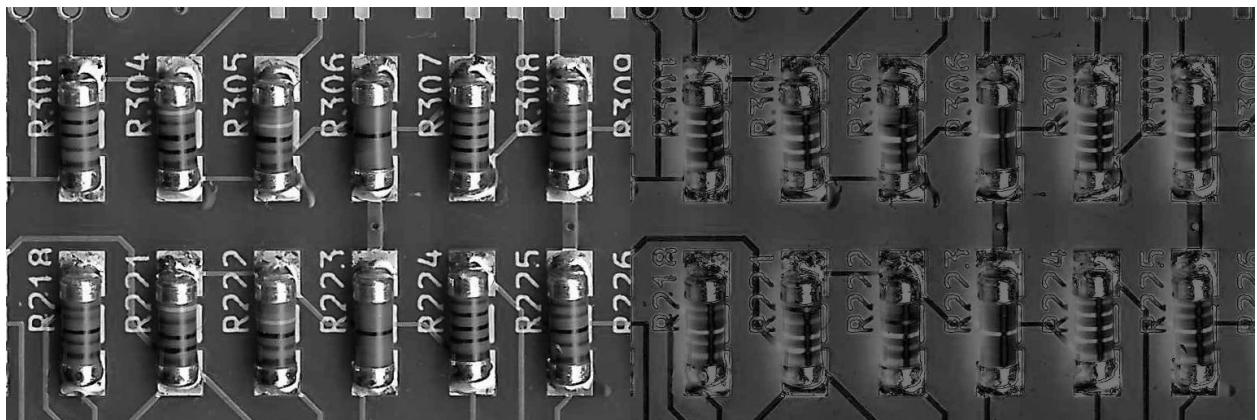
Subtract with saturation.

Image - Constant -> Result



Difference

Difference with saturation.

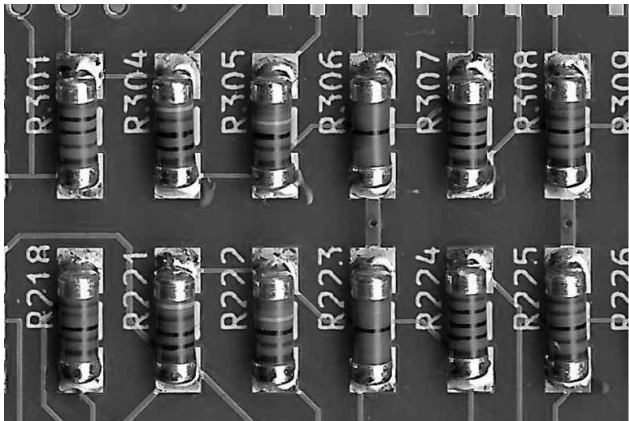


$\text{abs (Image - Constant)} \rightarrow \text{Result}$

Multiply

Multiply without saturation.

$\text{Image} * \text{Constant} \rightarrow \text{Result}$



Divide

Divide without saturation.

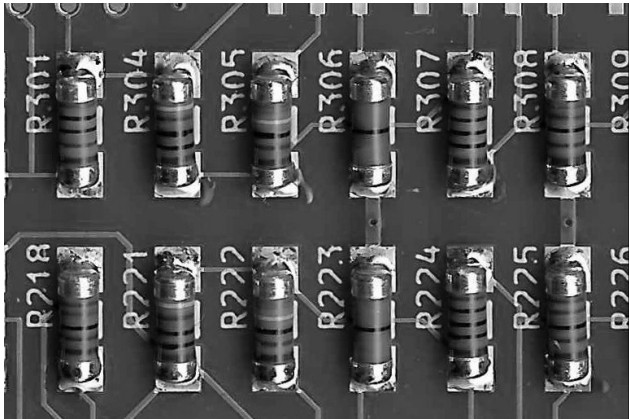


Image / Constant -> Result

Multiply (Blend)

Multiply with saturation.

Image * Constant -> Result

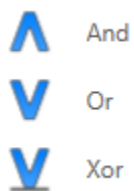
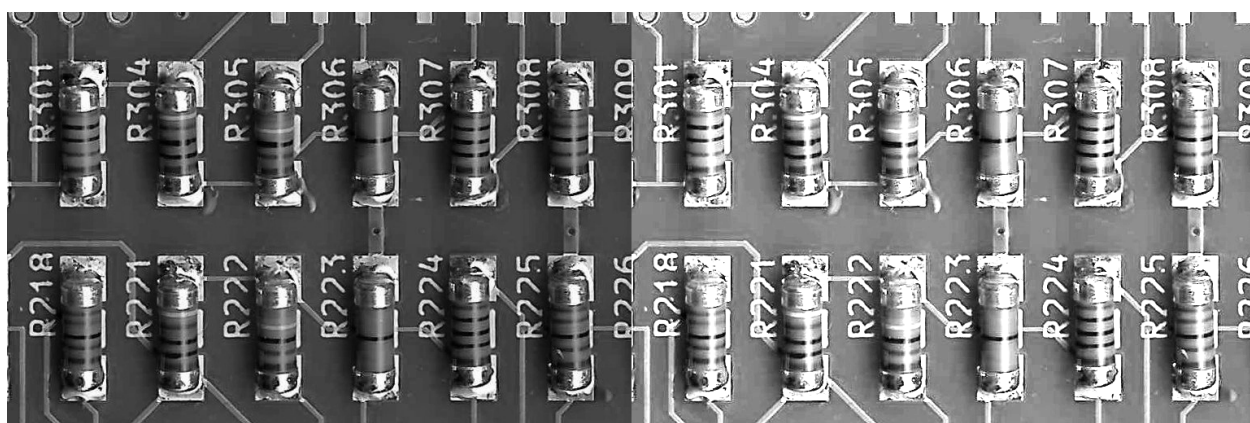
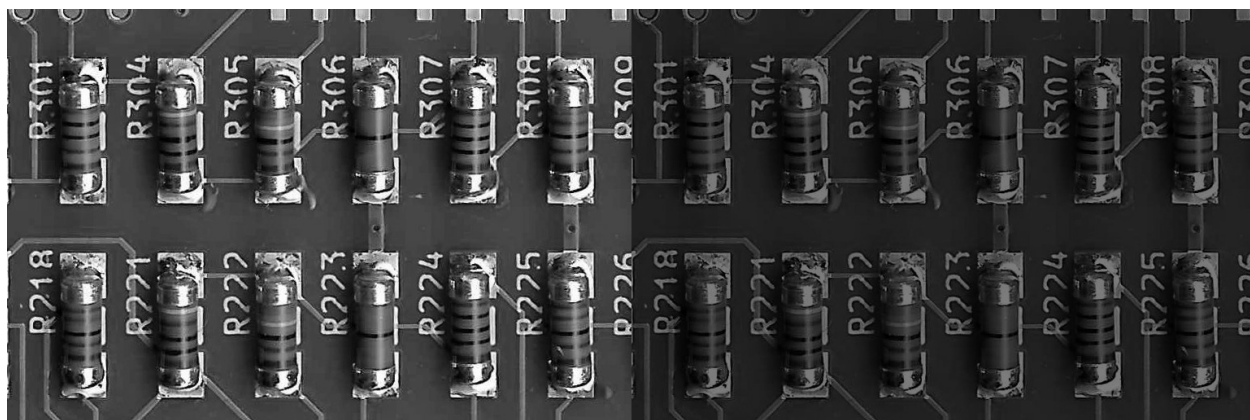
Divide (Blend)

Divide with saturation.

Image / Constant -> Result

Logic Operators

The following operators are available:



And

Logical And.

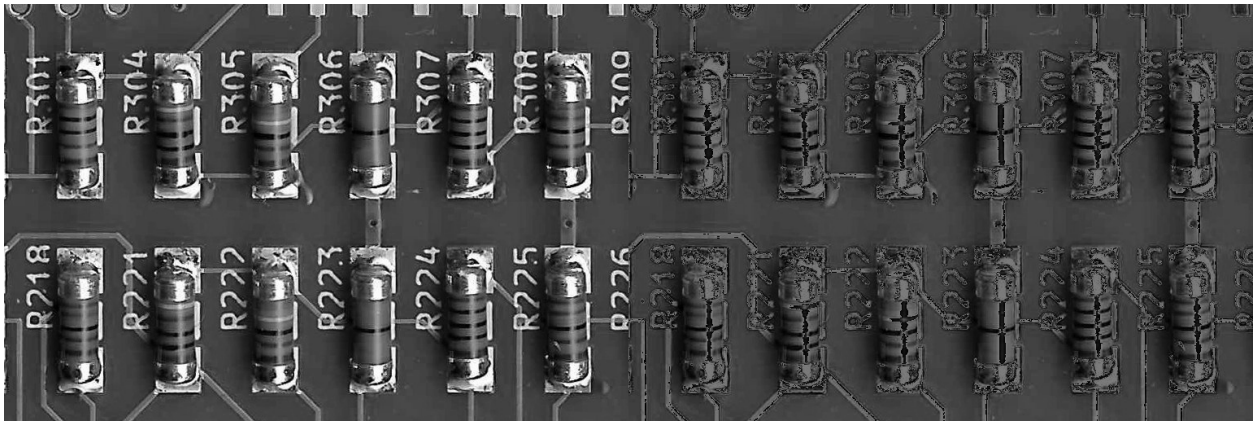


Image & Constant -> Result

Or

Logical Or.

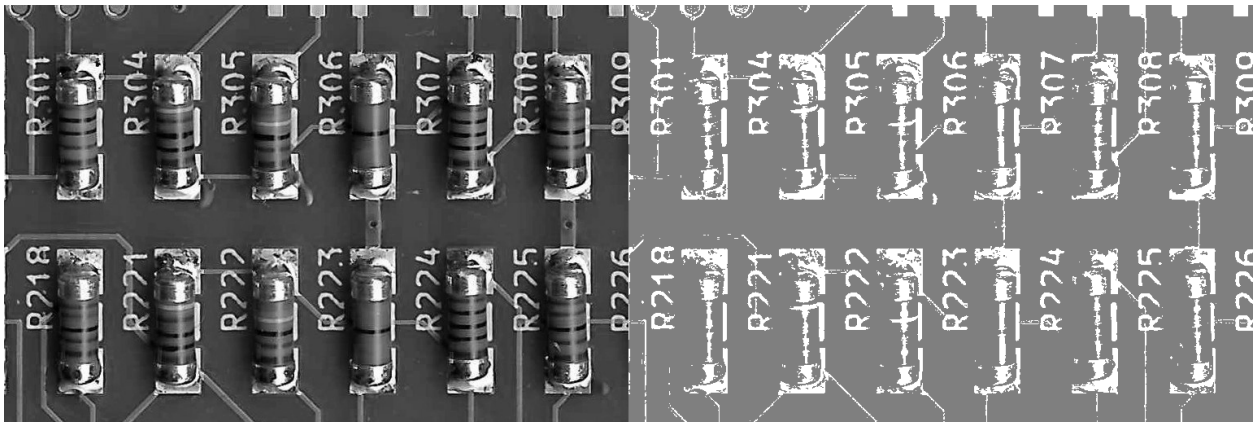


Image | Constant -> Result

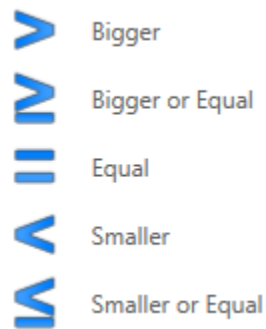
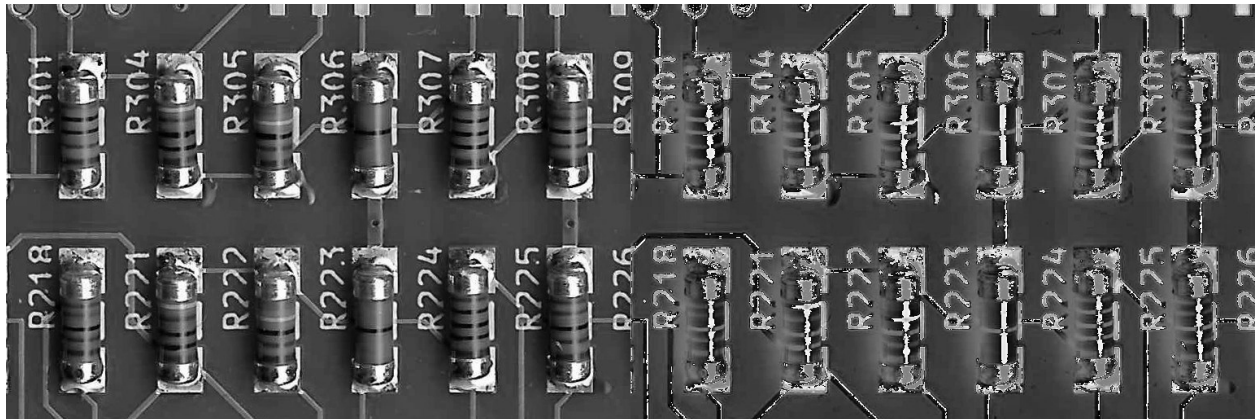
Xor

Logical Xor.

Image ^ Constant -> Result

Comparison Operators

The following operators are available:



Smaller

Compare smaller.

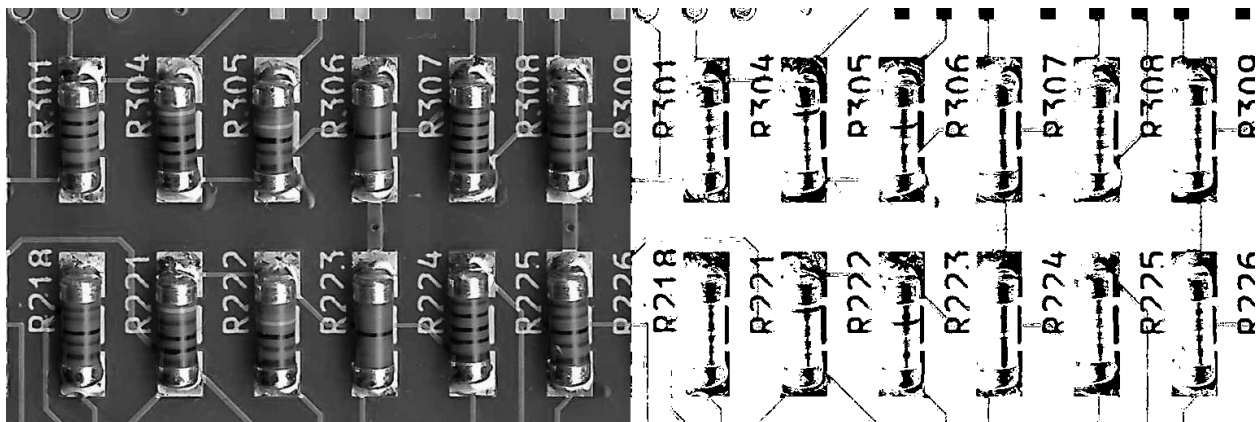
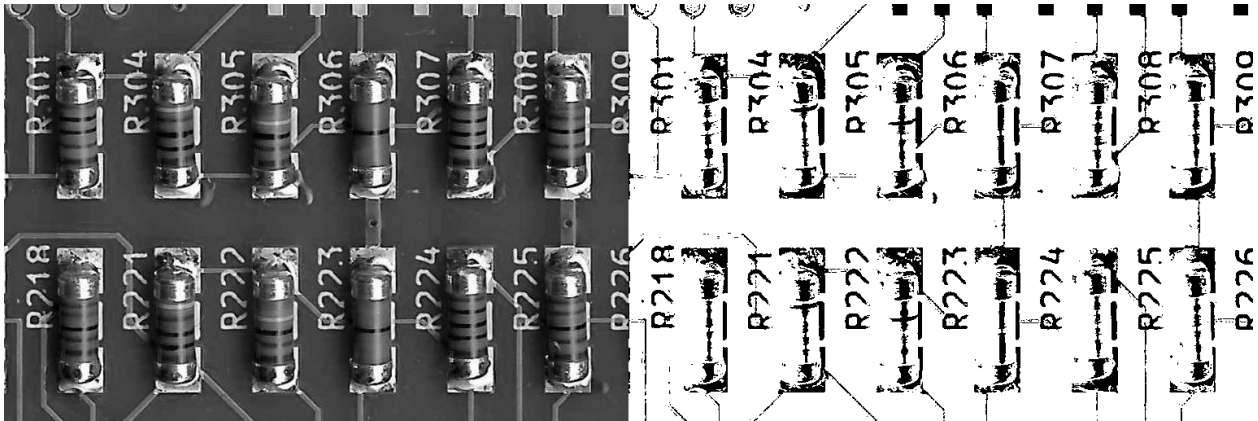


Image < Constant -> Result

Smaller or Equal

Compare smaller or equal.

Image <= Constant -> Result



Equal

Compare equal.

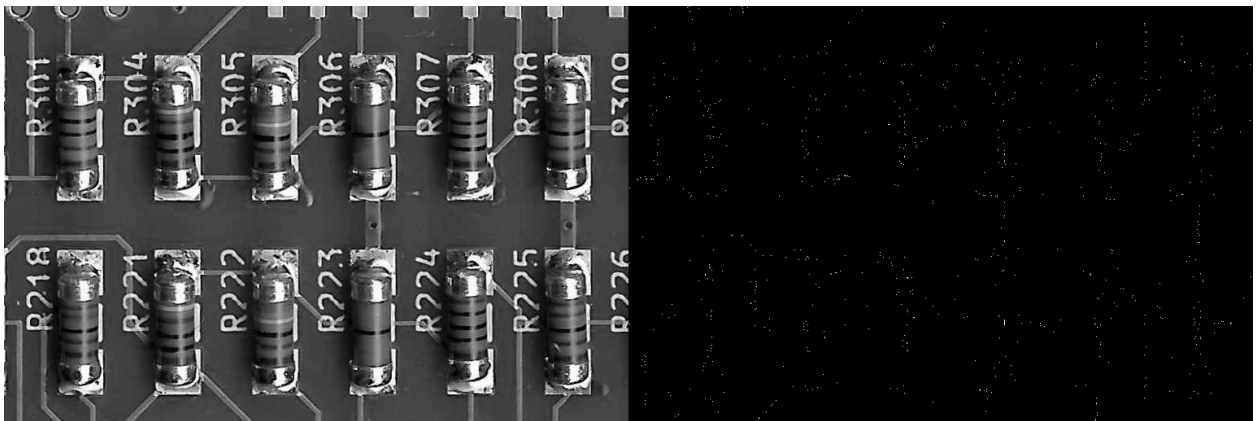


Image == Constant -> Result

Bigger or Equal

Compare bigger or equal.

Image >= Constant -> Result

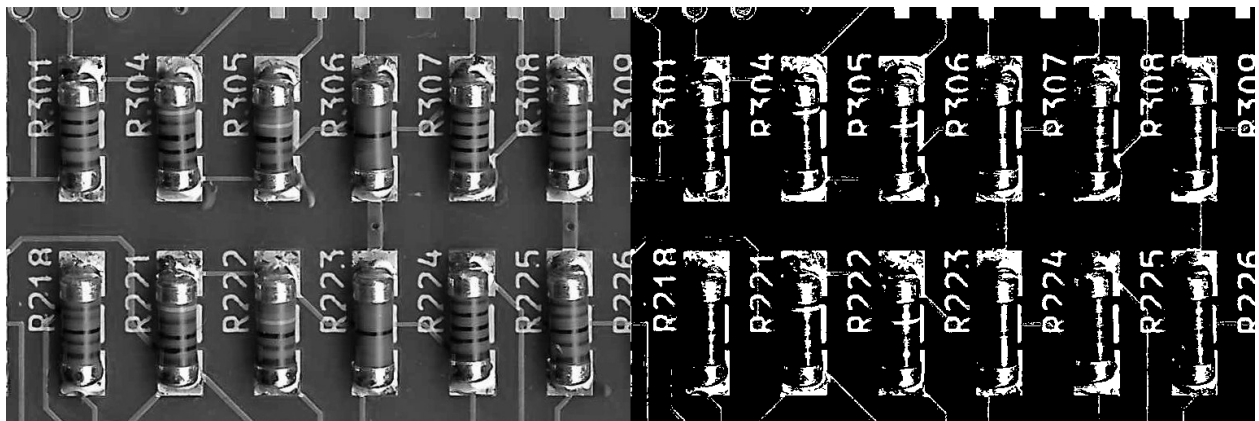
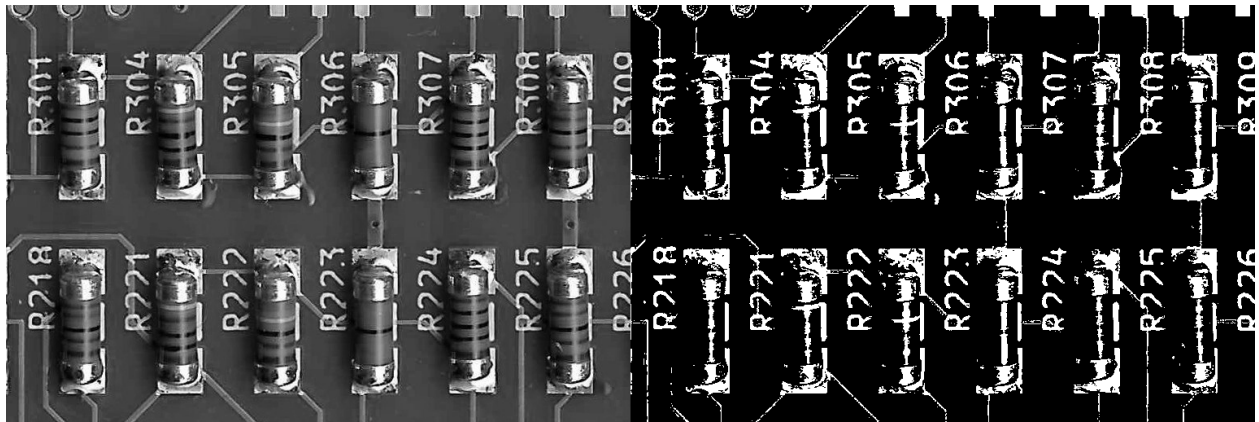
Bigger

Compare bigger.

Image > Constant -> Result

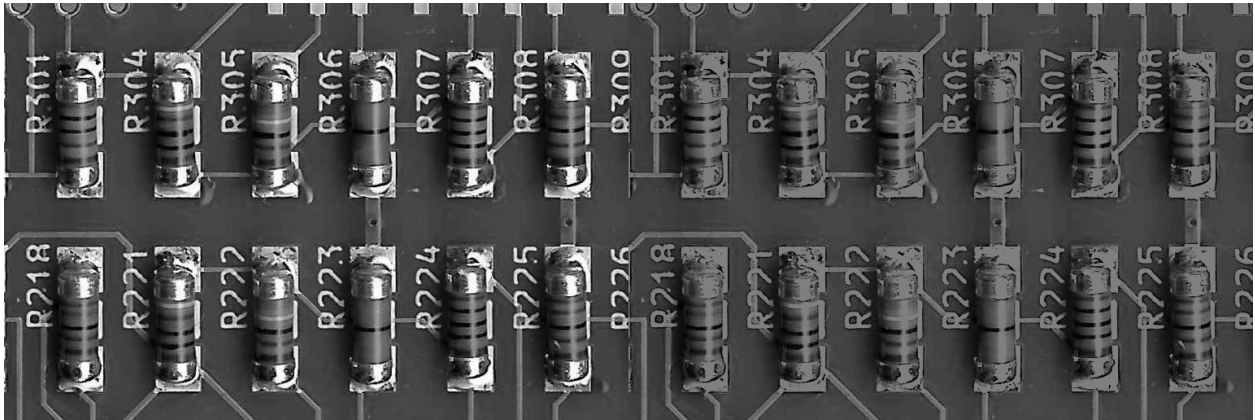
Min/Max Operators

The following operators are available:



Min

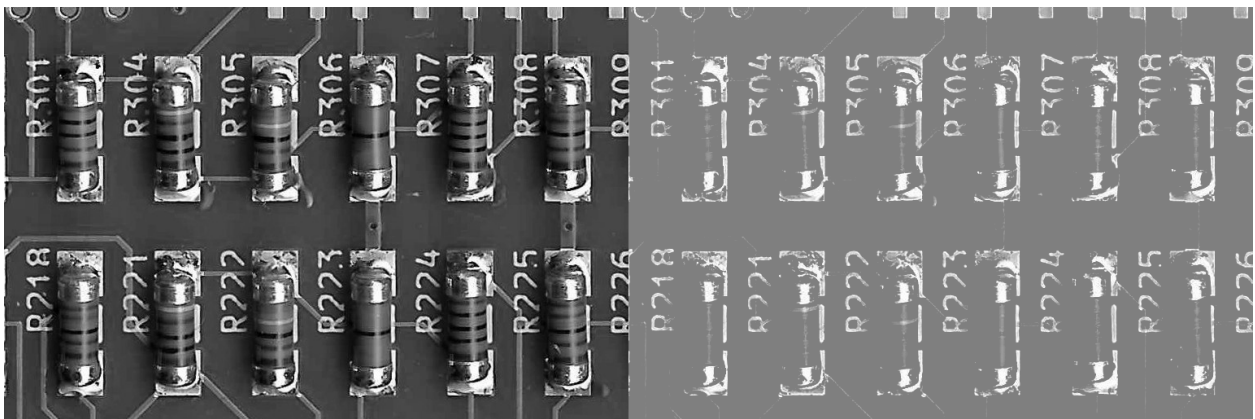
Minimum.



`min(Image, Constant) -> Result`

Max

Maximum.



`max(Image, Constant) -> Result`

Inputs**Image (Type: Image)**

The input image.

Constant (Type: object)

The constant.

Operator (Type: string)

Specifies the operator.

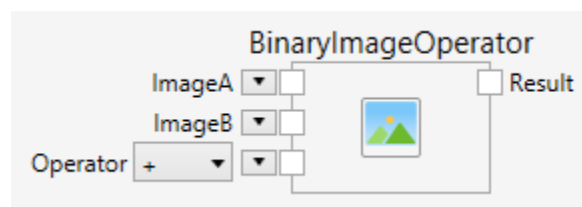
operator	operation
+	add
-	subtract
diff	difference
*	multiply
/	divide
*_blend	multiply (blend)
/_blend	divide (blend)
&	and
	or
^	xor
<	smaller
<=	smaller or equal
==	equal
>=	bigger or equal
>	bigger
min	minimum
max	maximum

Outputs**Result (Type: Image)**

The result image.

9.1.4 BinaryImageOperator

Applies a binary point operation between two images. Possible operations can be grouped into arithmetic, logic and comparison fields. Binary point operations are applied pixel by pixel, the same way for every pixel, for corresponding pixels of two images.



The following operators are available:








Add with saturation.

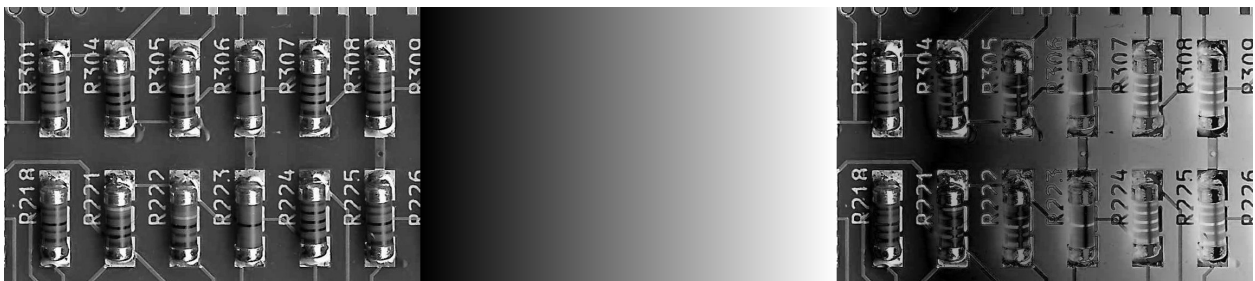
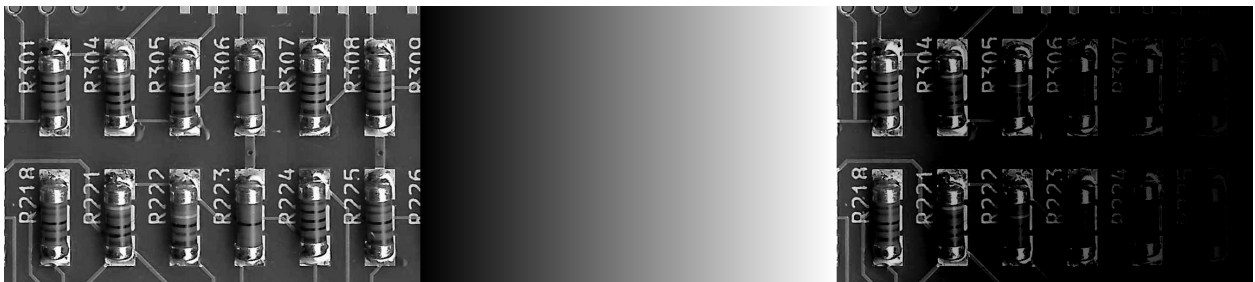
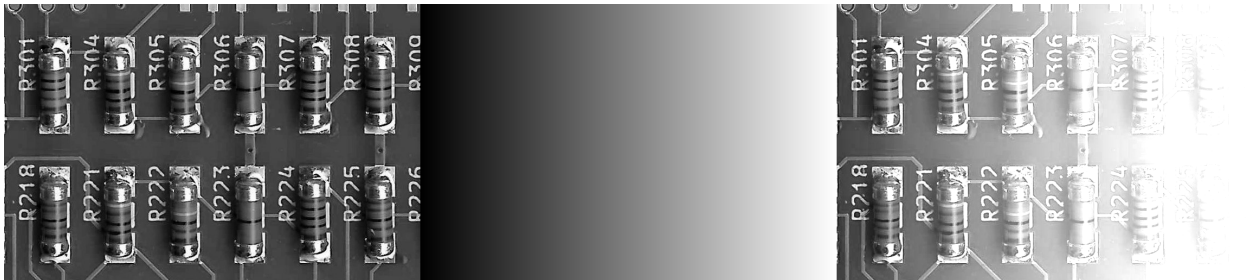
Image + Gradient -> Result

Subtract with saturation.

Image - Gradient -> Result

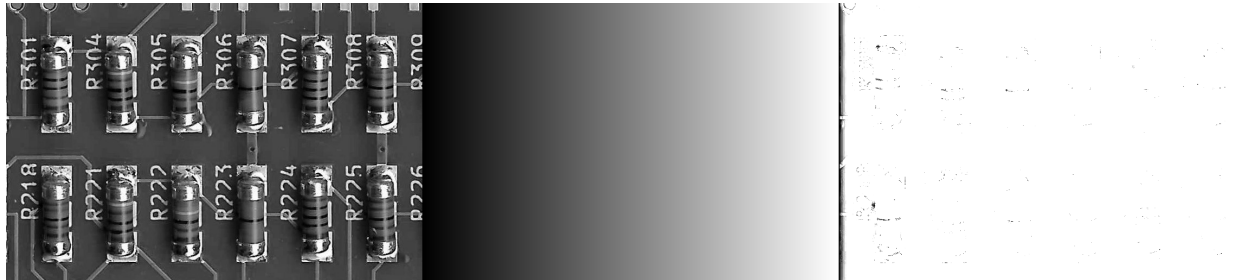
Difference with saturation.

-  Add
-  Subtract
-  Difference
-  Multiply
-  Divide
-  Multiply (Blend)
-  Divide (Blend)



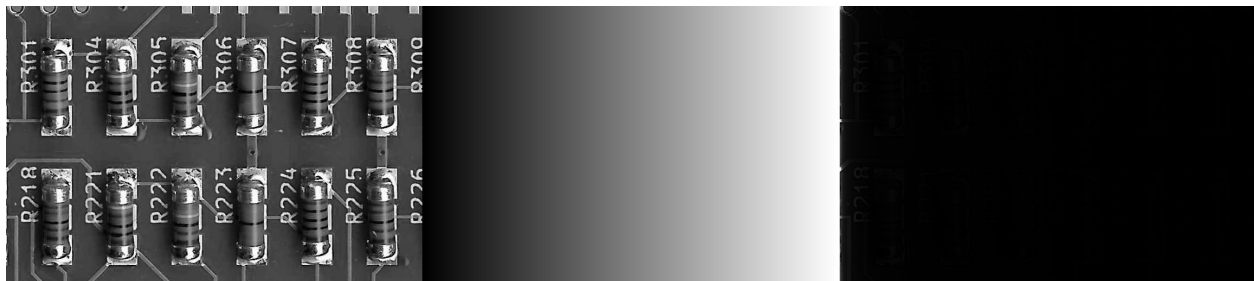
$\text{abs}(\text{Image} - \text{Gradient}) \rightarrow \text{Result}$

Multiply without saturation.



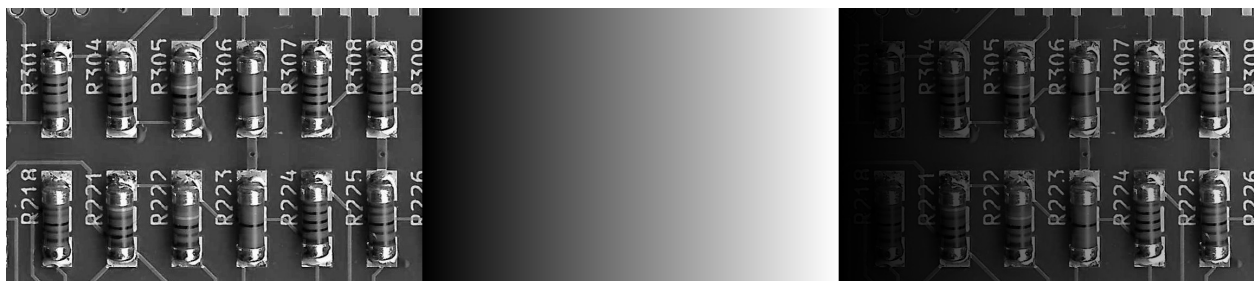
$\text{Image} * \text{Gradient} \rightarrow \text{Result}$

Divide without saturation.



$\text{Image} / \text{Gradient} \rightarrow \text{Result}$

Multiply with saturation.



$\text{Image} * \text{Gradient} \rightarrow \text{Result}$

Divide with saturation.

$\text{Image} / \text{Gradient} \rightarrow \text{Result}$

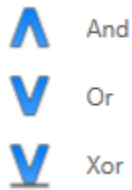
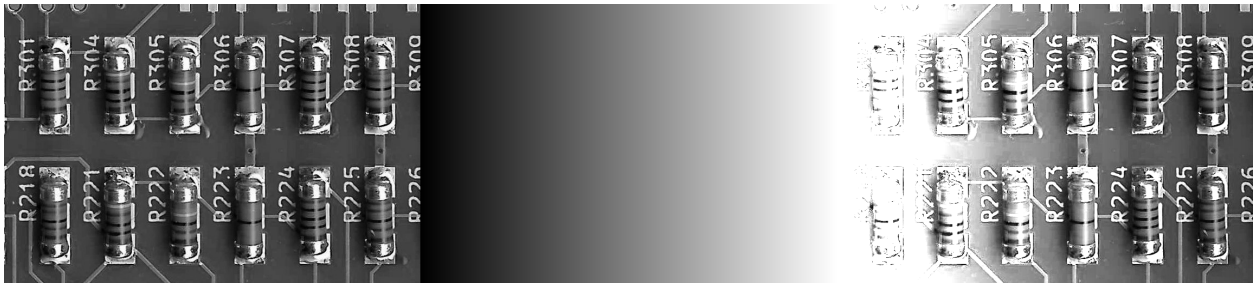
Logic Operators

The following operators are available:

And

Logical And.

$\text{Image} \& \text{Gradient} \rightarrow \text{Result}$



Or

Logical Or.

Image | Gradient -> Result

Xor

Logical Xor.

Image ^ Gradient -> Result

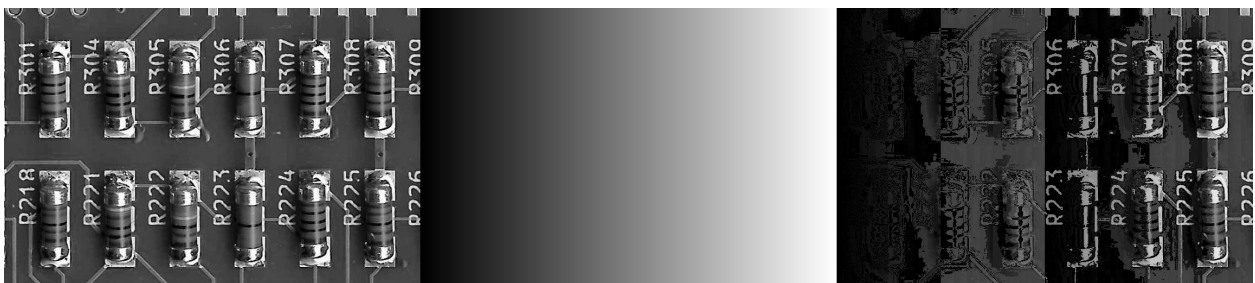
Comparison Operators

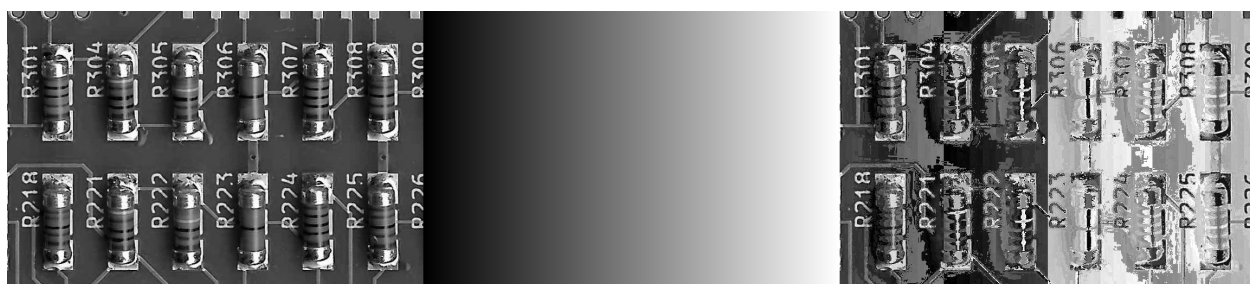
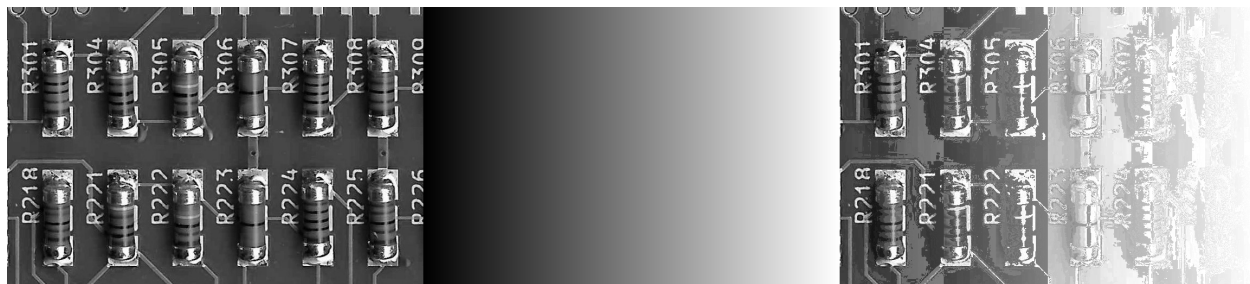
The following operators are available:






Smaller

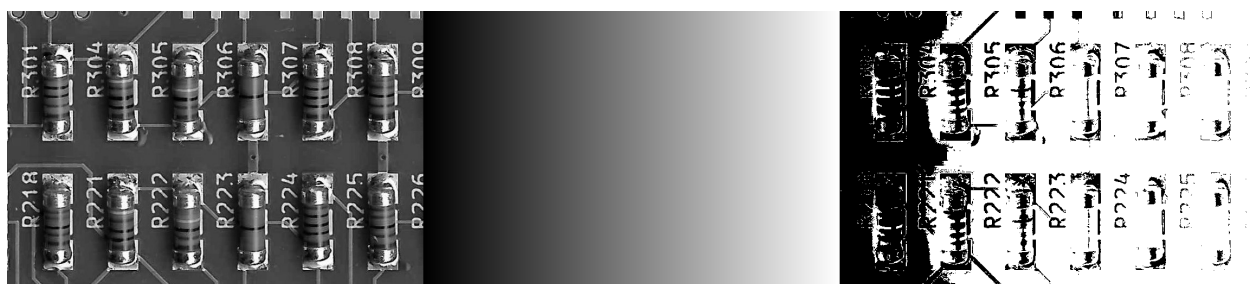
Compare smaller.

Image < Gradient -> Result





-  Bigger
-  Bigger or Equal
-  Equal
-  Smaller
-  Smaller or Equal



Smaller or Equal

Compare smaller or equal.

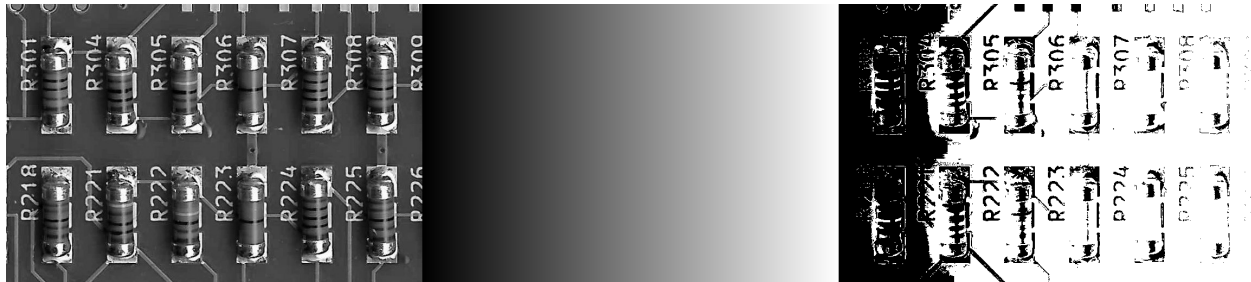


Image \leq Gradient \rightarrow Result

Equal

Compare equal.

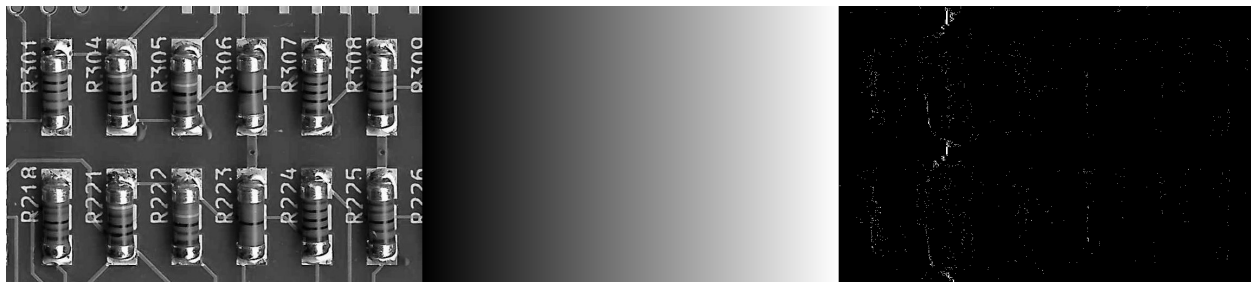


Image $=$ Gradient \rightarrow Result

Bigger or Equal

Compare bigger or equal.

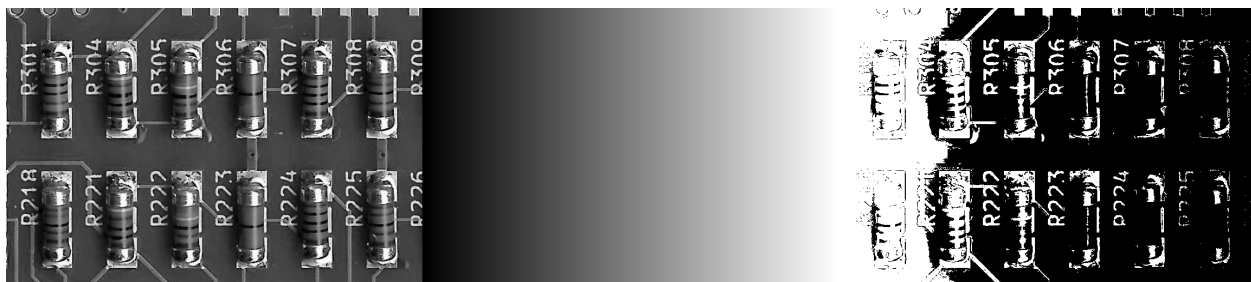


Image \geq Gradient \rightarrow Result

Bigger

Compare bigger.

Image $>$ Gradient \rightarrow Result



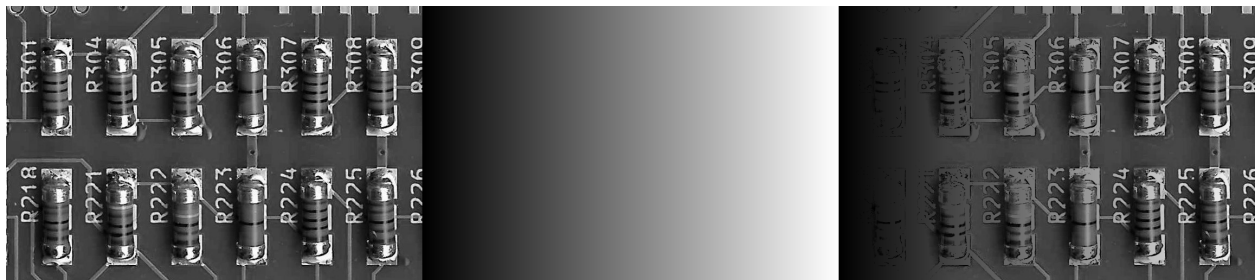
Min/Max Operators

The following operators are available:



Min

Minimum.



`min(Image, Gradient) -> Result`

Max

Maximum.

`max(Image, Gradient) -> Result`

Inputs

ImageA (Type: Image)

The first input image.



ImageB (Type: Image)

The second input image.

Operator (Type: string)

Specifies the operator.

operator	operation
+	add
-	subtract
diff	difference
*	multiply
/	divide
*_blend	multiply (blend)
/_blend	divide (blend)
&	and
	or
^	xor
<	smaller
<=	smaller or equal
==	equal
>=	bigger or equal
>	bigger
min	minimum
max	maximum

Outputs

Result (Type: Image)

The result image.

9.2 Color

The term color space describes the representation of colors with numbers. An infinite number of color spaces is in existence and a large number is in everyday use today.

Physically, color is a sensation that can be described with a spectrum, a mixture of light waves of different wavelengths between 380 nm and 700 nm. The spectrum is a physical quantity that can be measured by appropriate means.

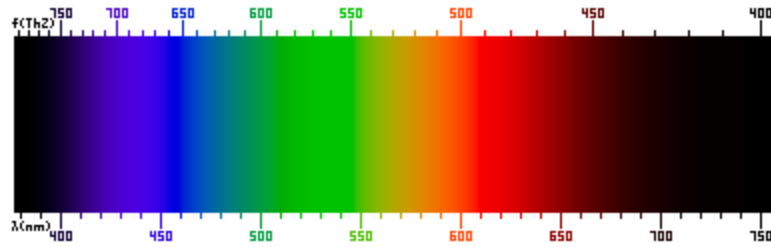


Fig. 9.1: The Visible Spectrum

However, color is also a human sensation that is formed in the human brain by means of stimulation via the eye. This makes it a bit difficult to talk about color, since some people might perceive it slightly different. In fact, there are people which exhibit certain forms of color blindness such that their perception is different from people with normal color vision.

Since there seems to be slightly different color perception even among non-color-blind people, the CIE has defined a normal observer with a defined color perception.

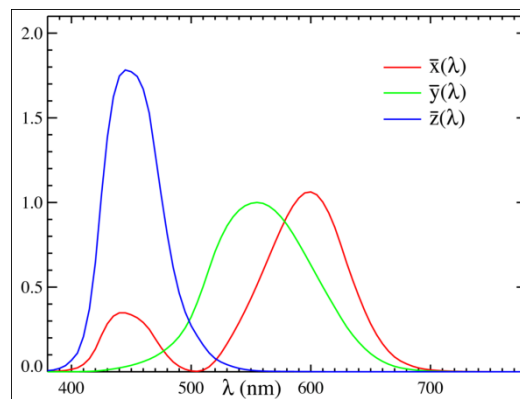


Fig. 9.2: The CIE Standard Observer Color Matching Functions

The normal observer has been defined in 1931 by testing around 200 people with normal vision. These people have been asked to mix a color that they observed on a 2 degree view centered in the middle of the retina with three colored light sources (red, green and blue). The numbers of the mixtures have been recorded and served to derive the normal spectrum. In 1964 a similar experiment has been carried out to determine the 10 degree normal spectrum.

Color primaries (red, green and blue) are mixed to approximate the color spectrum in most television and computer displays.

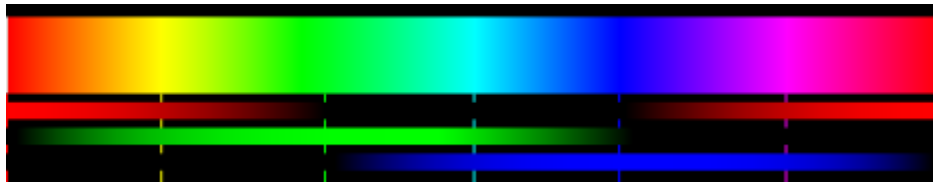


Fig. 9.3: The Color Display Spectrum

9.2.1 Use of Color Spaces in Image Reproduction and Television

Image reproduction in print or television faces very difficult problems, since images are often viewed in dramatically different conditions than they were taken. For example take a scene in the open field on a bright sunny day which has been filmed and is viewed by the audience in a dark cinema at total darkness. Yet, the color impression should be natural for the audience. Similar problems exist in print, where paper, ink and process differences make it very difficult to achieve satisfying results.

While the color space conversion functions in nVision can be used to compensate for these effects, **nVision** has not been designed specifically to provide the equivalent of a color management system.

9.2.2 Use of Color Spaces in Image Analysis

In image analysis color spaces are often used to provide a means for qualitative separation of colors. Depending on the specific needs, some color spaces are more suitable than others, and therefore the need of color space transformation arises.

The color space transformations available in nVision have been designed to target the needs in image analysis.

9.2.3 The CIE Color Spaces

With the standard observer color matching functions, the CIE defined the XYZ color space. The tristimulus values for a color with a spectral power distribution $I(\lambda)$ are given in terms of the standard observer by:

$$X = \int I(\lambda)\bar{x}(\lambda)d\lambda$$

$$Y = \int I(\lambda)\bar{y}(\lambda)d\lambda$$

$$Z = \int I(\lambda)\bar{z}(\lambda)d\lambda$$

Since the human eye has three types of color sensors, a full plot of all visible colors is a three-dimensional figure. However, if brightness is stripped off, chromaticity remains. For example brown is a darker version of yellow, they have the same chromaticity but different brightness.

The XYZ color space was designed so that the Y parameter was a measure of the brightness of a color. The chromaticity of a color could then be specified by the parameters x and y:

$$x = \frac{X}{X + Y + Z}$$

$$y = \frac{Y}{X + Y + Z}$$

Because of the normalization, the third parameter z is not needed:

$$z = \frac{Z}{X + Y + Z} = 1 - x - y$$

The derived color space specified by x, y and Y is known as the CIE xyY color space and is widely used to specify colors in practice.

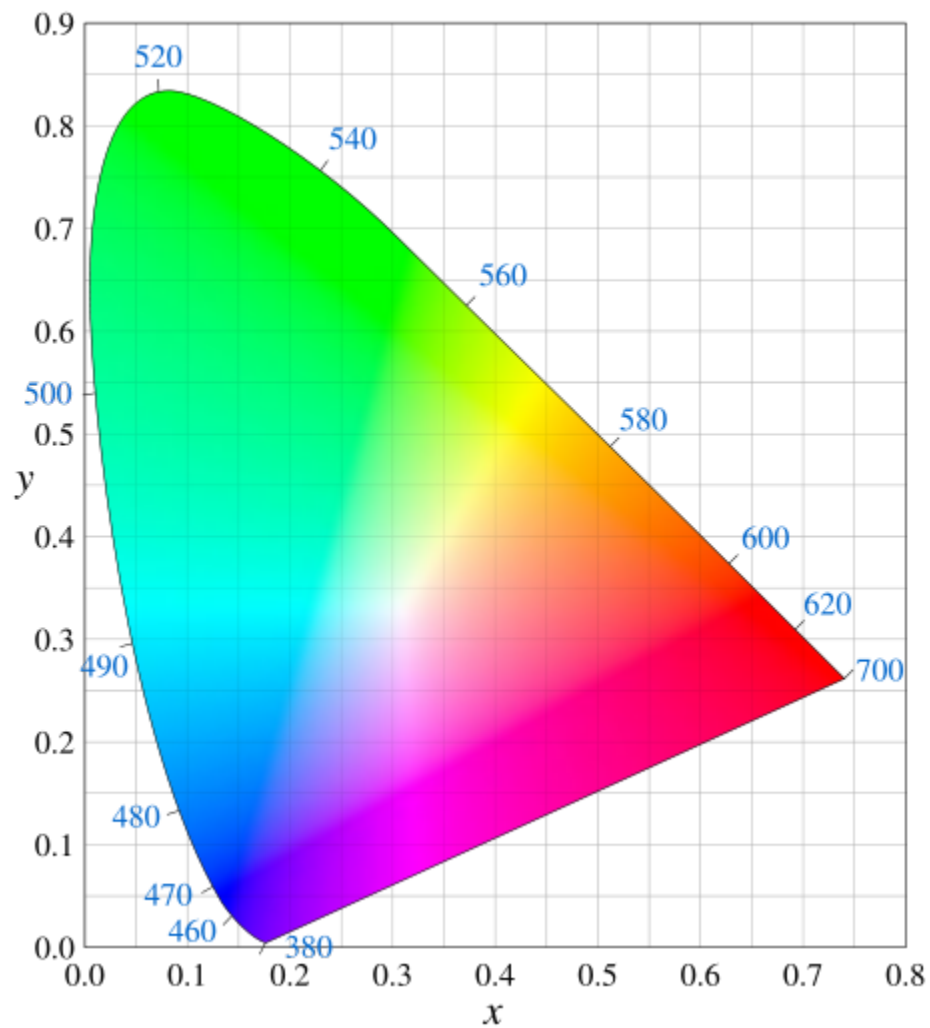


Fig. 9.4: The CIE Color Space Chromaticity Diagram

9.2.4 The RGB Color Space

There is not only one RGB color space. Instead, more information must be presented to precisely define the specific variant of RGB. Notably, the XYZ coordinates of the red, green and blue Primaries must be known, as well as the white point.

A very good source of information is the website of Bruce Lindbloom, notably the page titled “RGB Working Space Information”.

Most images nowadays use the sRGB color space, and this is also the default chosen by the nVision software.

9.2.5 The HLS and HSI Color Spaces

These color models are based on an artist’s concepts of tint, shade and tone. The coordinate system is cylindrical and the models are designed to provide intuitive color specification.

The HLS (hue, lightness, saturation) color model forms a double cone in a cylindrical space. Hue is the angle around the vertical axis, with red at 0 degrees, green at 120 degrees and blue at 240 degrees. The colors occur around the perimeter in the same order as in the CIE chromaticity diagram. Lightness is along the vertical center axis and ranges from black at the bottom over the various grey shades to white at the top. Saturation is the radial distance from the center axis and specifies how much color and grey are mixed.

The HSI (hue, saturation, intensity) color model forms a cone, where the top plane contains white in the center and saturated colors around the perimeter. The bottom tip of the cone is black. As with the HLS model, hue is the angle around the vertical axis, with red at 0 degrees, green at 120 degrees and blue at 240 degrees.

The mathematics of the both models are described in “Foley, van Dam: Computer Graphics: Principles and Practice”.

Both color spaces make it easy to segment images based on colors.

9.2.6 The L*a*b* Color Space

The L*a*b* color space (sometimes also called CIELAB) is a color opponent space. It is physically non-linear to provide perceptual uniformity. The difference between colors can then be calculated simply by computing their Euclidean distance. While HLS/HSI provide are intuitive, but not uniform, L*a*b* adds uniformity.

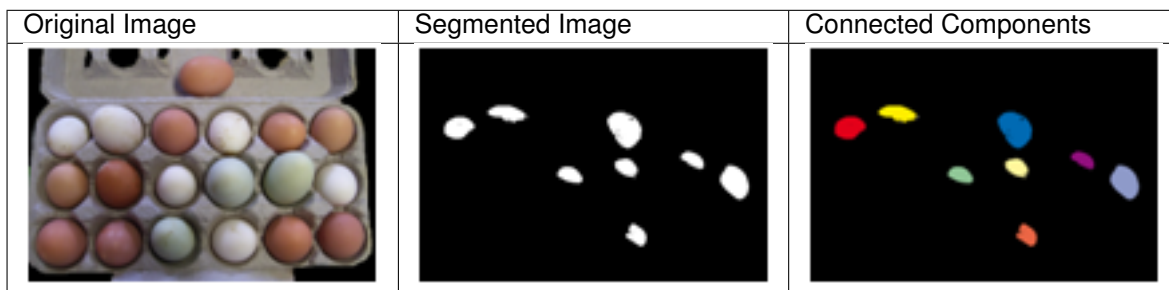
The mathematics of the L*a*b* color space is described in “Reinhard: Color Imaging”.

9.2.7 Color Space Conversion

nVision has functions to convert images between different color spaces.

10.1 Blob Analysis

A group of connected pixels is commonly called a blob. Here is an example of an image containing a few blobs. The original image is segmented and the connected components are determined and visualized by color.



Blob analysis takes a set of connected pixels that are usually generated by imaging real-life objects, and calculates features from the pixel set, such as their area, their position, etc.

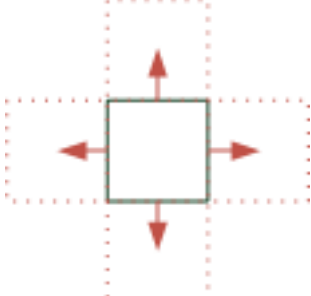
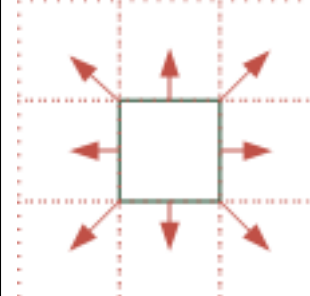
Often blobs are produced by some segmentation process, and one of the simplest segmentation methods is binary thresholding: every pixel with a value below the threshold is part of the background and every other pixel is part of some blob. Exactly which blob the pixel is a part of is determined by a connectivity analysis process sometimes called labelling or connected components analysis.

The smallest possible blob consists of a single pixel, and larger blobs consist of several connected pixels.

Have a look at the image above: Does this picture contain one or two blobs? It depends on how you look at it. If you only allow connections via horizontal and vertical neighbours, the picture contains two blobs. However, if diagonal connections count as well, the picture contains one blob only.



Fig. 10.1: Does this picture contain one or two blobs?

4-connected	8-connected
	

By convention, one often uses 8-connectedness for the foreground and 4-connectedness for the background pixels. According to this convention, the following picture contains one white object with two black holes.

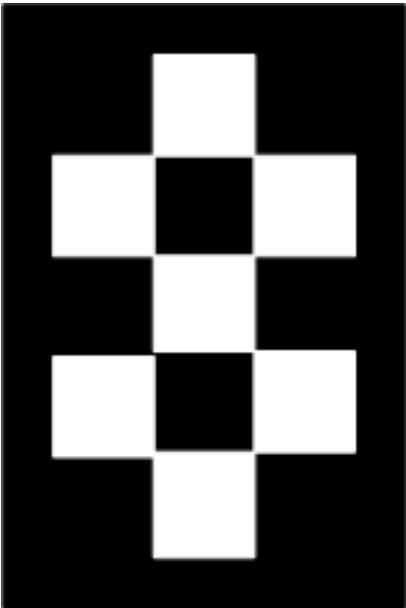


Fig. 10.2: One white object with two black holes

One can summarize the relations of the above picture with the following table.
The table clearly shows that there are only two useful connectivity configurations:

Foreground Connectivity	Background Connectivity	Objects / Holes
4	4	7 / 2
4	8	7 / 0
8	4	1 / 2
8	8	1 / 0

- One (white) object with two (black) holes, or
- Seven (white) objects with no holes.

The other possible configurations do not make any sense, regardless how you look at them.

By default, **nVision** assumes that foreground pixels are 8-connected and background pixels are 4-connected.

nVision uses regions to represent blobs. Blob analysis is then the process of calculating region features. Because of the efficient run-length storage of regions, these features can be calculated very quickly. Since a region does not contain any pixel values, region features do not need to touch pixels. For a surprisingly large number of features pixel values are not necessary, such as the area or the center of gravity. Here is an example of how a `region_list` can be created by thresholding and connected components analysis:

However, there is also a class of features that can only be calculated when you take pixel values into account, such as the mean brightness of a blob. **nVision** has a set of functions that can calculate pixel based features as well, but keep in mind that the performance requirements of calculating these features are higher. For pixel-based features, you still need a region to specify the object.

Here is a list of the region features:

Feature	Description
Area	The area as a number of pixels.
Left	The leftmost column position (inclusive).
Right	The rightmost column position (exclusive).
Top	The topmost row position (inclusive).
Bottom	The bottommost row position (exclusive).
Width	The width (axis parallel).
Height	The height (axis parallel).
Aspect Ratio	The axis parallel aspect ratio.
Bounds	The axis-parallel bounding box.
Perimeter	The perimeter.
Compactness	The compactness.
Circularity	The circularity.
Sphericity	The sphericity.
Roughness	The roughness.
Roundness	The roundness.
Convex Hull	The convex hull.
Convex Area	The area of the convex hull.
Perforation	The perforation.
Convexity	The convexity.
Convex Perimeter	The perimeter of the convex hull.
Maximum Feret Diameter	The maximum diameter of the convex hull.
Minimum Feret Diameter	The minimum diameter of the convex hull.
Minimum Bounding Circle	The minimum bounding circle of the region.
Inner Contour	The inner contour (lies on the blob).
Inner Contour Perimeter	The length of the inner contour.
Outer Contour	The outer contour (lies on the background).

Continued on

Table 10.1 – continued from previous page

Feature	Description
Outer Contour Perimeter	The length of the outer contour.
Raw Moments	The moments based on the region.
Normalized Moments	The normalized moments based on the region.
Centroid	The center of gravity based on the region.
Central Moments	The central moments based on the region.
Equivalent Ellipse	The equivalent ellipse based on the region.
Equivalent Ellipse Eccentricity	the equivalent ellipse based on the region.
Scale Invariant Moments	The scale invariant moments based on the region.
Hu Moments	Hu moments based on the region.
Flusser Moments	Flusser moments based on the region.
Holes	The holes of the region in the form of a region_list of connected components.
Number of Holes	The number of holes of the region.
Area of Holes	The total area of the holes of the region.
Fiber Length	A length measurement of fiber-like objects.
Fiber Width	A thickness measurement of fiber-like objects.
Bounding Rectangle	A bounding rectangle of the region. This rectangle has the same orientation as the equivalent ellipse.
Minimum Area Bounding Rectangle	A bounding rectangle of the region. This rectangle has the smallest possible area.
Minimum Perimeter Bounding Rectangle	A bounding rectangle of the region. This rectangle has the smallest possible perimeter.

Here is a list of the pixel based features:

Feature	Description
Minimum Position	The position of the minimum.
Minimum	The minimum pixel value.
Maximum Position	The position of the maximum.
Maximum	The maximum pixel value.
Total	The total value.
Mean	The mean value.
Standard Deviation	The standard deviation.
Skewness	The skewness.
Channel Minimum	The per-channel minimum pixel value.
Channel Maximum	The per-channel maximum pixel value.
Contrast	The contrast.
Weber Contrast	The contrast according to Weber's formula.
Michelson Contrast	The contrast according to Michelson's formula.
Horizontal Profile	The horizontal profile.
Vertical Profile	The vertical profile.
Histogram	The histogram.
Raw Moments	The moments based on the grey values.
Normalized Moments	The normalized moments based on the grey values.
Centroid	The center of gravity based on the grey values.
Central Moments	The central moments based on the grey values.
Equivalent Ellipse	The equivalent ellipse based on the grey values.
Equivalent Ellipse Eccentricity	The equivalent ellipse based on the grey values.
Scale Invariant Moments	The scale invariant moments based on the grey values.
Hu Moments	Hu moments based on the grey values.
Flusser Moments	Flusser moments based on the grey values.

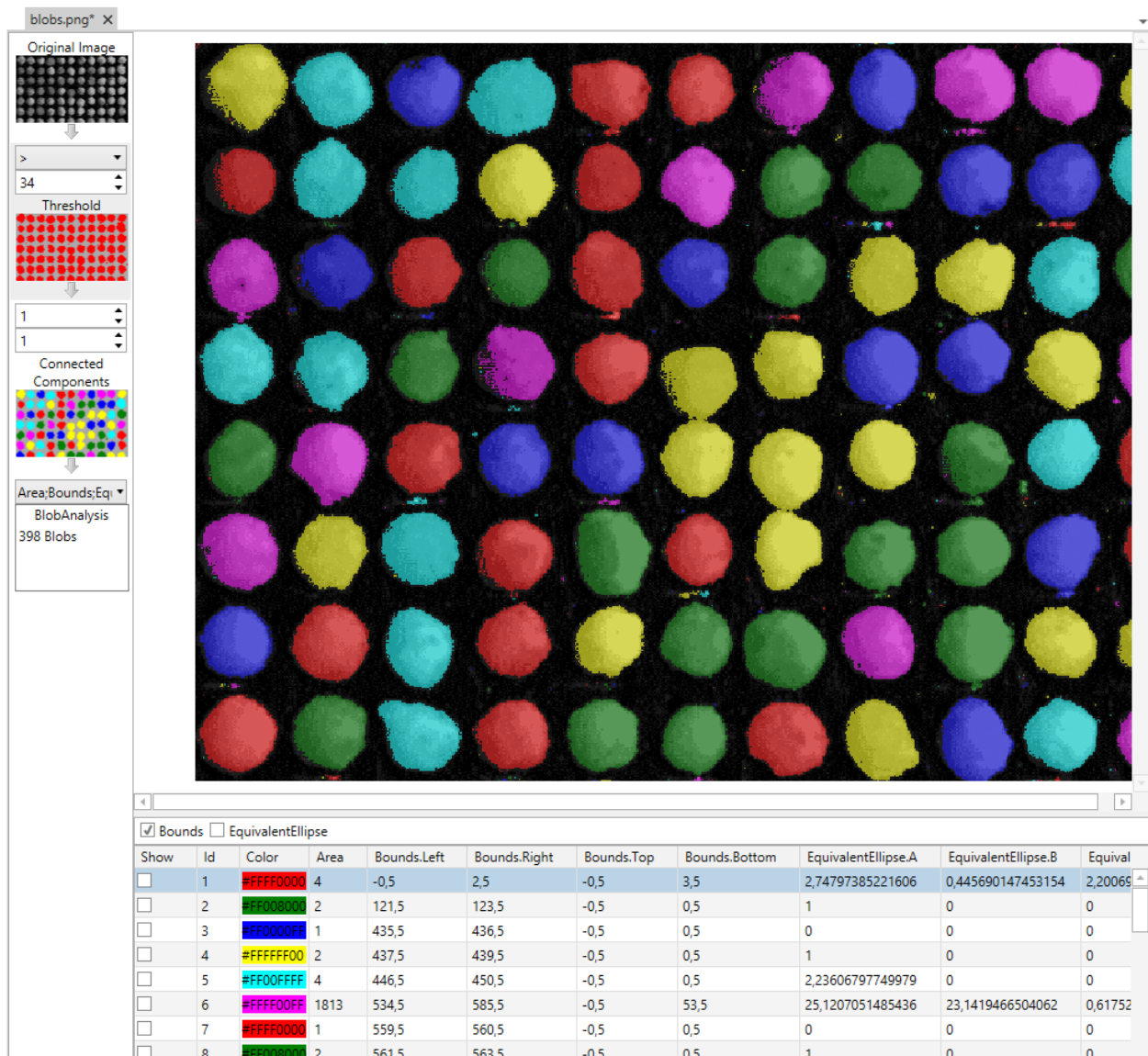


Fig. 10.3: Segmentation, connected components analysis and blob analysis.

10.2 Region Features

The simplest feature you may want to calculate - which is not a region feature in the strict sense - could be the number of objects in an image.

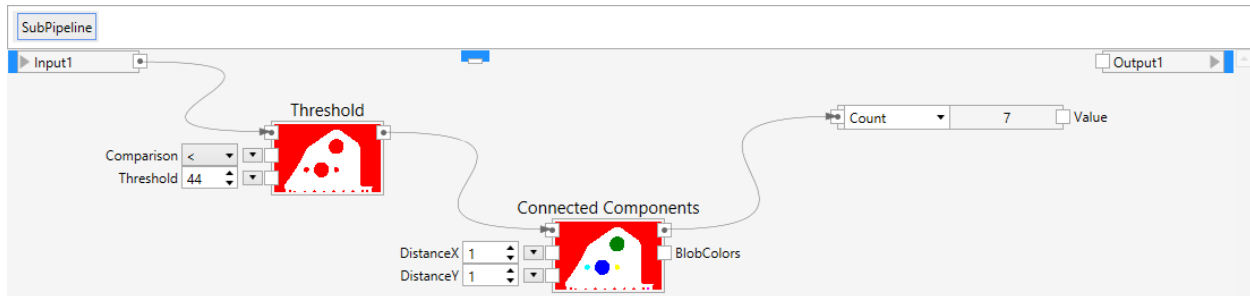


Fig. 10.4: Calculating the number of objects in an image.

The second simplest feature is the region area, and now we are really talking about region features.

10.2.1 Area

The area of a region is an integer value and specifies the number of pixels in the object. It is calculated efficiently by summing the lengths of all region chords together.

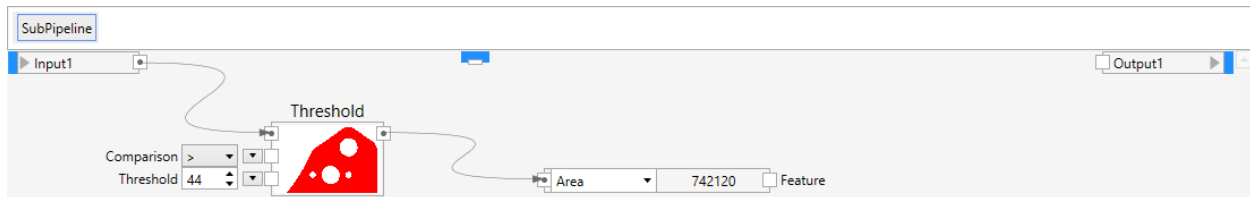


Fig. 10.5: Calculating the area.

The area property is often used to filter out small or large objects. Another way to calculate this feature is with the `blob_features` class:

10.2.2 Left

This calculates the minimal x coordinate (inclusive) of the region.

Regions that have a left value of 0 (or <0) are regions touching (or crossing) the left border. This feature is sometimes used to filter out objects touching the left border.

Since this is an integer coordinate, it is conceptually in the middle of the leftmost pixel.

10.2.3 Right

This calculates the maximal x coordinate (exclusive) of the region.

Regions that have a right value equal to the image width (or bigger than the image width) are regions touching (or crossing) the right border. This feature is sometimes used to filter out objects touching the right border.

Since this is an integer coordinate, it is conceptually half a pixel right of the rightmost pixel.

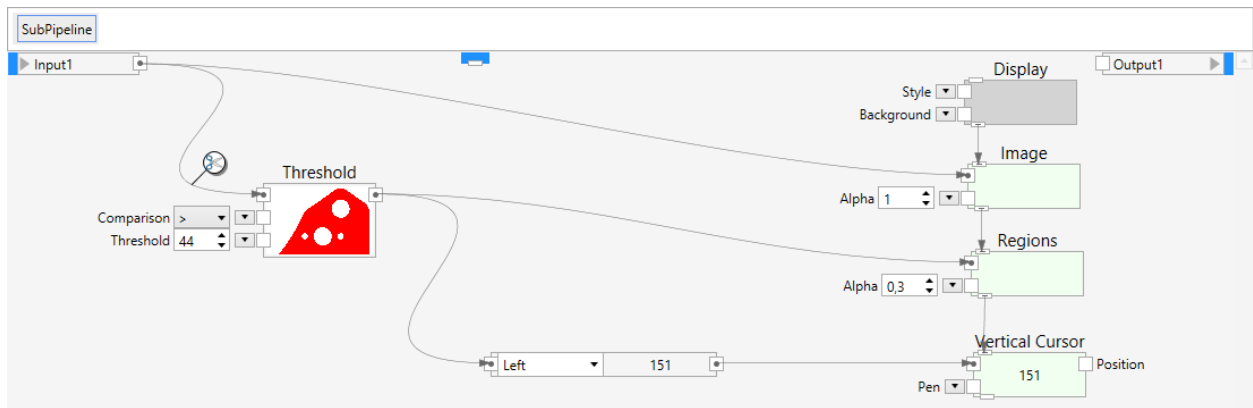


Fig. 10.6: Calculating the left position.

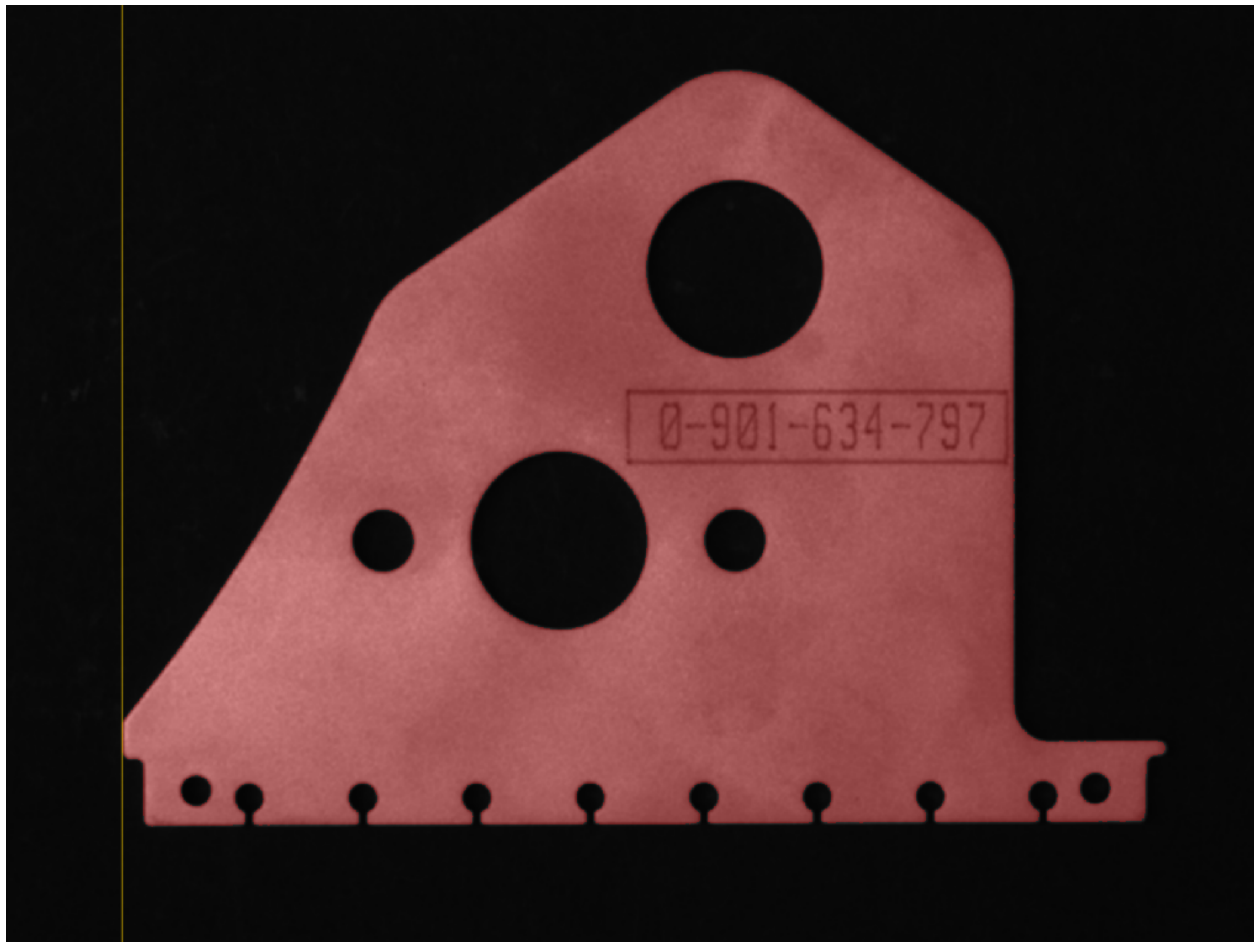


Fig. 10.7: Visualizing the left position.

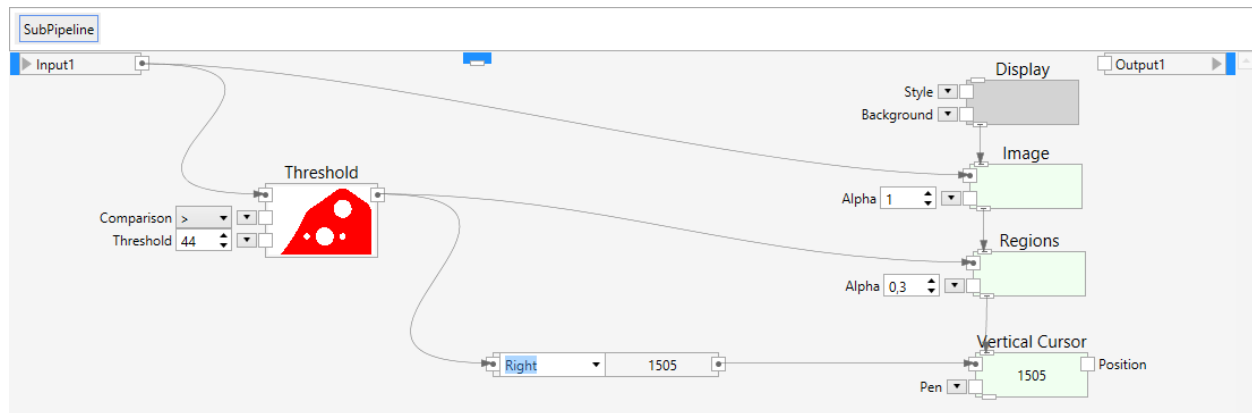


Fig. 10.8: Calculating the right position.

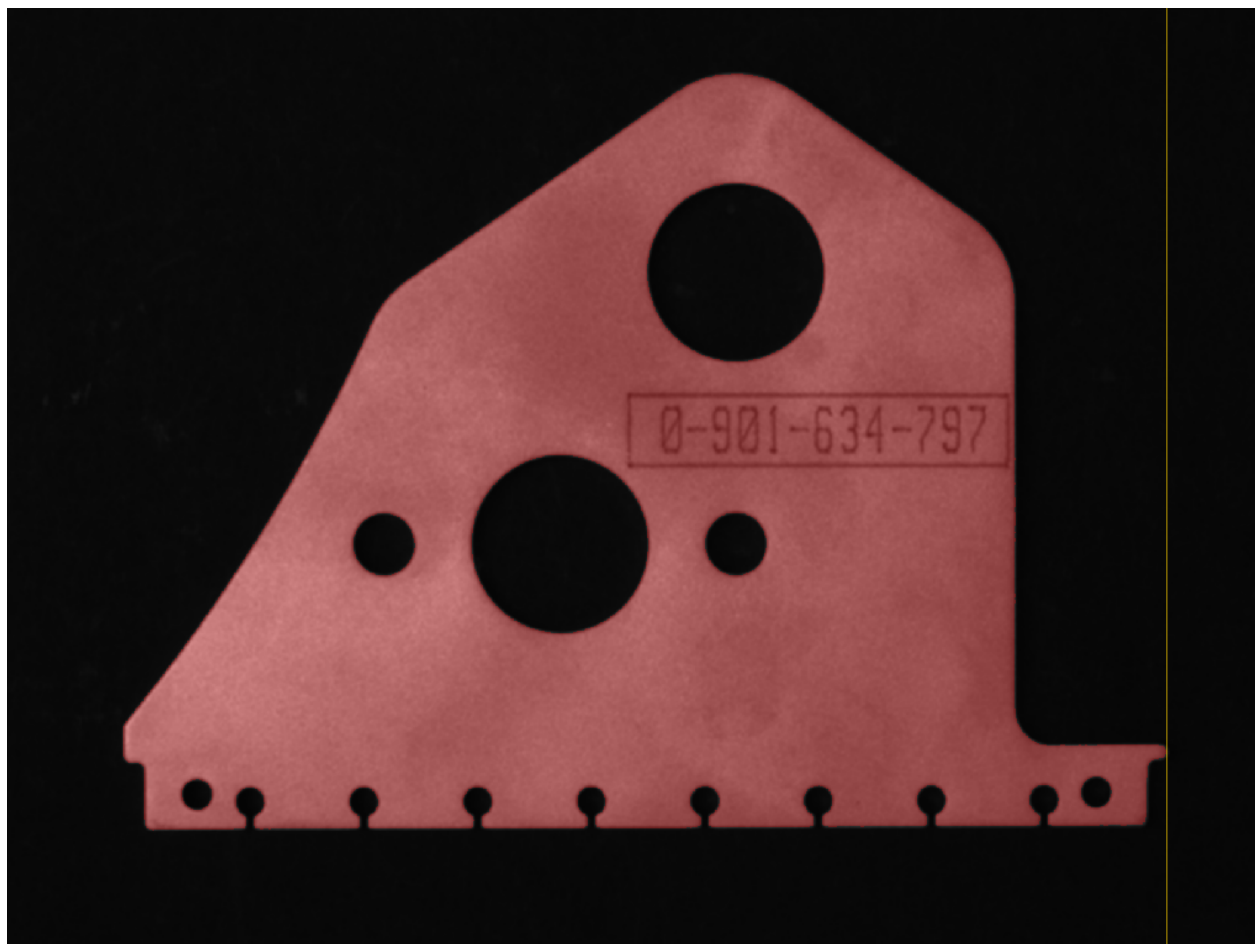


Fig. 10.9: Visualizing the right position.

10.2.4 Top

This calculates the minimal y coordinate (inclusive) of the region.

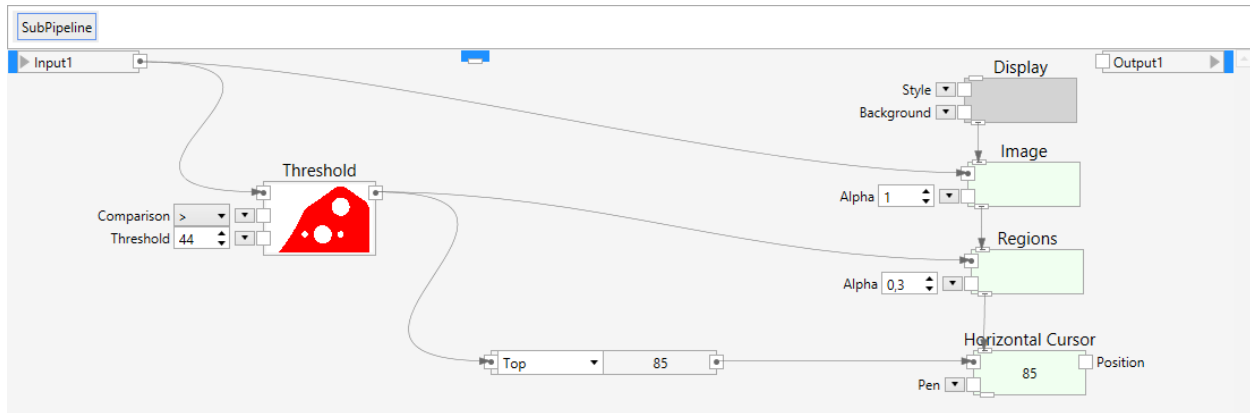


Fig. 10.10: Calculating the top position.

Regions that have a top value of 0 (or <0) are regions touching (or crossing) the top border. This feature is sometimes used to filter out objects touching the top border.

Since this is an integer coordinate, it is conceptually in the middle of the topmost pixel.

10.2.5 Bottom

This calculates the maximal y coordinate (exclusive) of the region.

Regions that have a bottom value equal to the image height (or bigger than the image height) are regions touching (or crossing) the bottom border. This feature is sometimes used to filter out objects touching the bottom border.

Since this is an integer coordinate, it is conceptually in the middle of the topmost pixel.

10.2.6 Width

This calculates the axis-parallel width of the region.

10.2.7 Height

This calculates the axis-parallel height of the region.

10.2.8 Aspect Ratio

This calculates the axis-parallel aspect ratio of the region. The aspect ratio is calculated according to the following formula:

$$r = \frac{w}{h}$$

where w is the width and h is the height.

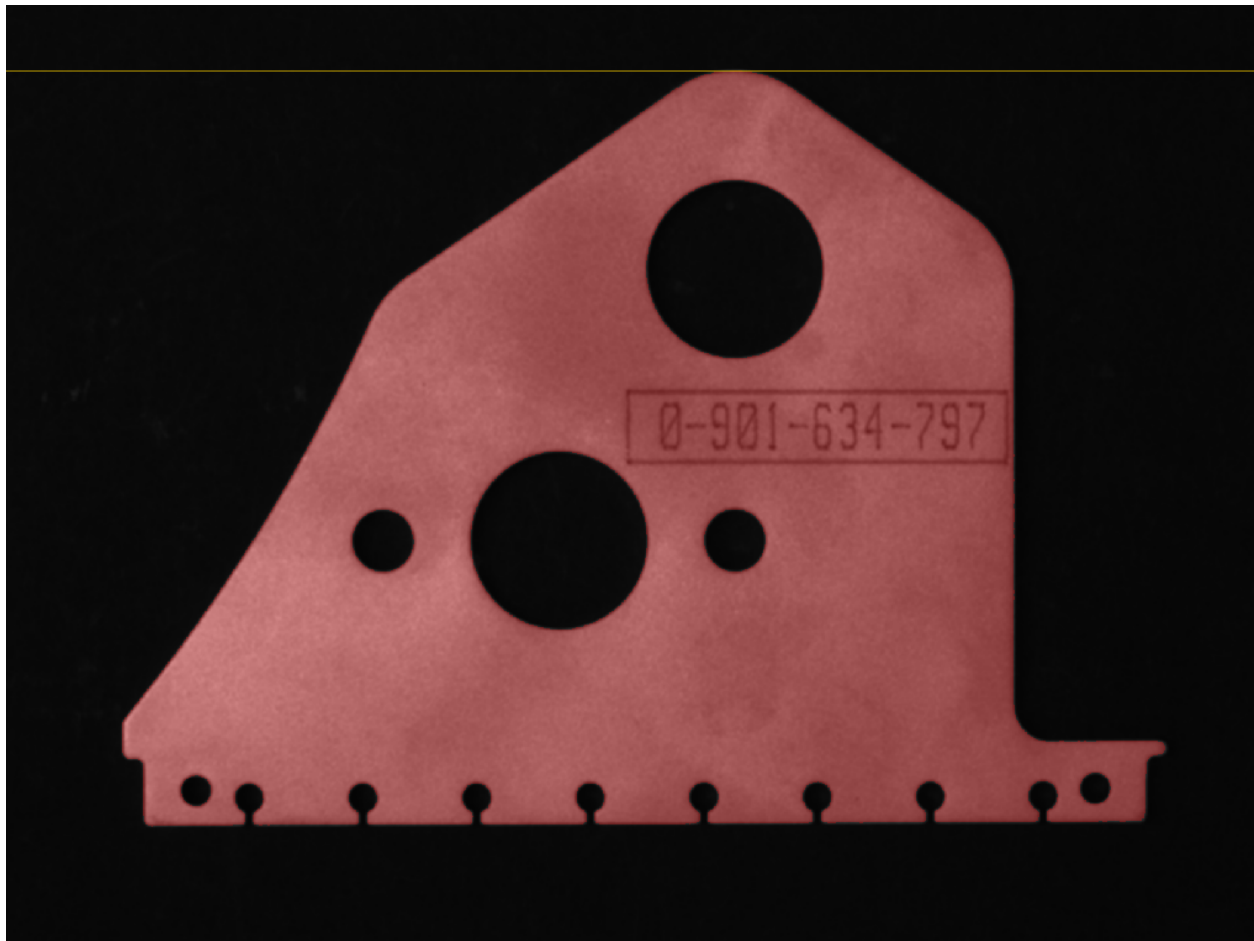


Fig. 10.11: Visualizing the top position.

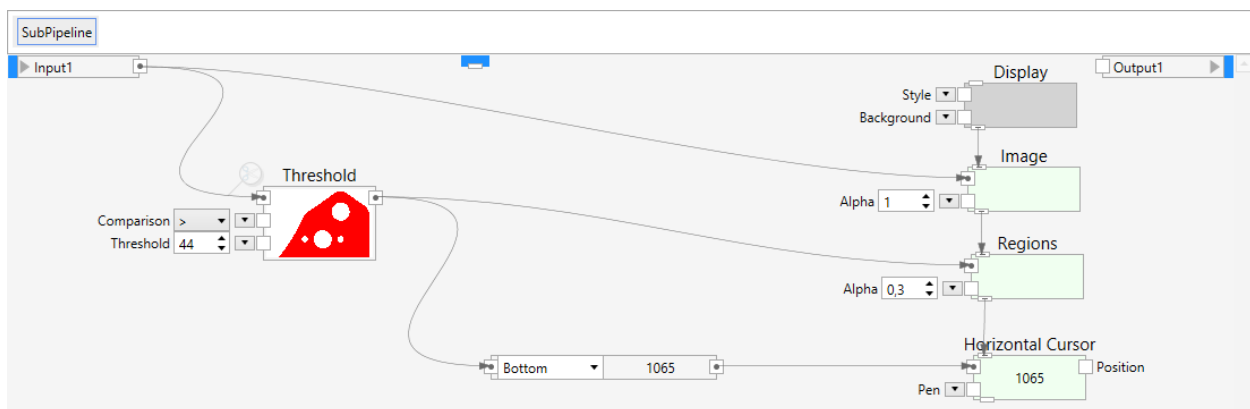


Fig. 10.12: Calculating the left position.

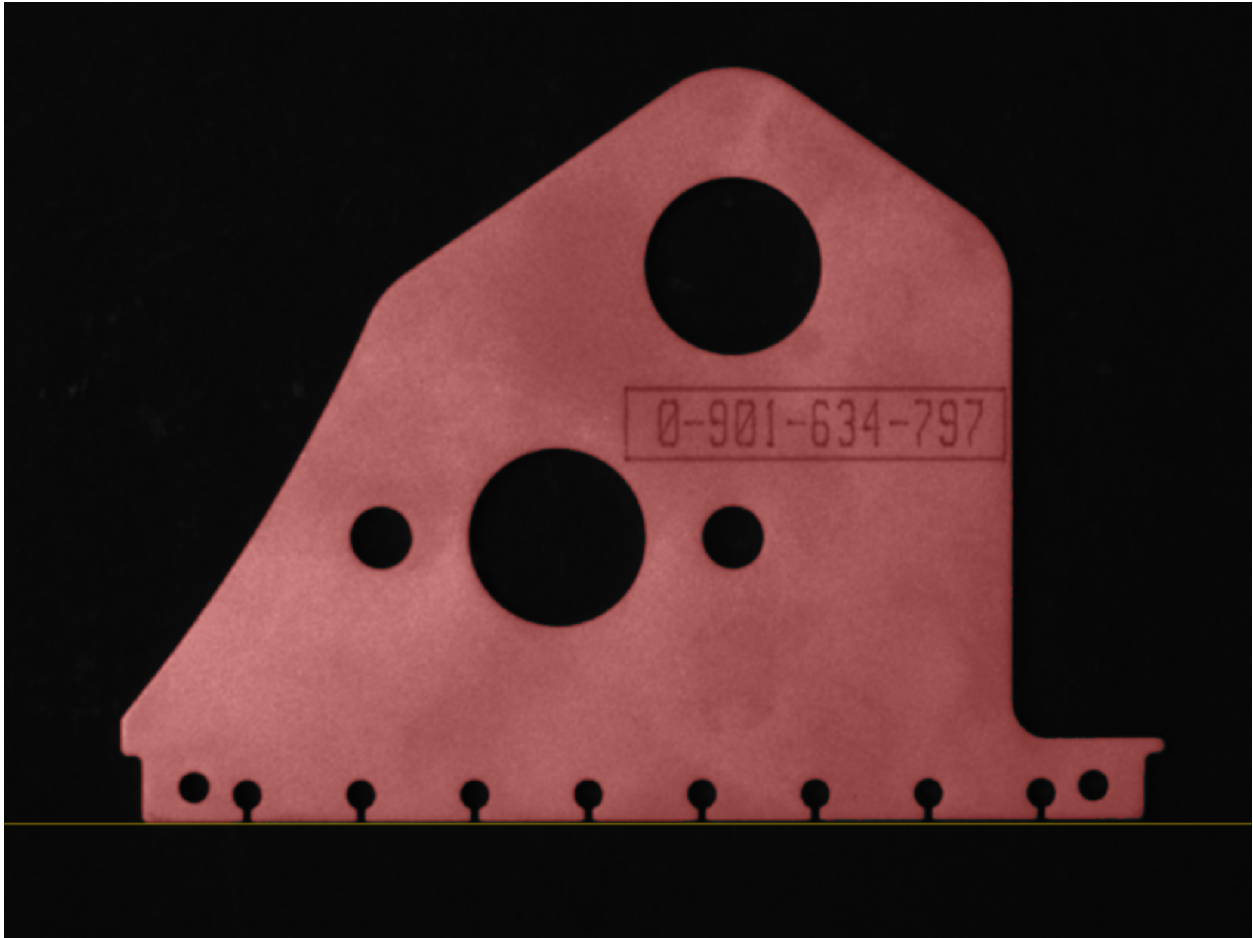


Fig. 10.13: Visualizing the bottom position.



Fig. 10.14: Calculating the width of an object.

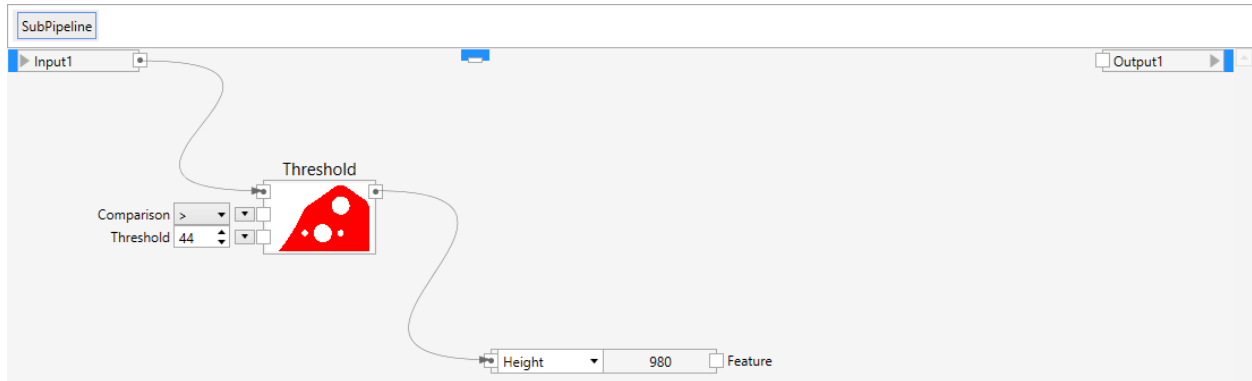


Fig. 10.15: Calculating the height of an object.



Fig. 10.16: Calculating the aspect ratio of an object.

10.2.9 Sphericity

This calculates the sphericity of the region. The sphericity is defined as the quotient of the diameter of a circle having the same area as the object and the maximum feret diameter of the object. The sphericity is calculated according to the following formula:

$$s = \frac{2 * \sqrt{\frac{a}{\pi}}}{f}$$

where a is the area and f is the maximum feret diameter.

10.2.10 Circularity

This calculates the circularity of the region. The circularity is defined as the quotient of the perimeter of a circle having the same area as the object and the perimeter of the object. The circularity is calculated according to the following formula:

$$c = \frac{2 * \sqrt{a * \pi}}{p}$$

where a is the area and p is the perimeter.

10.2.11 Roundness

This calculates the roundness of the region. The roundness is defined as the quotient of the area of the object and the area of the minimum bounding circle. The roundness is calculated according to the following formula:

$$r = \frac{a}{c}$$

where a is the area and c is the area of the minimum bounding circle.

10.2.12 Roughness

This calculates the roughness of the region. The roughness is defined as the quotient of the perimeter of the object and the perimeter of the convex hull. The roughness is calculated according to the following formula:

$$r = \frac{p}{c}$$

where p is the perimeter and c is the perimeter of the convex hull.

10.2.13 Bounds

This calculates the bounds of the region.

The bounds are an axis parallel rectangle that encompasses the whole region tightly.

10.2.14 Perimeter

The perimeter of a region specifies the length of the object outline.

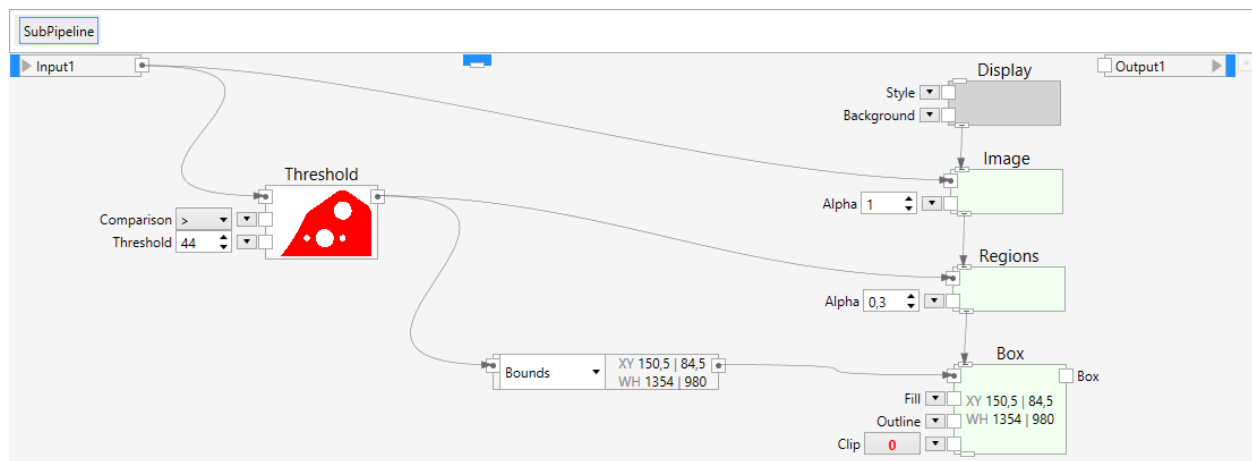


Fig. 10.17: Calculating the bounds of an object.

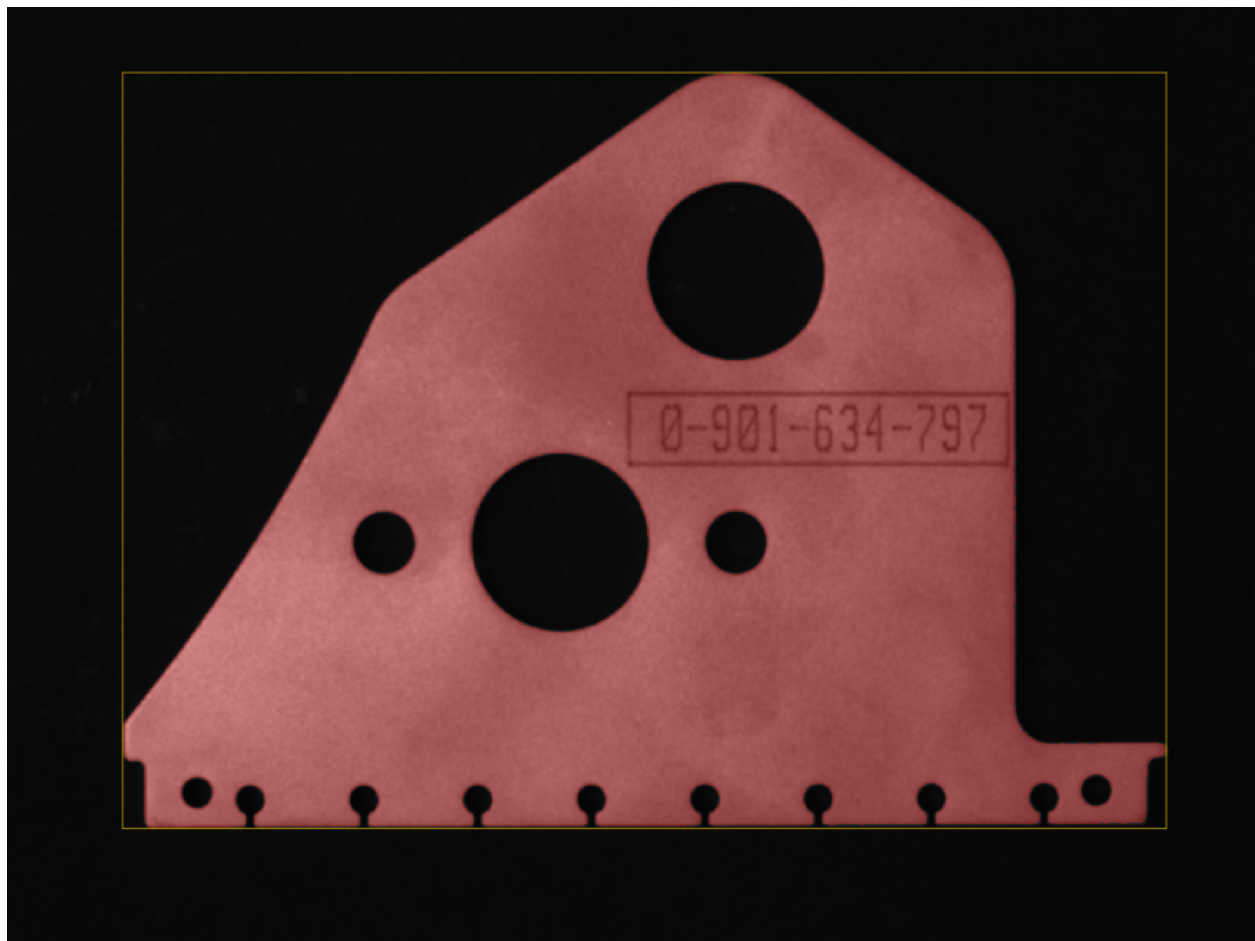


Fig. 10.18: Visualizing the bounds.

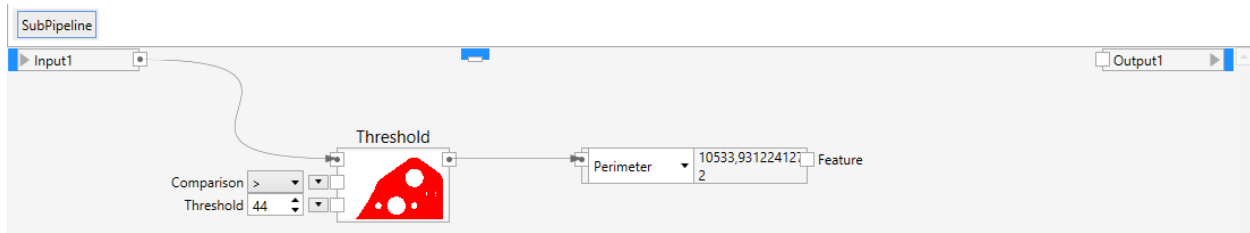


Fig. 10.19: Calculating the perimeter of an object.

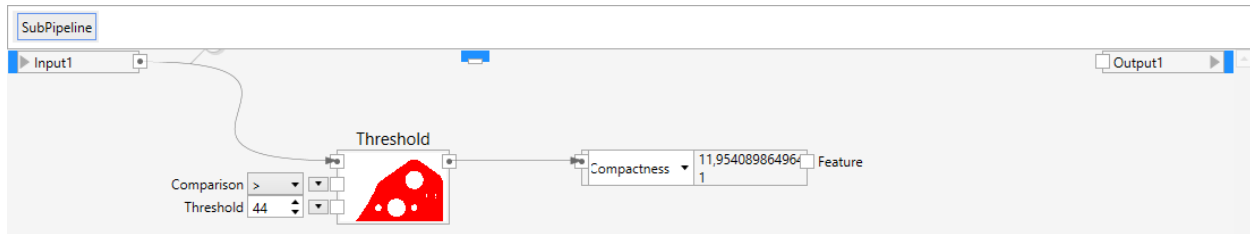


Fig. 10.20: Calculating the compactness of an object.

10.2.15 Compactness

This calculates the compactness of the region.

The compactness c is calculated from the region area a and the region perimeter p according to this formula:

$$c = \frac{p^2}{4\pi a}$$

10.2.16 Convex Hull

This calculates a polyline that constitutes the convex hull of the region.

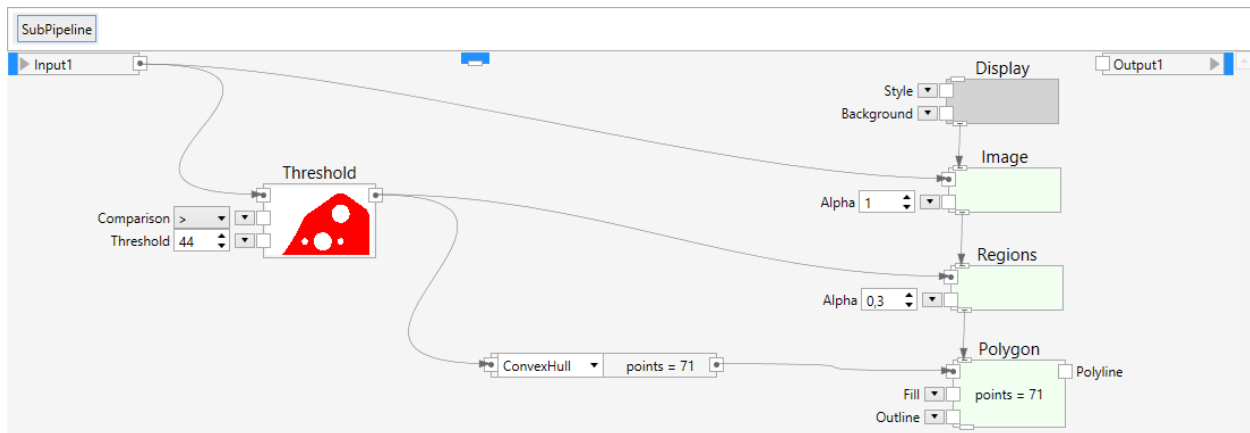


Fig. 10.21: Calculating the convex hull of an object.

The convex hull encompasses the region tightly, like a rubberband.

The convex hull can be used to calculate further features, such as the area of the convex hull, the perimeter of the convex hull, as well as the minimum and maximum width of the convex hull. The minimum and the maximum width of the convex hull are calculated using a rotating caliper algorithm and are often called the Feret diameters.

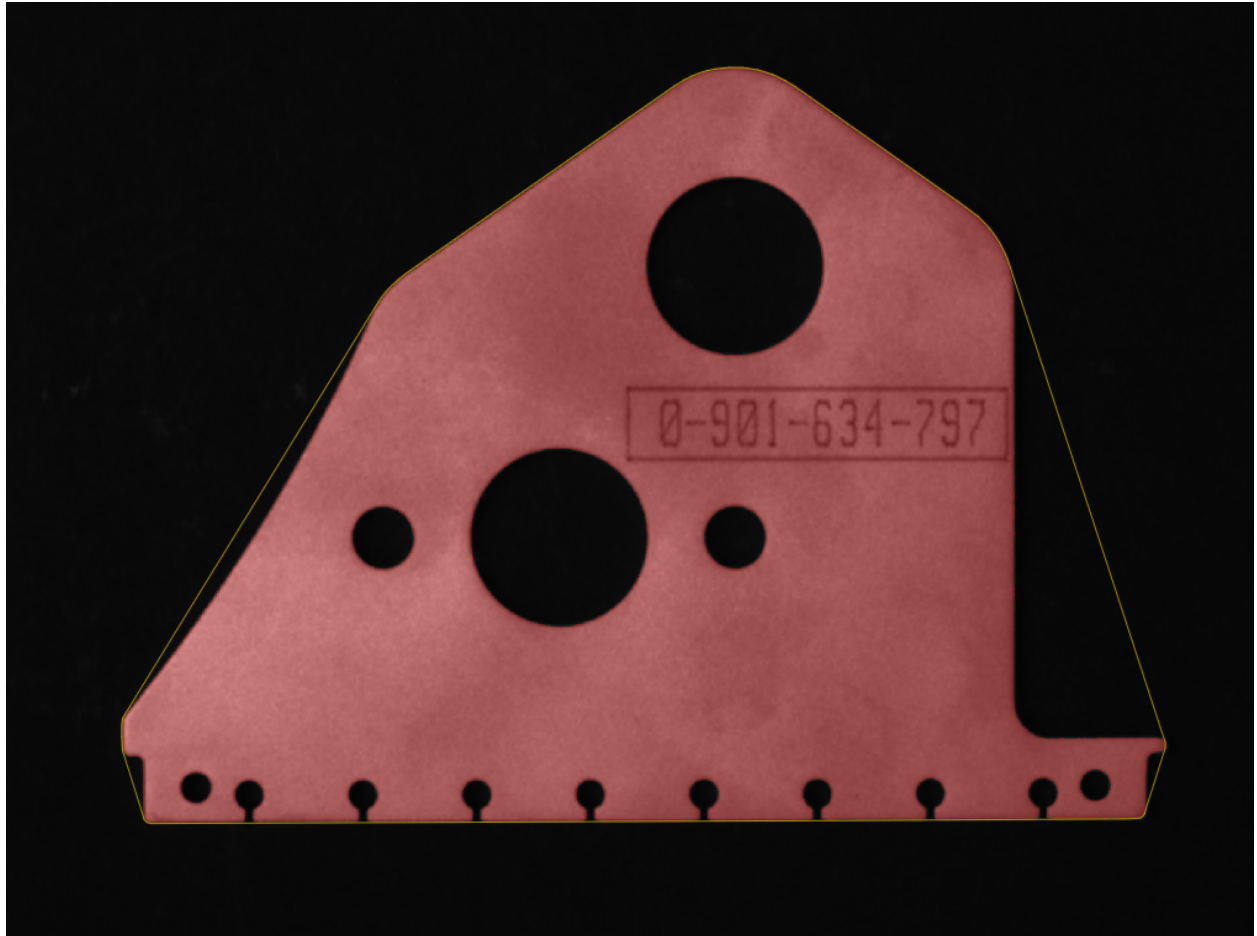


Fig. 10.22: Visualizing the convex hull.

10.2.17 Convex Area

This calculates the area enclosed by the convex hull.

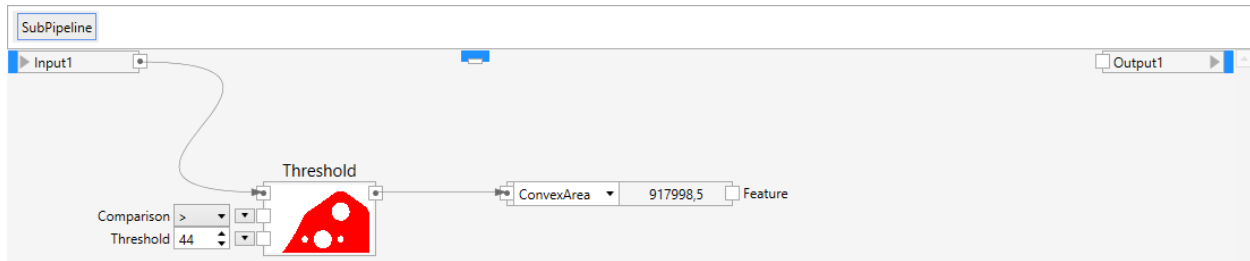


Fig. 10.23: Calculating the convex area of an object.

The convex area is always the same or bigger than the area of the original region. For the example image one can see right away that it is certainly bigger.

10.2.18 Perforation

This calculates the perforation of the region.

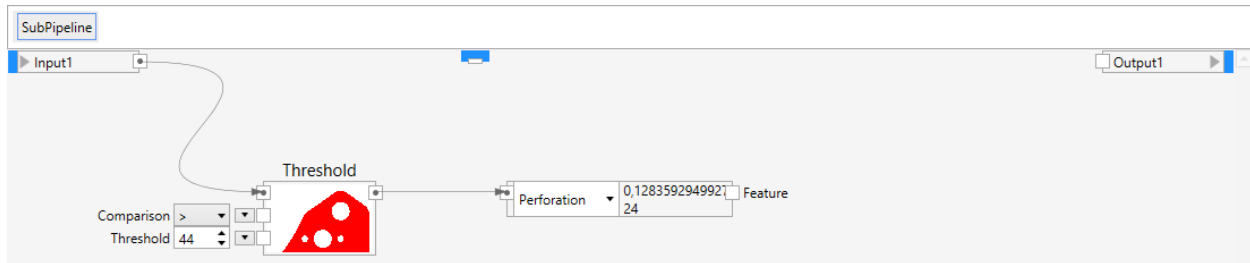


Fig. 10.24: Calculating the perforation of an object.

The perforation p is calculated from the region area a and the area of the holes a_h according to this formula:

$$p = \frac{a_h}{a}$$

The perforation is zero for a region that has no holes, and it increases the more holes the region has. The perforation is scale invariant.

10.2.19 Convexity

This calculates the convexity of the region.

The convexity c is calculated from the region area a and the convex area a_c according to this formula:

$$c = \frac{a}{a_c}$$

The convexity is 1 for a region that already is convex. It is smaller than 1 for non-convex regions. The convexity is scale invariant.

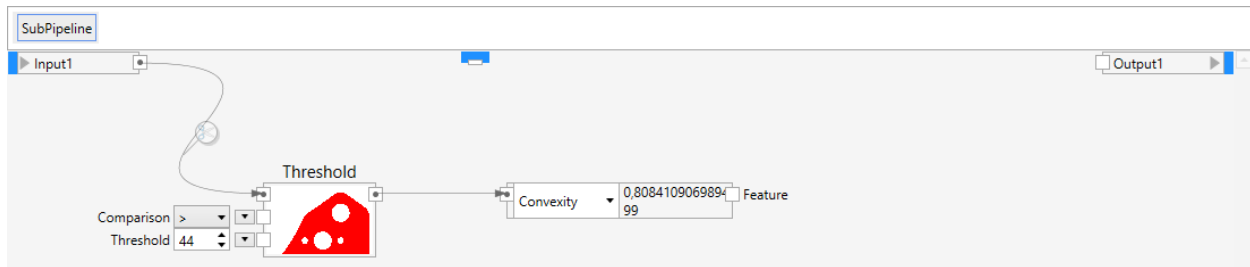


Fig. 10.25: Calculating the convexity of an object.

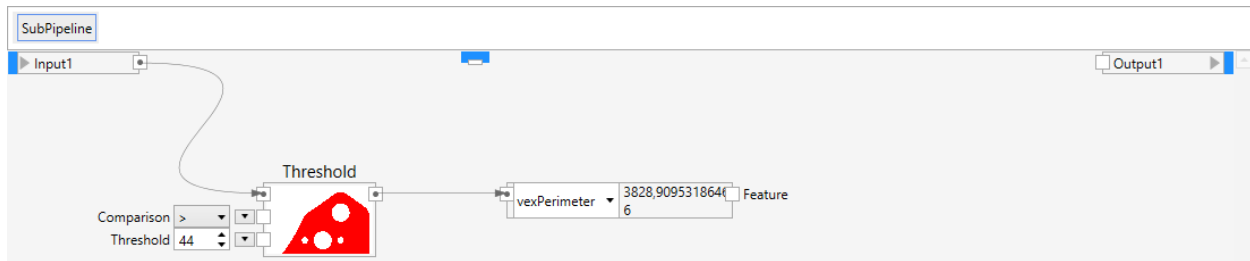


Fig. 10.26: Calculating the convex perimeter of an object.

10.2.20 Convex Perimeter

This calculates the perimeter of the convex hull polygon.

10.2.21 Minimum Feret Diameter

This calculates the minimum feret diameter of the convex hull polygon. The minimum feret diameter is calculated using a rotating calipers algorithm.

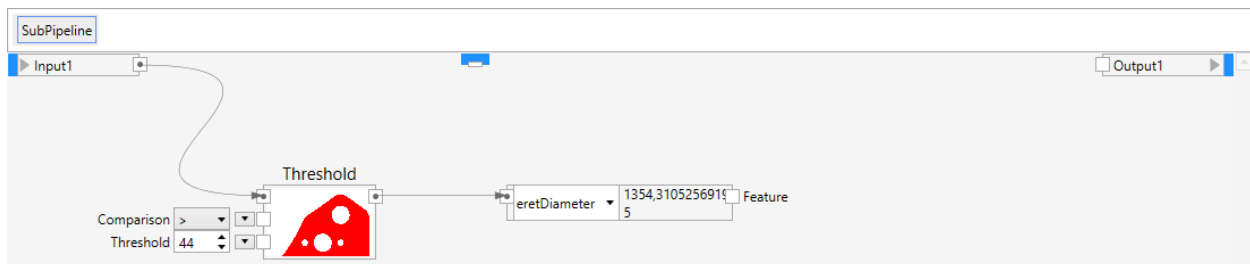
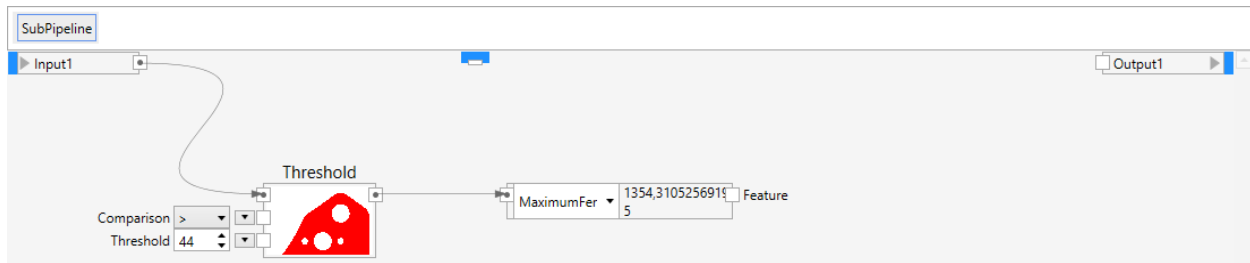


Fig. 10.27: Calculating the minimum feret diameter of an object.

10.2.22 Maximum Feret Diameter

This calculates the maximum feret diameter of the convex hull polygon. The maximum feret diameter is calculated using a rotating calipers algorithm.



10.2.23 Minimum Bounding Circle

This calculates the smallest bounding circle of the region.

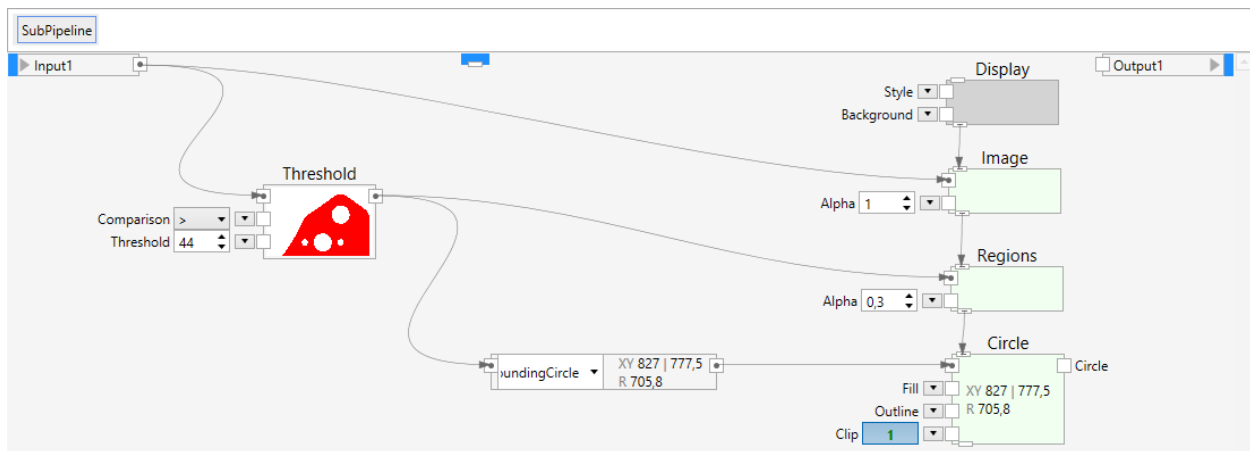


Fig. 10.28: Calculating the minimum bounding circle of an object.

10.2.24 Inner Contour Perimeter

This calculates the object perimeter using the inner contour. The inner contour is a counter-clockwise chain code of object boundary pixels. The inner contour exclusively lies on object pixels and is an 8-connected contour.

The inner contour is 8-connected. The contour length sums the contour segments for each of the chain code directions. Horizontal and vertical directions count as 1, diagonal directions count as $\sqrt{2}$.

10.2.25 Outer Contour Perimeter

This calculates the object perimeter using the outer contour. The outer contour is a clockwise chain code of background pixels just outside the object. The outer contour exclusively lies on background pixels and is a 4-connected contour.

The outer contour is 4-connected. The contour length sums the contour segments for each of the chain code directions. Horizontal and vertical directions count as 1, diagonal directions are not used.

10.2.26 Region-based Moments

This calculates the region-based raw moments. There is a specific chapter about moments that you can consult, if you want to learn more about moments and how they are implemented. Here, in the context of blob analysis, they are documented only briefly.

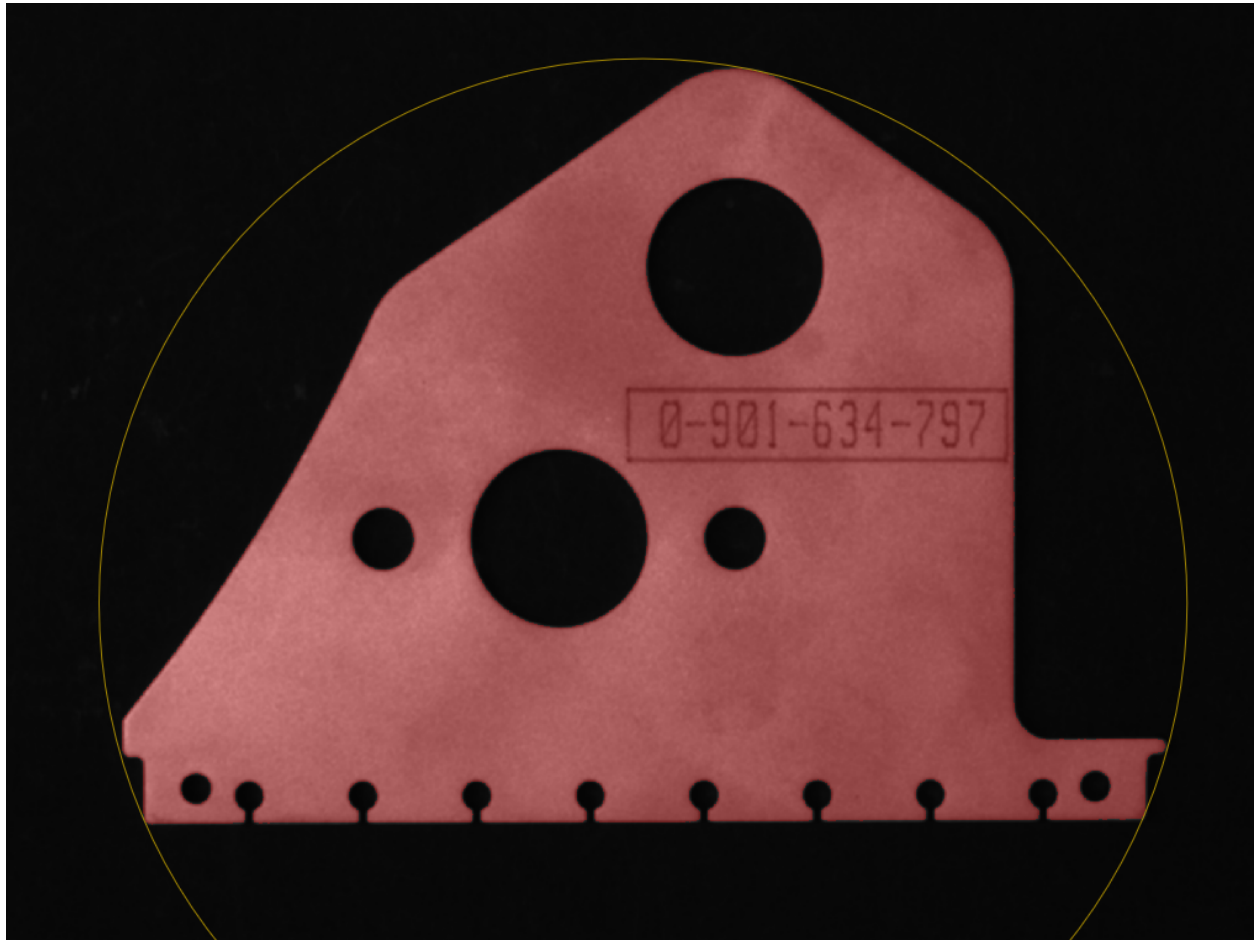


Fig. 10.29: Visualizing the minimum bounding circle of an object.

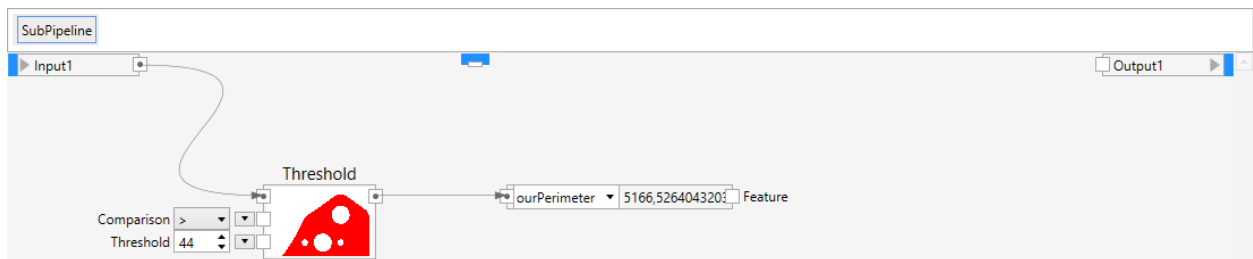


Fig. 10.30: Calculating the inner contour perimeter of an object.

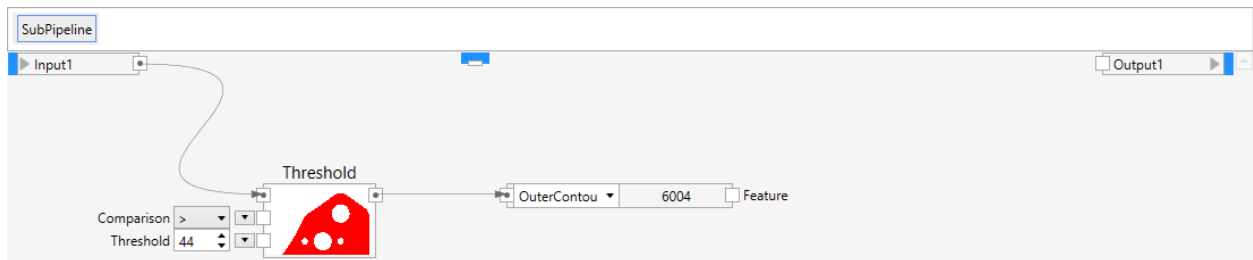


Fig. 10.31: Calculating the outer contour perimeter of an object.

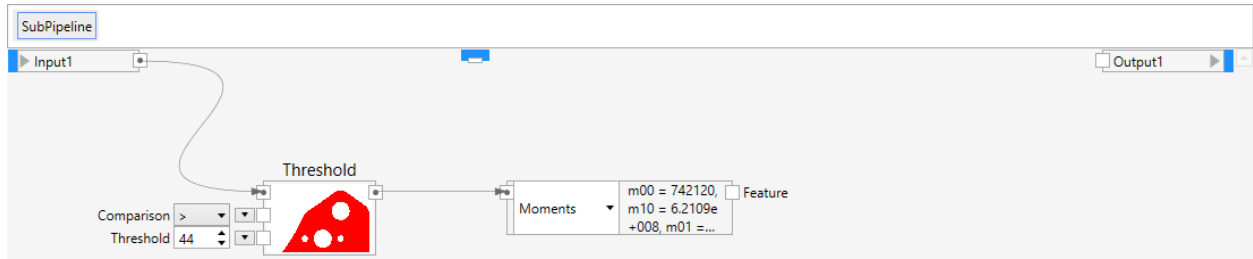


Fig. 10.32: Calculating the region based moments of an object.

The region-based raw moments are calculated up to the third order, according to the following formula:

$$M_{pq} = \sum_{xy} x^p y^q$$

The moments class has the properties m00, m10, m01, m20, m11, m02, m30, m21, m12 and m03 to read the respective moment.

10.2.27 Region-based Normalized Moments

This calculates the region-based normalized moments.

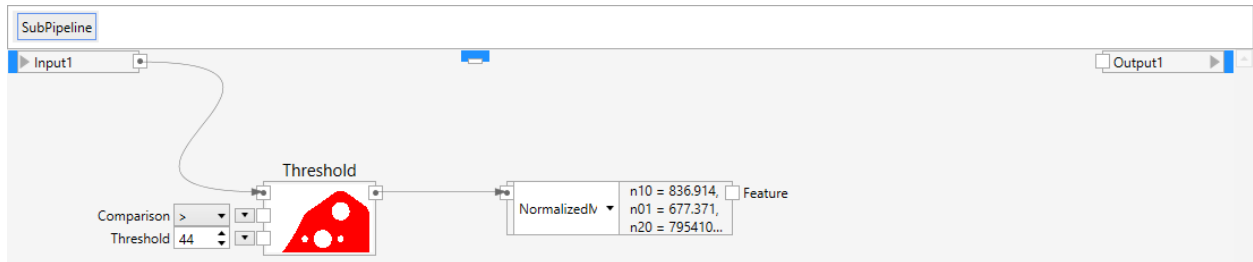


Fig. 10.33: Calculating the region based normalized moments of an object.

They are calculated up to the third order, according to the following formula:

$$N_{pq} = \frac{M_{pq}}{M_{00}}$$

Normalized moments are invariant with respect to scaling, but are not invariant with respect to translation and rotation. You can use the properties n10, n01, n20, n11, n02, n30, n21, n12 and n03 to read the respective normalized moments.

10.2.28 Region-based Centroid

This calculates the region-based object centroid. The centroid is a property of the normalized moments.

The centroid is the center of gravity and calculated according to the following formula:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{N_{10}}{N_{00}} \\ \frac{N_{01}}{N_{00}} \end{pmatrix}$$

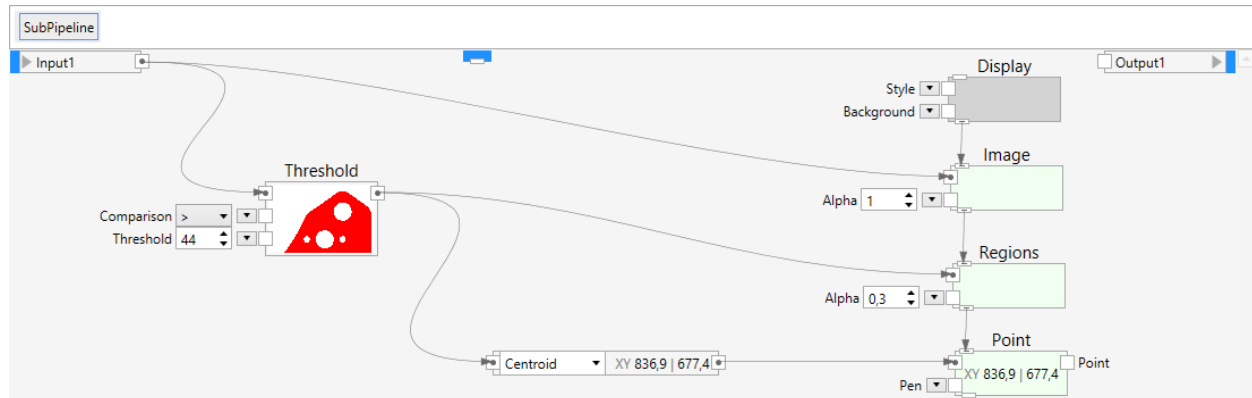
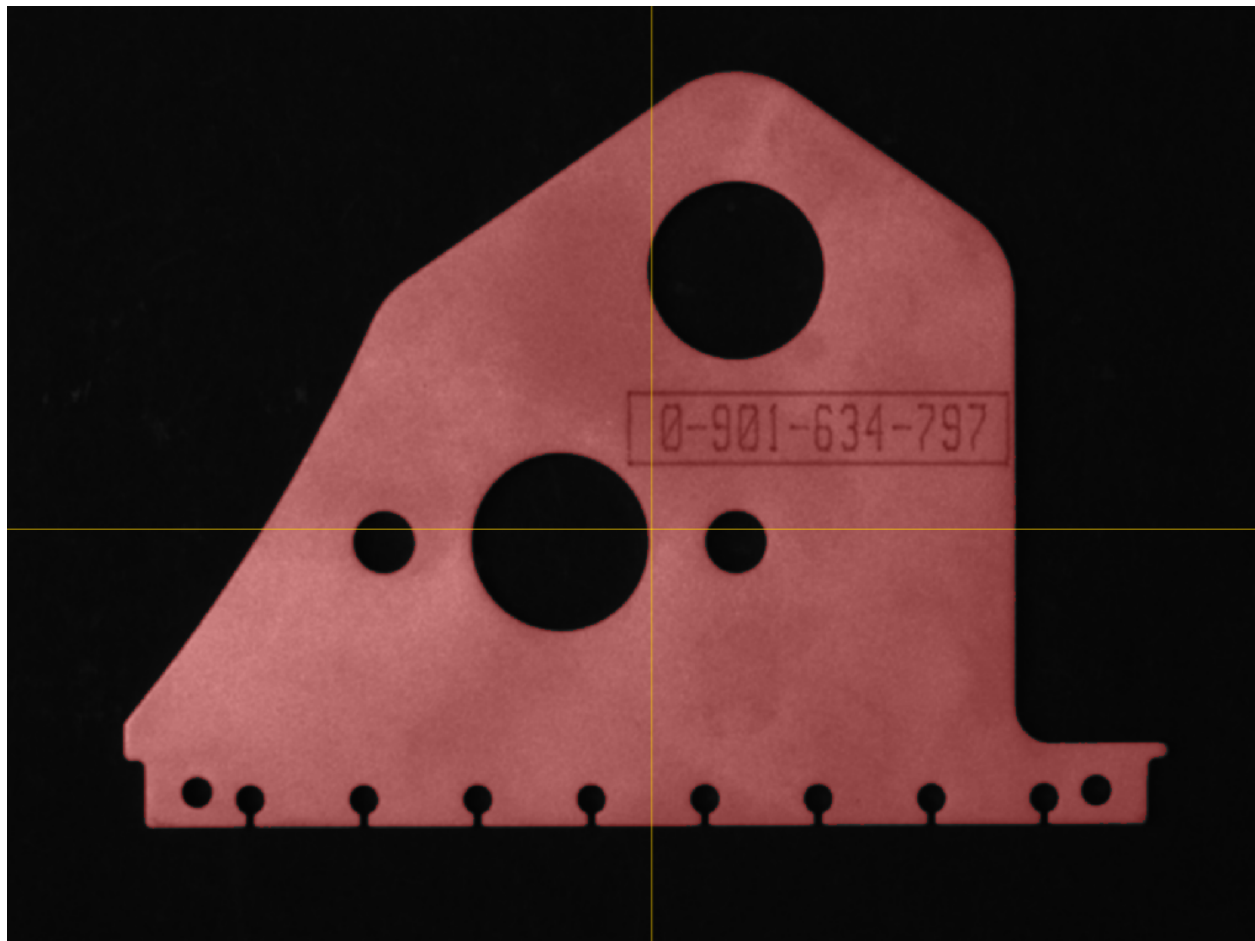


Fig. 10.34: Calculating the region based centroid of an object.



10.2.29 Region-based Central Moments

This calculates the region-based central moments.

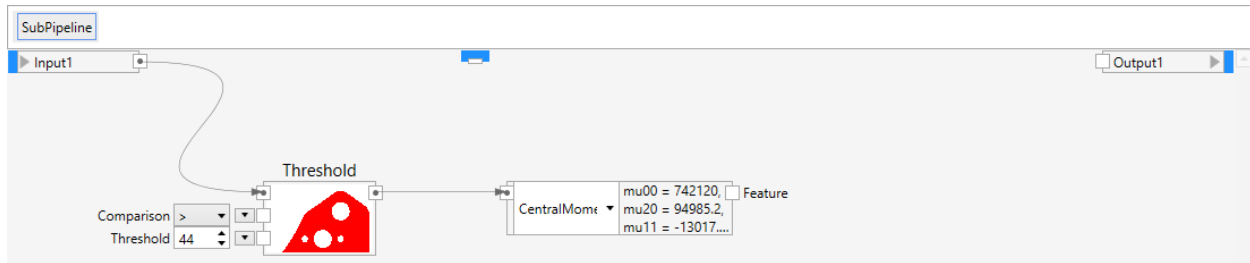


Fig. 10.35: Calculating the region based central moments of an object.

They are calculated up to the third order according to the following formula:

$$\mu_{pq} = \frac{1}{M_{00}} \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q$$

Central moments are invariant with respect to translation, but are not invariant with respect to rotation. You can use the properties mu20, mu11, mu02, mu30, mu21, mu12 and mu03 to read the respective central moments.

10.2.30 Region-based Equivalent Ellipse

This calculates the region_based equivalent ellipse of the object. The equivalent ellipse is built by combining properties from the normalized and the central moments.

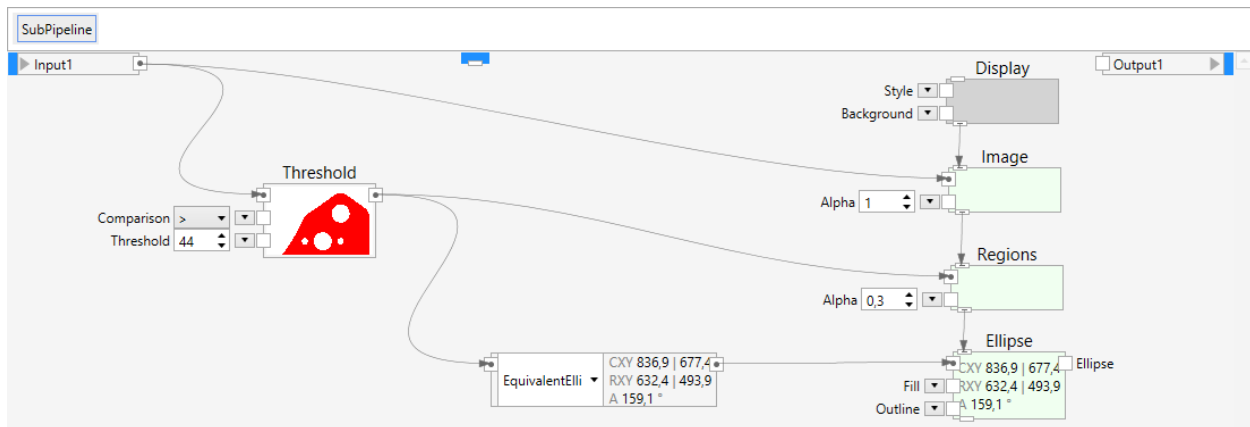


Fig. 10.36: Calculating the region based equivalent ellipse of an object.

10.2.31 Region-based Scale Invariant Moments

This calculates the region-based scale-invariant moments.

They are calculated up to the third order according to the following formula:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q}{2}+1}}$$

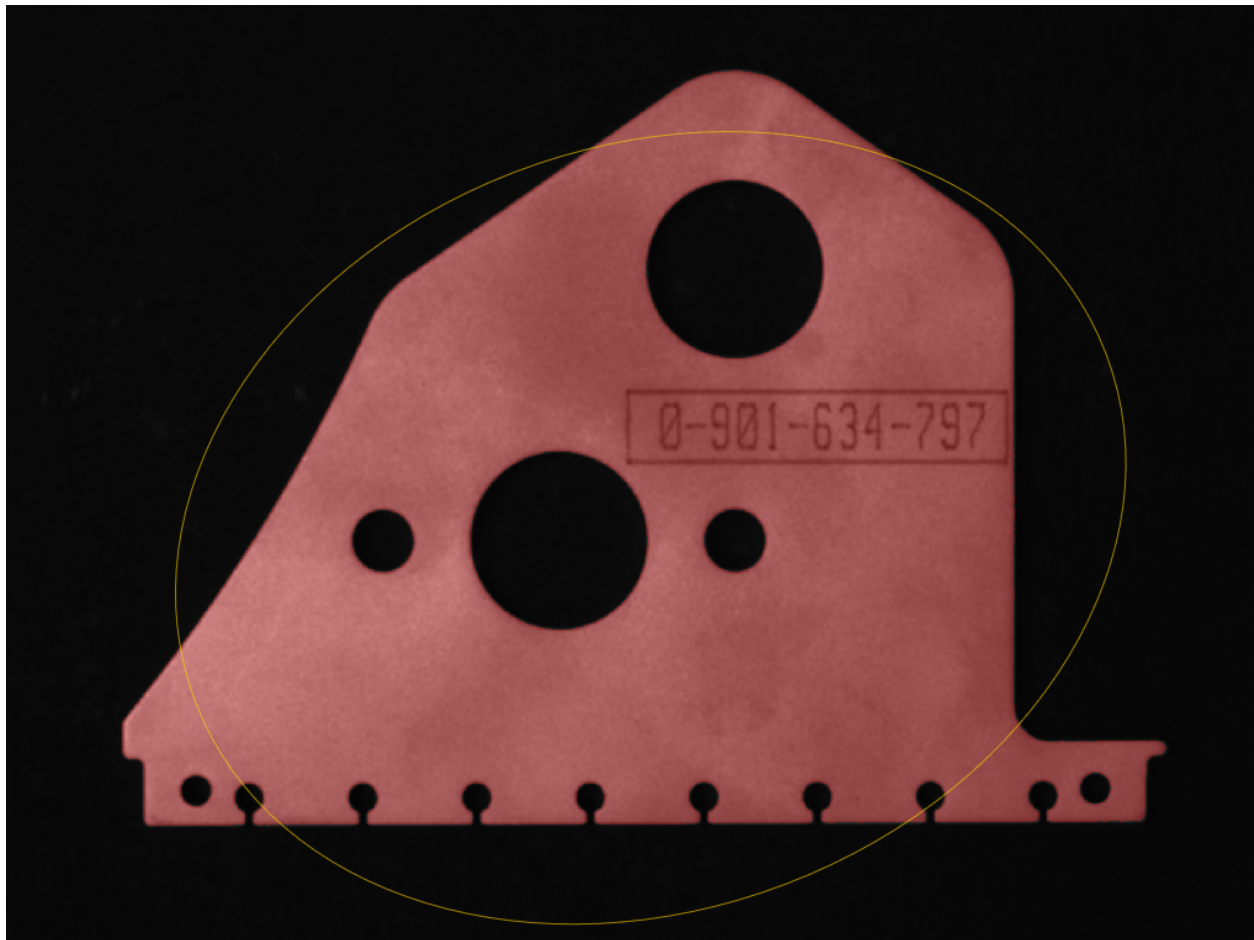


Fig. 10.37: Visualizing the region based equivalent ellipse of an object.



Fig. 10.38: Calculating the region based scale invariant moments of an object.

Scale-invariant moments are invariant with respect to translation and scaling, but are not invariant with respect to rotation.

You can use the properties eta20, eta11, eta02, eta30, eta21, eta12 and eta03 to read the respective scale invariant moments.

10.2.32 Region-based Hu Moments

This calculates the region-based Hu moments. The formulas are described in the chapter about moments.

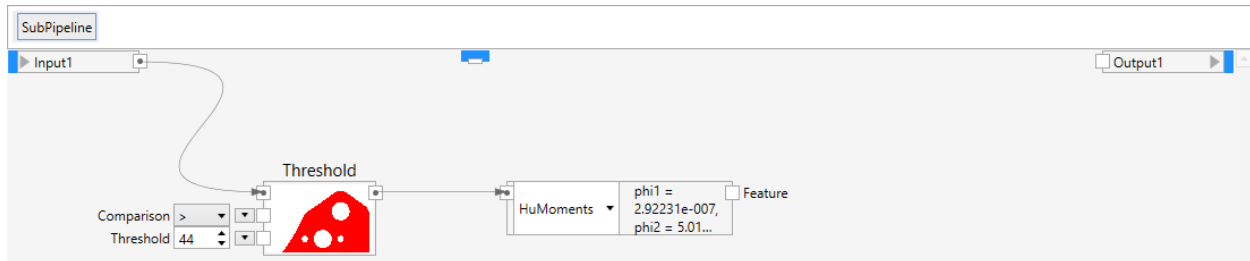


Fig. 10.39: Calculating the region based Hu moments of an object.

Hu's moments are invariant with respect to translation, scaling and rotation.

The formulas for Hu's moments are:

$$\begin{aligned}
 \phi_1 &= \eta_{20} + \eta_{02} \\
 \phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
 \phi_3 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
 \phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
 \phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
 &\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
 \phi_6 &= (\eta_{20} - \eta_{02}) [(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
 &\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
 \phi_7 &= (3\eta_{21} - 3\eta_{03})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
 &\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
 \end{aligned}$$

You can use the properties phi1, phi2, phi3, phi4, phi5, phi6 and phi7 to read the respective Hu moment.

10.2.33 Region-based Flusser Moments

This calculates the region-based Flusser moments. The formulas are described in the chapter about moments.

Flusser's moments are invariant with respect to affine transformations.

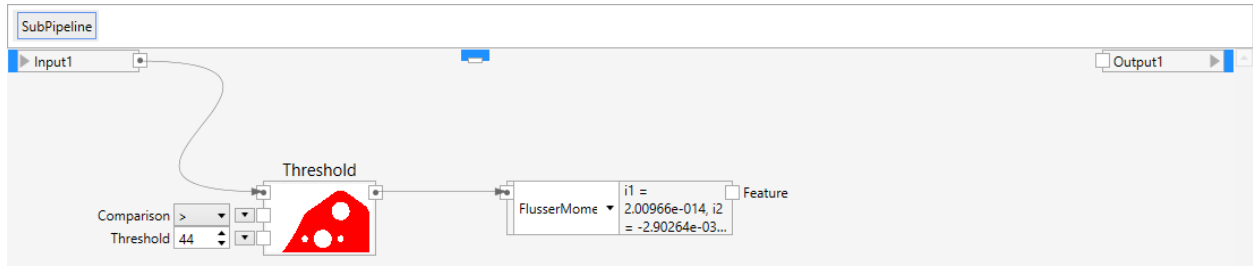


Fig. 10.40: Calculating the region based Flusser moments of an object.

The formulas for Flusser's moments are:

$$\begin{aligned}
 I_1 &= \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4} \\
 I_2 &= \frac{\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} + 4\mu_{30}\mu_{12}^3 + 4\mu_{21}^3\mu_{03} - 3\mu_{21}^2\mu_{12}^2}{\mu_{00}^7} \\
 I_3 &= \frac{\mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12})}{\mu_{00}^7} + \frac{\mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)}{\mu_{00}^7} \\
 I_4 &= \frac{\mu_{20}^3\mu_{03}^2 - 6\mu_{20}^2\mu_{11}\mu_{12}(\mu_{03} - 6\mu_{20}^2\mu_{02}\mu_{21})\mu_{03}}{\mu_{00}^{11}} \\
 &\quad + \frac{9\mu_{20}^2\mu_{02}\mu_{12}^2 + 12\mu_{20}\mu_{11}^2\mu_{21}\mu_{03} + 6\mu_{20}\mu_{11}\mu_{02}\mu_{30}\mu_{03}}{\mu_{00}^{11}} \\
 &\quad - \frac{18\mu_{20}\mu_{11}\mu_{02}\mu_{21}\mu_{12} + 8\mu_{11}^3\mu_{30}\mu_{03} + 6\mu_{20}\mu_{02}^2\mu_{30}\mu_{12}}{\mu_{00}^{11}} \\
 &\quad + \frac{9\mu_{20}\mu_{02}^2\mu_{21}^2 + 12\mu_{11}^2\mu_{02}\mu_{30}\mu_{12} - 6\mu_{11}\mu_{02}^2\mu_{30}\mu_{21}}{\mu_{00}^{11}} \\
 &\quad + \frac{\mu_{02}^3\mu_{30}^2}{\mu_{00}^{11}}
 \end{aligned}$$

You can use the properties i1, i2, i3, and i4 to read the respective flusser moments.

10.2.34 Number of Holes

This calculates the number of holes of the region.

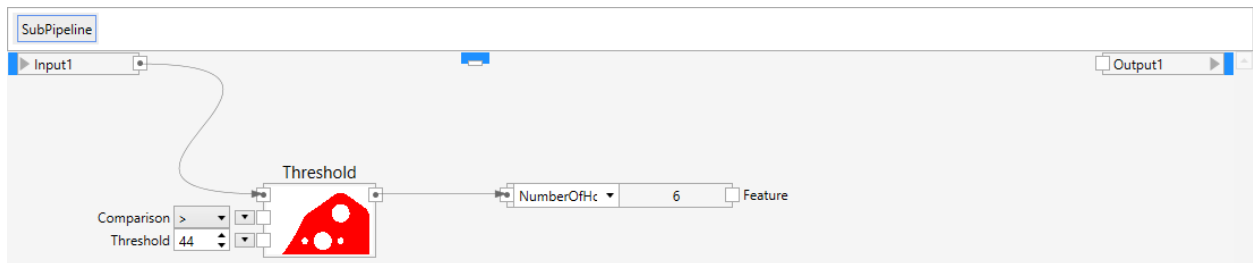


Fig. 10.41: Calculating the number of holes of an object.

10.2.35 Area of Holes

This calculates the total area of the holes of the region.



Fig. 10.42: Calculating the area of all holes of an object.

10.2.36 Fiber Length

This calculates an approximation of a fiber length. This measurement is only valid for elongated objects. The fiber length is approximated as half the perimeter.

10.2.37 Fiber Width

This calculates an approximation of a fiber width. The fiber width is approximated as twice the maximum value of the distance transform.

10.2.38 Bounding Rectangle

This calculates a bounding rectangle of the region.

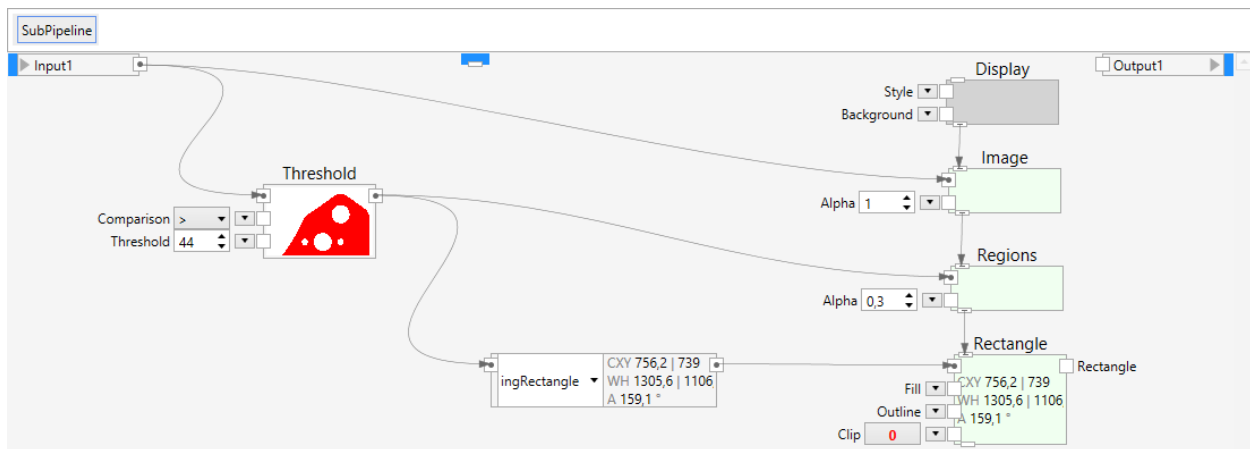
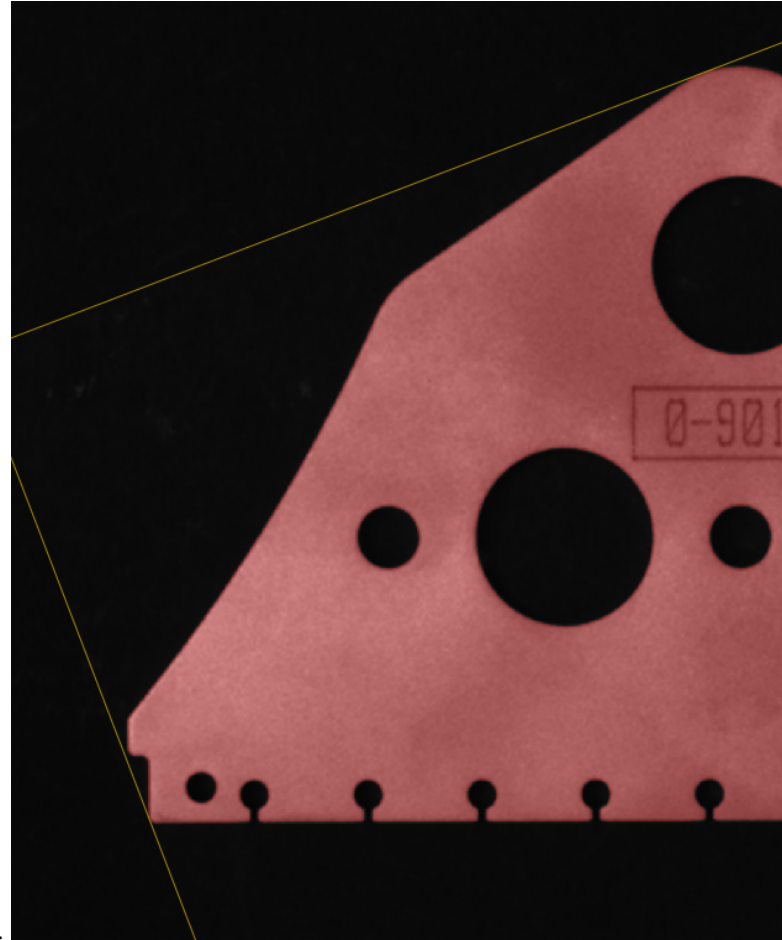


Fig. 10.43: Calculating the bounding rectangle of an object.



The rectangle has the same orientation as the equivalent ellipse.

10.2.39 Minimum Area Bounding Rectangle

This calculates a bounding rectangle of the region with a minimal area.

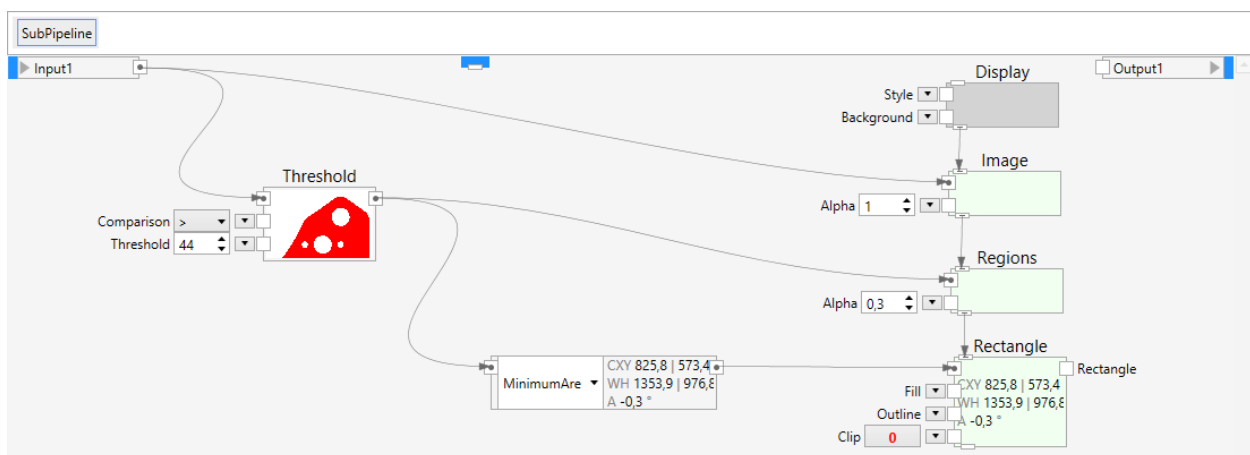


Fig. 10.44: Calculating the bounding rectangle of an object.

The result is the rectangle with the smallest area bounding the region.

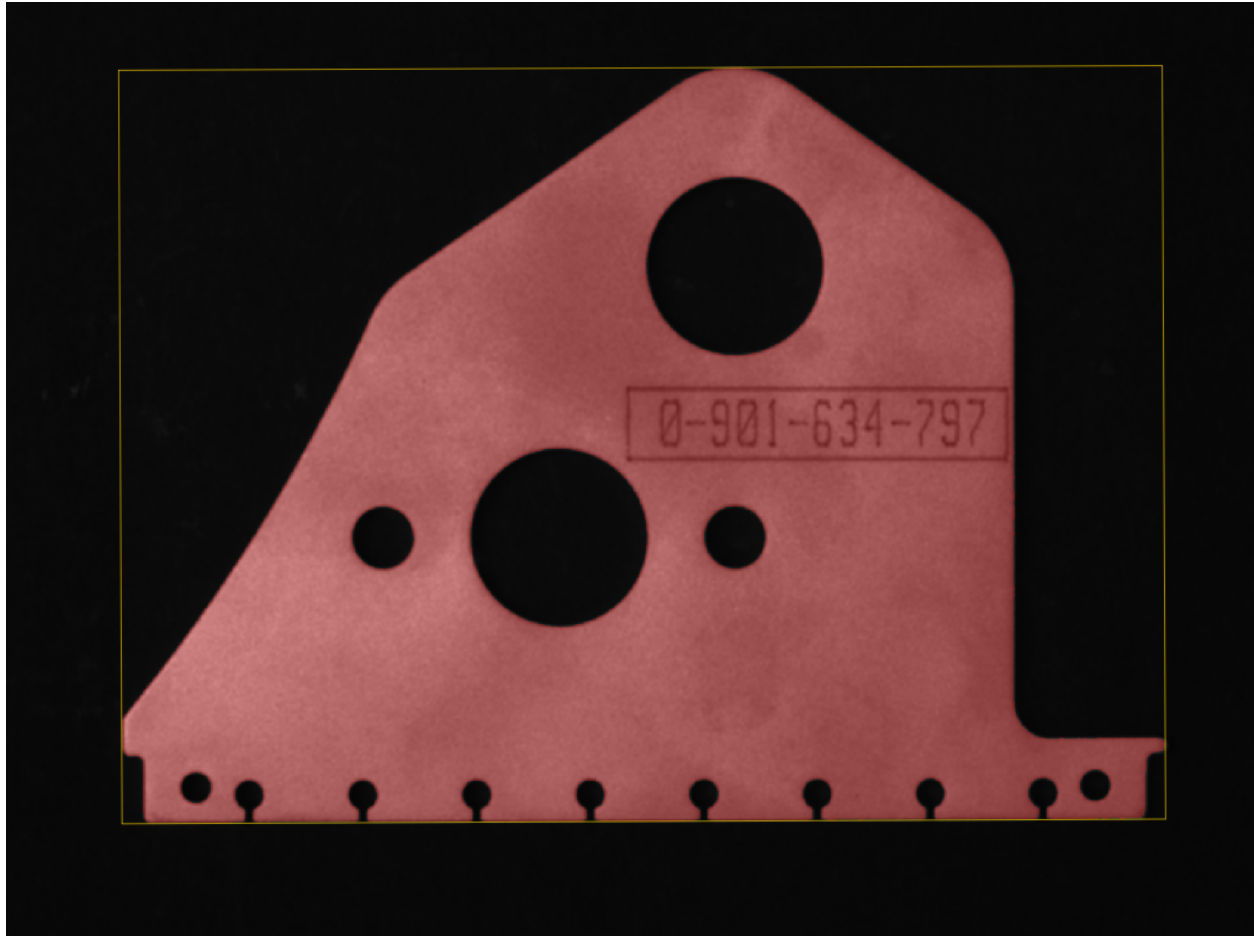


Fig. 10.45: Visualizing the bounding rectangle of an object.

10.2.40 Minimum Perimeter Bounding Rectangle

This calculates a bounding rectangle of the region with a minimal perimeter.

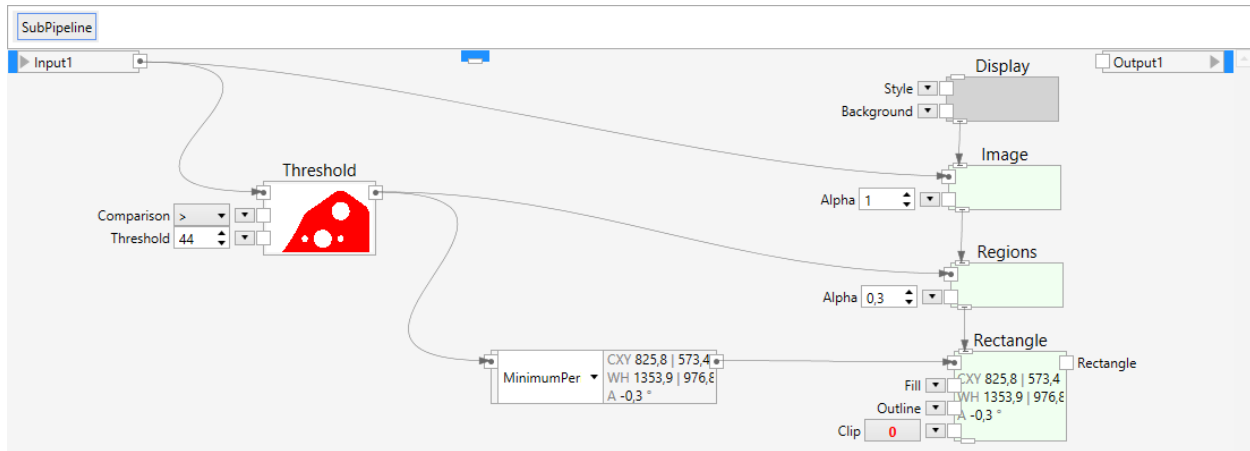


Fig. 10.46: Calculating the bounding rectangle of an object.

The result is the rectangle with the smallest perimeter bounding the region.

10.3 Pixel Based Features

In contrast to region-based features, pixel-based features use the pixel values in addition to the region definition. Pixel-based features are necessary since some features, such as the minimum pixel value, cannot be calculated given the region alone. However, the calculation of pixel-based features is slower than the calculation of region-based features.

10.3.1 Minimum Position

This calculates the minimum position value.

```
blob_features b(gray.get_view(), objects[0]);
point_3d<int> minimum = *b.get_minimum_position();
```

When calculating the minimum position for color images, the color is treated as a vector and the minimum is determined by comparing the Euclidean vector length.

10.3.2 Minimum

This calculates the minimum value.

When calculating the minimum for color images, the color is treated as a vector and the minimum is determined by comparing the Euclidean vector length.

10.3.3 Maximum Position

This calculates the maximum position value.

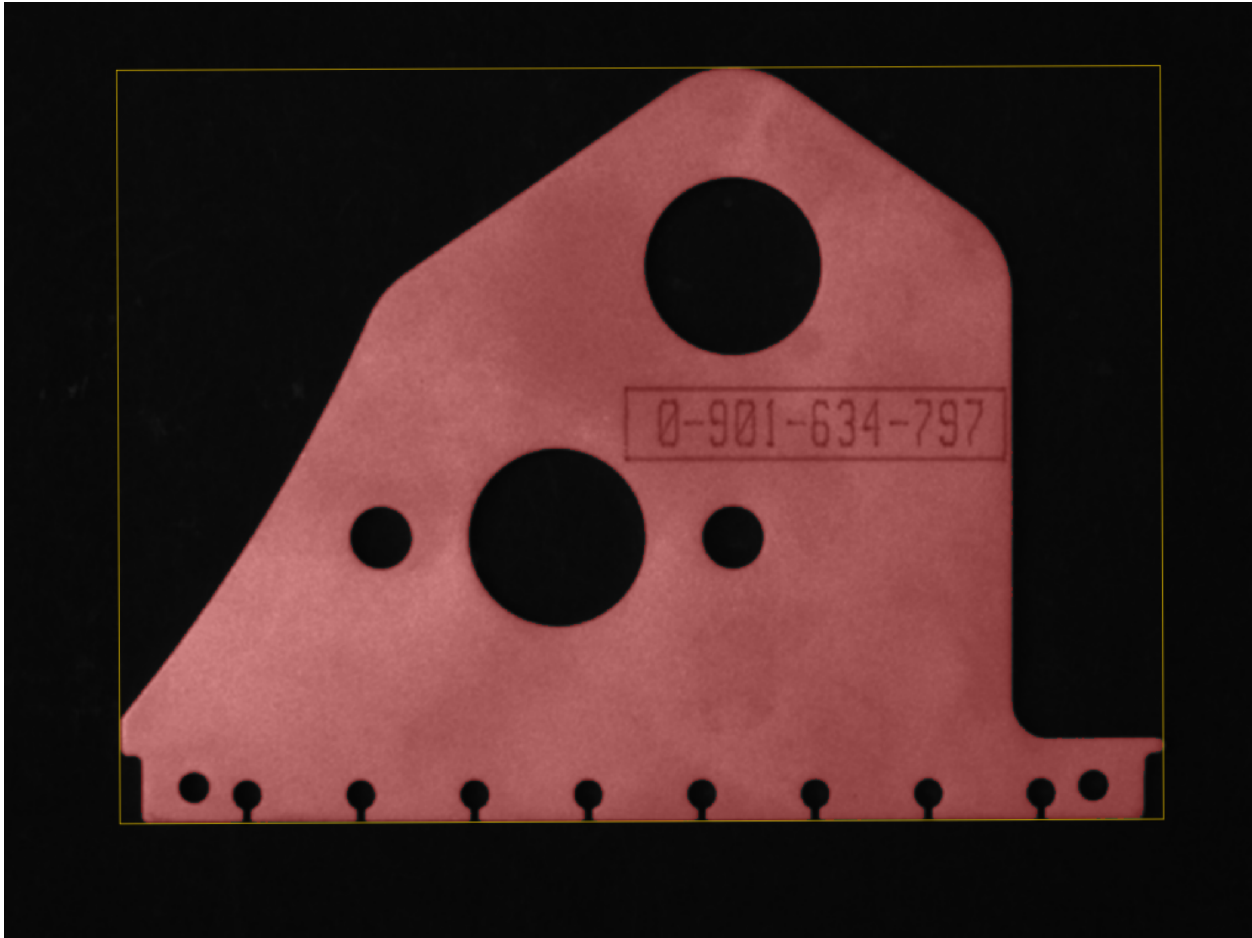


Fig. 10.47: Visualizing the bounding rectangle of an object.

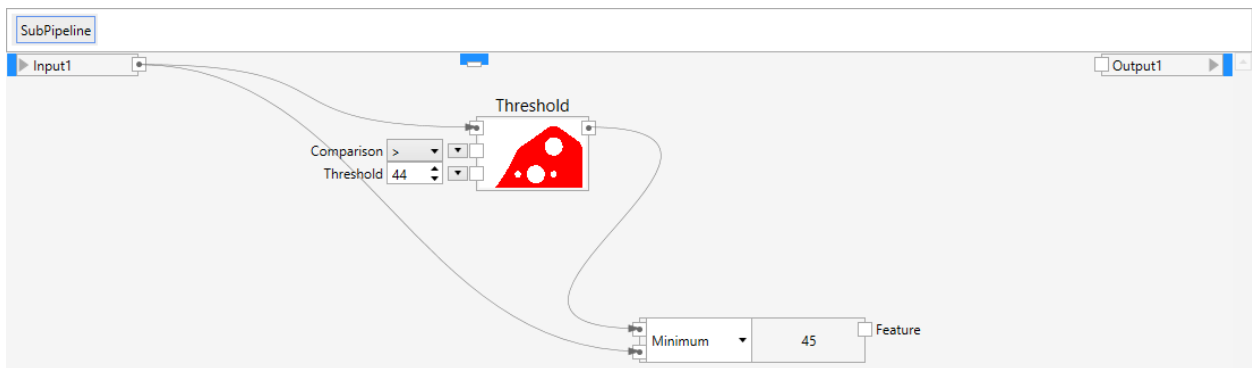


Fig. 10.48: Calculating the minimum value of an object.

```
blob_features b(gray.get_view(), objects[0]);
point_3d<int> maximum = *b.get_maximum_position();
```

When calculating the minimum position for color images, the color is treated as a vector and the minimum is determined by comparing the Euclidean vector length.

10.3.4 Maximum

This calculates the maximum value.

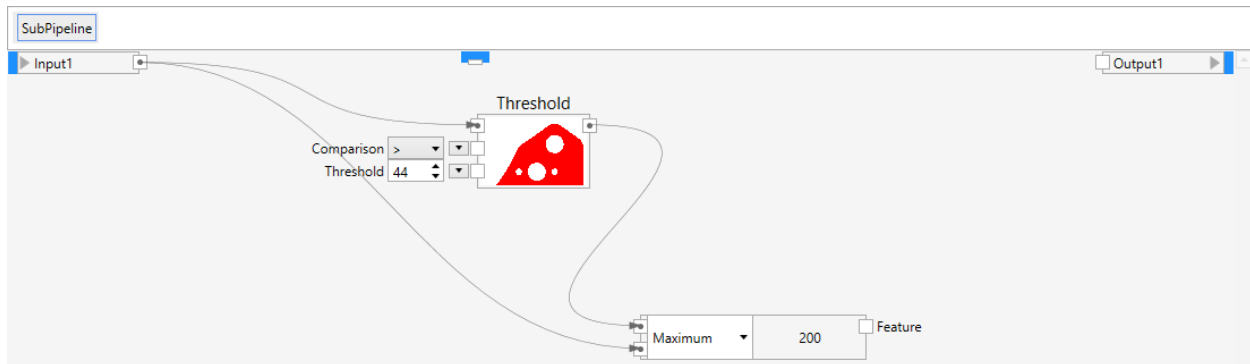


Fig. 10.49: Calculating the maximum value of an object.

When calculating the maximum for color images, the color is treated as a vector and the maximum is determined by comparing the Euclidean vector length.

10.3.5 Total

This calculates the total value.

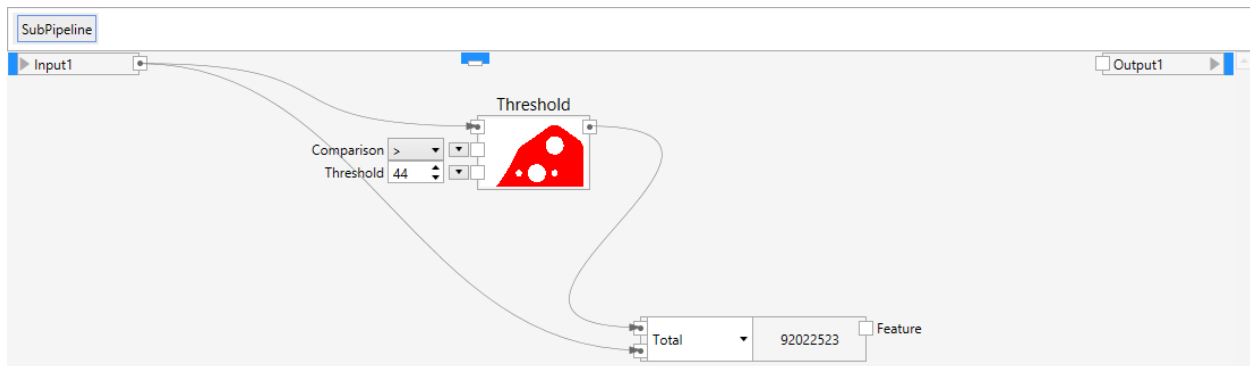


Fig. 10.50: Calculating the total value of an object.

The total is the sum of all pixel values of the object:

$$T = \sum I$$

10.3.6 Mean

This calculates the mean value.

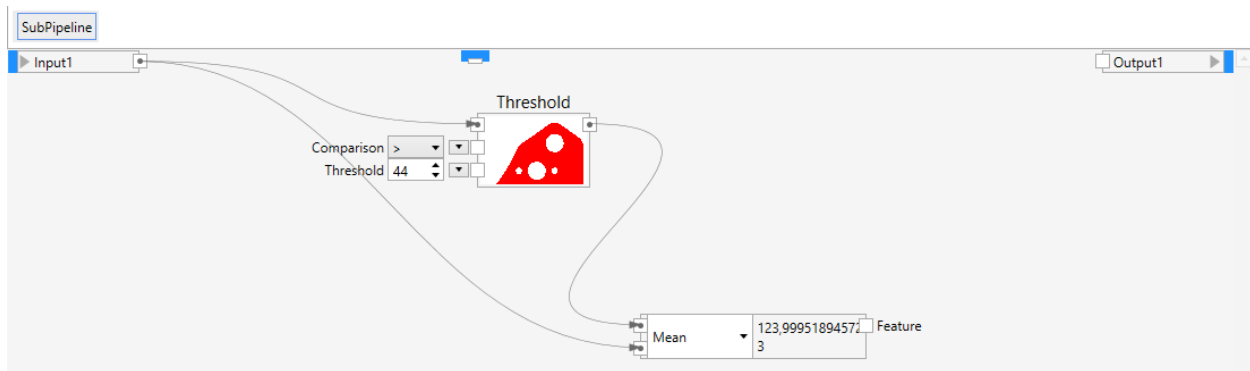


Fig. 10.51: Calculating the mean value of an object.

The mean is calculated according to the following formula:

$$M = \frac{T}{n}$$

where T is the total value and n is the number of pixels (i.e. the area) of the blob.

10.3.7 Standard Deviation

This calculates the standard deviation value. Standard deviation is a measure of how wide the distribution of values is.

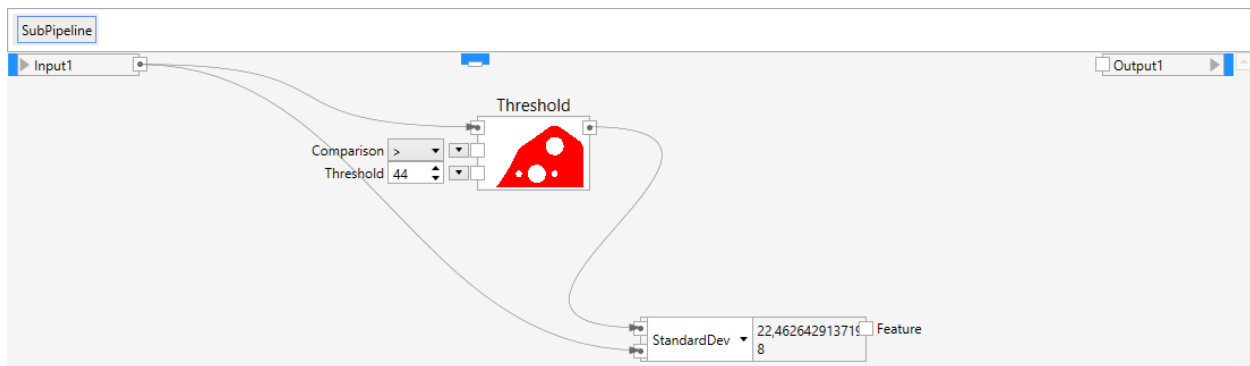


Fig. 10.52: Calculating the standard deviation of an object.

The standard deviation is calculated according to the following formula:

$$\sigma = \sqrt{\frac{1}{n} \sum (I - E)^2}$$

10.3.8 Skewness

This calculates the skewness value. Skewness can be understood as a measure of asymmetry in the distribution, i.e. how much it deviates from a gauss-shaped curve.

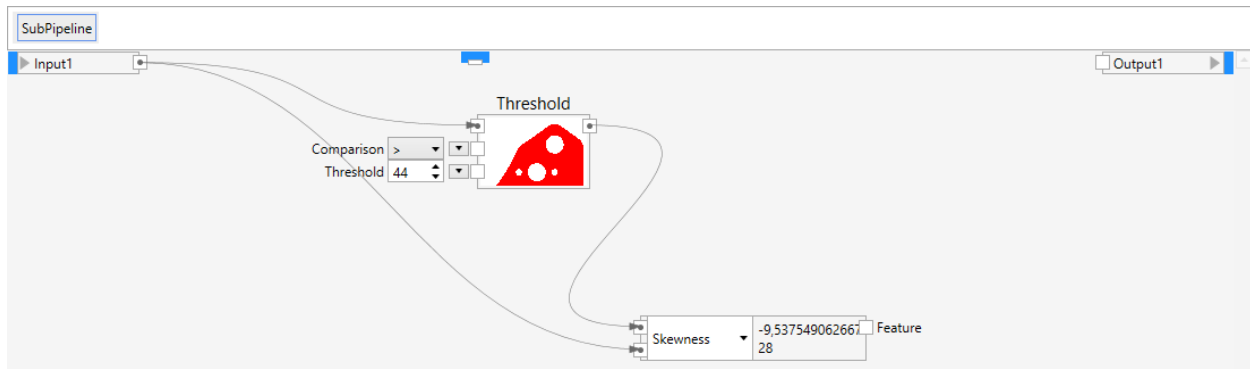


Fig. 10.53: Calculating the skewness of an object.

The skewness is calculated according to the following formula:

$$\sigma = \sqrt[3]{\frac{1}{n} \sum (I - E)^3}$$

10.3.9 Contrast

This calculates the channel-wise contrast.

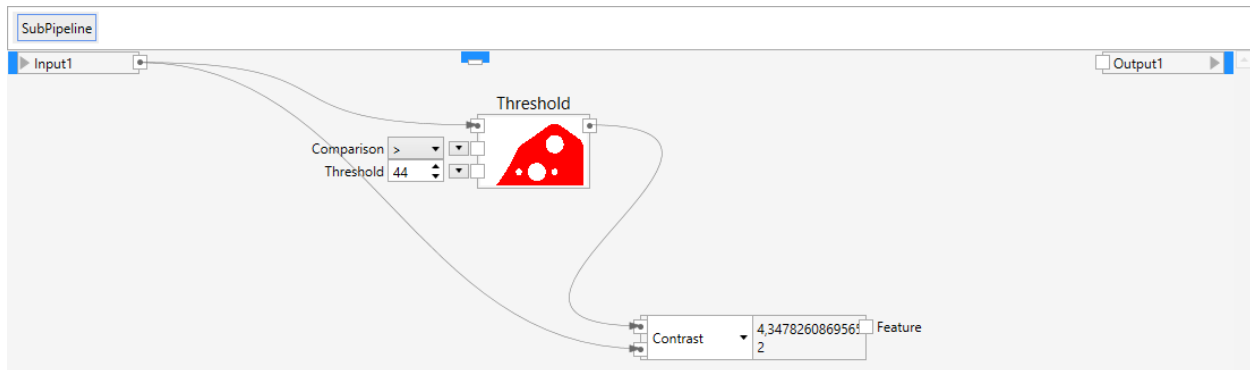


Fig. 10.54: Calculating the contrast of an object.

The contrast is calculated according to the following formula:

$$c = \frac{\max}{\min + 1}$$

10.3.10 Weber Contrast

This calculates the channel-wise contrast according to Weber's formula.

The contrast is calculated according to the following formula:

$$c = \frac{\max - \min}{\min + 1}$$

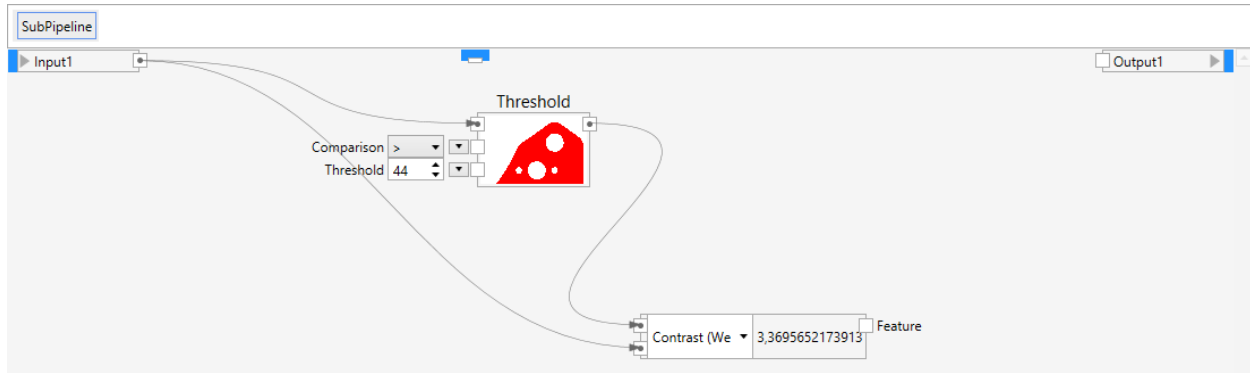


Fig. 10.55: Calculating the contrast of an object.

10.3.11 Michelson Contrast

This calculates the channel-wise contrast according to Michelson's formula.

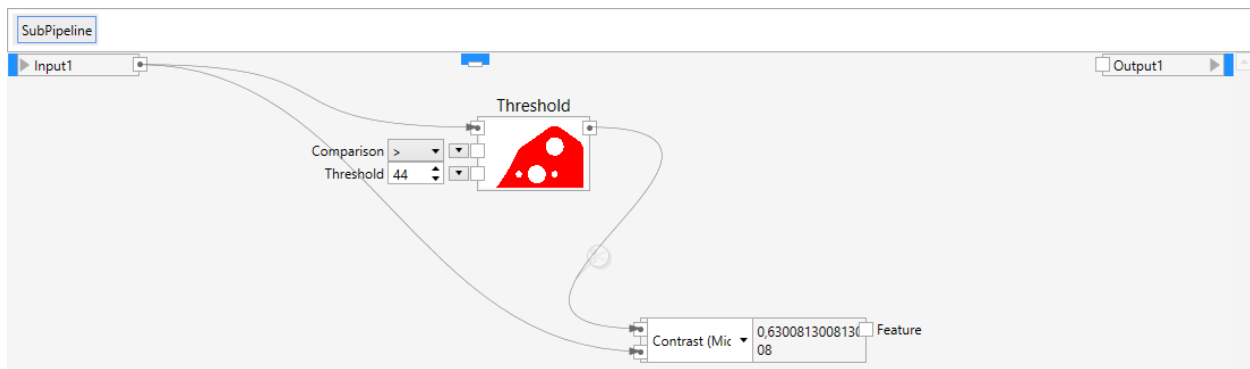


Fig. 10.56: Calculating the contrast of an object.

The contrast is calculated according to the following formula:

$$c = \frac{\max - \min}{\max + \min + 1}$$

10.3.12 Pixel-based Moments

This calculates the pixel-based raw moments. There is a specific chapter about moments that you can consult, if you want to learn more about moments and how they are implemented. Here, in the context of blob analysis, they are documented only briefly.

The pixel-based raw moments are calculated up to the third order, according to the following formula:

$$M_{pq} = \sum_{xy} x^p y^q I_{xy}$$

Pixel based moments are similar to region-based moments, but they make use of pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. The moments class has the properties m00, m10, m01, m20, m11, m02, m30, m21, m12 and m03 to read the respective moment.

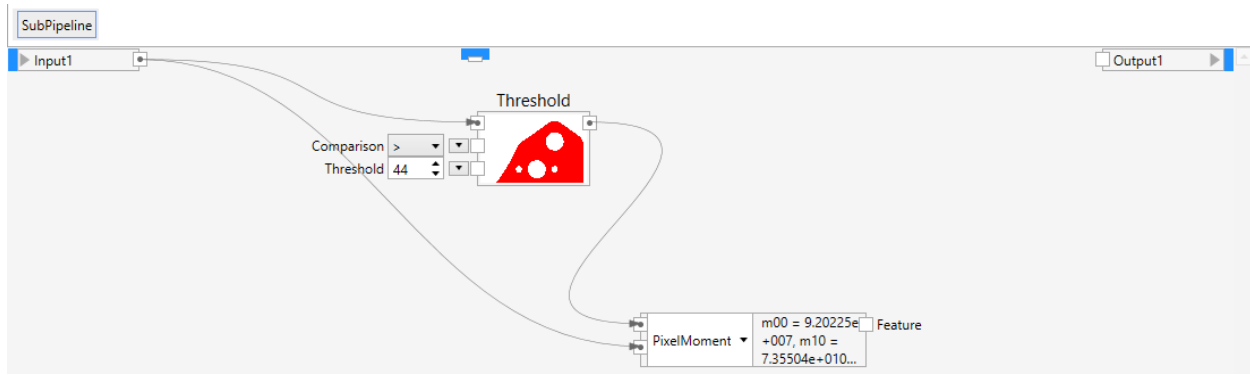


Fig. 10.57: Calculating the pixel based moments of an object.

10.3.13 Pixel-based Normalized Moments

This calculates the pixel-based normalized moments.

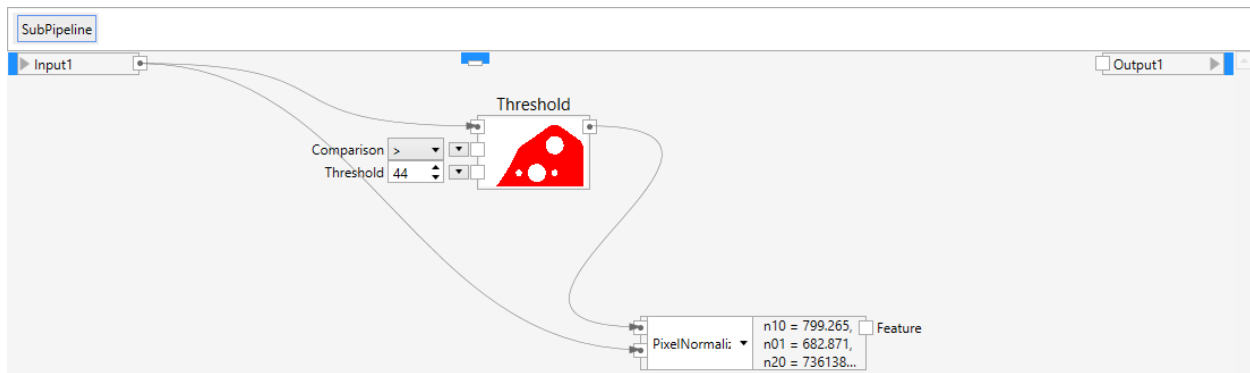


Fig. 10.58: Calculating the pixel based normalized moments of an object.

They are calculated up to the third order, according to the following formula:

$$N_{pq} = \frac{M_{pq}}{M_{00}}$$

Pixel based moments are similar to region-based moments, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. You can use the properties n10, n01, n20, n11, n02, n30, n21, n12 and n03 to read the respective normalized moments.

10.3.14 Pixel-based Centroid

This calculates the pixel-based object centroid. The centroid is a property of the normalized moments.

The centroid is the center of gravity and calculated according to the following formula:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{N_{10}}{N_{00}} \\ \frac{N_{01}}{N_{00}} \end{pmatrix}$$

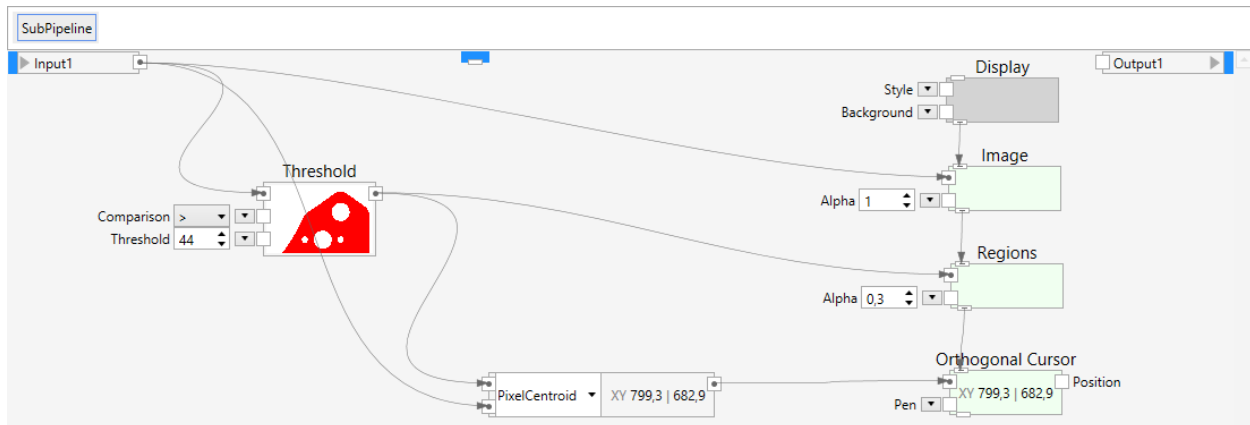


Fig. 10.59: Calculating the pixel based centroid of an object.

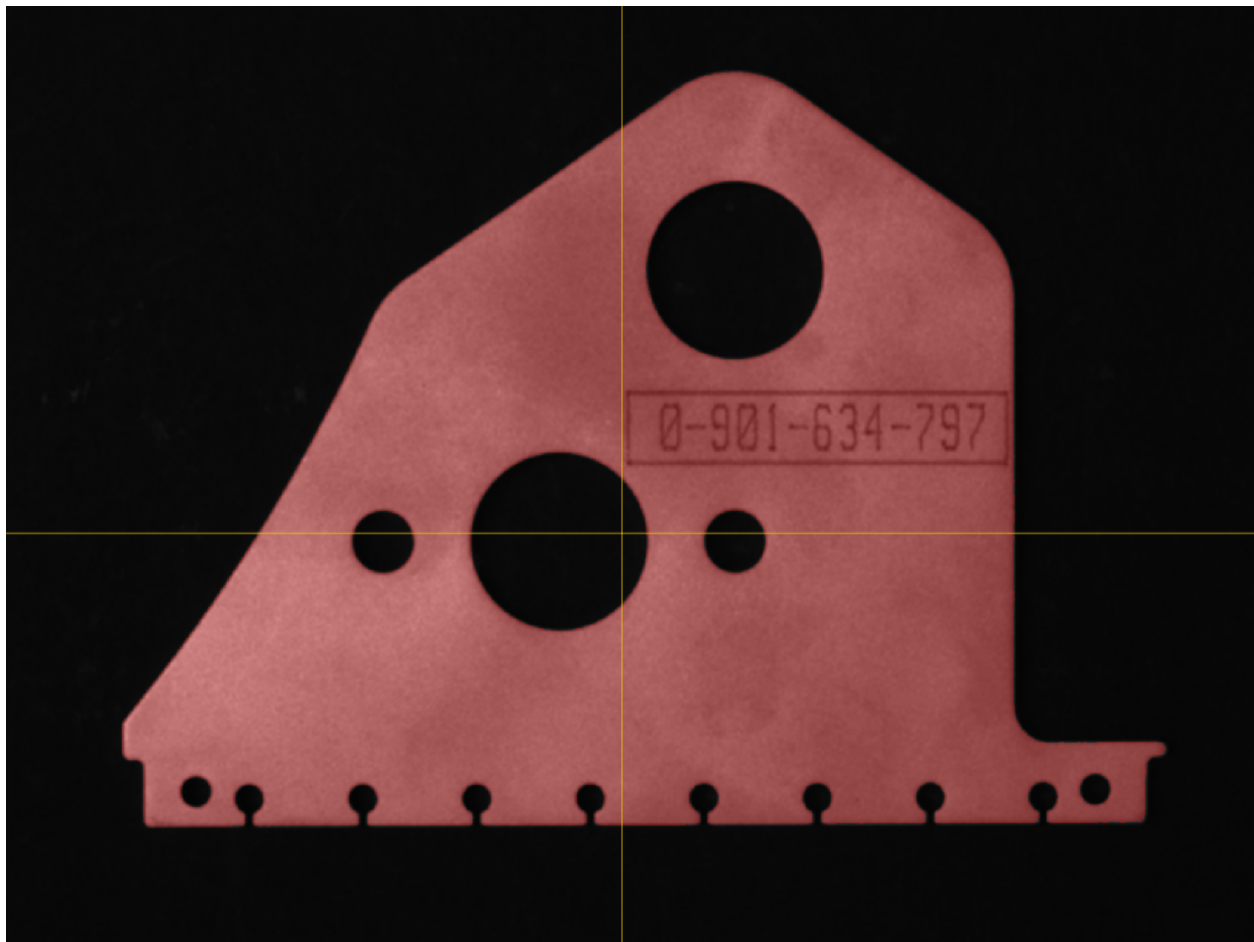


Fig. 10.60: Visualizing the pixel based centroid of an object.

Pixel based centroids are similar to region-based centroids, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated.

10.3.15 Pixel-based Central Moments

This calculates the pixel-based central moments.

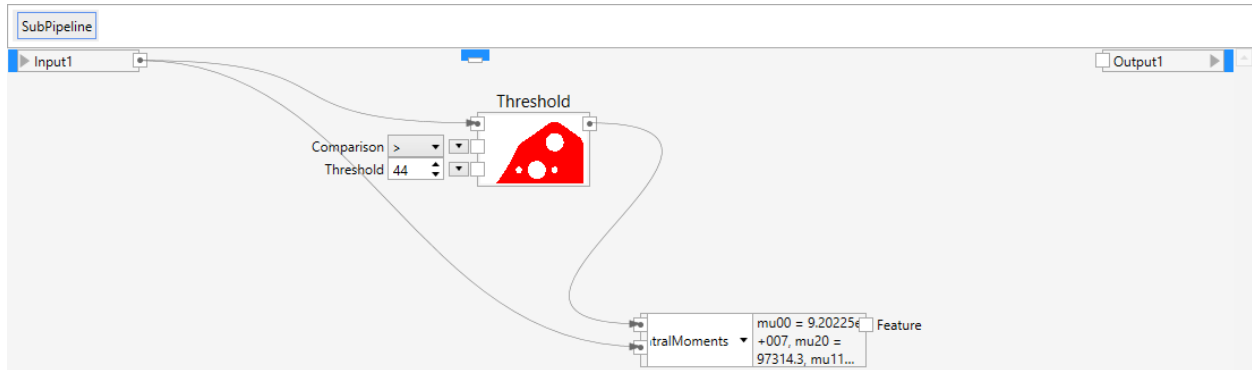


Fig. 10.61: Calculating the pixel based central moments of an object.

They are calculated up to the third order according to the following formula:

$$\mu_{pq} = \frac{1}{M_{00}} \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I_{xy}$$

Central moments are invariant with respect to translation, but are not invariant with respect to rotation. Pixel based moments are similar to region-based moments, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. You can use the properties mu20, mu11, mu02, mu30, mu21, mu12 and mu03 to read the respective central moments.

10.3.16 Pixel-based Equivalent Ellipse

This calculates the pixel-based equivalent ellipse of the object. The equivalent ellipse is built by combining properties from the normalized and the central moments.

Pixel based equivalent ellipses are similar to region-based equivalent ellipses, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated.

10.3.17 Pixel-based Scale Invariant Moments

This calculates the pixel-based scale-invariant moments.

They are calculated up to the third order, according to the following formula:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\frac{p+q}{2}+1}}$$

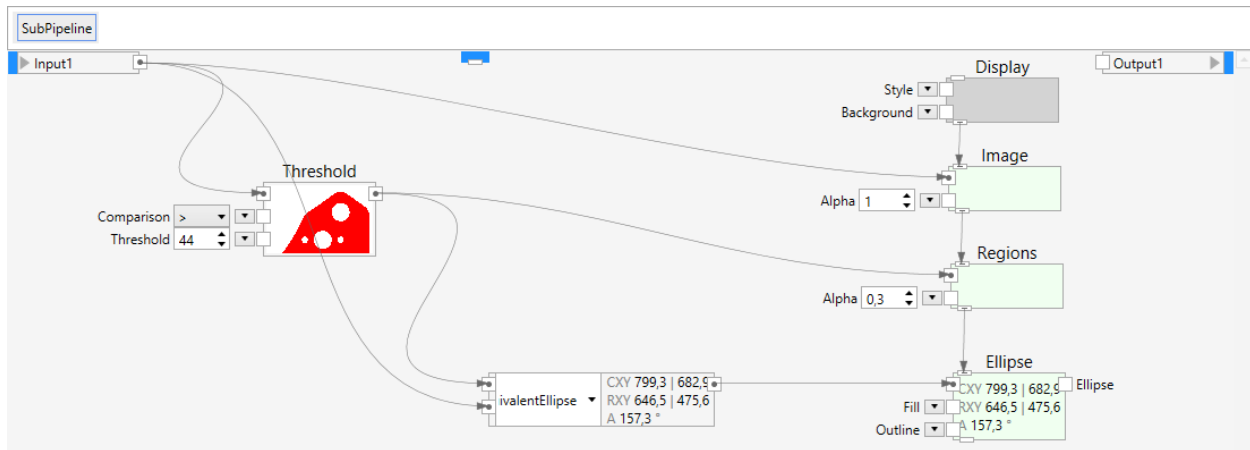


Fig. 10.62: Calculating the pixel based equivalent ellipse of an object.

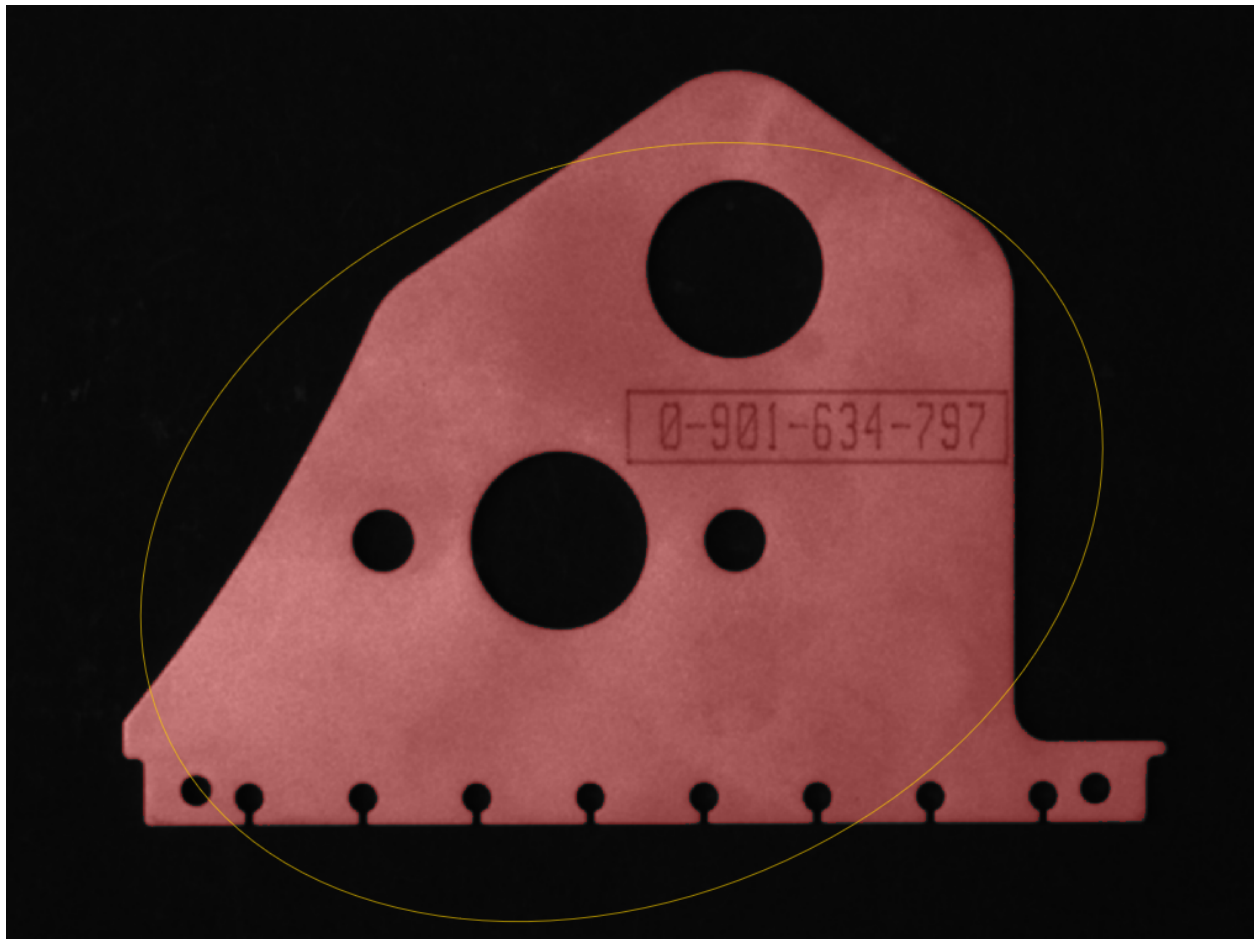


Fig. 10.63: Visualizing the pixel based equivalent ellipse of an object.

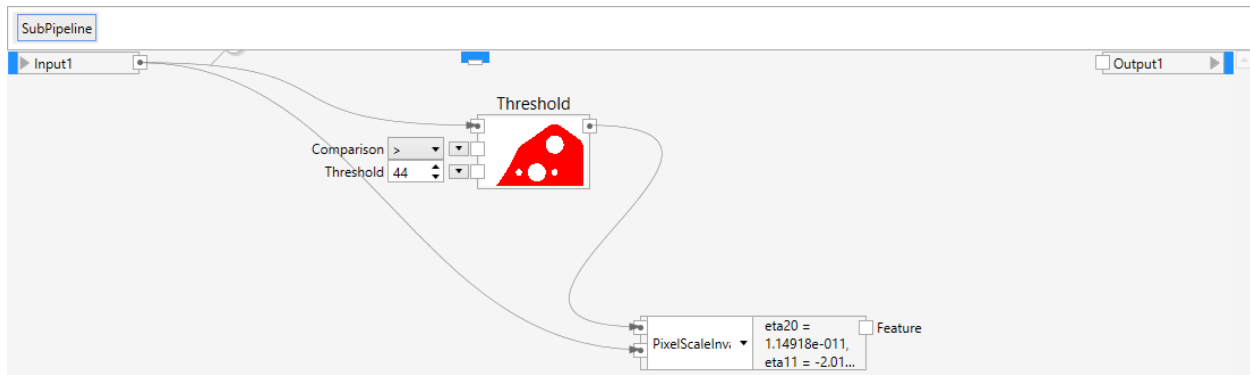


Fig. 10.64: Calculating the pixel based scale invariant moments of an object.

Scale-invariant moments are invariant with respect to translation and scaling, but are not invariant with respect to rotation.

Pixel based moments are similar to region-based moments, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. You can use the properties eta20, eta11, eta02, eta30, eta21, eta12 and eta03 to read the respective scale invariant moments.

10.3.18 Pixel-based Hu Moments

This calculates the pixel-based Hu moments.

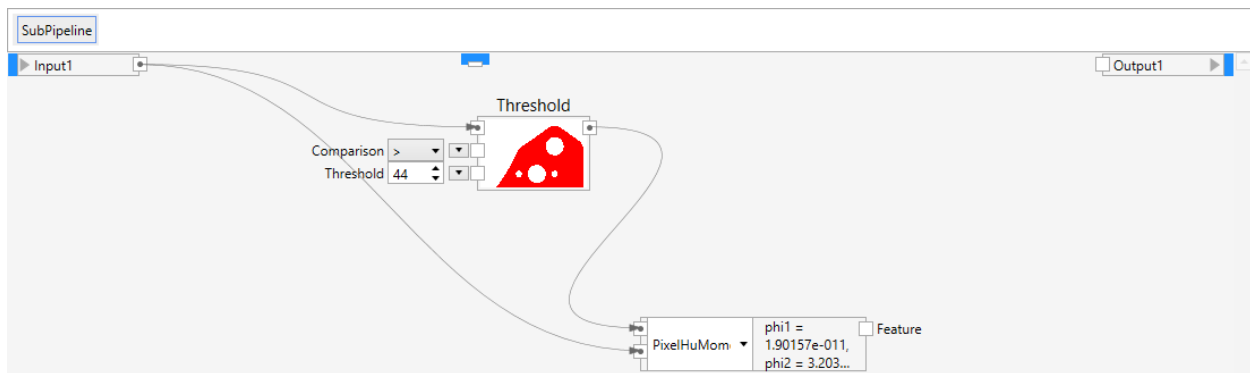


Fig. 10.65: Calculating the pixel based Hu of an object.

The formulas for Hu's moments are:

$$\begin{aligned}
 \phi_1 &= \eta_{20} + \eta_{02} \\
 \phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
 \phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
 \phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
 \phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
 &\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
 \phi_6 &= (\eta_{20} - \eta_{02}) [(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
 &\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
 \phi_7 &= (3\eta_{21} - 3\eta_{03})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
 &\quad - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
 \end{aligned}$$

Hu's moments are invariant with respect to translation, scaling and rotation.

Pixel based moments are similar to region-based moments, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. You can use the properties phi1, phi2, phi3, phi4, phi5, phi6 and phi7 to read the respective Hu moment.

10.3.19 Pixel-based Flusser Moments

This calculates the pixel-based Flusser moments.

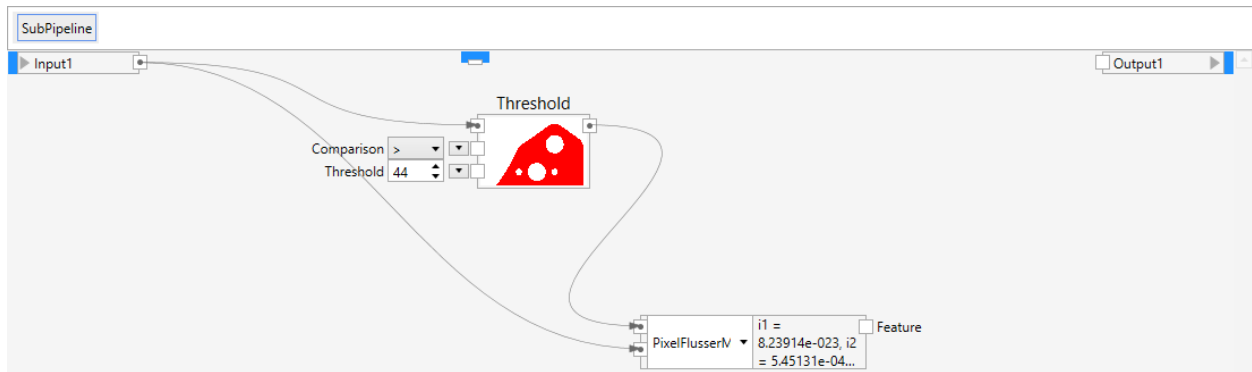


Fig. 10.66: Calculating the pixel based Flusser moments of an object.

The formulas for Flusser's moments are:

$$\begin{aligned}
 I_1 &= \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4} \\
 I_2 &= \frac{\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} + 4\mu_{30}\mu_{12}^3 + 4\mu_{21}^3\mu_{03} - 3\mu_{21}^2\mu_{12}^2}{\mu_{00}^7} \\
 I_3 &= \frac{\mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12})}{\mu_{00}^7} \\
 &\quad + \frac{\mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)}{\mu_{00}^7} \\
 I_4 &= \frac{\mu_{20}^3\mu_{03}^2 - 6\mu_{20}^2\mu_{11}\mu_{12}(\mu_{03} - 6\mu_{20}^2\mu_{02}\mu_{21})\mu_{03}}{\mu_{00}^{11}} \\
 &\quad + \frac{9\mu_{20}^2\mu_{02}\mu_{12}^2 + 12\mu_{20}\mu_{11}^2\mu_{21}\mu_{03} + 6\mu_{20}\mu_{11}\mu_{02}\mu_{30}\mu_{03}}{\mu_{00}^{11}} \\
 &\quad - \frac{18\mu_{20}\mu_{11}\mu_{02}\mu_{21}\mu_{12} + 8\mu_{11}^3\mu_{30}\mu_{03} + 6\mu_{20}\mu_{02}^2\mu_{30}\mu_{12}}{\mu_{00}^{11}} \\
 &\quad + \frac{9\mu_{20}\mu_{02}^2\mu_{21}^2 + 12\mu_{11}^2\mu_{02}\mu_{30}\mu_{12} - 6\mu_{11}\mu_{02}^2\mu_{30}\mu_{21}}{\mu_{00}^{11}} \\
 &\quad + \frac{\mu_{02}^3\mu_{30}^2}{\mu_{00}^{11}}
 \end{aligned}$$

Flusser's moments are invariant with respect to affine transformations.

Pixel based moments are similar to region-based moments, but they use pixel values in addition. The brighter a pixel is the more weight it has in the calculation of the moment.

If color images are used, they are converted to monochrome, before the respective pixel-based moment is calculated. You can use the properties i1, i2, i3, and i4 to read the respective user moments.

10.4 Object Filtering

Filtering is the process of removing objects that satisfy a certain condition. Examples of filtering are removal of small objects caused by noise, or removal of objects touching the image border, because these objects would be partial anyway and their features would be wrong.

Here is an example that removes objects with an area smaller than 100 pixels:

Here is another example that removes objects which are touching the left border:

Filters can be chained in order to create more complex conditions.

10.5 Gauging

nVision can be used to measure positions of edges in images. It uses projections along lines, within rectangles or along other curves or within other shapes. The projections are analyzed and edges can be measured fast and with subpixel accuracy. The precision that you can achieve depends on the characteristics of your system (noise in the image, sharpness of the edges, etc.), but in general you can expect 1/20th of a pixel precision with an optimal setup.

The first step in measuring edges is the selection of an area of interest. This can be a line segment, an oriented rectangle, or even a circular ring. From this area of interest, an intensity profile is calculated. Usually you would use real image intensities, but you can also measure edges in a different color space, if you are interested in the location of a color change.

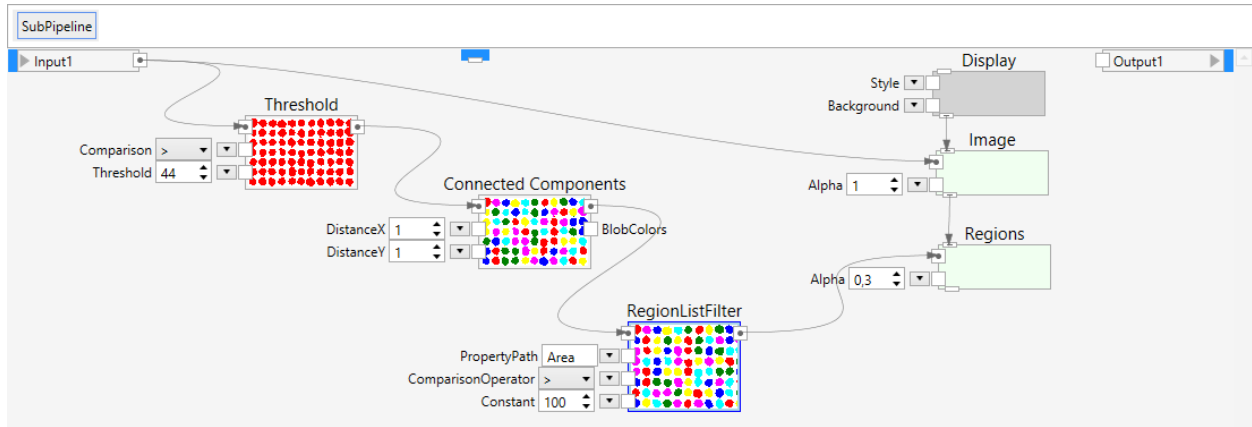


Fig. 10.67: Removing small objects.

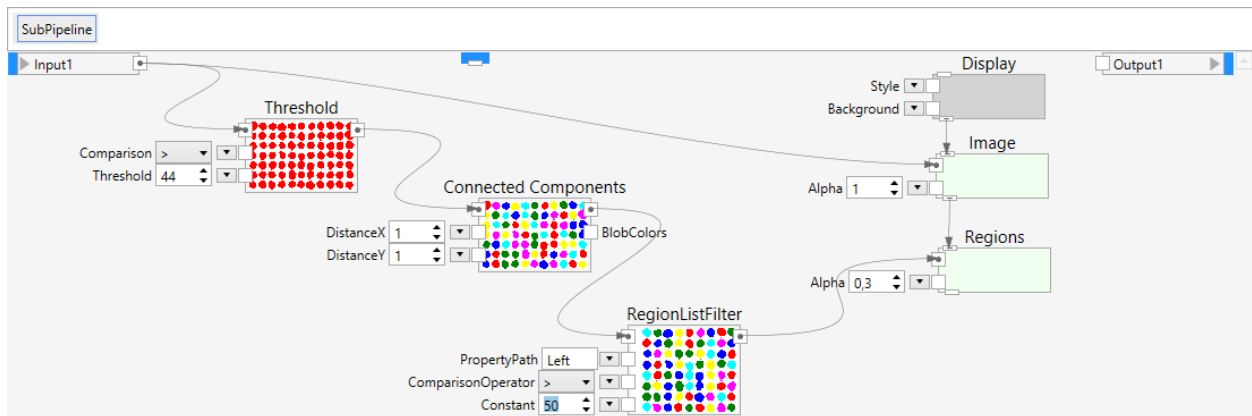


Fig. 10.68: Removing objects near left boundary.

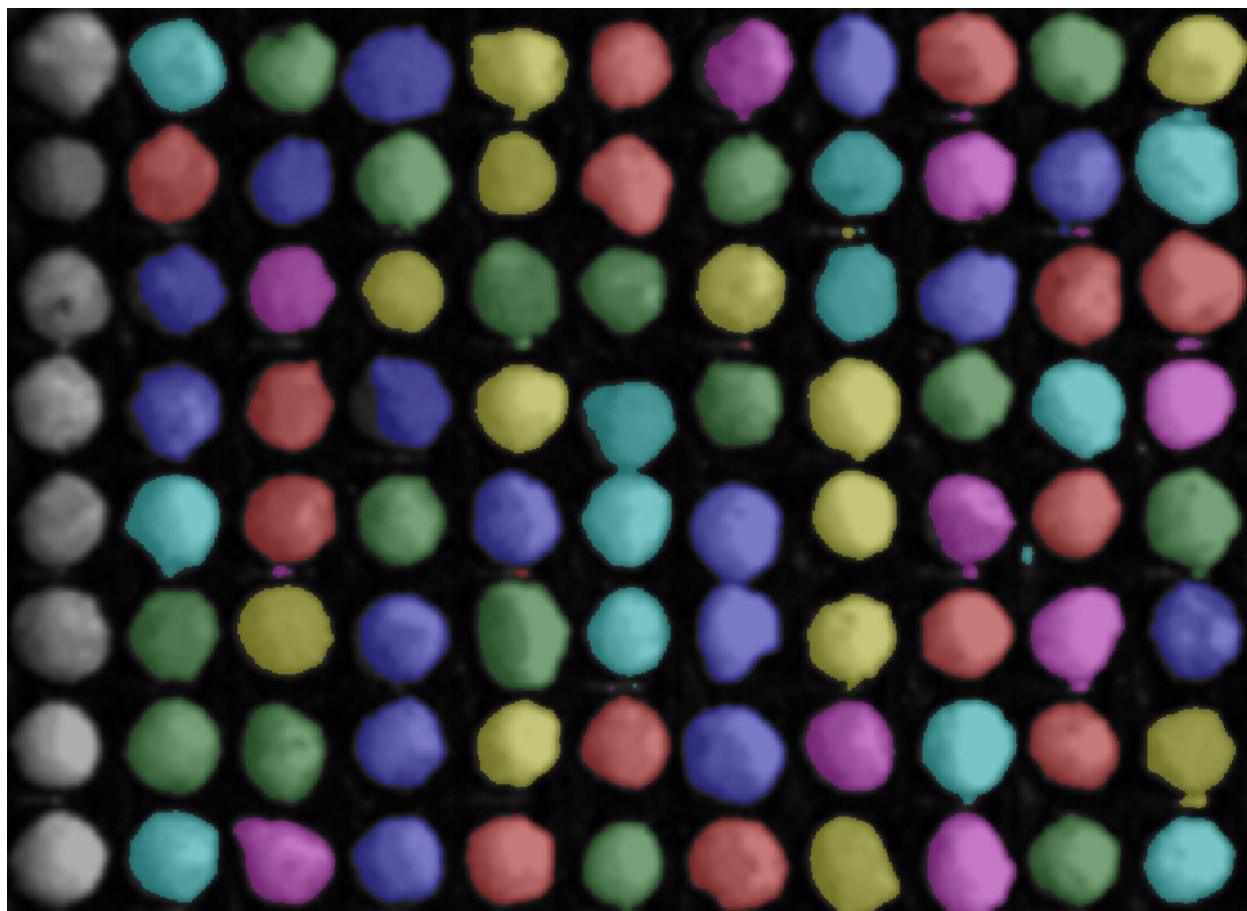
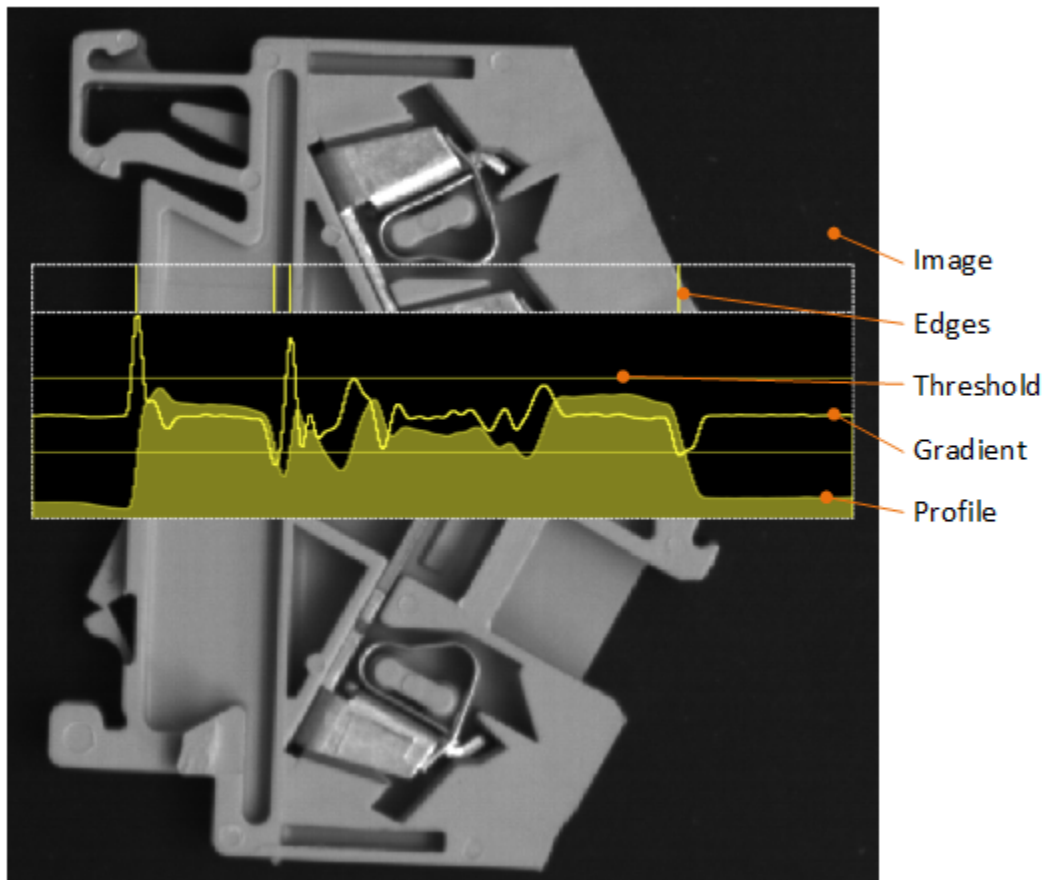


Fig. 10.69: Objects near left border have been removed..

The intensity profile is then smoothed in order to reduce noise and then the gradient is calculated. The extrema within the gradient are searched and their strength is compared to a threshold. If the extrema fall below the threshold, they are discarded; otherwise the extrema represent real edges. The position of the edge is then calculated with subpixel precision, as well as the interpolated grey and gradient values at this location.

Here is the full edge information visualized within a tool.



10.6 Identification

10.6.1 Barcode Decoding

nVision supports decoding barcodes.

Several one-dimensional symbologies are supported. Here is an alphabetical list of the supported symbologies:

Symbology	Dimensionality	Link to Description
Code39	1D	http://en.wikipedia.org/wiki/Code_39
Code93	1D	http://en.wikipedia.org/wiki/Code_93
Code128	1D	http://en.wikipedia.org/wiki/Code_128
EAN	1D	http://en.wikipedia.org/wiki/International_Article_Number_(EAN)
UPC	1D	http://en.wikipedia.org/wiki/Universal_Product_Code

The decoder node has a selection list, where you can select the symbologies that you want to decode.

Code 39
Code 93
Code 128
EAN 8
EAN 13
ITF
UPC A
UPC E
UPC EAN Extension

Fig. 10.70: The symbology selection list.

The decoder expects one code in the image that is given. It expects the code to sit roughly in the middle and the scanning is horizontal. This means that the bars should be vertical. A virtual horizontal scan line that sits in the middle should cross the whole code.

Here is an example of a perfectly positioned code:

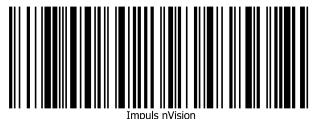


Fig. 10.71: Perfect position of code for the decoder.

The code in the following image cannot be decoded:

The code in the following image does not decode, because the virtual scan line does not cross the whole code:

The barcode decoder does not do anything else to locate a barcode. If your circumstances are different, you can use other preprocessing tools, to bring the barcode in a suitable position for the decoder. If the barcode is not properly rotated, you can use the rotation tools. If the barcode is not properly located, you can use the crop tool to cut out the relevant portion for the barcode decoder. If contrast or brightness are not sufficient, you can use image preprocessing.

The decoder can be inserted in the linear pipeline or in a sub-pipeline.

In a sub-pipeline, there is an additional region input as well as a boolean input, that tries more location steps at the expense of longer decoding time.

In addition, also the output of the decoder can be handled in more detail. The decoder delivers a list of results, and each result contains the decoded string, the name and identifier of the recognized symbology and the position where the code was found.

10.6.2 Matrixcode Decoding

nVision supports decoding matrix codes.

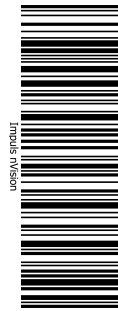


Fig. 10.72: The decoder cannot decode this image, because the bars are horizontal.



Fig. 10.73: The decoder cannot decode this image, because the virtual scan line does not cross the whole code.



Fig. 10.74: In a linear pipeline, the image flows in from the top and the symbologies can be selected.

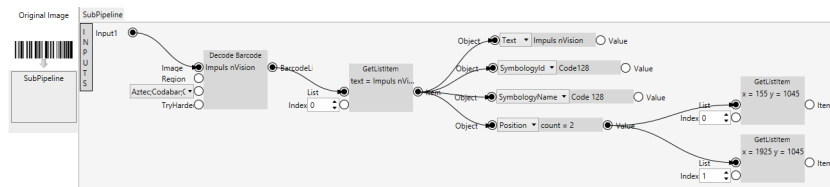


Fig. 10.75: In a sub-pipeline, more control can be executed.

Several two-dimensional symbolologies are supported. Here is an alphabetical list of the supported symbolologies:

Symbology	Dimensionality	Link to Description
Aztec	2D	http://en.wikipedia.org/wiki/Aztec_Code
Data Matrix	2D	http://en.wikipedia.org/wiki/Data_Matrix
PDF417	2D	http://en.wikipedia.org/wiki/PDF417
QR Code	2D	http://en.wikipedia.org/wiki/QR_code

The decoder node has a selection list, where you can select the symbolologies that you want to decode.

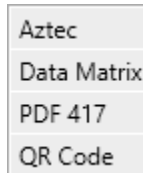


Fig. 10.76: The symbology selection list of the matrixcode decoder.

The decoder expects one code in the image that is given. It expects the code to sit roughly in the middle.

Here is an example of a perfectly positioned code:



Fig. 10.77: Perfect position of code for the decoder.

The matrix code decoder does not do anything else to locate a code. If your circumstances are different, you can use other preprocessing tools, to bring the matrix code in a suitable position for the decoder. If the matrix code is not properly located, you can use the crop tool to cut out the relevant portion for the matrix code decoder. If contrast or brightness are not sufficient, you can use image preprocessing.

The decoder can be inserted in a sub-pipeline.

10.7 Camera Calibration

In many measurement applications, measurements are to be taken in world units, such as meters. A typical camera transforms those world units into pixels. Camera calibration aims to perform the inverse transformation, in order to convert distances expressed in pixels back to world units. Camera calibration establishes a correspondence between coordinates in the real world and coordinates in the camera image.

The real world is three-dimensional and a camera creates a two-dimensional picture of it. The resulting projection usually is a perspective projection, where distances between points far away from the camera are smaller than the same distances between points nearer to the camera.

If you can control the optical setup in a way that the optical axis is orthogonal to an observed planar scene, you can setup the calibration by defining a scaling factor only, like in the following example.

The screenshot shows the definition of a scale with the **Define Scale** tool in the **Adjust Camera** group of the **Home** tab on the ribbon.

Once the scale has been defined, it will be used by the spatial measurement tools, and they will return their measured values in the world coordinates that have been defined with the **Define Scale** tool. It is important to know that the

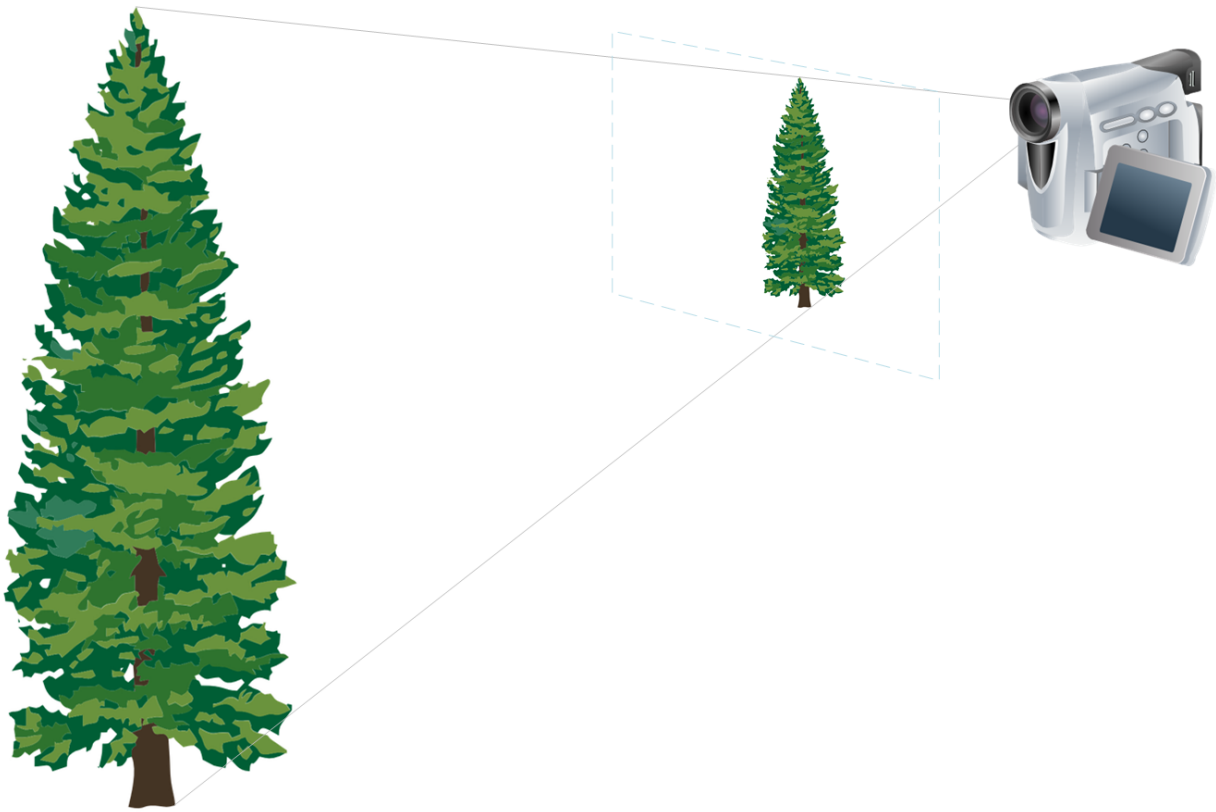


Fig. 10.78: Real World and Camera Correspondence

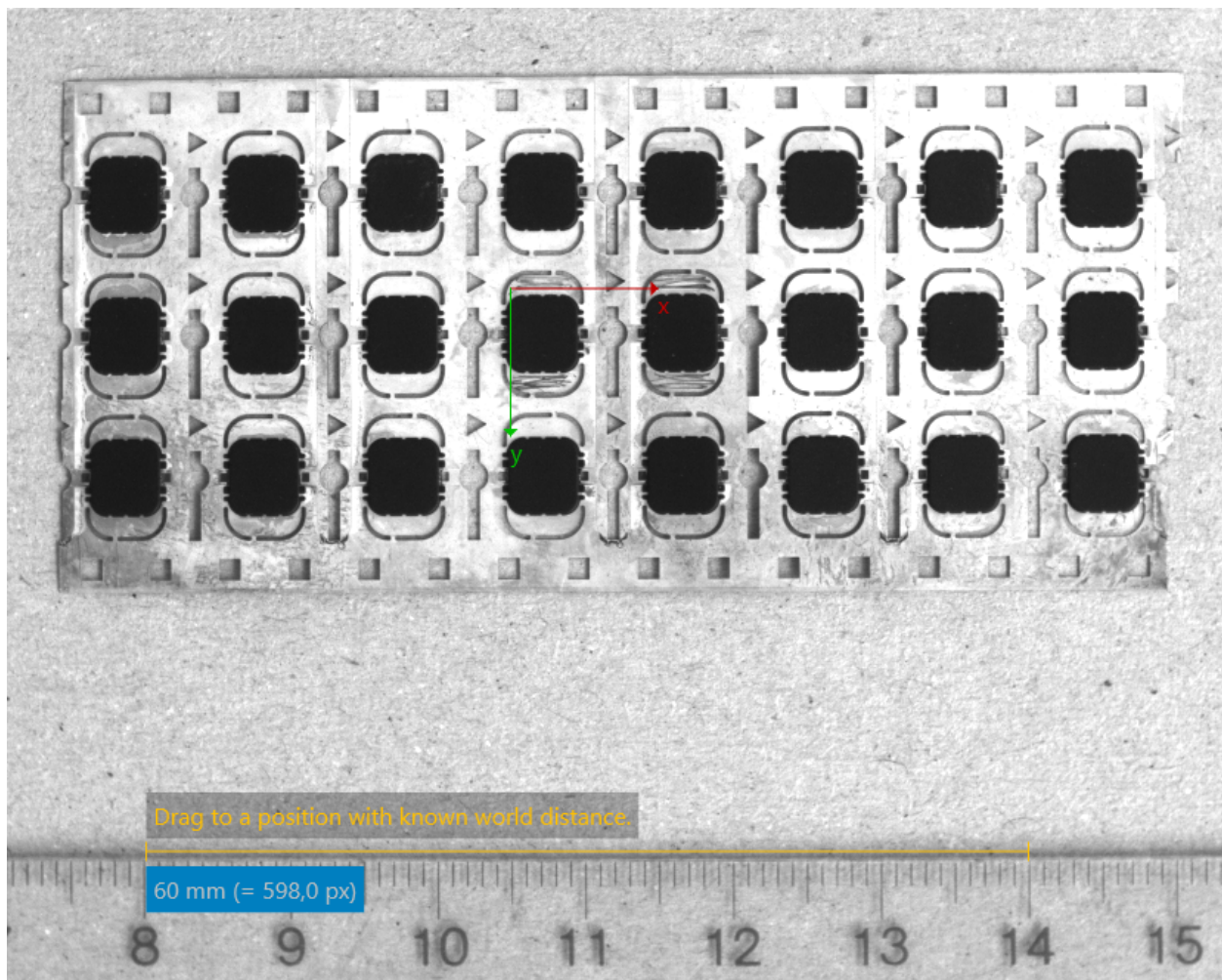


Fig. 10.79: Define a scaling factor

method with the scaling factor will only work, if you have an orthogonal camera setup. If the real setup deviates, the accuracy of the method will suffer.

Another, more automatic way to calibrate the camera is to use a calibration target. **nVision** uses calibration targets in order to find the correspondence between coordinates in the camera coordinate system and the respective coordinates in the real world. The resultant calibration is valid only for the two-dimensional plane where the calibration target has been put in the real world.

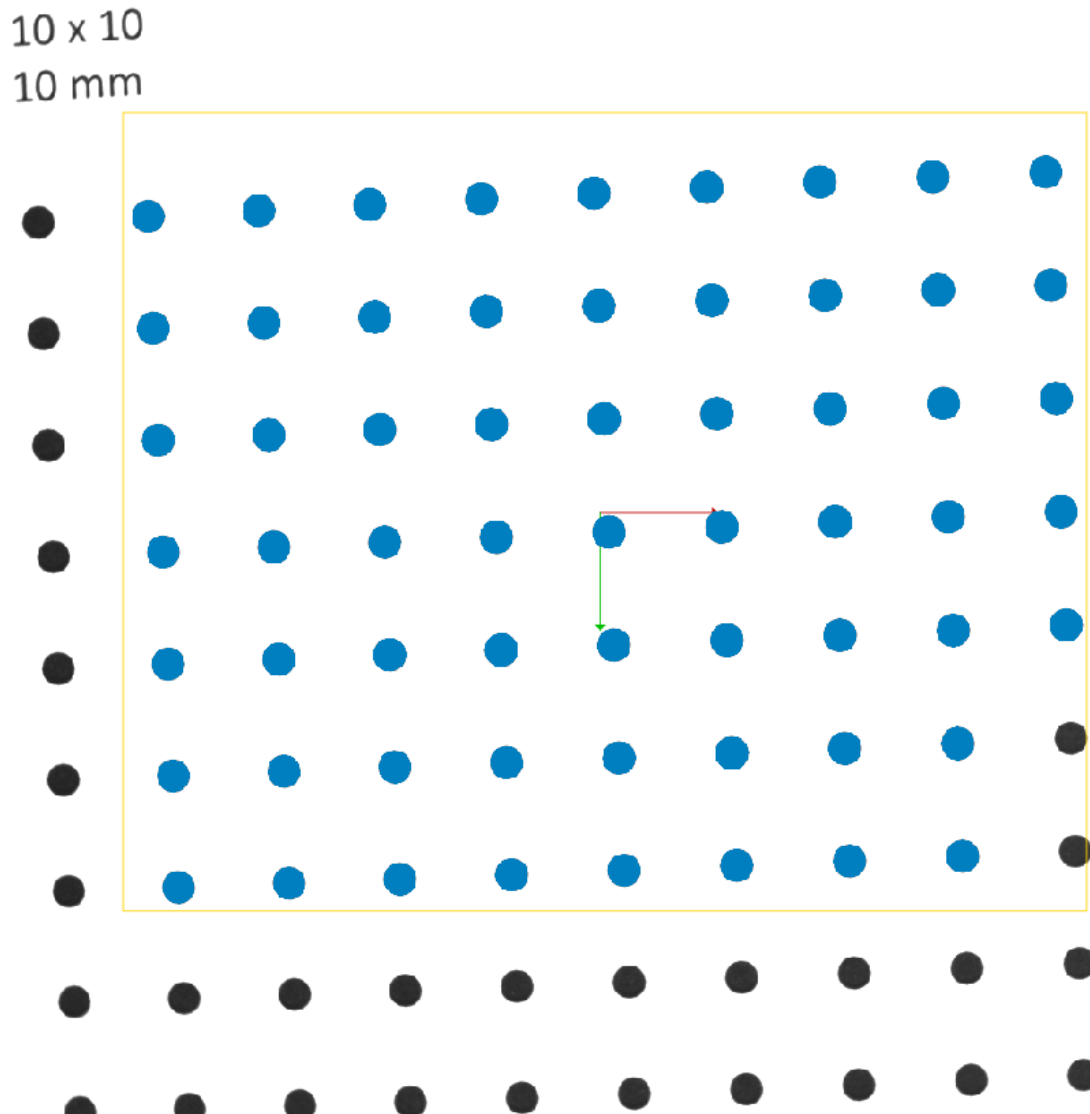








Fig. 10.80: Using a calibration target

The calibration target consists of a pattern with known geometry and known distances. The calibration procedure takes an image of the pattern and measures the pattern. The measured positions are with respect to the camera coordinate system. Since the coordinates of the calibration target are known, they can be related to the measured positions and the correspondence can be established. Once this correspondence is known, any coordinate in the image coordinate system can be transformed to the real world coordinates (in the plane of the calibration target).

nVision has a set of tools that help with camera calibration. The **Define Scale** and **Calibrate** tools can be used to

determine the calibration. The **Distance**, **Distance (Two Points)**, **Circle** and **Area Size** tools return their results in world units if used with calibration.

Tool	Icon	Description
Define Scale		Interactively define a scaling factor to be used in camera calibration.
Calibrate		Use a calibration target to automatically determine the camera calibration.
Distance		Measure a distance along a line in world coordinates.
Distance (Two Points)		Measure a distance between two points in world coordinates.
Circle		Measure a circle position and radius in world coordinates.
Area Size		Measure the size of an area in world coordinates.

Mathematically, the background of the camera calibration is a perspective transformation between pixel coordinates and real world coordinates. To find the values of the perspective transformation, the correspondence of points in the real world (they come from the known distance of the calibration target) and the points in the image (they are measured from the image of the calibration target) is needed. At least four correspondent points are needed, but **nVision** can handle any number of additional points. Once this correspondence is known, it is saved as a prespective transformation matrix along with the unit of the real world coordinates.

Various vendors produce high quality calibration targets in different size and made from different material. See the Edmund Optics website for more information: <http://www.edmundoptics.com/test-targets/distortion-test-targets/fixed-frequency-grid-distortion-targets/>, <http://www.edmundoptics.com/test-targets/distortion-test-targets/diffuse-reflectance-grid-distortion-targets/3046/>.

A cheap way to create calibration targets is to print them out with a printer. The dots should be arranged in a rectangular way and the horizontal and vertical distances of all dots to their neighbors must be the same.

11.1 Geometry

Many functions in image processing and image analysis deal with geometric entities, i.e. when you determine the position of a template or the length of a structure. **nVision** has a set of geometric features, which help managing, visualizing and manipulating geometries.

nVision provides a set of geometric primitives as well as a set of geometric transformations. Geometric primitives are things like points, vectors, lines, etc. Geometric transformations are translations, rotations, scalings, etc. Geometric transformations allow to transform geometric primitives, i.e. move, scale or rotate them, etc. Geometric primitives have features: the length and orientation of a line segment are examples. In addition, geometric primitives can be used to perform geometric calculations: examples are the calculation of the intersection between two lines, or the calculation of a convex hull polygon, given a set of points.

Besides the geometric primitives, **nVision** has graphic counterparts for some to support the visualization and the interactive manipulation of the geometric primitives. For example, a **PointWidget** displays a point on some surface with a color and diameter that can be chosen via a **Pen**. The point widget also supports interactive manipulation, i.e. the user can drag the point on the surface with the mouse.

11.2 Geometric Primitives

The geometric primitives are used to model geometries. They are drawn here with respect to a coordinate system used in images, where the positive x-axis extends to the right and the positive y-axis extends downwards. In such a coordinate system, positive angles are in clockwise direction.

11.2.1 Direction

A **Direction** is a vector of unit length in two-dimensional space. It serves the same purpose as an angle, i.e. it represents a direction in the two-dimensional Euclidean space R^2 .

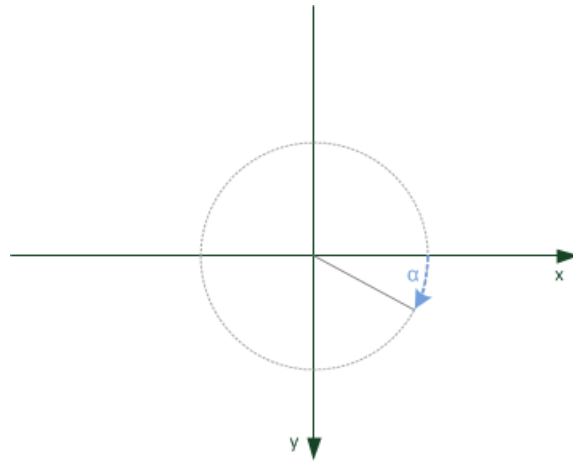


Fig. 11.1: The direction geometric primitive.

You can default-construct a direction (corresponding to an angle of 0 degrees), you can convert it to an angle, calculate the orthogonal and opposite directions, add and subtract directions as well as find the bisector between two directions.

11.2.2 Vector

A `Vector` is a vector of arbitrary length in two-dimensional Euclidean space R^2 .

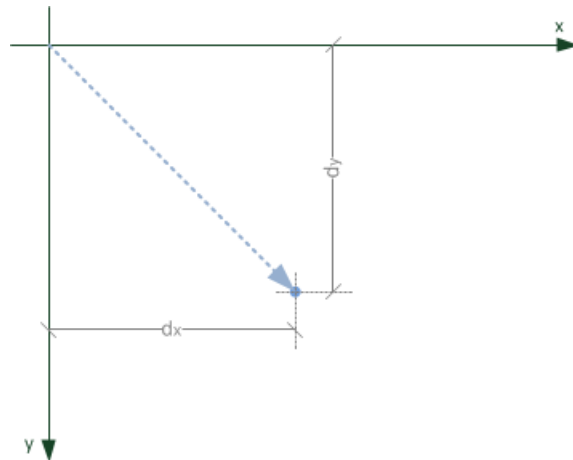


Fig. 11.2: The vector geometric primitive.

You can default-construct a vector (of length 0) or you can pass a coordinate pair. With a given vector, you can calculate its direction and length. You can also read back its horizontal and vertical extents. You can also find its direction, normal vector or calculate the dot product with another vector.

11.2.3 Point

A `Point` is a point at an arbitrary position in two-dimensional Euclidean space R^2 .

You can default-construct a point (at the origin) or you can pass a coordinate pair. You can also read back its horizontal and vertical position. The difference of two points is the directed distance between them, i.e. it is a vector.

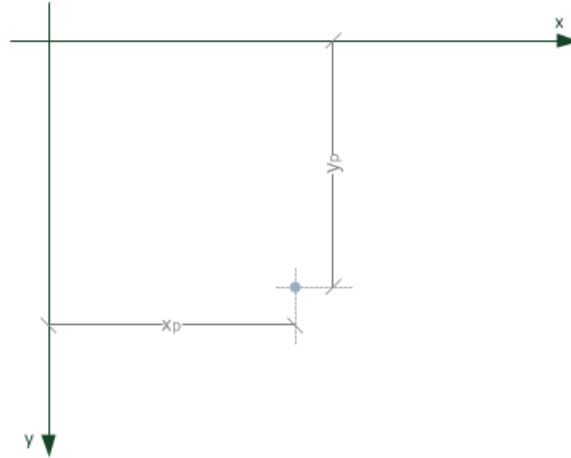


Fig. 11.3: The point geometric primitive.

11.2.4 Line

A `Line` is a line at an arbitrary position and direction in two-dimensional Euclidean space R^2 that extends into infinity at both ends.



Fig. 11.4: The line geometric primitive.

You can default-construct a line (horizontal at the origin) or you can pass a point and a direction. There are various ways you can modify a line, which boil down to the fact that you can modify its origin and direction.

11.2.5 Ray

A `Ray` is a line starting at an arbitrary position and extending into infinity along its direction in two-dimensional Euclidean space R^2 .



Fig. 11.5: The ray geometric primitive.

You can default-construct a ray (horizontal starting at origin along the positive x-axis) or you can pass a point and a direction. There are various ways you can modify a ray, which boil down to the fact that you can modify its origin and direction.

11.2.6 LineSegment

A `LineSegment` is a line between two arbitrary points in two-dimensional space R^2 .

You can default-construct a line segment (between origin and point $[1, 0]$) or you can pass a point, direction and extent, or you can pass two points. There are various ways you can modify a line segment, which boil down to the fact that you can modify its origin, direction and extent.



Fig. 11.6: The line segment geometric primitive.

11.2.7 Box

A `Box` is a rectangle of arbitrary size located at an arbitrary point in two-dimensional Euclidean space R^2 , whose sides are parallel to the coordinate axes.



Fig. 11.7: The box geometric primitive.

You can default-construct a box or you can pass a point and a vector.

11.2.8 Rectangle

A `Rectangle` is characterized by a center point P , two radii rx and ry and an angle α .



Fig. 11.8: The rectangle geometric primitive.

A rectangle can be created from a center point, a diagonal vector and an angle given in radians.

11.2.9 Circle

A `Circle` is located at its center point P and contains the set of points having distance radius to the center point.

You can default-construct a circle or you can pass a point and a radius. A circle is characterized by a center point P and the radius r .

11.2.10 Ellipse

An `Ellipse` is characterized by a center point P , two radii rx and ry and an angle α .

An ellipse can be created from a center point, a diagonal vector and an angle given in radians.

11.3 Geometric Transformations

This chapter explains geometric transformations, such as translation, rotation, etc. The available transformations are listed and their properties are explained.

The various translations can be ordered and structured. Starting with the identity transform everything is invariant. The translation transform leaves everything but the position invariant, the rotation transform leaves everything but the rotation invariant. Combining translation and rotation yields a rigid transform, which leaves everything but position



Fig. 11.9: The circle geometric primitive.



Fig. 11.10: The ellipse geometric primitive.

and rotation invariant. Combining a rigid transform with an isotropic scaling transform yields a similarity transform, which does no longer preserve area and length. An affine transform no longer leaves angles invariant and a projective transform does not preserve parallelism.

The diagram shows the various transforms in a hierarchy and the little drawings hint about the transformations that are carried out. **nVision** implements functions to transform various geometric primitives with these transformations. In addition, transformations can be determined by relating sets of points (often called passpoints) that correspond in both world and image coordinates.

11.3.1 Translation

Translation moves geometry in the plane by a translation vector.

It has two degrees of freedom, namely the horizontal and vertical displacements (i.e. the components of the translation vector).

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is translated or shifted by adding offsets to it. This can be written as:

$$x' = x + t_x$$

$$y' = y + t_y$$

or in matrix form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

By using homogeneous coordinates, the equation can also be written like this:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

We will use the notation with homogeneous coordinates from now on.

11.3.2 Rotation

Rotation rotates geometry in the plane by some angle.

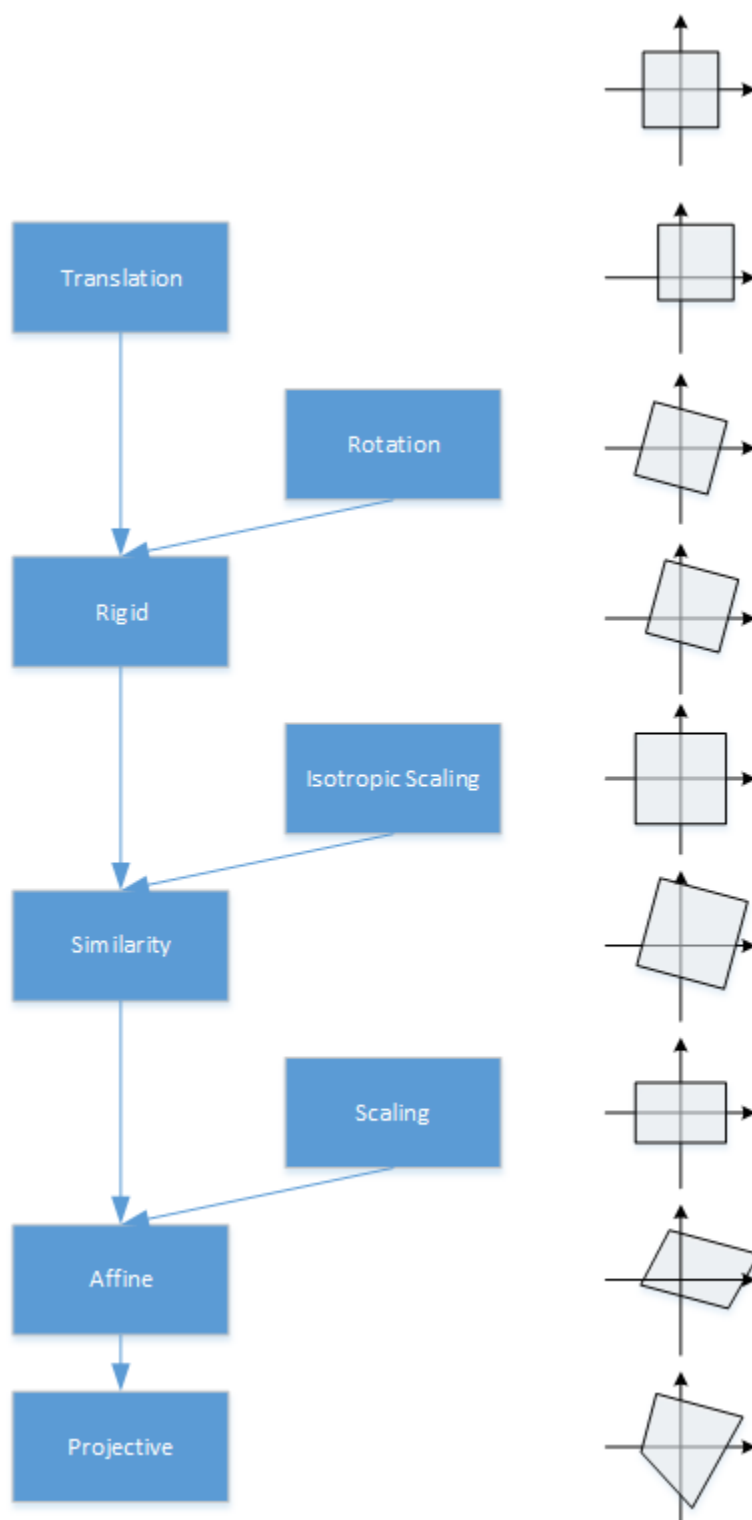


Fig. 11.11: The hierarchy of the geometric transformations.

It has one degree of freedom, namely the rotation angle.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is rotated around the origin by multiplying with a rotation matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.3 Rigid

A rigid transformation combines translation and rotation.

It has three degrees of freedom, namely the horizontal and vertical displacements (i.e. the components of the translation vector) and the rotation angle.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with a rigid matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & t_x \\ \sin(\phi) & \cos(\phi) & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.4 Isotropic Scaling

An isotropic scaling scales geometry in the plane by some factor.

It has one degree of freedom, namely the scaling factor.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with an isotropic scaling matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.5 Similarity

A similarity transformation combines a rigid transformation with isotropic scaling.

It has four degrees of freedom, namely the horizontal and vertical displacements (i.e. the components of the translation vector), the rotation angle and the scaling factor.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with a similarity matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s * \cos(\phi) & -s * \sin(\phi) & t_x \\ s * \sin(\phi) & s * \cos(\phi) & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.6 Scaling

A scaling scales geometry in the plane by different factors in the horizontal and vertical direction.

It has two degrees of freedom, namely the horizontal and vertical scaling factors (i.e. the components of the scaling vector).

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with a scaling matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.7 Affine

An affine transformation has six degrees of freedom.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with an affine matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.3.8 Projective

A projective transformation has eight degrees of freedom.

A point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

in the two-dimensional plane is transformed by multiplying with a projective matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

11.4 Graphics

11.4.1 Coordinates

Graphics are drawn within a coordinate system. In order to facilitate easy cooperation with image coordinates, the coordinate system has its origin in the upper left corner, positive x coordinates extend to the right and positive y coordinates extend to the bottom. The graphics system is two-dimensional. There is no z dimension in graphics that would correspond with the z dimension in images. The graphics system is connected to the system of hierarchical widgets. Each widget lives in the coordinate system of its parent widget, but it may establish a different coordinate system for itself and its children. For example, the `WidgetImage` shifts its coordinate system by half a pixel right and down, in order to make sure that integer coordinates are in the middle of a pixel. If you draw graphics on top of an image (inside an `ImageWidget`), the lines will be centered on the image pixels if you draw them with integer coordinates.

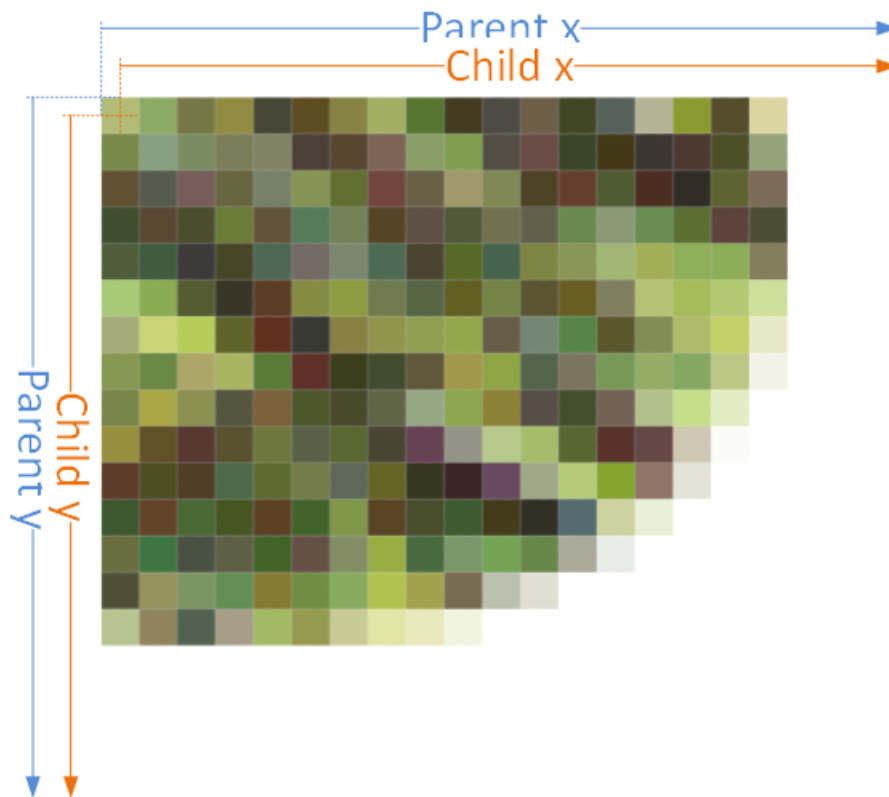


Fig. 11.12: Image coordinates

11.4.2 Brush

A brush is needed to fill a geometric primitive. **nVision** provides a solid color brush. A solid color brush can be created from a color and an alpha value.

A brush can be created by setting values for the primary colors red, green and blue, as well as its opacity (which is the opposite of transparency).

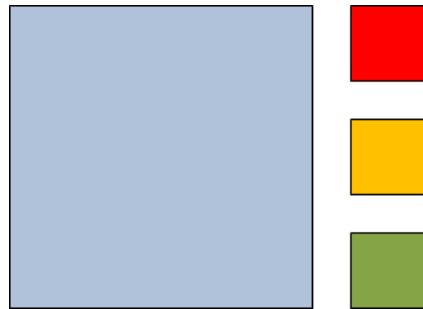


Fig. 11.13: Solid color brushes

11.4.3 Pen

A pen is needed to outline a geometric primitive. A pen has a certain brush, a width, a dash style, a dash offset, a cap style for start, end and dash caps, a line join style and a miter limit. It is used to draw outlines.

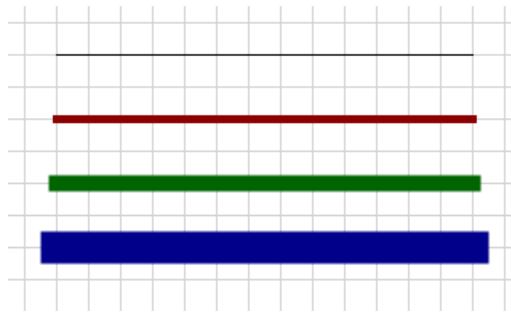


Fig. 11.14: Pen color and thickness

The picture shows various pens with different colors and increasing width from top to bottom. The fact that the line ends seem to extend more to the left and right with the thicker lines comes from the fact that a square end line cap is used by default.

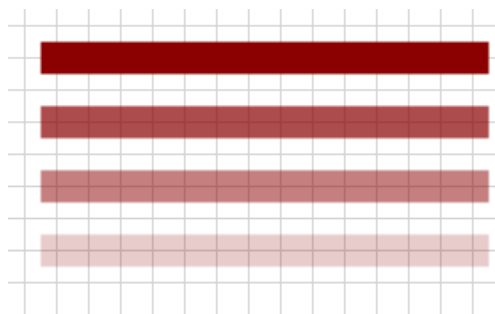


Fig. 11.15: Pen opacity on white background

A pen's opacity or transparency can be adjusted with its alpha value (alpha = 1.0 means fully opaque, alpha = 0.0 means fully transparent). The resulting color is a mixture of the background with the pen color, where the alpha value controls the weight of the mixture.

A pen can have flat, square, round or triangle shaped end caps, as shown in the picture. The caps can be specified separately for the begin and end of the line.

A pen can have various dash styles: solid, dashed, dotted, etc.

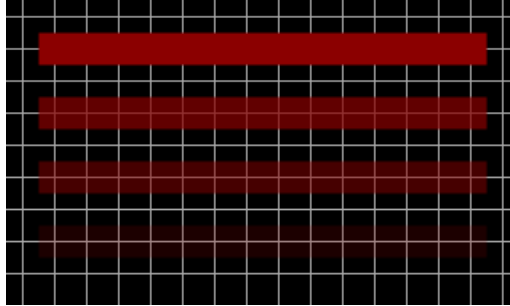


Fig. 11.16: Pen opacity on black background

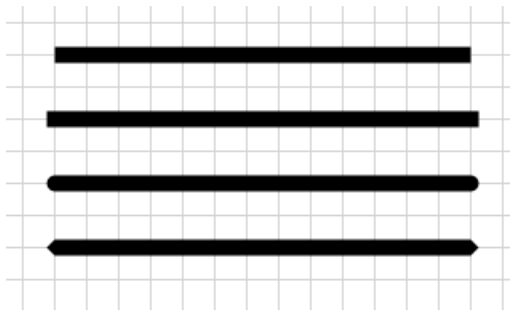


Fig. 11.17: Pen end caps

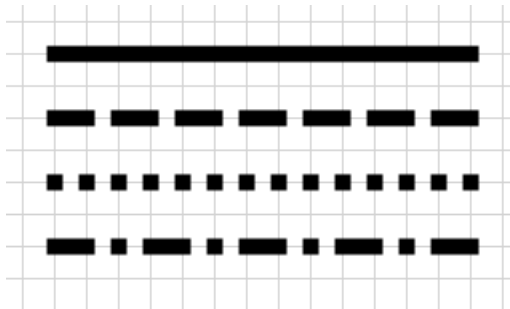


Fig. 11.18: Pen dash style

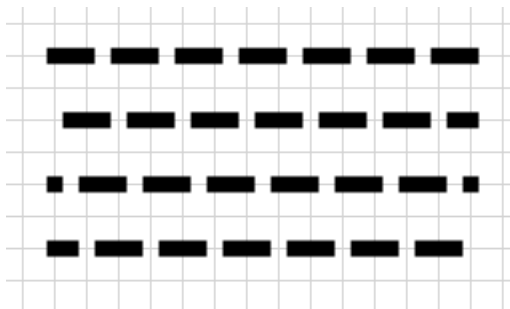


Fig. 11.19: Pen dash offset

The dash pattern can be shifted left or right with the dash offset. Positive values shift to the left, negative values shift to the right. The actual shift amount is the pen width multiplied by the dash offset.

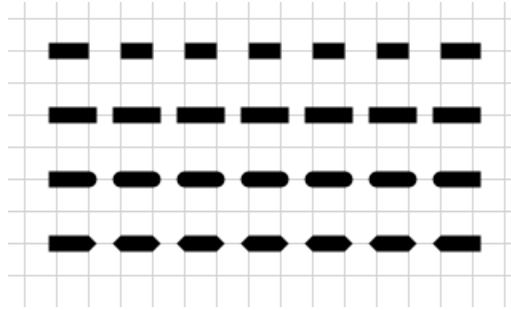


Fig. 11.20: Pen dash caps

Similar to the end caps, a pen can have also dash caps.



Fig. 11.21: Pen miter join



Fig. 11.22: Pen bevel join

A polygon can have various joins at the vertices: miter join, bevel join, round join.

The miter limit controls how far joins can extend, when the angle between the lines is small.

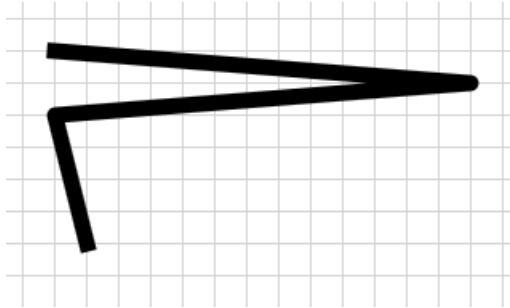


Fig. 11.23: Pen round join

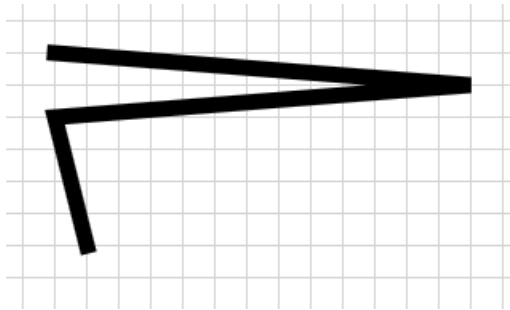


Fig. 11.24: Pen miter or bevel join

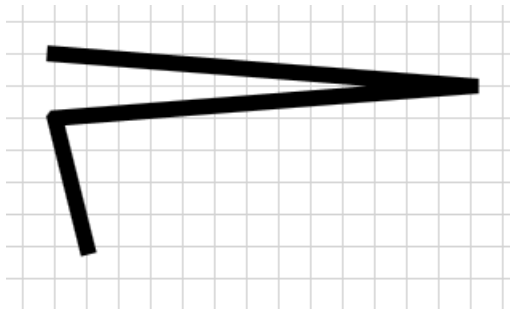


Fig. 11.25: Pen miter join with miter limit 1.0

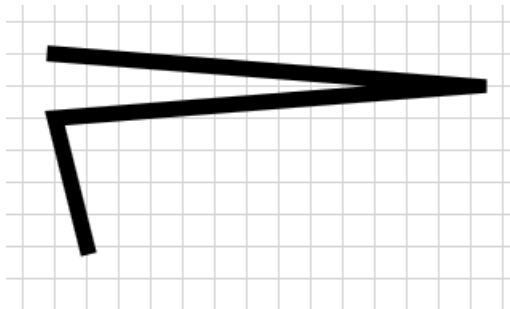


Fig. 11.26: Pen miter join with miter limit 2.0



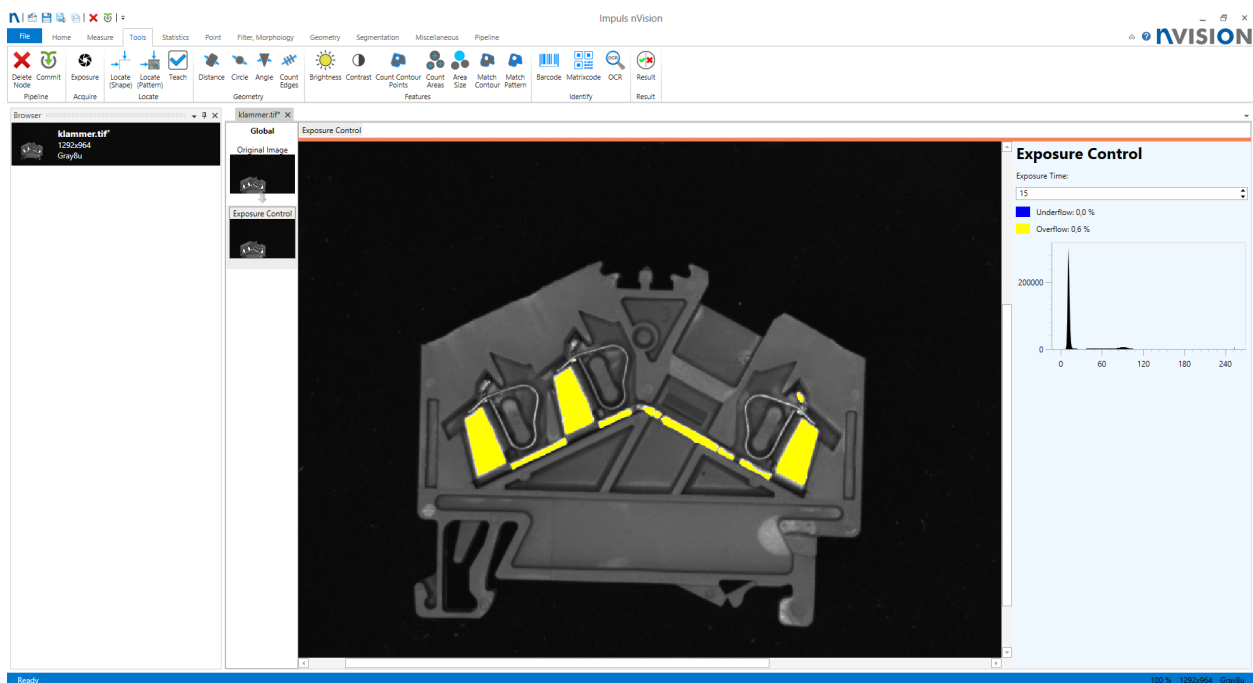
Fig. 11.27: Pen miter join with miter limit 3.0

CHAPTER 12

User Interface Creation

nVision can create and display HMIs (human machine interfaces or custom user interfaces). This facility is used within the **nVision Designer** at various places, and it can also be exploited by users.

The HMI system is fully integrated in the pipeline programming system. Here is an example of an HMI in **nVision**:

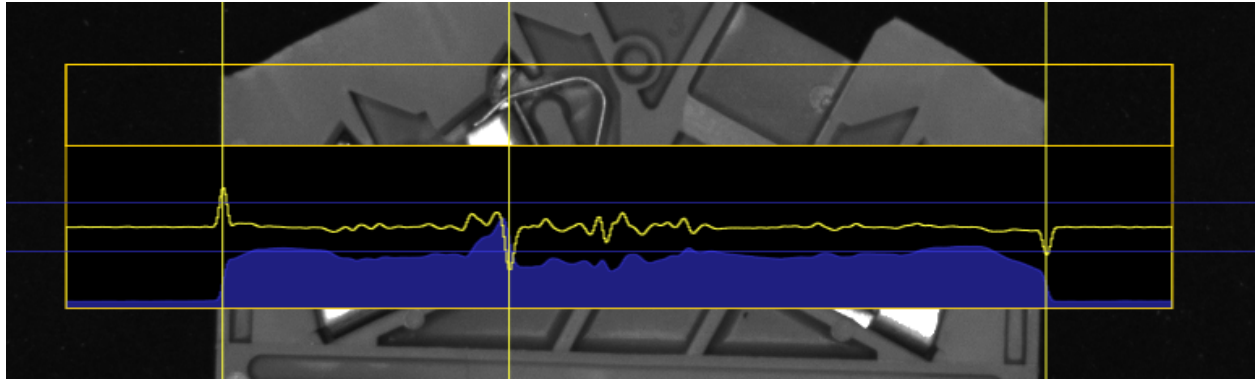


In the workspace area you see a picture at the left and a dialog area at the right. The picture occupies 3/4th and the dialog occupies 1/4th of the available workspace width. This HMI example is taken from the *Exposure* tool.

The HMI system is split into imaging centric widgets (used to display images, histograms and other imaging data) and in dialog centric controls (used for standard purposes like entering texts and numbers, making selections, pressing buttons). Both parts are integrated with each other and integrated with the pipeline.

The imaging centric widgets support display of imaging related things, with an additional set of interactive graphical tools. They support layering with arbitrary depth and structure and allow you to create complex displays that may be used for visualization purposes or for editing purposes.

The edge inspector is an example of a tool with a complex graphical widget display.



12.1 Controls

The **nVision** HMI system has a set of controls that can be grouped into layout, input and display purposes.

12.1.1 Window

The root of the HMI is the window.

The **Window** node creates the connection to the **nVision** workspace area.

At the bottom of the **Window** node is a pin that allows it to connect one child to the window. This child can take all the space in the window. Usually, this will be a control from the layout group of controls.

12.1.2 Layout Controls

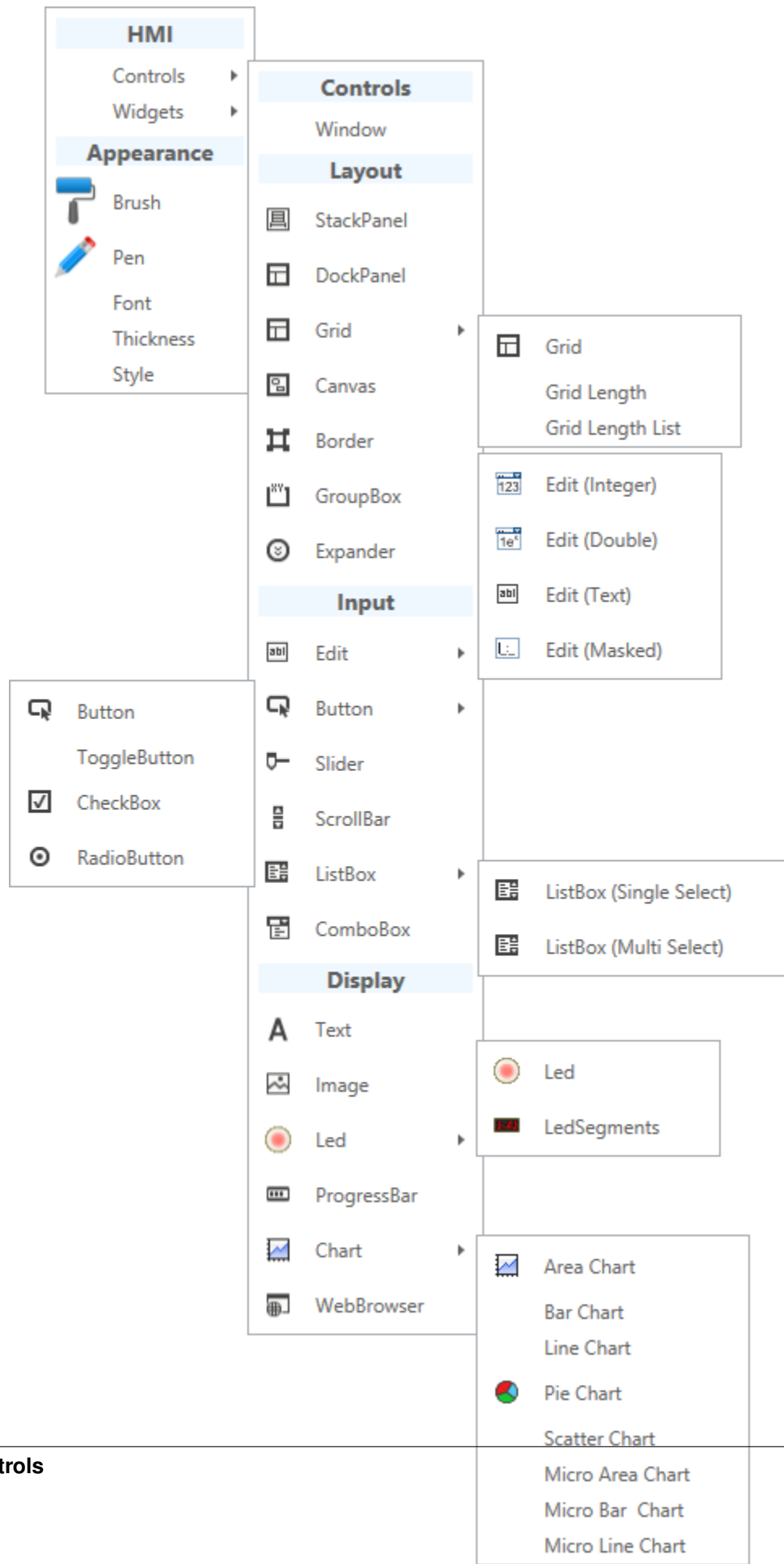
nVision has layout controls that adapt to the size of the available space (like the **StackPanel**, **DockPanel** and **Grid**), as well as the **Canvas**, which places controls at specific pixel positions.

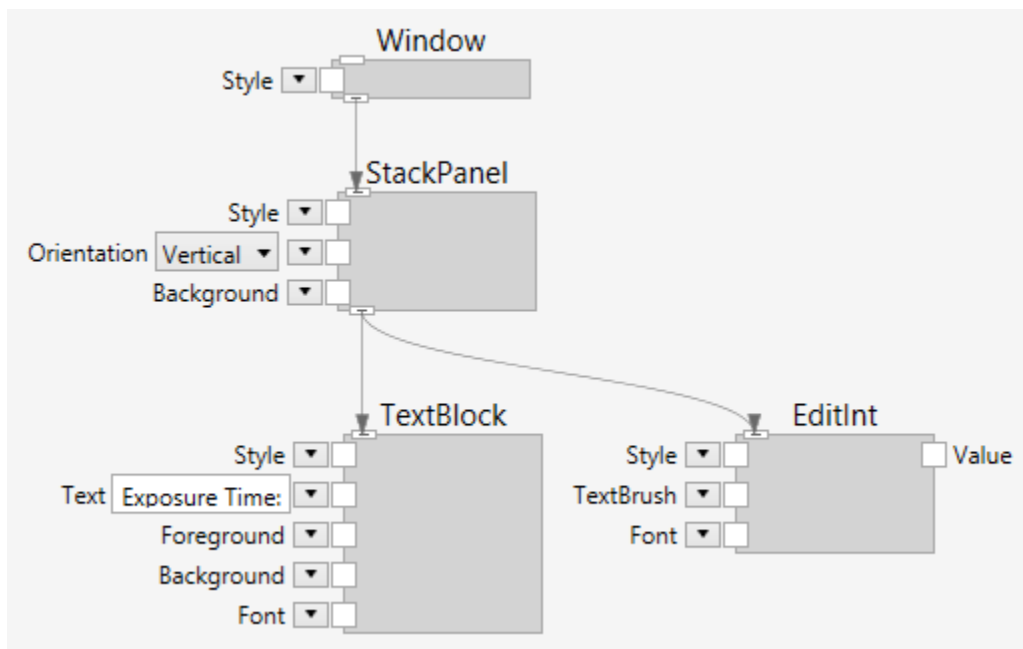
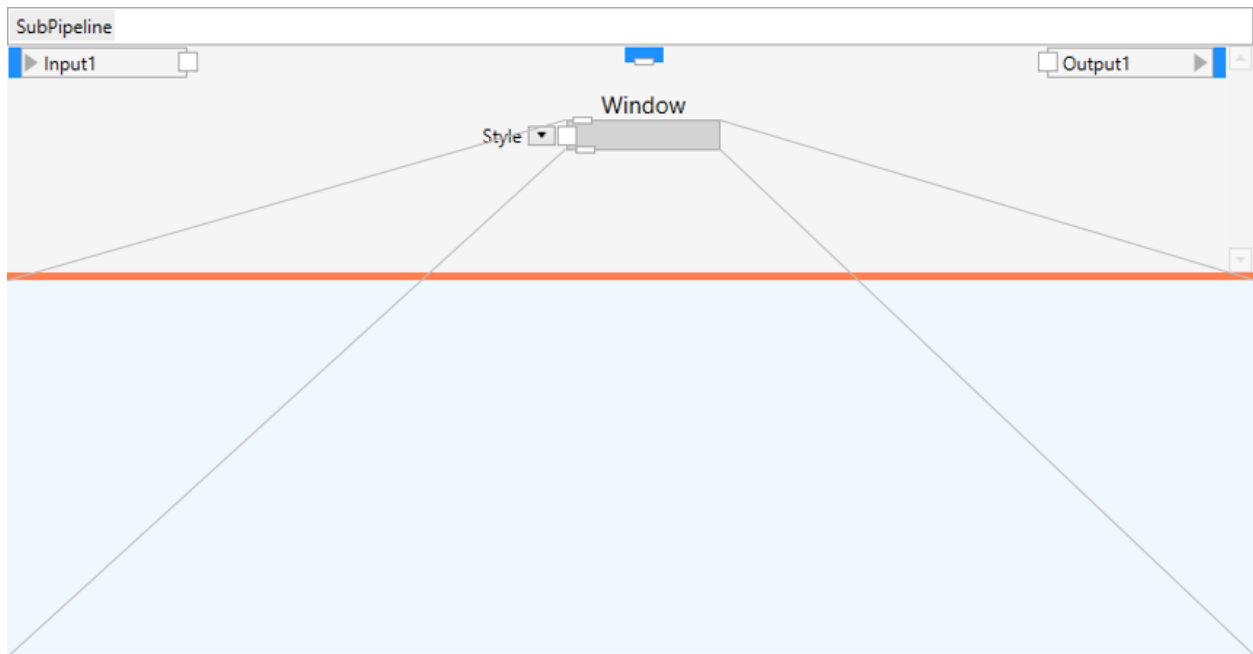
In addition, there are the **Border**, **GroupBox** and **Expander** controls, which group child controls (but are no layout controls in the strict sense).

StackPanel

The **StackPanel** stacks its child controls in the vertical or horizontal direction.

Here is an example that stacks a text label and an edit control vertically. This definition creates the following user interface:





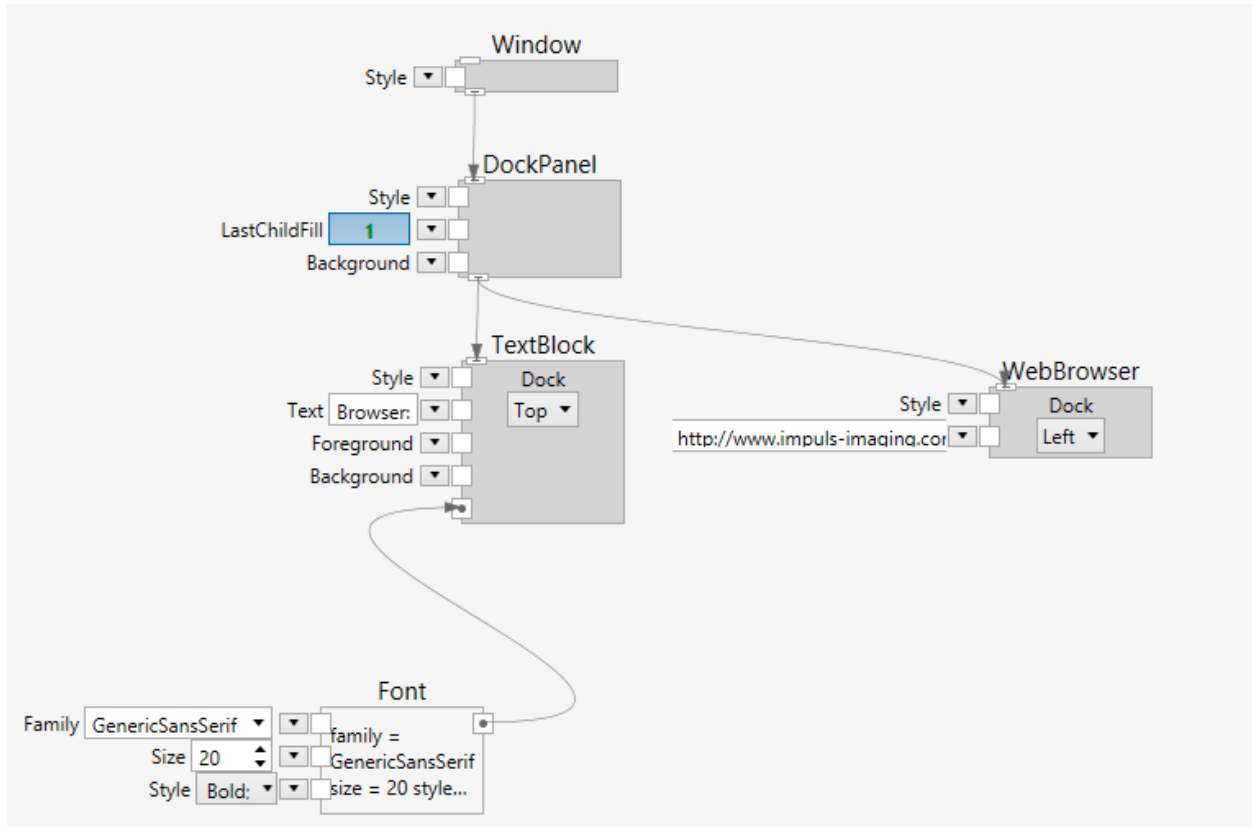
Exposure Time:

0

DockPanel

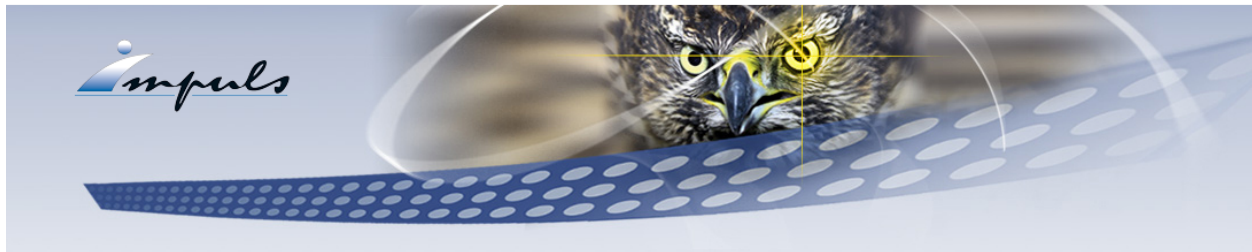
The **DockPanel** docks its child controls to one of the four sides *Left*, *Top*, *Right* or *Bottom*, where the respective child takes at much space as it needs.

Here is an example that docks a text label to the top and a webbrowser to the left, where the webbrowser fills the available space fully. This definition



creates the following user interface:

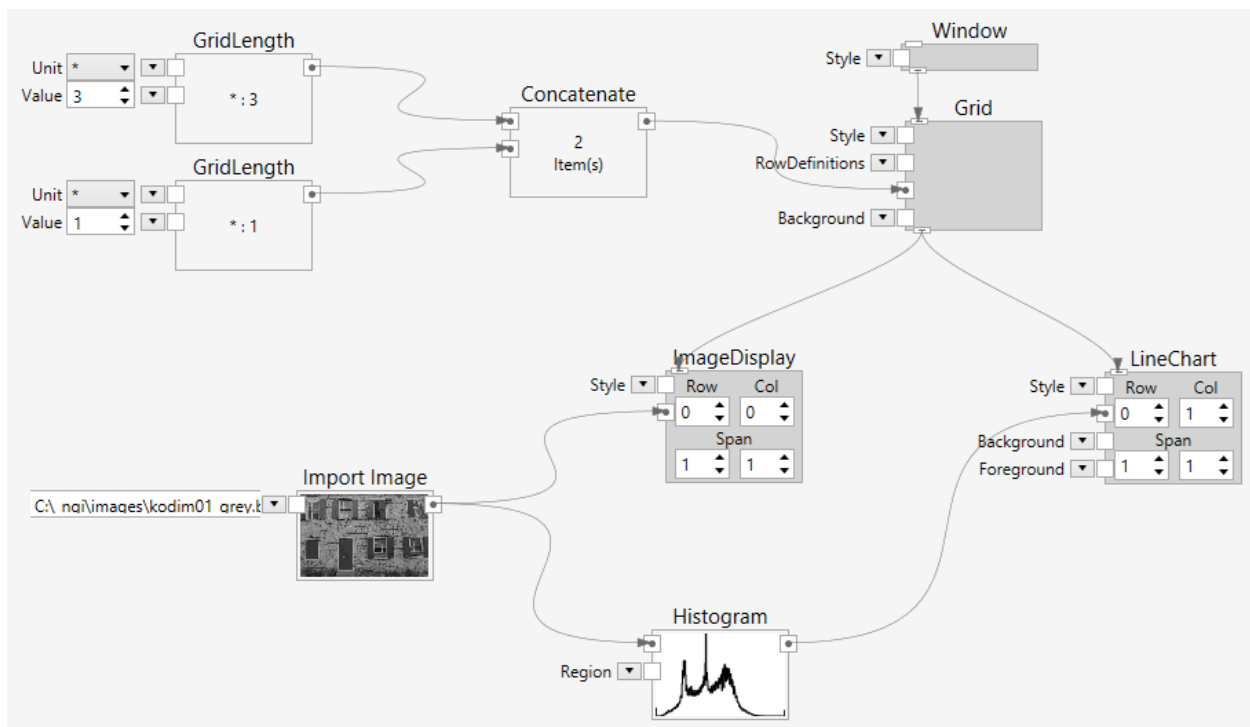
Browser:



Grid

The **Grid** allows you to split the available area into a rectangular grid with m rows and n columns. The row and column definitions are build with a list of **GridLength** nodes. The **GridLength** node specifies the size (absolute or relative) of the specific row or column.

Here is an example that splits the area into two columns and one row, where the left column has 3x size and the right column has 1x size. An image is displayed in the left column and its histogram is displayed in the right column. This definition:



creates the following user interface:

12.1.3 Canvas

The **Canvas** allows you to put controls at specific positions on the canvas.

Here is an example that puts four images on a canvas. This definition:

creates the following user interface:

Border

The **Border** puts a border around its child.

Here is an example that puts a border around an image. This definition:

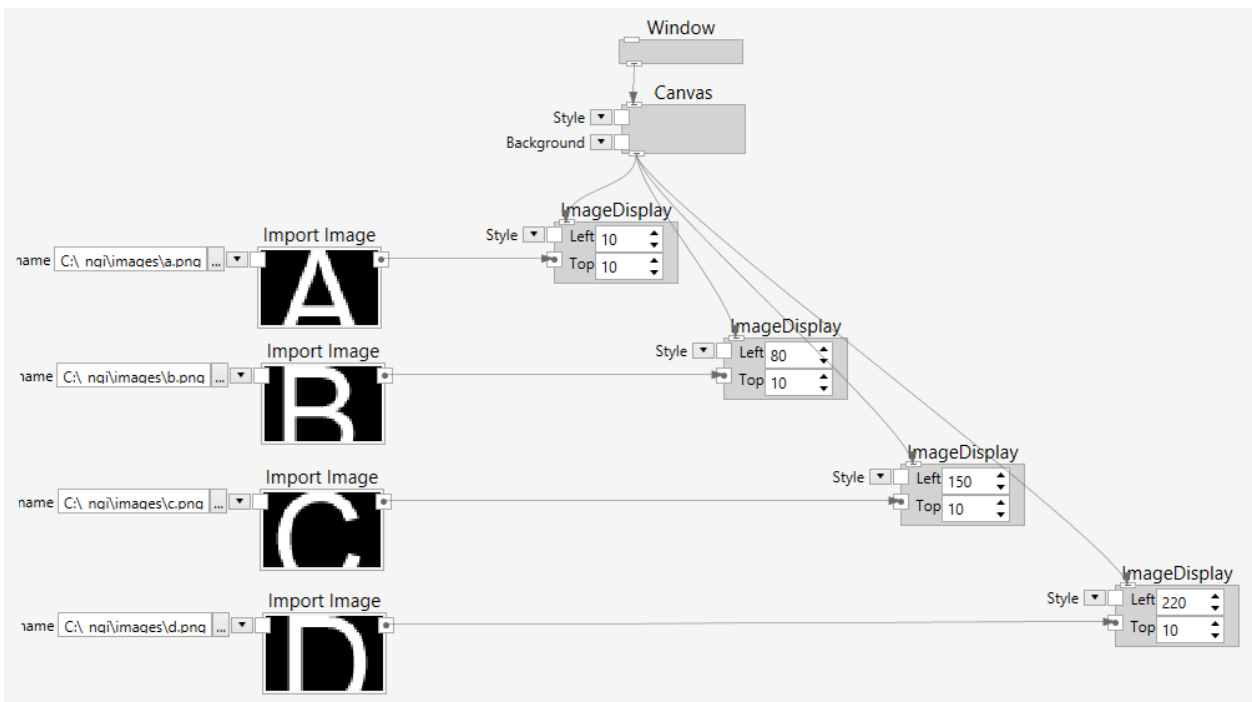
creates the following user interface:

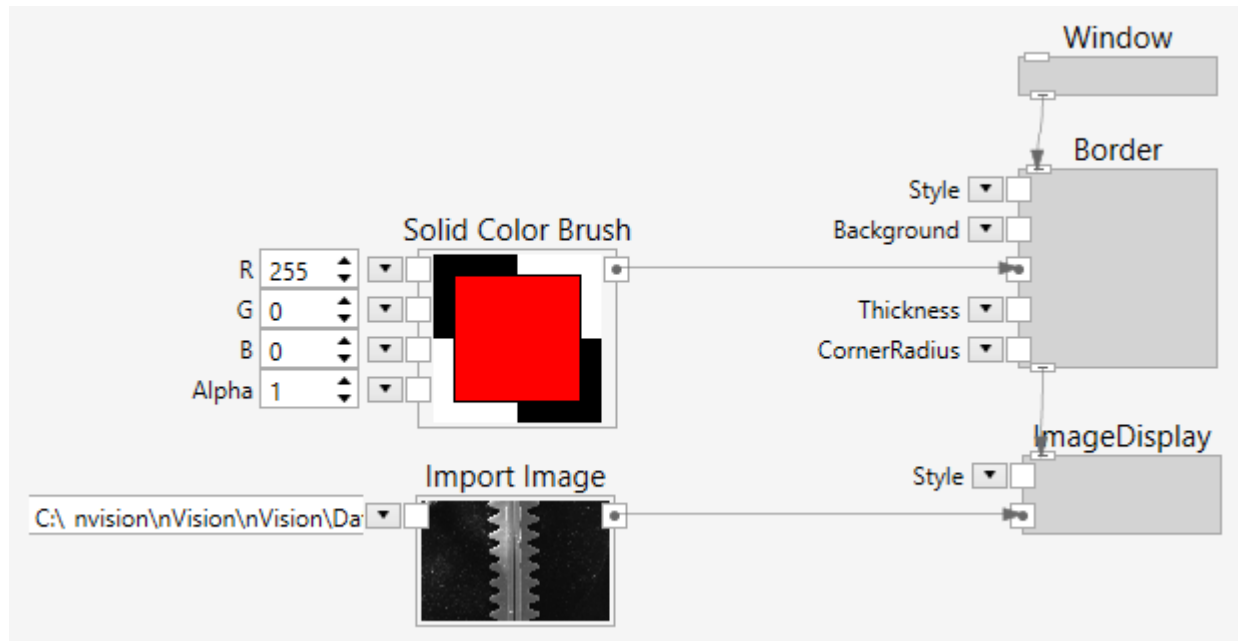
GroupBox

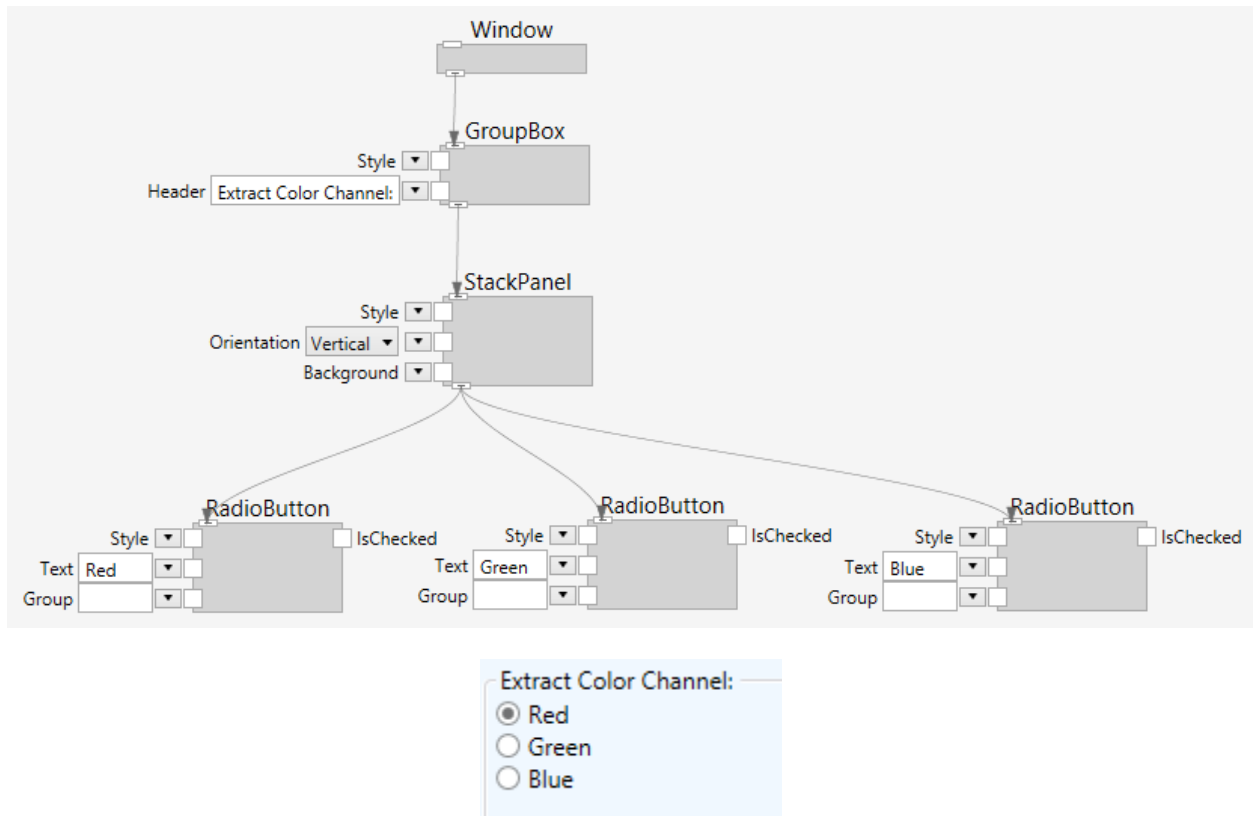
The **GroupBox** puts a group box around its child.

Here is an example that puts a groupbox around three radio buttons. This definition:

creates the following user interface:







Expander

The **Expander** puts an expandable and collapsible area around its child.

Here is an example that puts an expander around a profile. This definition:

creates the following user interface:

The profile display can be hidden or shown by clicking the little arrow button at the top-right corner of the expander.

12.1.4 Input Controls

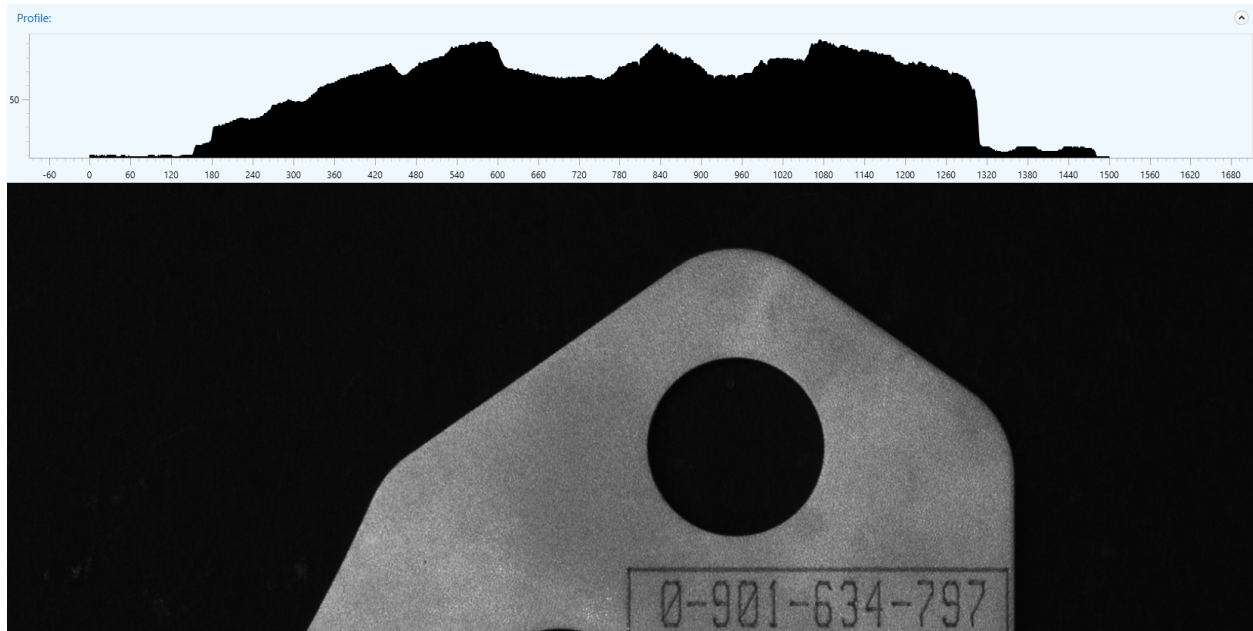
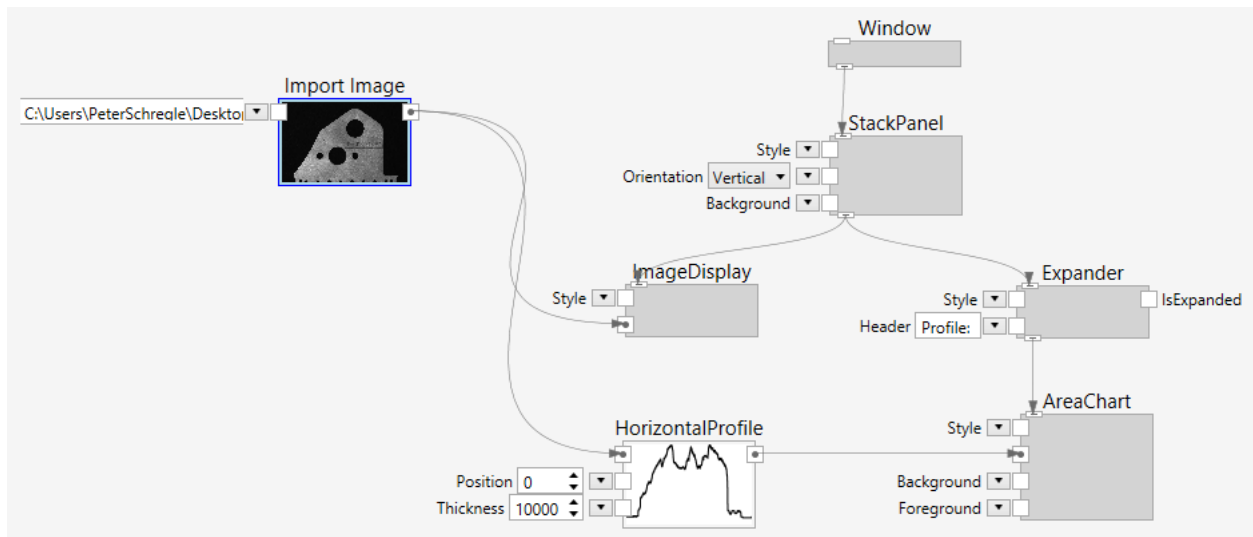
Input controls are used to get input data from a user. This can be a text, a number, or anything else. Various controls are available for this purpose.

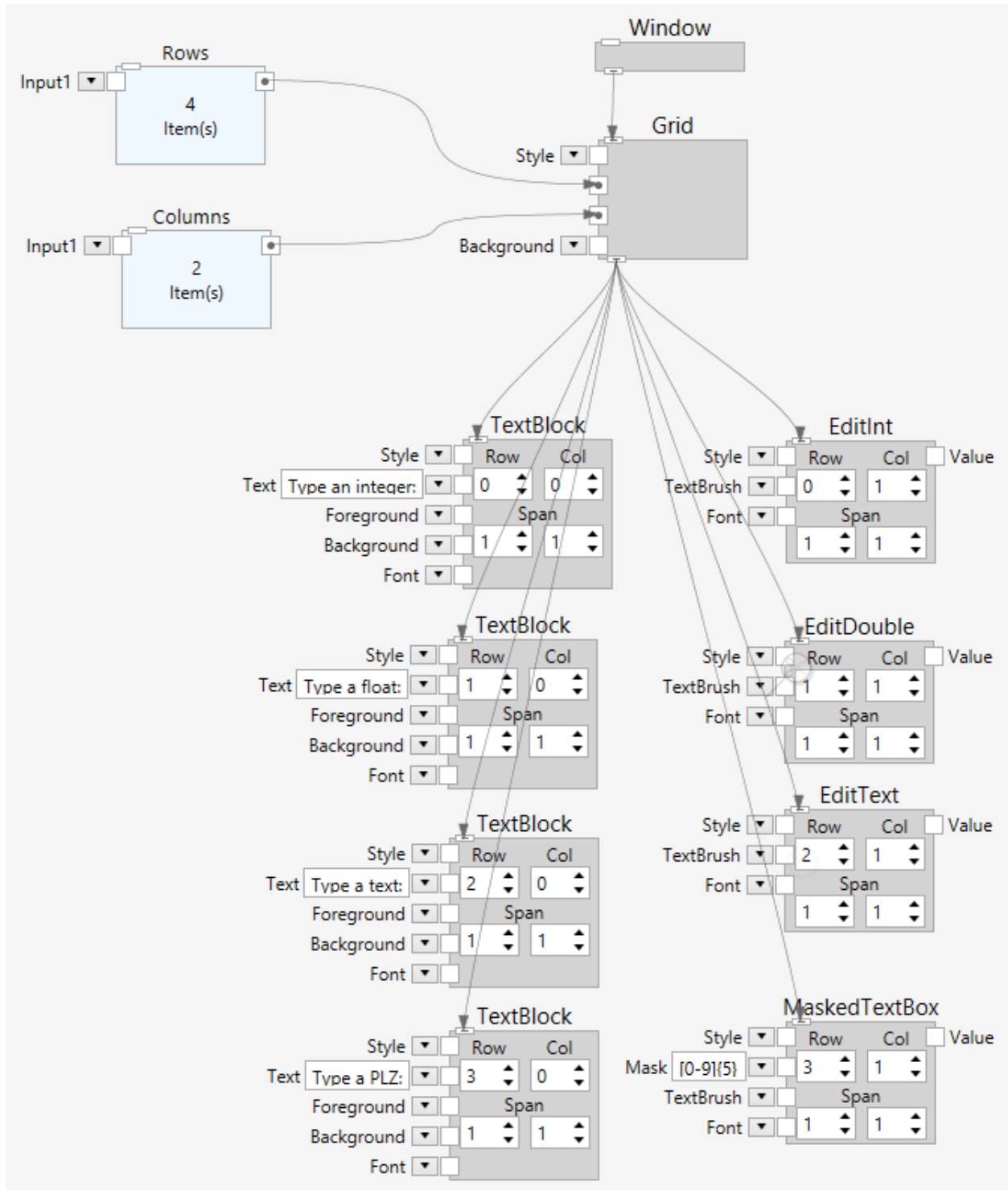
Edit

Edit controls provide a means for the user to type data. The HMI system provides edit controls for integer numbers, for floating point numbers, for arbitrary texts and for masked texts (masks provide a means to determine the *shape* or *pattern* of the text that can be entered).

Here is an example that shows the various edit controls. This definition:

creates the following user interface:





Type an integer:	2
Type a float:	0,5
Type a text:	Hello
Type a PLZ:	86842

Button

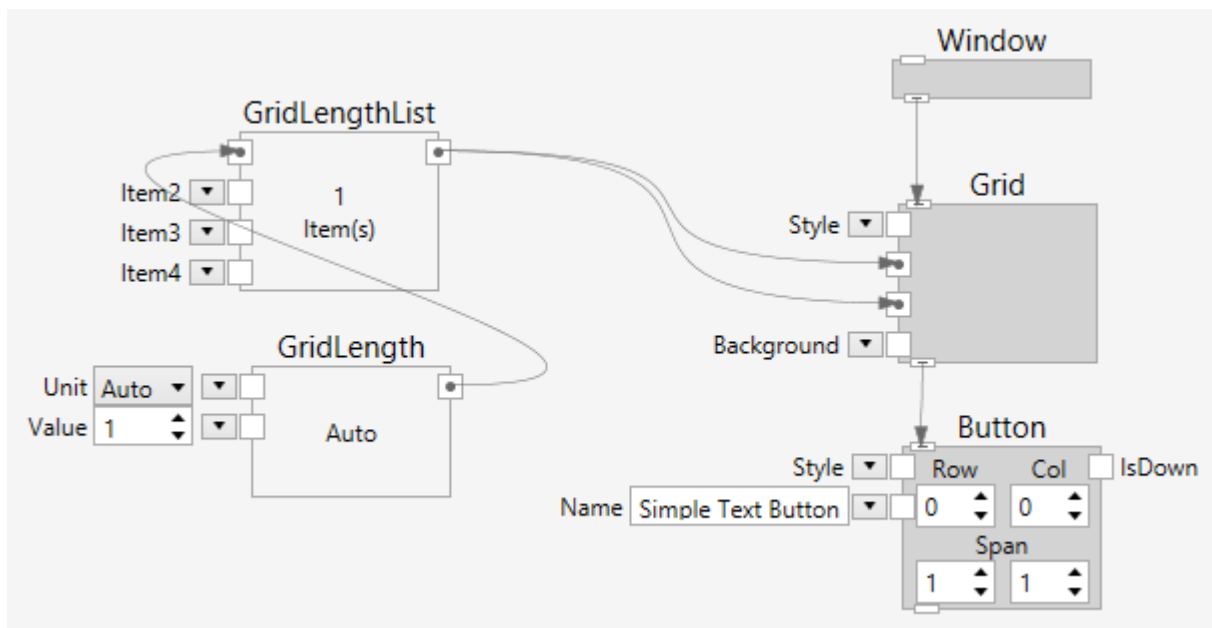
Buttons are used to switch or visualize state, as well as to trigger actions. The **Button** is a push-button, which is released when you no longer push it. The **ToggleButton** has two states, up or down. The **CheckBox** also has two states, checked or not, but it visually looks different. The **RadioButton** is used in groups, where only one of the buttons can be checked.

All buttons have one child, where you can connect an arbitrary complex HMI control hierarchy. All buttons also have a *Name* input, which you can alternatively use to quickly create normal text buttons.

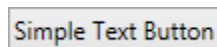
Button

A **Button** is down as long as it is held down with the mouse. When it is no longer held down, it releases itself to the up state.

Here is an example that shows a simple text button. This definition:



creates the following user interface:



And this more complicated example shows a button with an image and a text. This definition:

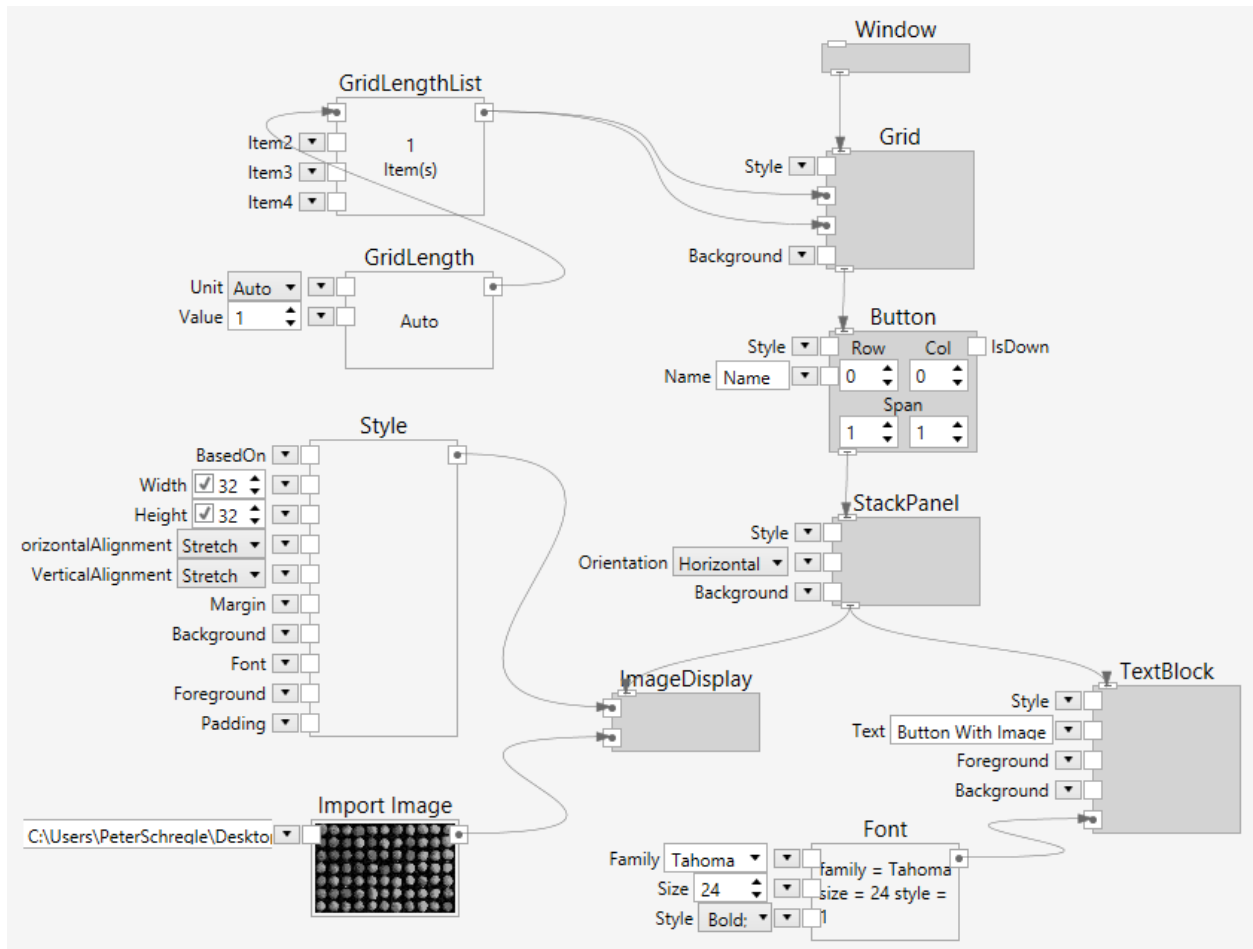
creates the following user interface:

ToggleButton

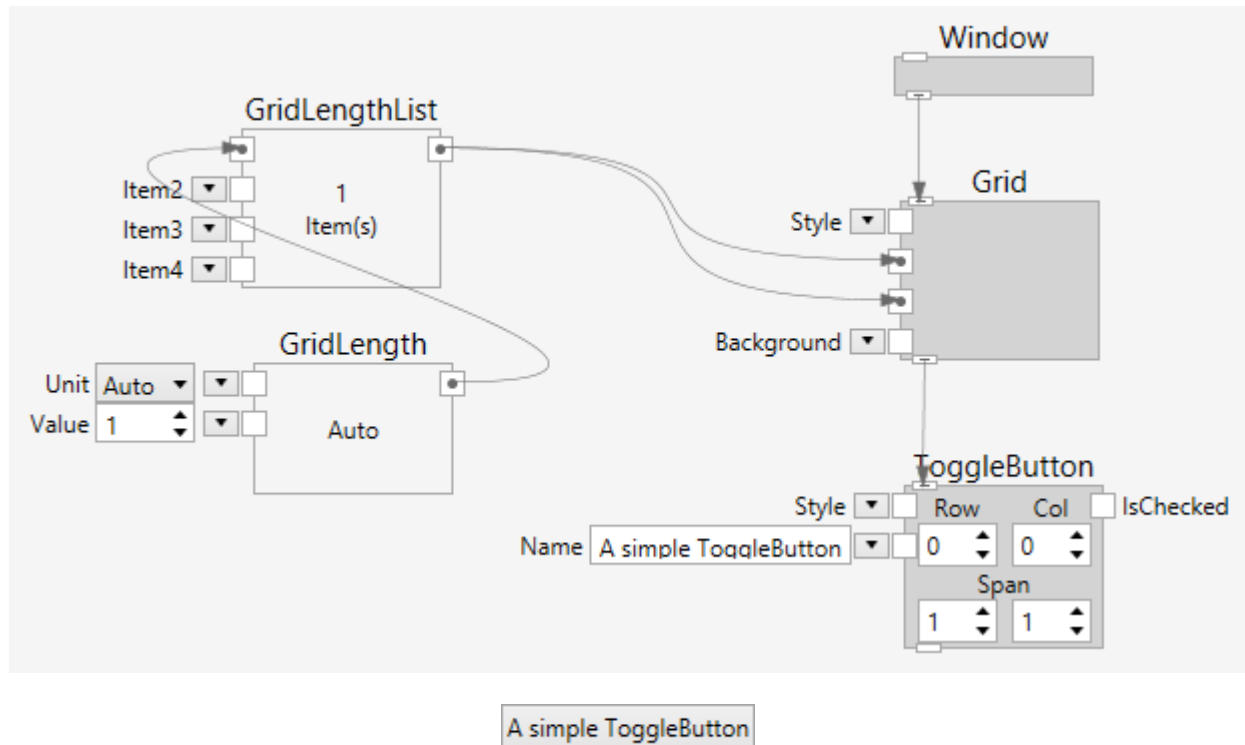
A **ToggleButton** toggles between the up and down states when it is clicked.

Here is an example that shows a simple toggle button. This definition:

creates the following user interface:



 **Button With Image**



CheckBox

A **CheckBox** toggles between the checked and unchecked states when it is clicked.

Here is an example that shows a simple **CheckBox**. This definition:

creates the following user interface:

RadioButton

A **RadioButton** is used in a group with other **RadioButtons**. Only one of them is checked at a time. If another **RadioButton** in the same group is checked, the previously checked one is unchecked.

Here is an example that shows a simple **RadioButton**. This definition:

creates the following user interface:

Slider

A **Slider** can be used to move to or visualize a position.

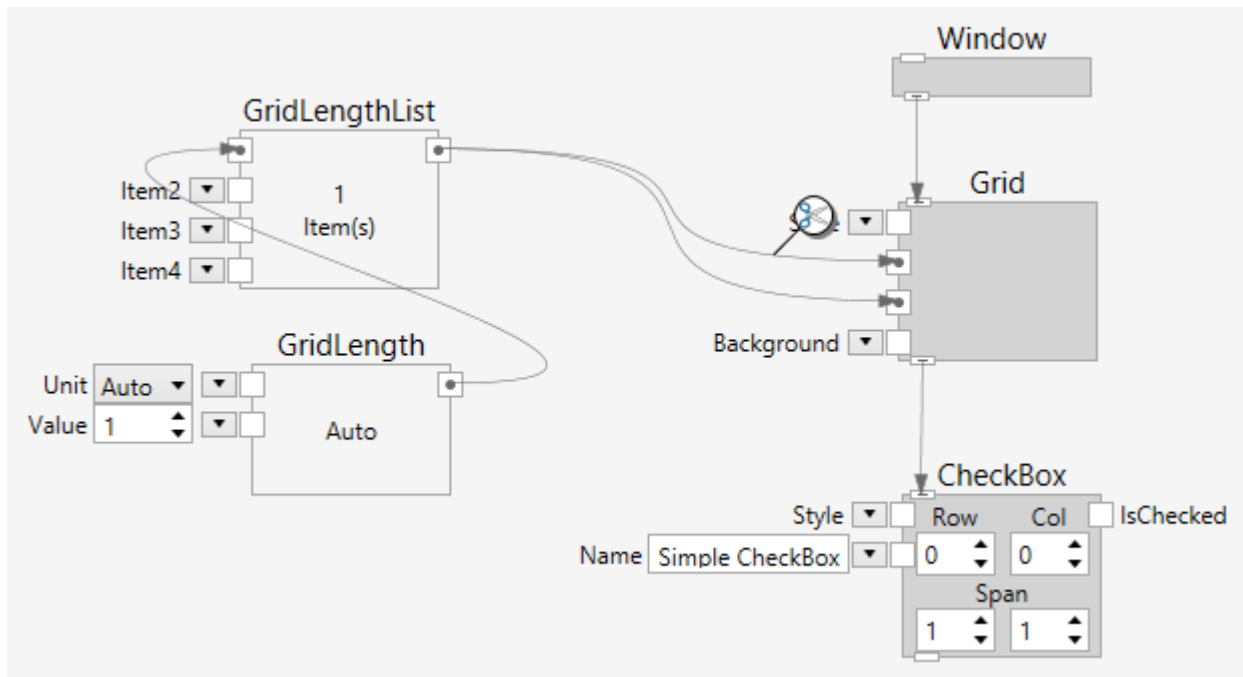
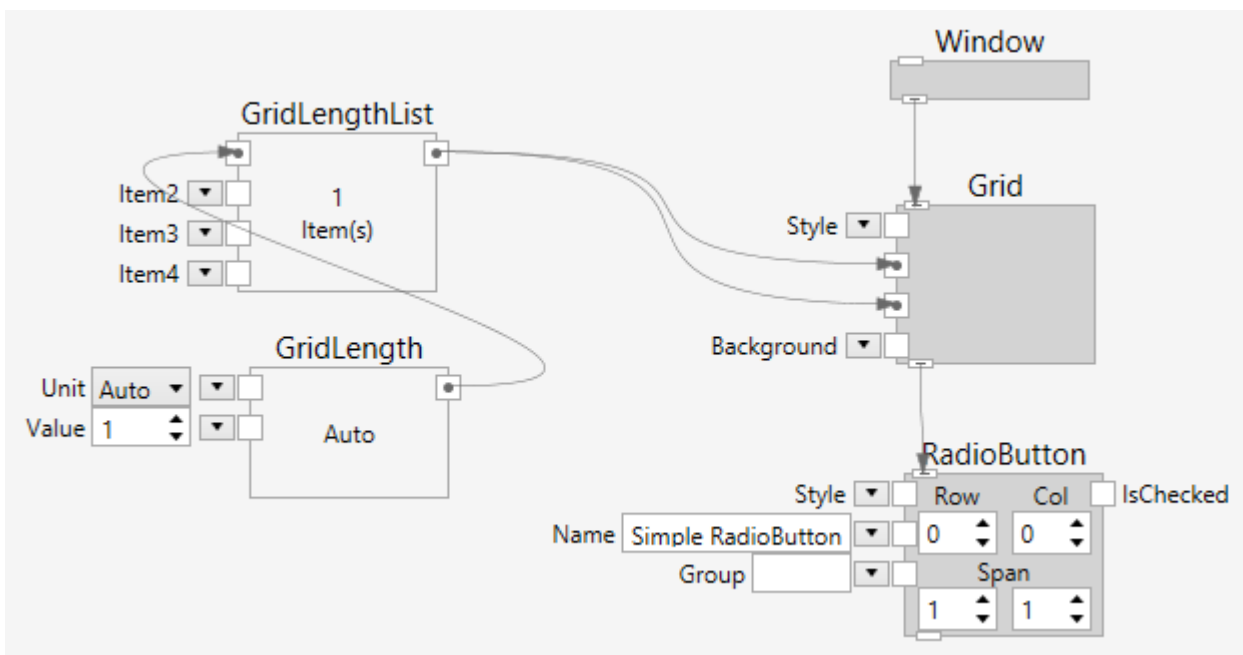
Here is an example that shows a **Slider**. This definition:

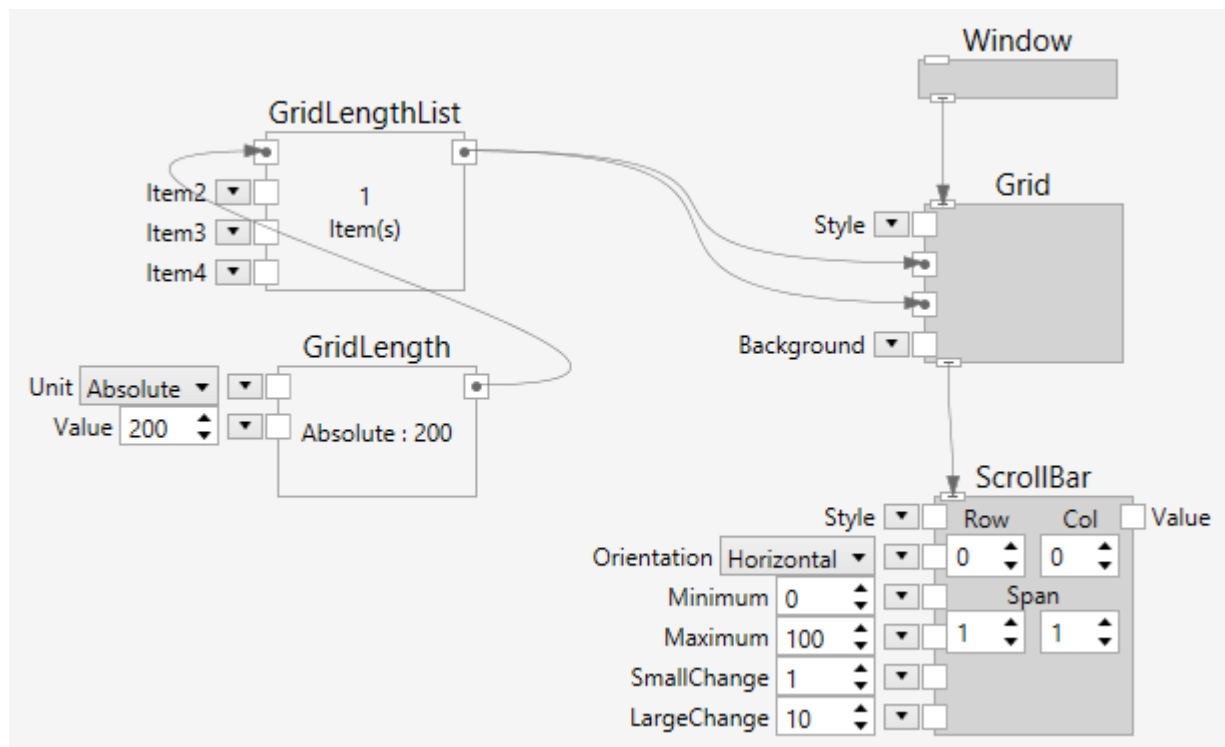
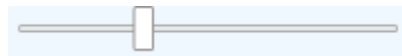
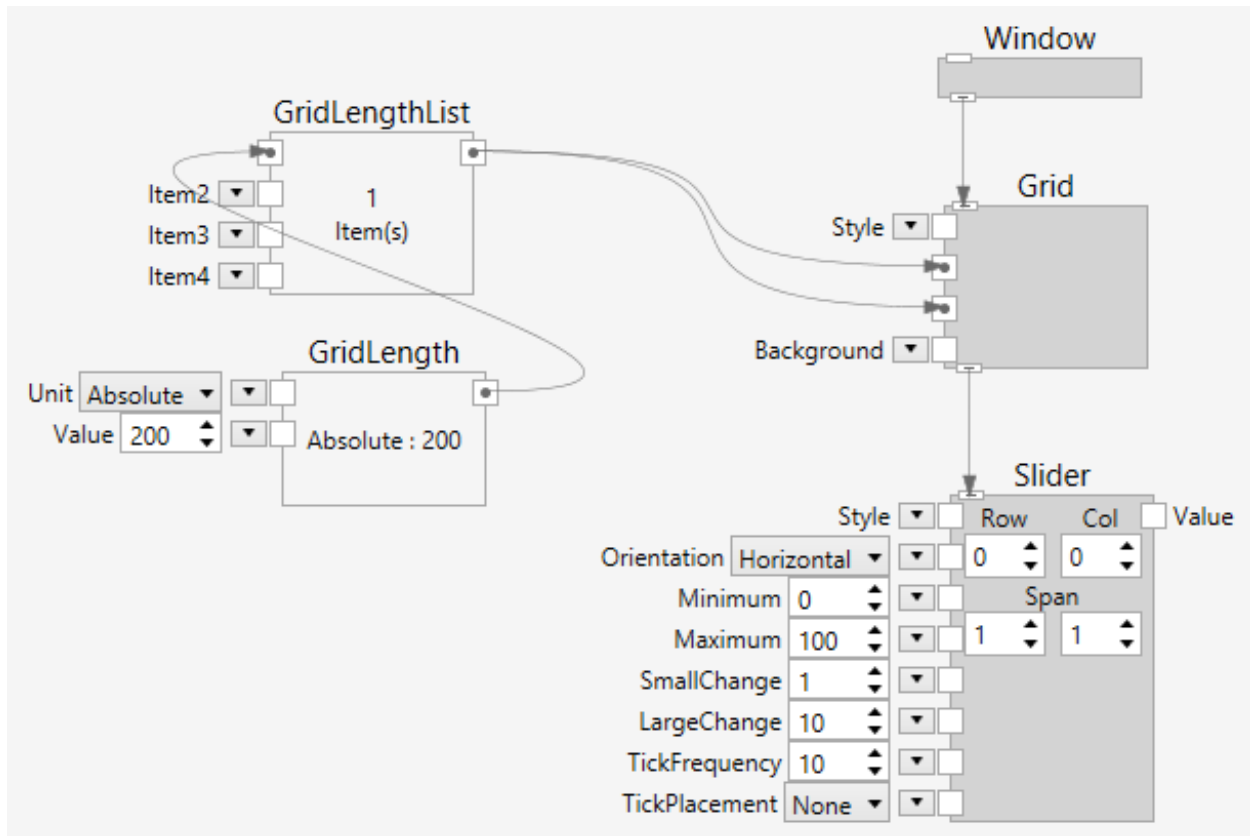
creates the following user interface:

ScrollBar

A **ScrollBar** can be used to scroll to or visualize a position.

Here is an example that shows a **ScrollBar**. This definition:


☐ Simple CheckBox

☐ Simple RadioButton



creates the following user interface:



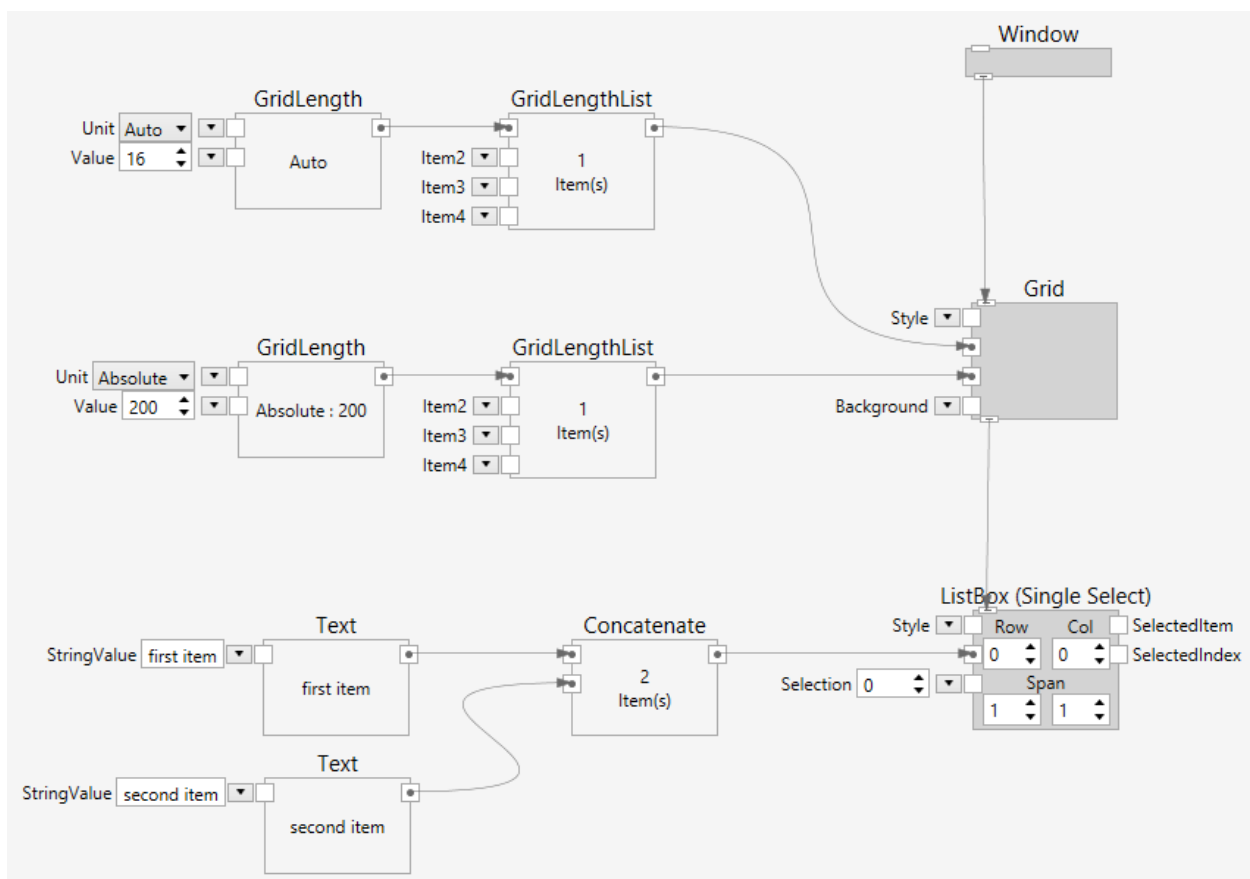
Listbox

The HMI system has several ways to present lists of strings.

ListBox (Single Select)

A single select **ListBox** can be used to select one from a number of entries.

Here is an example that shows a **ListBox**. This definition:



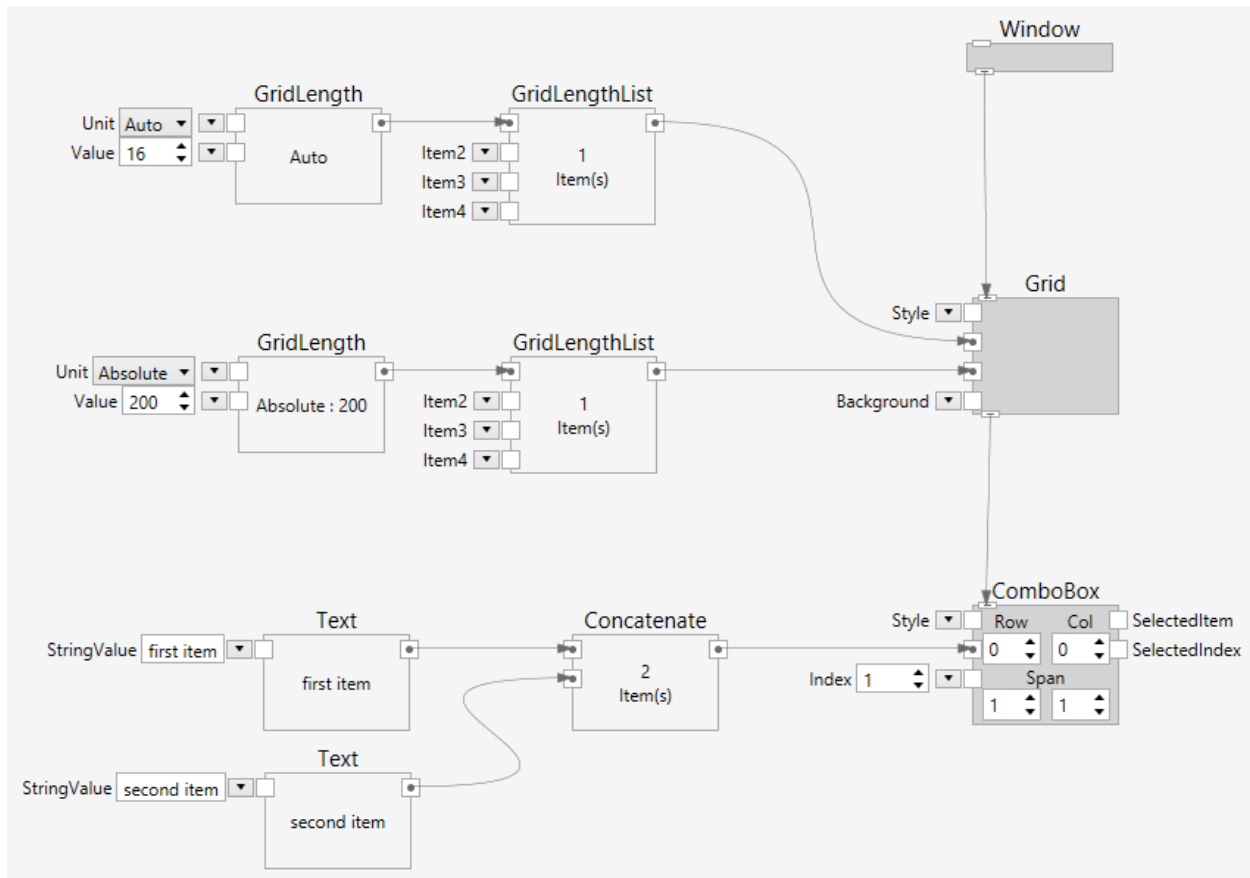
creates the following user interface:



ListBox (Multi Select)

A multi select **ListBox** can be used to select one from a number of entries.

Here is an example that shows a **ListBox**. This definition:



creates the following user interface:



ComboBox

A **ComboBox** can be used to select one from a number of entries. It has a collapsed and expanded state.

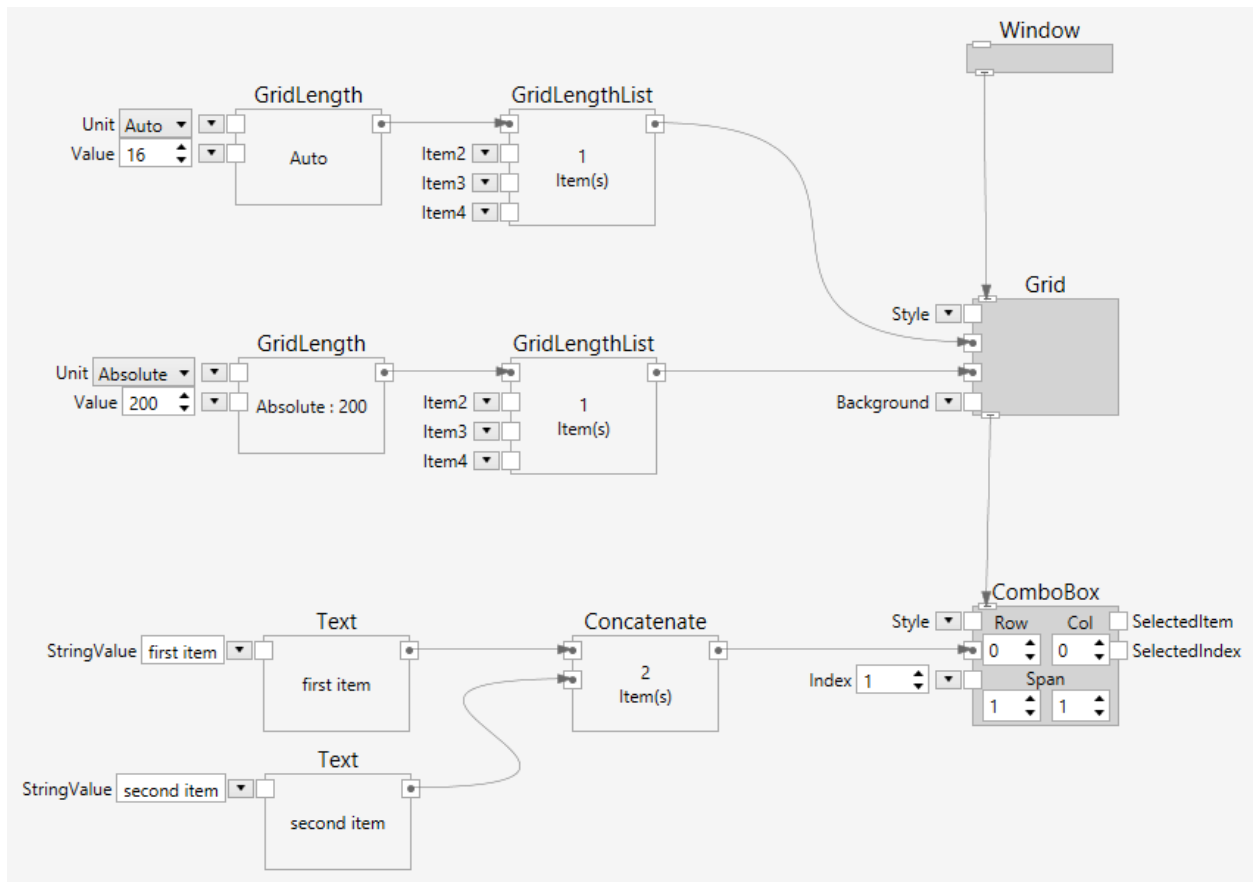
Here is an example that shows a **ComboBox**. This definition:

creates the following user interface:

and in expanded state:

12.1.5 Display Controls

Display controls are used to display information for a user. The information can be texts, images, charts or even webpages.



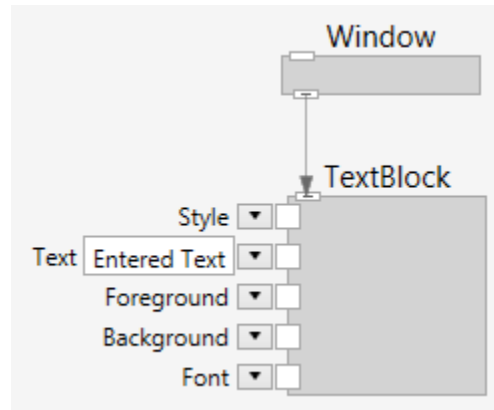
second item ▼

second item ▼
first item
second item

Text

A **TextBlock** can be used to display text.

Here is an example that shows a **TextBlock**. This definition:



creates the following user interface:

A

Image

An **Image** can be used to display an image.

Here is an example that shows an **Image**. This definition:



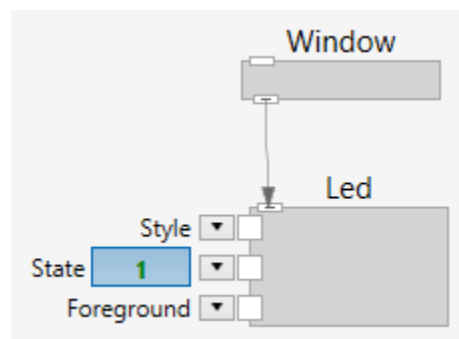
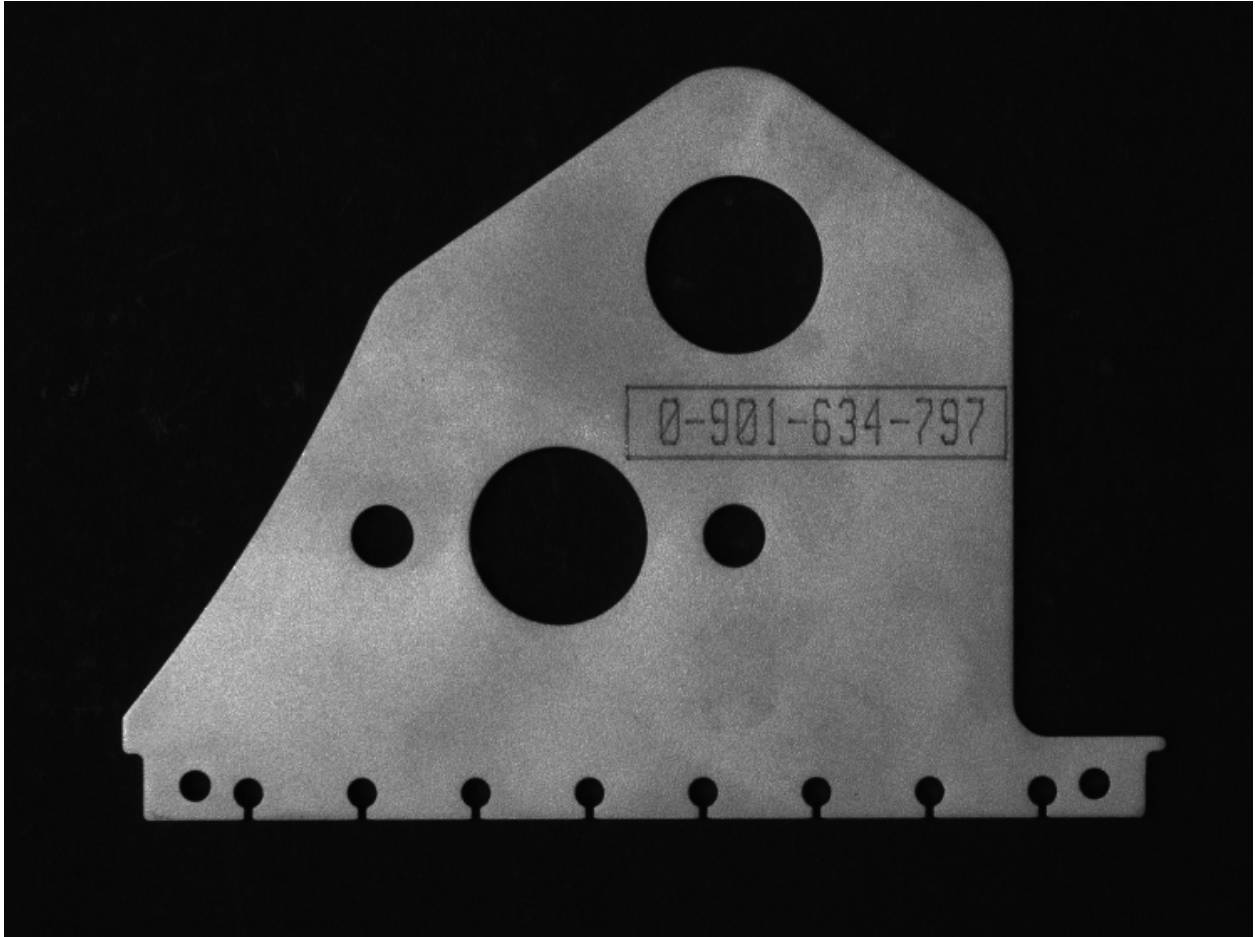
creates the following user interface:

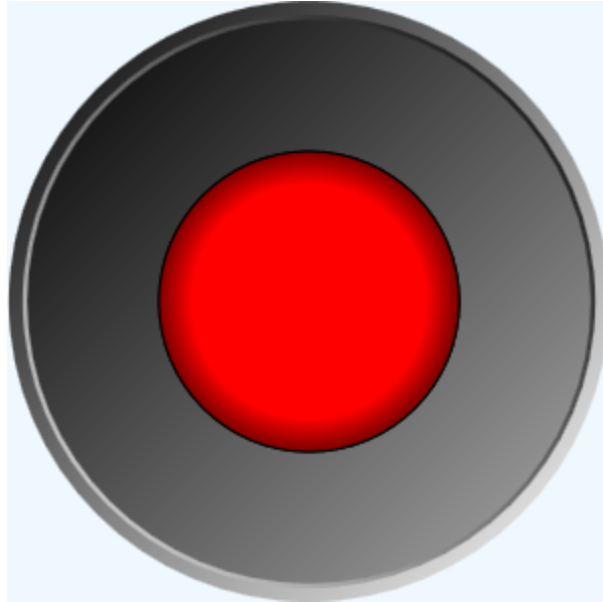
Led

An **Led** can be used to display a state.

Here is an example that shows an **Led**. This definition:

creates the following user interface:

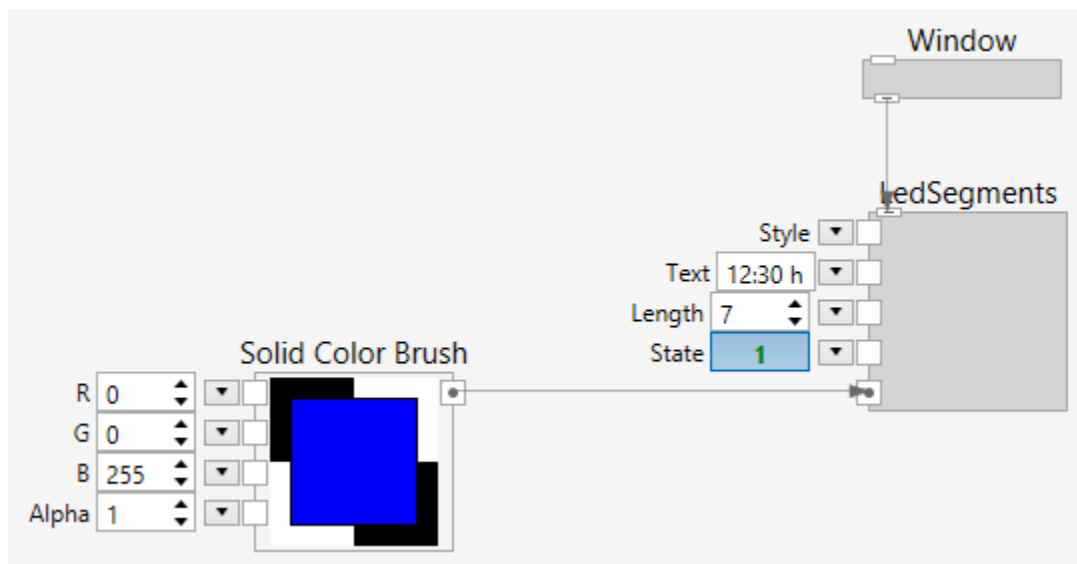




LedSegments

An **LedSegments** can be used to display text in a segmented led-style display.

Here is an example that shows an **LedSegments**. This definition:



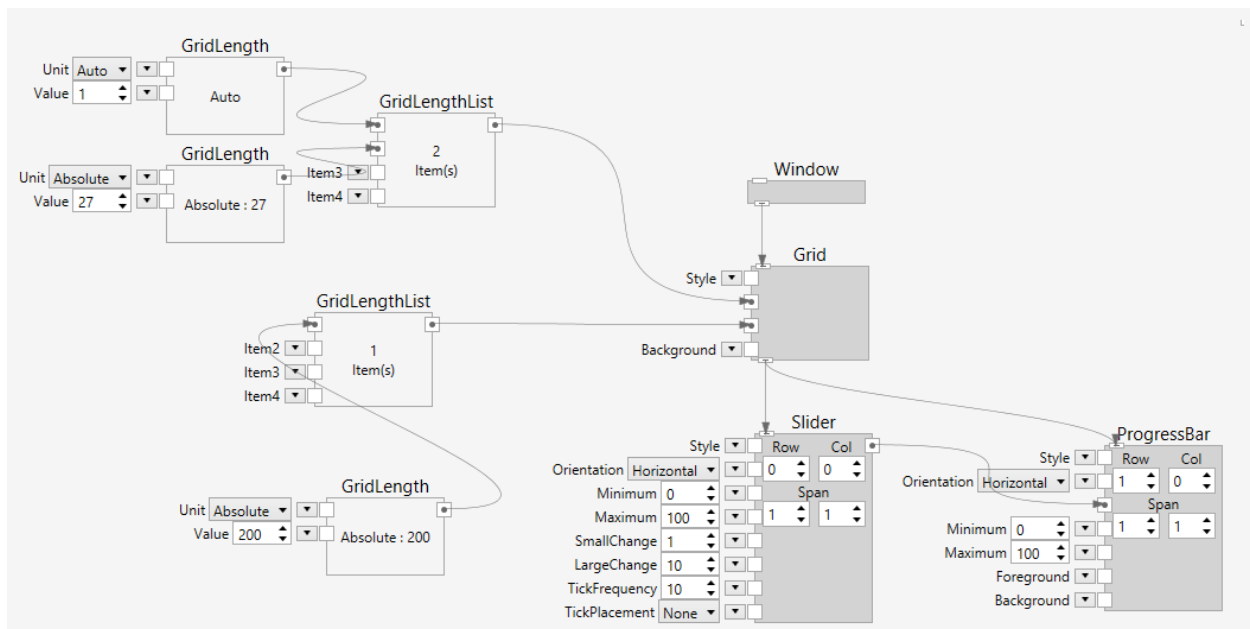
creates the following user interface:

ProgressBar

A **ProgressBar** can be used to visualize a value.

Here is an example that shows a **ProgressBar**, coupled to a **Slider**. This definition:

creates the following user interface:



Chart

The HMI system can display various types of charts.

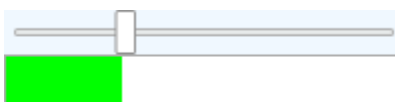
WebBrowser

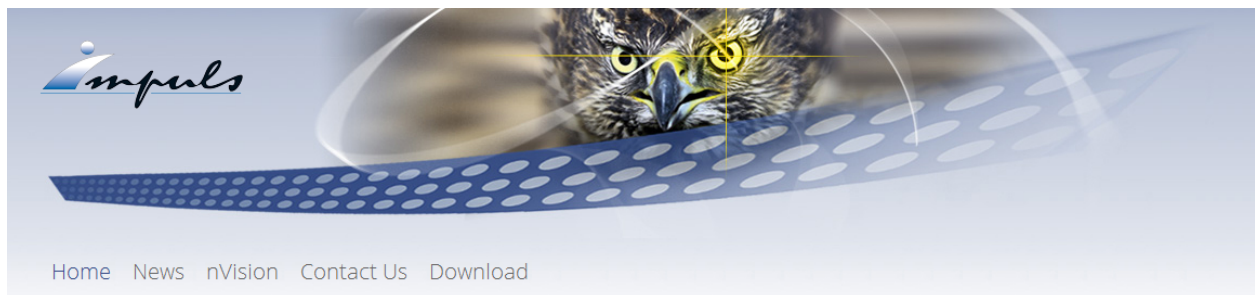
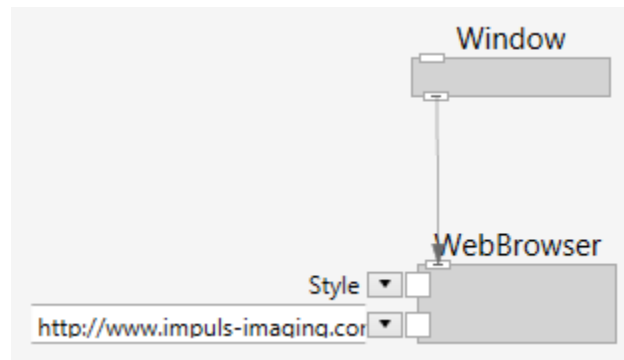
A **WebBrowser** can be used to display a webpage.

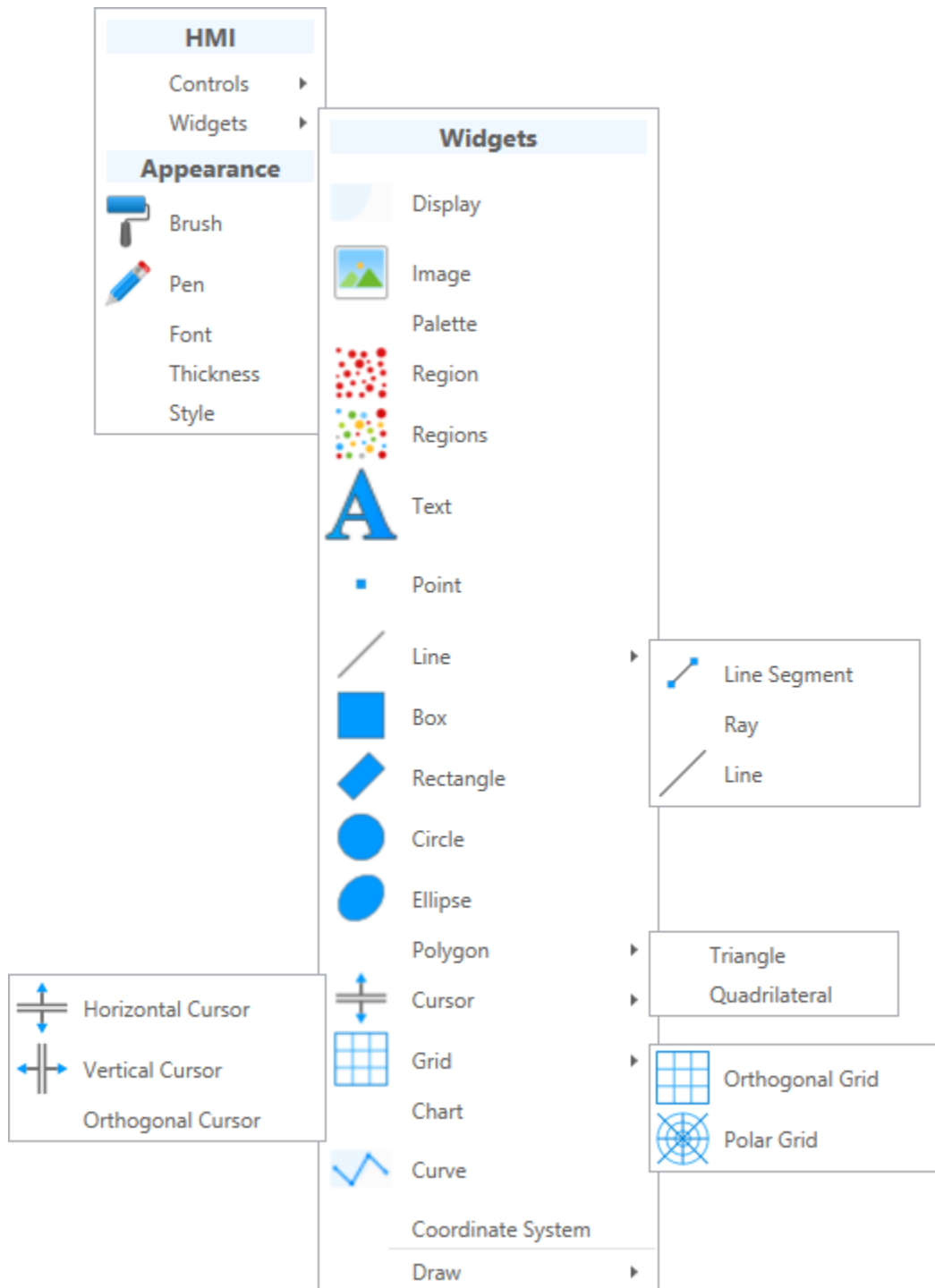
Here is an example that shows a **WebBrowser**. This definition:
creates the following user interface:

12.1.6 Styling

12.2 Widgets







13.1 Pipelines

Pipelines are the programs that **nVision** executes. Data flows through a pipeline.

Nodes are the building blocks of a pipeline. Nodes have **Pins**, which are used to connect nodes via **Connections**.

Here is a simple pipeline consisting of two nodes:



A pipeline builds a directed graph. Data flows in the direction that is visualized with the arrows of the connections. Cycles in the graph are forbidden.

In the above example, the *Text* node has a *StringValue* **Input Pin**, where the text *Hello World!* was typed. The **Output Pin** of the node was connected to the **Input Pin** of the *Uppercase* node. Each node displays the value of its (first) **Output Pin**.

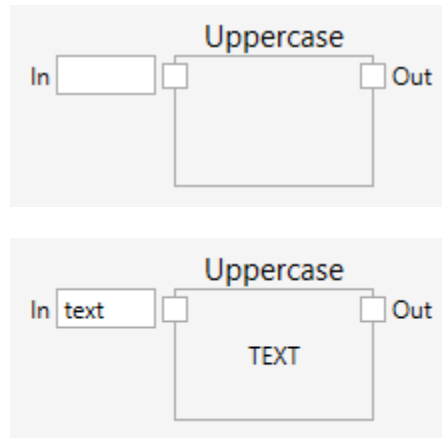
Pipelines are executed in a live fashion. At any time any value changes, the pipeline re-runs and produces new outputs. Pipeline execution is demand driven and lazy. Nodes execute only if and when needed.

13.2 Nodes

Nodes are the building blocks in a pipeline.

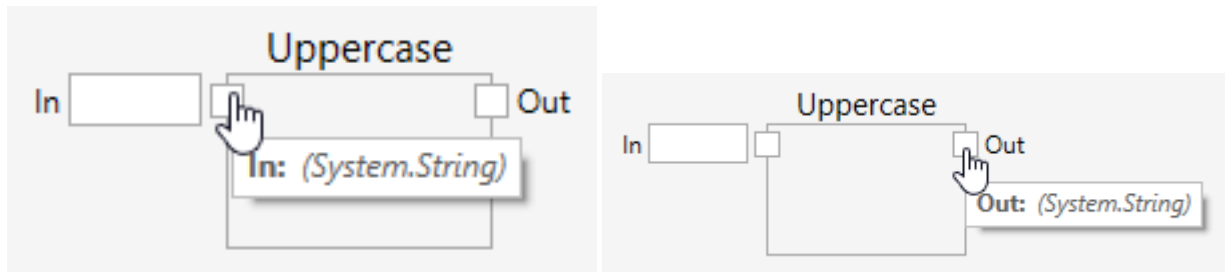
A node may have **Input Pins** and/or **Output Pins**. The **Pins** are used to connect **Nodes** with **Connections**.

This is a **Node** with one **Input Pin** labelled *In* and one **Output Pin** labelled *Out*. It takes its input text and uppercases each letter and provides the result on its output.

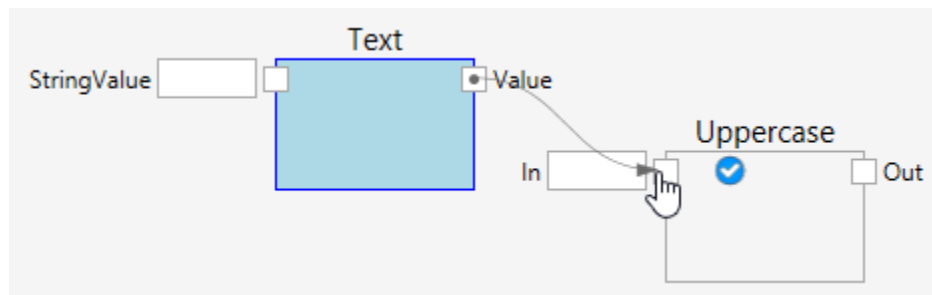


13.3 Pins

Nodes may have **Input Pins** and/or **Output Pins**. **Pins** are typed.



Connections can be made only between **Pins** with compatible **Types**.



Pins are compatible, if their **Types** are equal or if the **Type** flowing on the connection can be converted automatically.

Pins with incompatible **Types** cannot be connected.

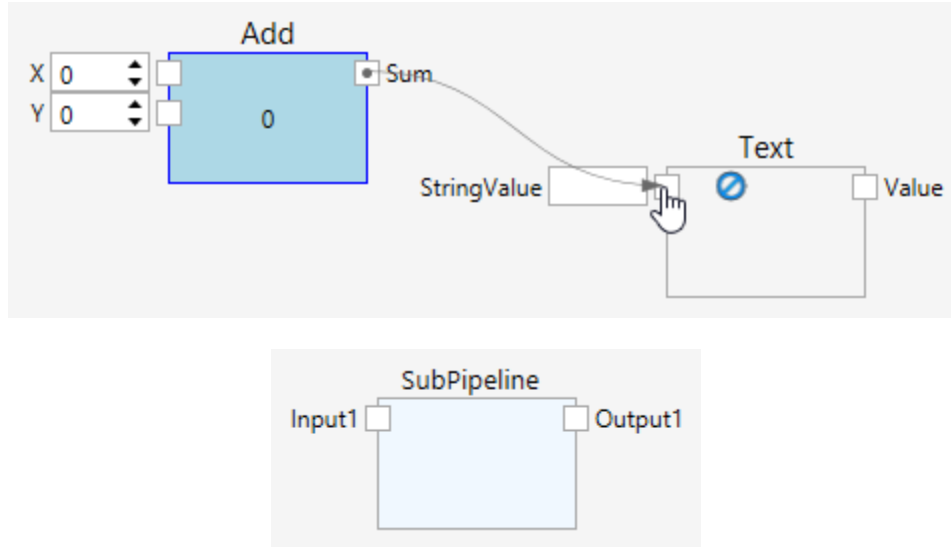
13.4 Subpipelines

Subpipelines can be used to group a set of connected **Nodes** into one **Node**.

As any other **Node**, a **Subpipeline** has **Input Pins** and/or **Output Pins**.

By default, a **Subpipeline Node** has one **Input Pin** and one **Output Pin**.

The name of the **Subpipeline** can be edited by clicking it:



Inside the **Subpipeline** (double-click the **Node** to get inside) it looks like this:

Additional **Pins** of a **Subpipeline Node** can be defined with the *Add Input Pin* and *Add Output Pin* commands on the context menu. The name of the **Pin** can be edited by clicking on it. A **Pin** can also be removed with the *Delete Pin* command (right click somewhere on the **Pin** definition - but not on the text).

The **Type** of a **Subpipeline Pin** is undefined as long as it is not connected. As soon as the **Pin** is connected (either from the inside, or from the outside) its **Type** is set (determined by the **Type** of the **Pin** on the other end of the connection).

13.5 Transform

The **Transform Node** takes a list of items and transforms it into a list of other items.

Transform goes through the input list item by item, and transforms each into a new item. New items are then combined into the output list.

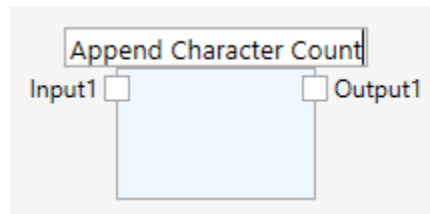
At the outside, transform takes the appearance of a **Subpipeline**. As any other **Node**, **Transform** has **Input Pins** and/or **Output Pins**.

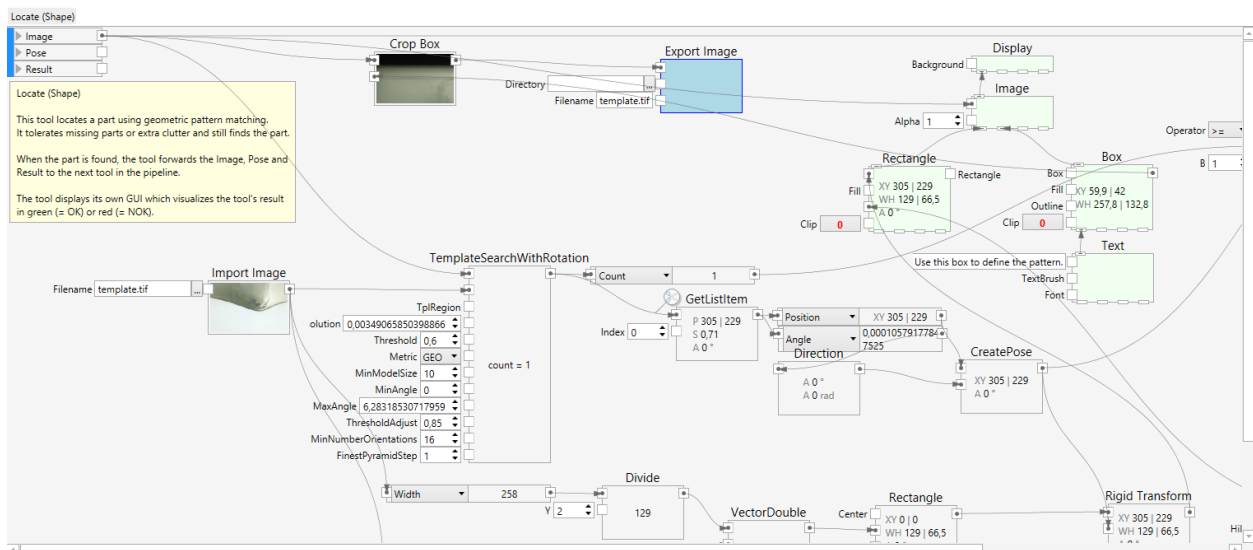
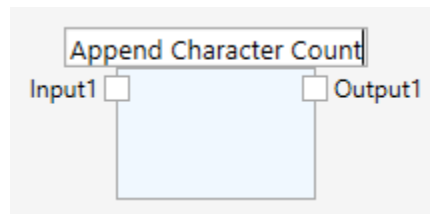
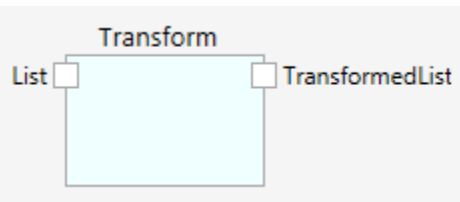
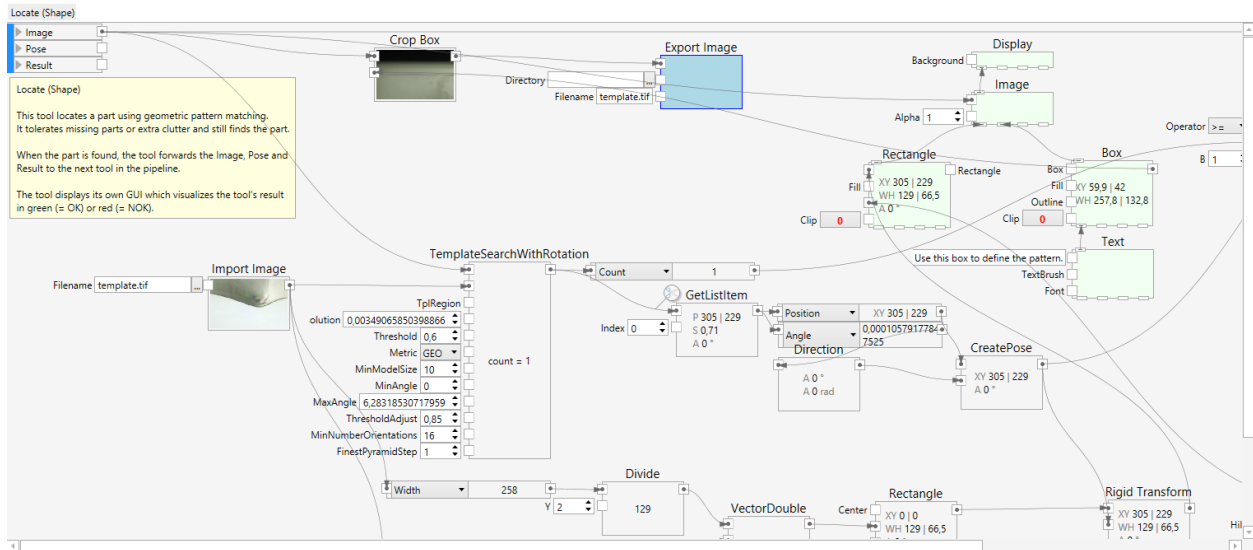
By default, the transform **Node** has one **Input Pin** and one **Output Pin**.

The name of the **Transform Node** can be edited by clicking it:

Inside the **Transform** (double-click the **Node** to get inside) it looks like this:

At the inside, **Transform** takes the appearance of a **Subpipeline**, but with predefined **Input pins** *Item* and *ItemIndex*. These predefined **Pins** carry one item of the input list and its index in the input list, respectively.





Additional **Pins** of a **Transform Node** can be defined with the *Add Input Pin* and *Add Output Pin* commands on the context menu. The name of the **Pin** can be edited by clicking on it. A **Pin** can also be removed with the *Delete Pin* command (right click somewhere on the **Pin** definition - but not on the text).

The **Type** of a **Transform Pin** is undefined as long as it is not connected. As soon as the **Pin** is connected (either from the inside, or from the outside) its **Type** is set (determined by the **Type** of the **Pin** on the other end of the connection).

13.6 Types

Data flowing in pipelines is typed.

Primitive Types are:

type	type-name	description
boolean	<code>System.Boolean</code>	
integer (32 bit, signed)	<code>System.Int32</code>	
text	<code>System.String</code>	textual data, such as <i>Text</i> .

Input Pins and **Output Pins** are typed. With most pins this type is fixed, but on some pins the type may change while the pipeline is edited.

Pins that may change are the input and output pins of subpipelines and their list processing cousins. These pins take over a specific type from the first connection that is made to them, either from the outside or from the inside.

13.7 Conversions

Connections between two **Pins** provide the means for data to flow. Since the **Pins** define the **Type** of data, connections can only be made, if the **Types** at both ends of the connection are the same, or if the data can be converted.

Widening conversions are made automatically, if they are available. Behind the scenes, a conversion **Node** is created, but this conversion **Node** is hidden.

Converting an integer number to a floating point number is widening (any integer number can be represented as a floating point number as well) and will be made automatically.

Converting a floating point number to an integer number is narrowing (the fractional part of a floating point number needs to be cut away to convert) and is not made automatically.

14.1 Digital IO

nVision has support for digital IO. This is helpful to control machinery from image processing applications or to react upon input signals sent from machinery. Digital IO can either use digital IO modules or a PLC (programmable logic controller).

14.1.1 Advantech Adam-60xx Ethernet Modules

nVision supports various digital IO modules from Advantech (<http://www.advantech.com>).



The modules are connected via LAN or wireless LAN to a PC and support a variety of input/output lines. The modules may have additional features, such as counters, etc. but these are not supported.

module

bus

features

ADAM-6050

LAN

12 digital inputs, 6 digital outputs

ADAM-6050W

WLAN

12 digital inputs, 6 digital outputs

ADAM-6051

LAN

12 digital inputs, 2 digital outputs

ADAM-6051W

WLAN

12 digital inputs, 2 digital outputs

ADAM-6052

LAN

8 digital inputs, 8 digital outputs

ADAM-6060

LAN

6 digital inputs, 6 digital outputs

ADAM-6060W

WLAN

6 digital inputs, 6 digital outputs

ADAM-6066

LAN

6 digital outputs

The following nodes are available:

node

purpose

Adam6000 Module

An Advantech Adam-6000 module abstraction, which is needed to initialize the module and get information from it.

Read Inputs

This is needed to read from digital inputs.

Write Outputs

This needed to write to digital outputs.

The **Adam6000 Module** node establishes a connection to an Adam device. The IP Address and Port inputs are used to identify the module. The Type input is used to select the connected type of the module. The Module output can be fed into the **Read Inputs** or **Write Outputs** nodes.

The **Read Inputs** node reads the values from the electrical inputs. The DigIn output contains the state of the electrical inputs in the form of a binary integer.

The **Write Outputs** node writes the value from its DigOut input to its electrical outputs.

The Sync inputs of both the **Read Inputs** and **Write Outputs** nodes can be used to establish a defined order.

The **Read Inputs** node can be polled at regular intervals using the **nVision** timer.

14.1.2 Beckhoff TwinCAT3

nVision supports direct connection to a Beckhoff PLC. (<http://www.beckhoff.de>).

You need to install the Beckhoff TwinCAT3 software in order to use the modules from within **nVision**.

The following nodes are available:

node

purpose

Beckhoff PLC

Establishes the connection to the PLC.

Read Symbol

Reads from a symbol in the PLC.

Write Symbol

Writes to a symbol in the PLC.

The **Beckhoff PLC** node establishes a connection to a PLC over the ADS protocol. It needs an AMS NetId and a Port as input, because this is needed for the unique identification of ADS devices. The output of the **Beckhoff PLC** node is the PLC itself, which can be fed into the **Read Symbol** or **Write Symbol** nodes.

The **Read Symbol** node reads a symbol from the PLC. The symbol can be chosen from the list at the Symbol input port. This list contains all available symbols in the connected PLC. The **Read Symbol** node has two outputs. The output port Value gives the value of the accessed symbol. The output port Info gives information about the accessed symbol. These information can be accessed e.g. by the **Get Property** node.

The **Write Symbol** node writes a symbol to the PLC at the input port. The symbol can be chosen from the list at the Symbol input port. This list contains all available symbols in the connected PLC. The new value of the symbol can be set on the input port Value.

The Sync inputs of both the **Read Symbol** and **Write Symbol** nodes can be used to establish a defined order.

The **Read Symbol** node can be polled at regular intervals using the **nVision** timer.

14.1.3 Data Translation OpenLayers

nVision supports various digital IO modules from Data Translation (<http://www.datatranslation.com>).



The modules are connected via USB or PCI buses to a PC and support a variety of input/output lines. The modules may have additional features, such as counters, etc. but these are not supported.

module

bus

features

DT9817

USB

28 digital inputs/outputs

DT9817-H

USB

28 digital inputs/outputs

DT9817-R

USB

8 digital inputs, 8 digital outputs

DT9835

USB

64 digital inputs/outputs

DT335

PCI

32 digital inputs/outputs

DT351

PCI

8 digital inputs, 8 digital outputs

DT340

PCI

32 digital inputs, 8 digital outputs

You need to install the Data Translation OpenLayers software in order to use the modules from within **nVision**. We have used OpenLayers 7.5 to implement the support, so OpenLayers 7.5 or higher is needed.

The following nodes are available:

node

purpose

OpenLayers System

Provide information about the installed DT OpenLayers software and the connected devices.

OpenLayers Device

A DT OpenLayers module abstraction, which is needed to initialize the module and get information from it.

Read Inputs

This is needed to read from digital inputs.

Write Outputs

This needed to write to digital outputs.

The **OpenLayers System** node enumerates the OpenLayers devices connected to the PC. The Modules output delivers a list of connected devices. The Version output displays the version of the installed OpenLayers software.

The **OpenLayers Module** node establishes a connection to an OpenLayers device. The Name input is needed to identify the module. You can find out the name by inspecting the output of the **OpenLayers System** node. The Module output can be fed into the **Read Inputs** or **Write Outputs** nodes.

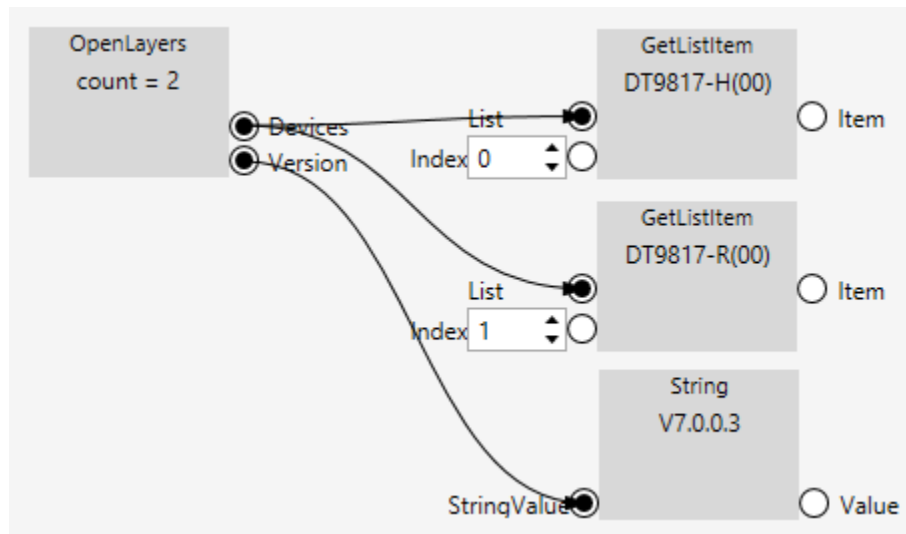
The **Read Inputs** node reads the values from the electrical inputs. The DigIn output contains the state of the electrical inputs in the form of an integer.

The **Write Outputs** node writes the value from its DigOut input to its electrical outputs.

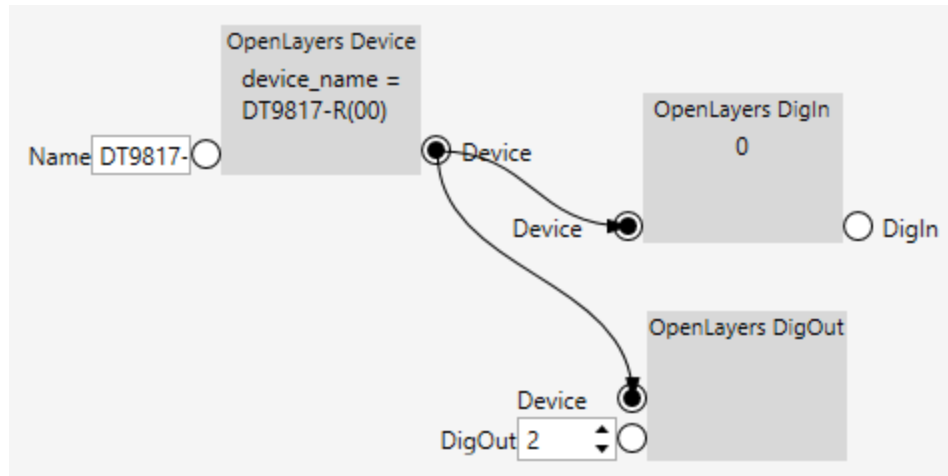
The Sync inputs of both the **Read Inputs** and **Write Outputs** nodes can be used to establish a defined order.

The **Read Inputs** node can be polled at regular intervals using the **nVision** timer.

Here is an example of how to find out which DT OpenLayers devices are connected.



If you know the name of a DT OpenLayers device, you can also type it directly to initialize the module. The value that you want to output on the digital output is typically not typed in, but calculated somewhere else. Also, the value that is input via the digital inputs is typically used somewhere else in the pipeline.



14.1.4 Imago VisionBox AGE-X

nVision supports the VisionBox AGE-X industrial PCs from Imago Technologies.

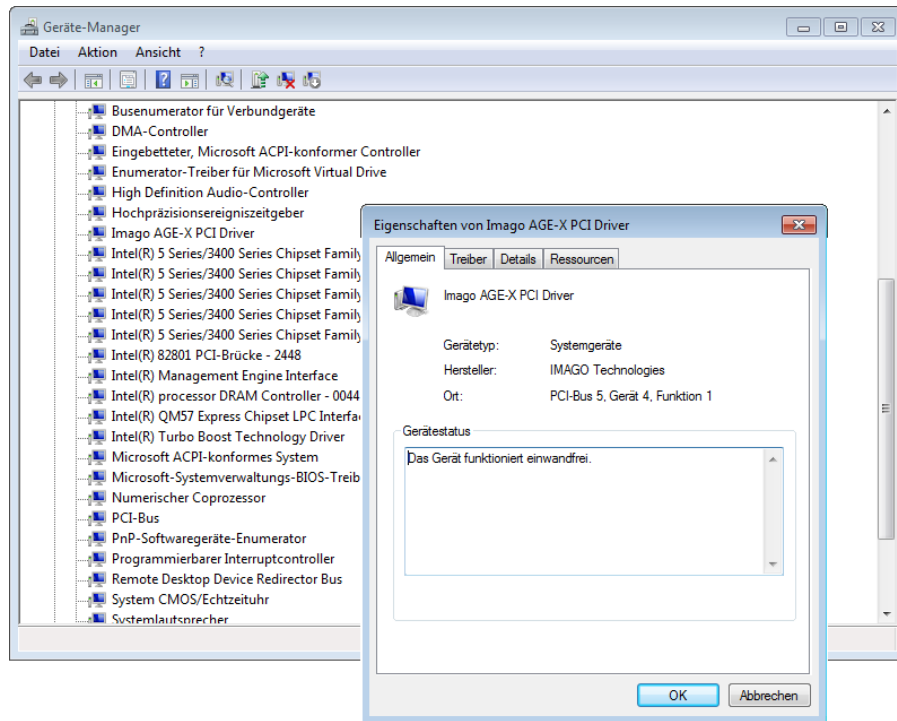


The VisionBox AGE-X is developed for machine vision and has special features for it. But inside it works like a normal PC. The VisionBox AGE-X can boot from a USB 2.0 or eSATA device. The idea is to work with an external hard disk with a normal operation system and an IDE. Boot the system from it and develop directly on the AGE-X. Impuls **nVision** can be installed on the VisionBox AGE-X. Tip: The Microsoft Windows remote desktop (RDP) is very powerful. Using it you can develop from your normal desktop.

Installation

VisionBox AGE-X device driver

After booting Windows the first time the OS asks for some driver. The delivery package contains all drivers you need. Other drivers can (must if you need this functionality) be installed manually. Select the desired installer and follow the instructions. One of these drivers is for the AGE-X functionality.



Imago SDK

To use the VisionBox AGE-X within nVision, the Imago SDK needs to be installed. The installer includes the SDK. Run the installer for your system: `AGE-X Setup 32bit.exe` or `AGE-X Setup 64bit.exe`. Select the desired installer and follow the instructions.

nVision pipelines that make use of the Imago nodes can run on a VisionBox AGE-X only. They can be loaded on a different PC (if the Imago SDK is installed on this PC), but they obviously cannot run on non-VisionBox hardware.

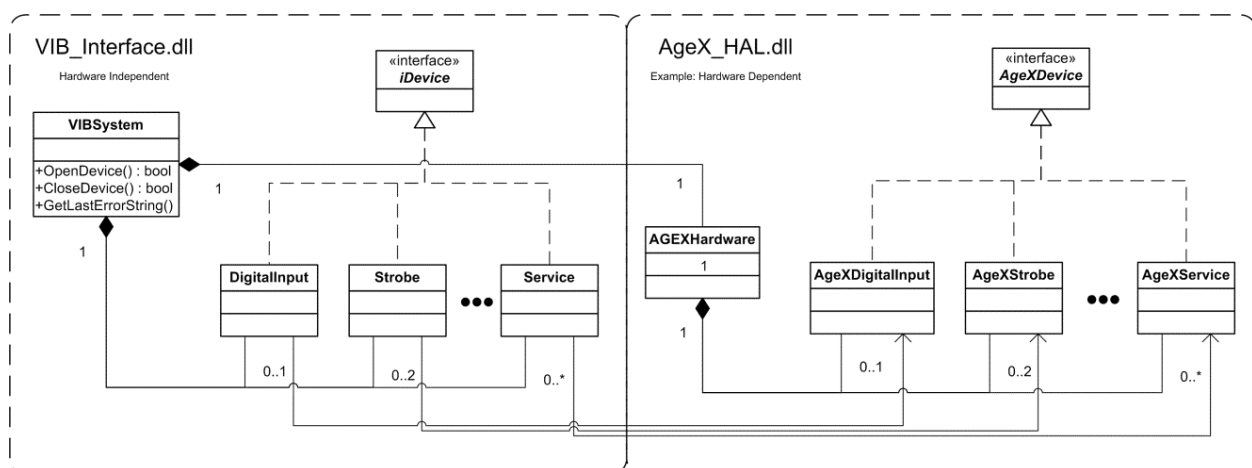
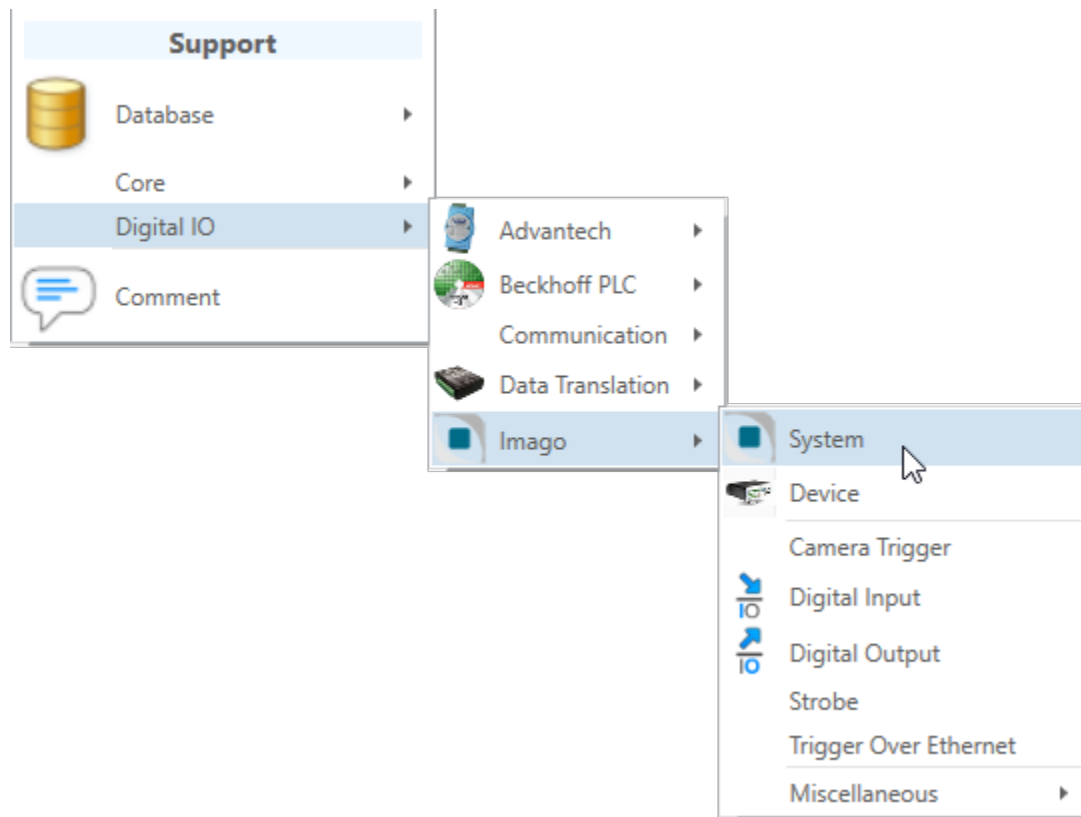
Build an Impuls nVision project

Start nVision, load an image, open a pipeline and you will find the AGE-X nodes in the menu.

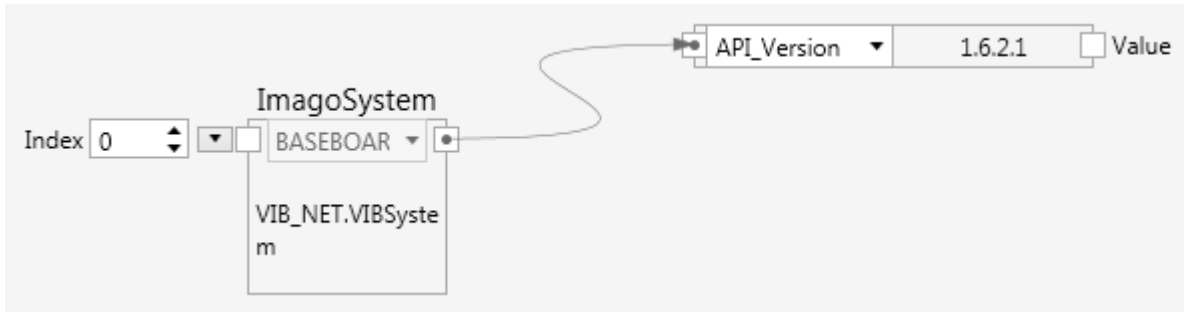
The next pages give an overview of the AGE-X functionality and the best way to use it within **nVision**. This is only an introduction of the general use. The Imago SDK contains a `VIB_NET.chm` reference for the whole functionality and function description.

Step 1 - Opening the system

On the first step we create the main factory. Before we start let's have a look to the structure of the framework:



This shows a simplified class diagram of the underlying framework. `VIB_NET` is a platform independent API to access the hardware functionality. `VIBSystem` is a factory which creates all devices. One device class groups functions around a physical interface / unit, for example digital inputs or strobe unit. If the box has two independent channels, you can create two instances of each type. For example the AGE-X has two strobe channels. Finally the `VIBSystem` contains / handles a hardware dependent factory to build a concrete device implementation. It allows one program to run on different Imago PC based boxes that will be created in the future without changes or rebuilds.



In order to read the version string from the `ImagoSystem` node, use the `GetProperty` node (`Ctrl-P`) and connect the `ImagoSystem` node output to it.

Since opening the system usually is a one-time step, it can be done globally; either in the system globals of **nVision** or in the global settings of the pipeline.

Step 2 - Using simple devices

An `ImagoDevice` node, which is not connected to an `ImagoSystem` node shows all possible devices, but they are all greyed out.

Once an `ImagoDevice` node is connected to the `ImagoSystem` node, the available devices are displayed prominently and can be easily selected.

The devices of a Baseboard system are different than those of a Network system.

Since opening the devices usually is a one-time step, it can be done globally; either in the system globals of **nVision** or in the global settings of the pipeline.

Here is an example that shows how to set a digital output. The `DIGITAL_OUTPUT` Operation node shows a number of methods that can be called on a digital output.

The example shows how to set a single bit on a digital output.

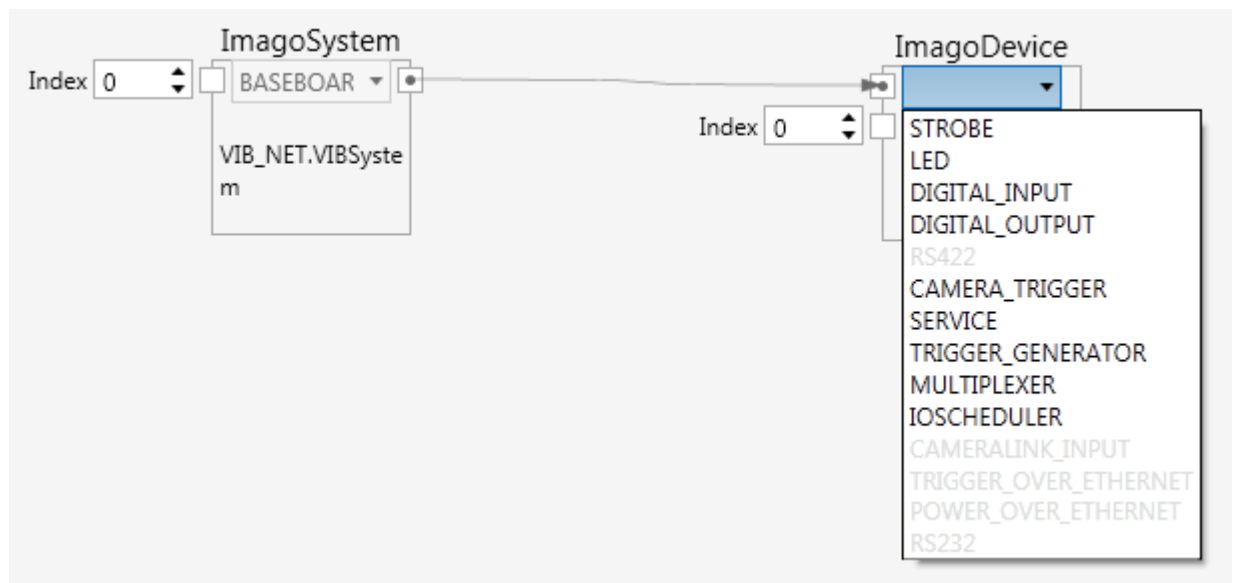
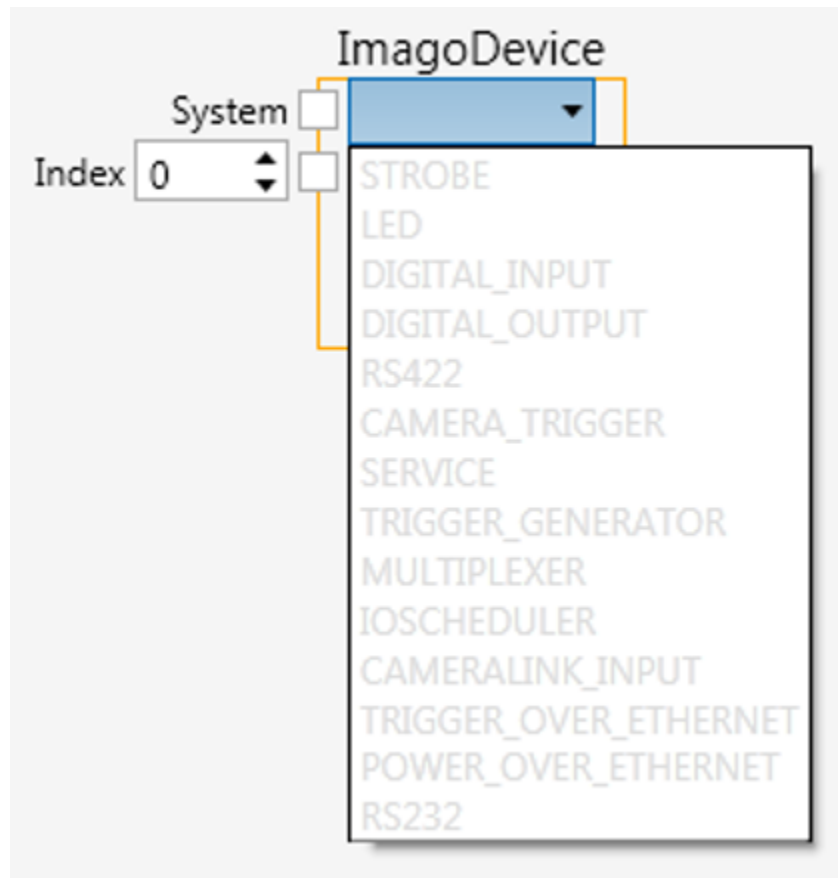
Step 3 - Using a complex device

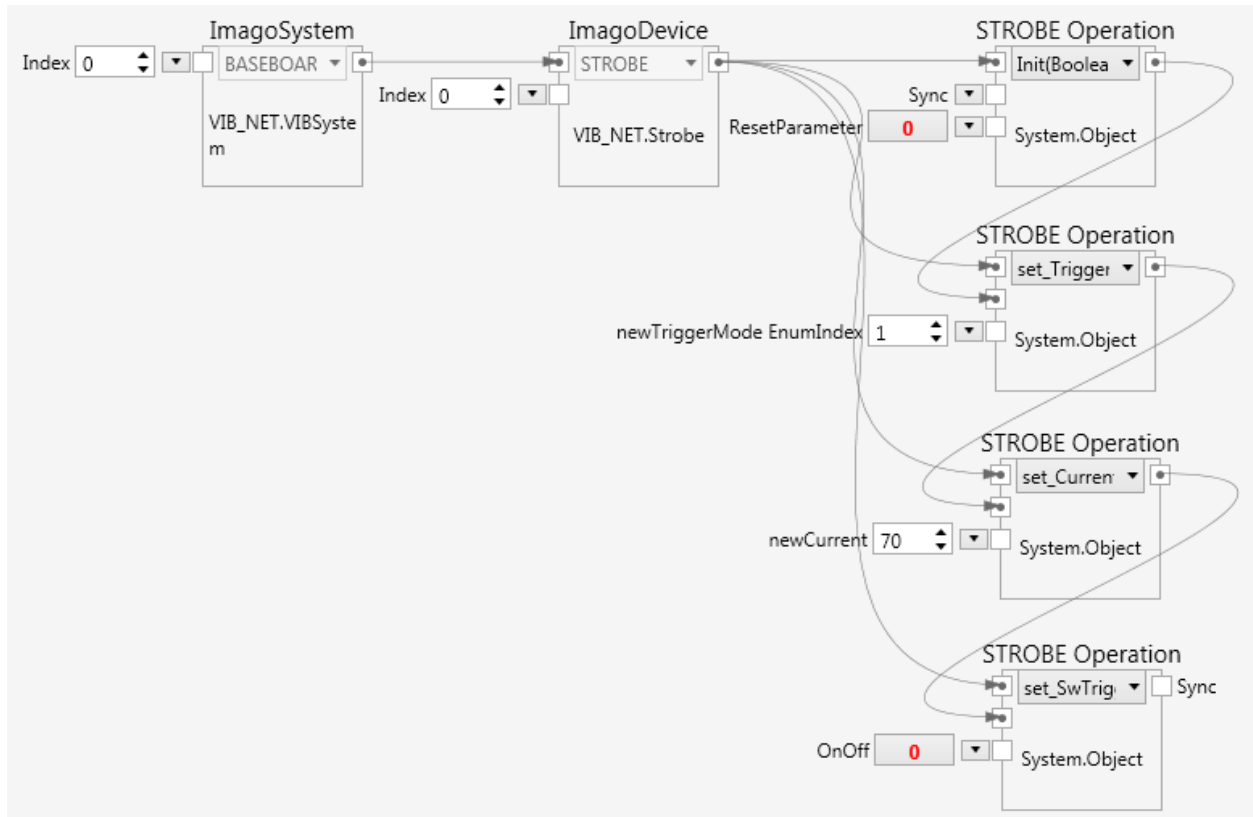
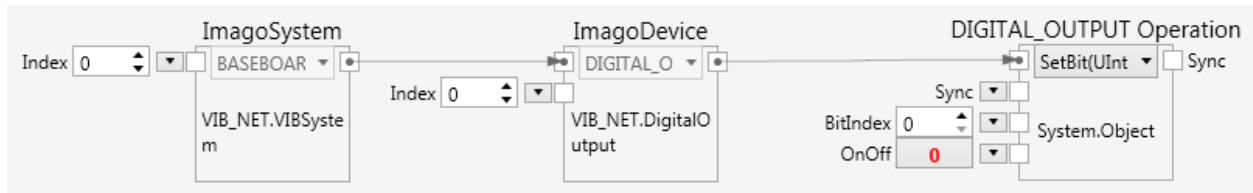
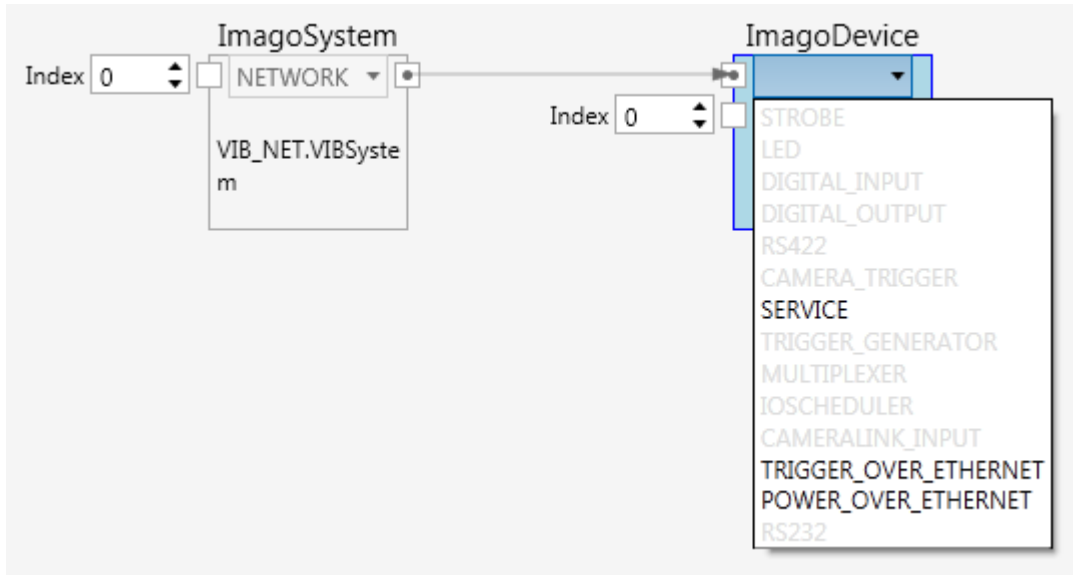
A more complex device is the strobe out. This example cannot explain the strobe unit in detail. Please refer to the function reference in the Imago SDK documentation. To run the code successfully, please connect a zero Ω bridge at the first output.

Note: Every channel of the strobe unit must be calibrated (by default done by Imago), otherwise the initialization will fail.

To use the strobe, several operations must be executed in a row:

- Initialize the strobe
- Set the trigger mode
- Set the desired current





Setting the trigger mode and current needs to be done after initializing the device. In order to establish this order, the Sync output of `Init()` is connected to the Sync input of `set_TriggerMode()`, and the Sync output of `set_TriggerMode()` is connected to the Sync input of `set_Current()`. Connecting the Sync ports in this way establishes an order of execution in a graph of operations, where otherwise the order of execution would be undefined.

Good to know

- Every Imago SDK function is thread safe.
- The Imago SDK framework is process safe.
- The device nodes show the available methods of a specific device. They also list property setters in the form of `set_Xxxx()`. Property setters can be distinguished from methods, which are listed in the form `XxxxxYyyyy()`.
- Properties of the Imago nodes can be read with the `GetProperty` node (Ctrl-P).
- Every function in the Imago SDK can be called with the graphical nodes.

14.2 CAD Drawing Support

nVision can import CAD drawings and convert them to images. Once the CAD drawing has been converted to an image, the usual image processing methods can be applied. Applications of particular interest are:

- match CAD drawings to acquired images, and then inspect differences.
- use CAD drawings to define regions of interest for part inspection.

nVision supports import of drawings from Autocad DXF files and Trumpf GEO files.

14.2.1 Import DXF files

DXF is a format defined and controlled by Autodesk (<http://www.autodesk.com>). Used within AutoCad, it has become the de-facto standard for CAD drawing interchange - even between different vendors. While the DXF format is syntactically simple and easy to parse, the semantics are complicated, because the format is not documented very well. Also, it has been changed and extended in a somewhat ad-hoc manner during the many years since the first versions of AutoCad, and this respecting and carrying on this heritage makes it difficult.

For this reason one cannot simply state that DXF drawing import will work in any case. **nVision** supports a partial 2D subset only. Particularly, the following primitives are supported:

primitve

features

line

a line segment

arc

a circular arc

circle

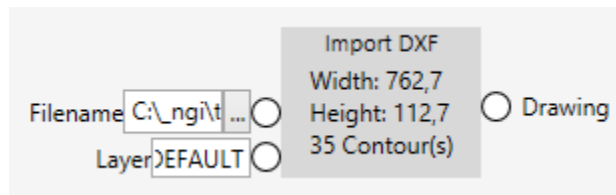
a full circle

polygon

a polygon

Files containing additional primitives usually can be read, but the unknown items are ignored.

When importing a DXF file, you need to specify the filename and the name of the layer you want to import. If there are multiple layers that you need to import, you can use several import nodes in parallel.



14.2.2 Import GEO files

GEO is the format used by laser cutting machines of TRUMPF (<http://www.trumpf.com>).

nVision supports a subset of the GEO format related to 2D.

primitve

features

line

a line segment

arc

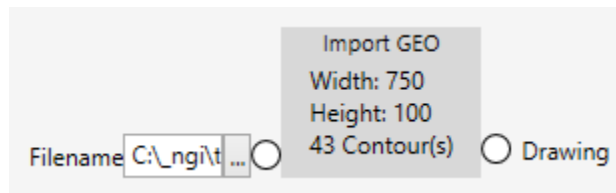
a circular arc

circle

a full circle

Files containing additional primitives usually can be read, but the unknown items are ignored.

When importing a GEO file, you need to specify the filename.



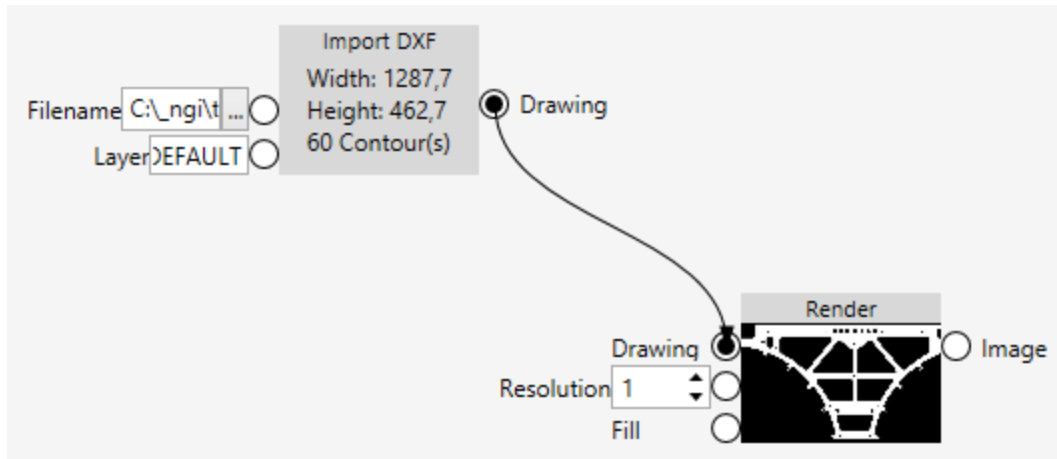
14.2.3 Render CAD drawings

Both import nodes create a CAD drawing when successful. The drawing has several properties that allow you to find out more information about the drawing, such as the height and width (in mm) as well as the contours contained in it.

Also, the drawing can be rendered into an image.

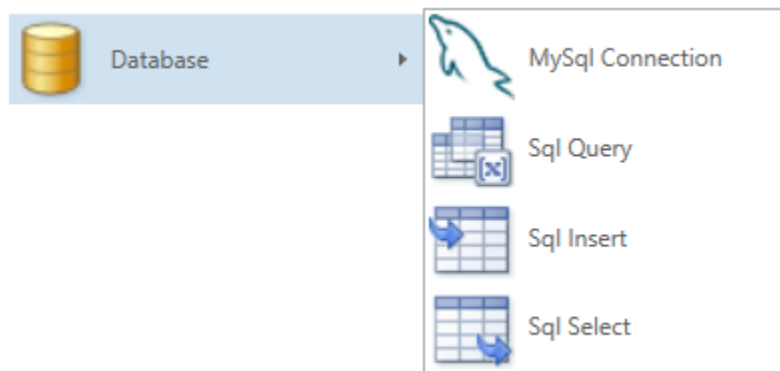
When rendering a CAD drawing, the resolution parameter tells you how fine the rendering will be. A value of 1 means that one pixel will have a size (area) of 1 mm (1 mm²). A value of 0.1 means that one pixel will have a size (area) of 0.1 mm (0.01 mm²). The actual size of the rendered image is determined by the width and height of the imported drawing (in mm), divided by the resolution parameter. The smaller (finer) the resolution parameter is, the bigger the rendered image will be.

The Fill parameter controls whether the drawing contours will be filled or outlined only.



14.3 Database Support

nVision can connect to databases and read values from or write value to them.



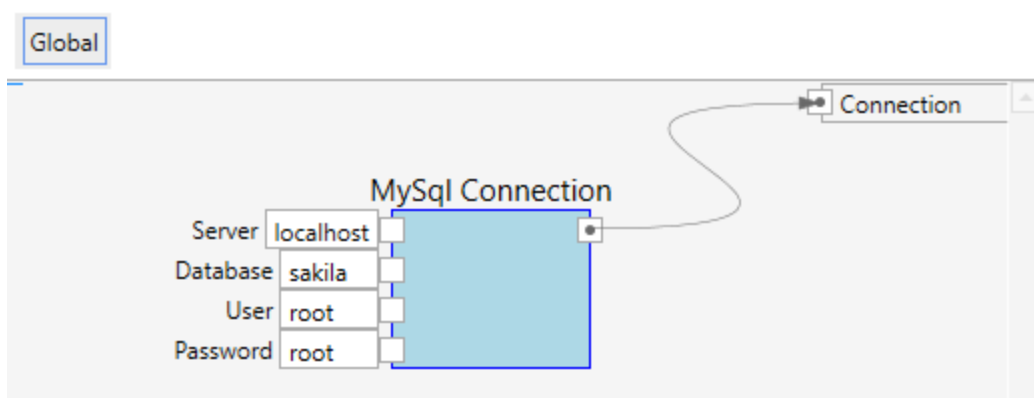
In order to use the database related nodes, you need to have a MySQL database server running somewhere. You can find more information about MySQL on the website (<http://www.mysql.com>). The samples assume that you have a local installation and that you have the `sakila` sample database accessible.

14.3.1 Establish Connection

The first step is to establish a connection to the database. The connection can be to a locally running database, or to a database running on a foreign machine. In order to connect to a database you need to specify the server name. To use the local machine, you would use `localhost` or `127.0.0.1` as the server name. To use a foreign name, you would use the IP address or the DNS name of the server. Then you need to specify the database name that you want to connect to. This database must exist on the server. Finally, you need to specify the user and the password for the connection.

Since a connection has a somewhat global nature, you can put the connection into the **Global** or the **System Global** pipeline.

Once you have a connection, you can use it to execute database queries. Queries come in the form of select, insert and generic query statements. Since the different query nodes perform IO and do not behave in a functional manner, they have the optional possibility to establish an order using their `Sync` inputs and outputs.



14.3.2 Execute Select

The select node allows you to construct a simple Sql `SELECT` statement easily.

The picture shows how you work with the select node. You start by establishing the connection, then you select the table from the database (`film`) and finally you choose the columns. In the picture we have constructed the following select statement: `SELECT title, description FROM film;`

The select returns a list of rows that reflect the result of the query, and each row consists of a list of strings; the result is a list of a list of strings (`List<List<String>>`).

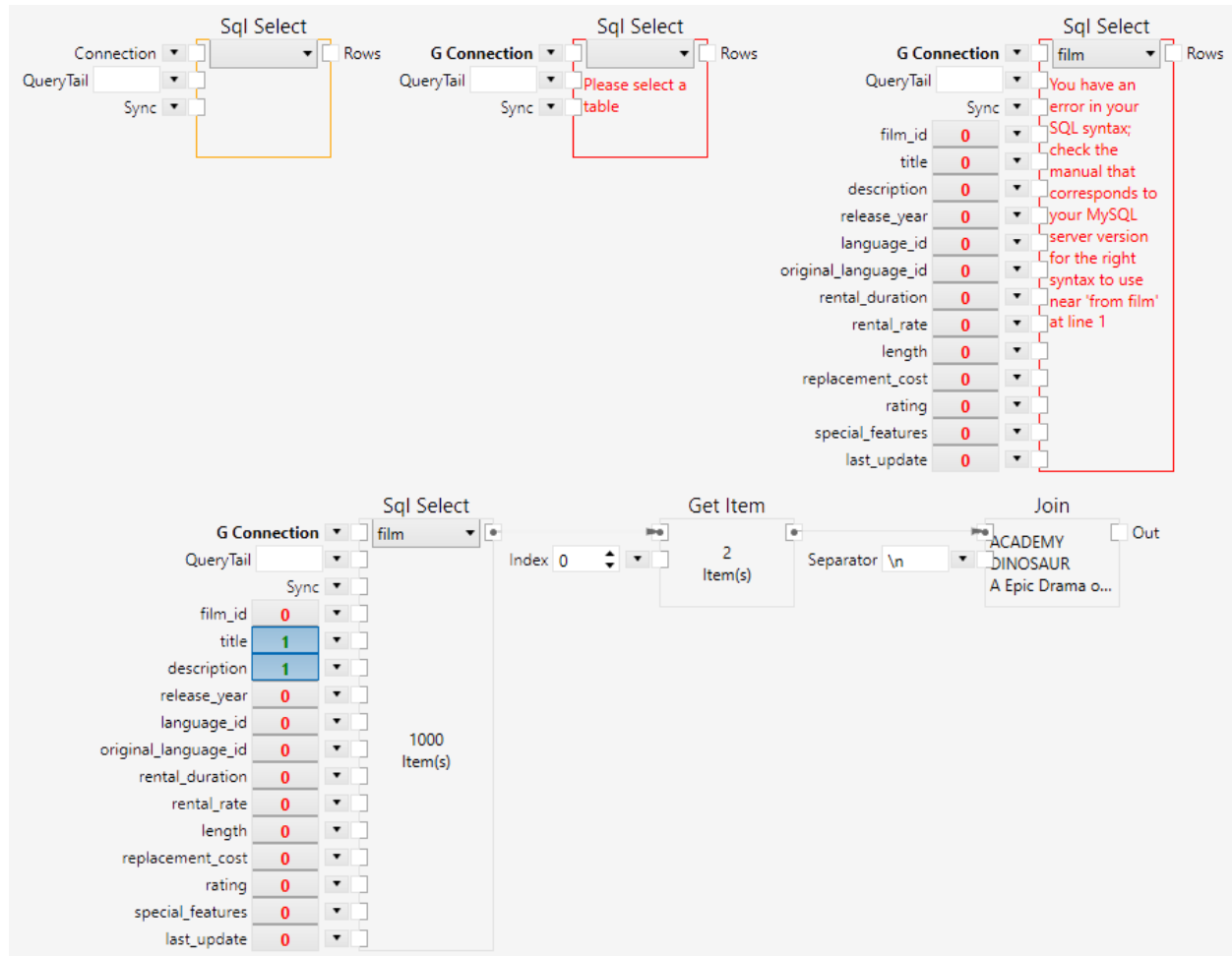
The `QueryTail` parameter allows you to append additional clauses to the end of the query (just before the semi-colon). Possible clauses are `WHERE . . .`, `GROUP BY . . .`, `ORDER BY . . .`, `LIMIT . . .`, etc. For a full explanation of the possible select queries have a look at the MySQL documentation (<https://dev.mysql.com/doc/refman/5.6/en/select.html>).

14.3.3 Execute Insert

The insert node allows you to construct a simple `INSERT` statement easily.

14.3.4 Execute Query

The query node allows you to construct any query that you can run against your database, not restricted to just `SELECT` or `INSERT` statements. Examples are `CREATE TABLE . . .`, `DELETE . . .`, `DESCRIBE . . .` and `EXPLAIN . . .` to name a few different statements. Have a look at the MySQL documentation (<https://dev.mysql.com/doc/refman/5.6/en/sql-syntax.html>) to find out more about the possible statements.



Connection Sync

Sql Insert

G Connection Sync

Sql Insert

Please select a table

G Connection Sync

Sql Insert

film

Incorrect integer value: '' for column 'film_id' at row 1

film_id

title

description

release_year

language_id

original_language_id

rental_duration

rental_rate

length

replacement_cost

rating

special_features

last_update

G Connection Sync

Sql Insert

film

Sync

film_id 1001

title THE BIG LEBOWSKI

description "The Dude" Lebowsky, mistaken for a millionaire Lebowsky, seeks restitution for his ruined rug and enlists his bowling buddies to help get it.

release_year 1998

language_id 1

original_language_id 1

rental_duration 6

rental_rate 1

length 117

replacement_cost 30

rating R

special_features

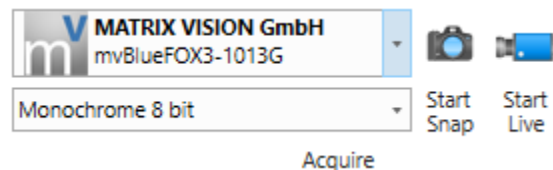
last_update 13.11.2015

System.Object

15.1 Ribbon

15.1.1 Home

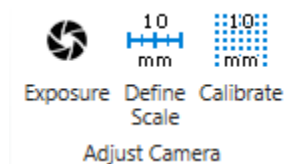
The **Home** tab groups commands that are used often.



You can select a camera and its mode with the combo boxes in the **Acquire** group. You can also use the **Start Snap** and **Start Live** commands to acquire a still image or start continuous live acquisition.

nVision supports multiple cameras (if they are connected) and you can use the controls to start one acquisition, change the camera and start another acquisition.

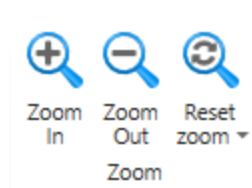
The commands in the **Adjust Camera** group control camera related settings.



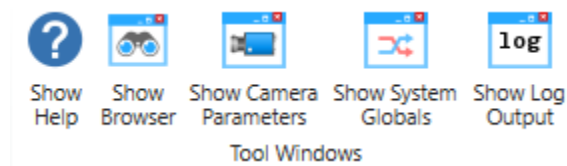
The **Exposure** tool lets you set the exposure time of a camera, and **Define Scale** and **Calibrate** perform spatial camera calibration using a known length or a calibration target.

The commands in the **Zoom** group are used to zoom in an out, as well as to reset the zoom factor.

You can also use keyboard shortcuts to zoom in (Ctrl + +), zoom out (Ctrl + -) or reset the zoom factor to 1 (Ctrl + Alt + 0), or set the zoom factor (Ctrl + 0) so that the display fits the window size.



The **Tool Windows** group has commands to show or hide the **Help**, **Browser**, **Camera Parameters**, **System Globals** and **Log Output** windows.



The **Help** (F1) window shows help information.

The **Browser** (F2) shows the loaded images and text files in a tiled fashion, along with additional information.

The **Camera Parameters** (F3) window shows the GenICam parameters of the selected camera.

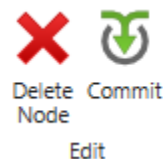
The **System Globals** (F4) window shows a pipeline editor that can be used to define global values that are available in all pipelines.

The **Log Output** (F5) window shows logging information.

15.1.2 Identify

The **Identify** tab groups commands that can read barcodes or text.

The **Edit** group contains commands for pipeline editing.



The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Codes** group contains commands for part identification with barcodes or matrix codes.

The **Barcode** and **Matrixcode** commands can be used to decode or verify 1-dimensional barcodes or 2-dimensional matrix codes of various symbologies.

The **Text** group contains the command for part identification with text.

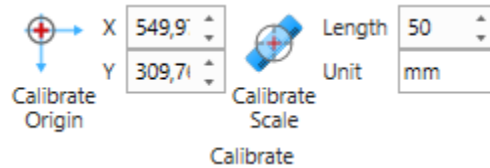
The **OCR** command can be used to decode or verify text.

The tools are meant to be chained one after the other, where one of the location tools is often the first tool in a chain of tools.



15.1.3 Interactive Measurement

The **Interactive Measurement** tab groups commands for interactive measurement.



The **Calibrate Origin** command is used to set the origin of an image.

The (x,y) pixel coordinates of the origin can either be typed into the controls on the ribbon or dragged with the rectangular crosshair cursor that appears when the command button is pressed.

The **Calibrate Scale** command sets the scale for interactive measurements. In order to calibrate the scale you align the scale tool with some known length in the image and enter the length and unit into the controls.

The **Overlay** group contains commands for graphical overlays.

The **Rectangular Grid** command overlays a rectangular grid that respects the origin.

The **Circular Grid** command overlays a circular grid that respects the origin.

The **Scale Bar** command overlays a scale bar in the lower left area of the image that visualizes the scale.

The **Picker** command adds an interactive tool that allows to pick grayscale or color values at some position from an image.

The position and the radius of the picking area can be dragged interactively.

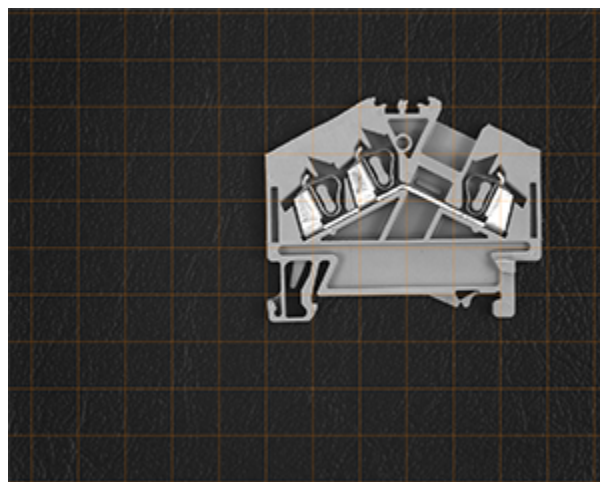
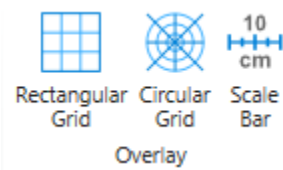
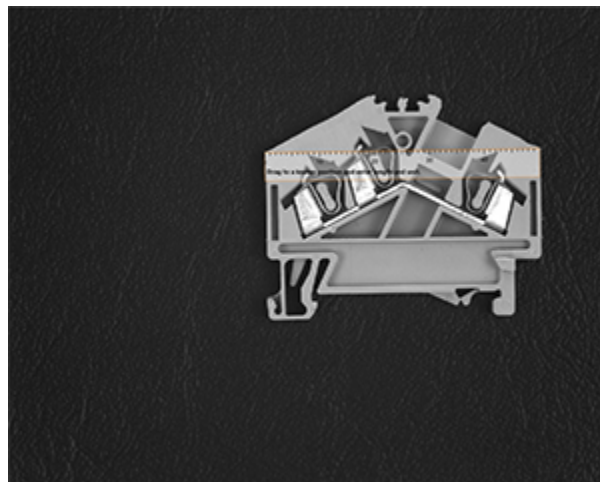
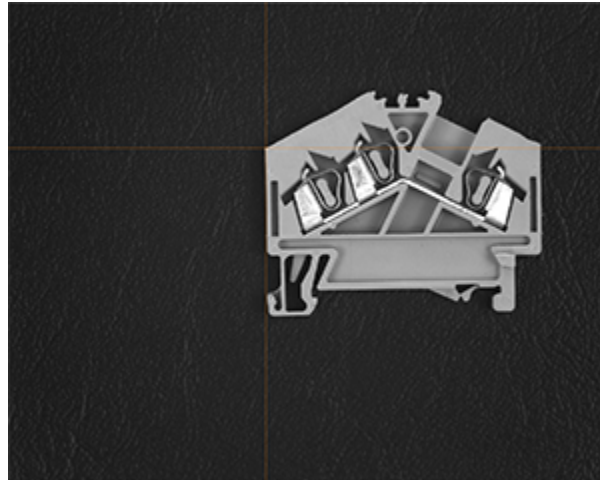
The **Measure** group contains commands for interactive measurements.

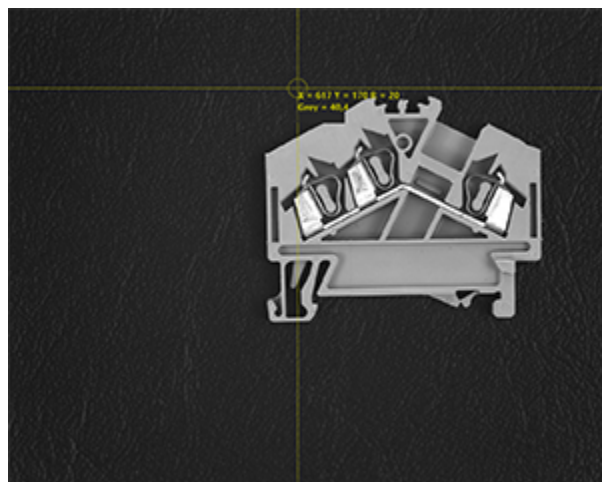
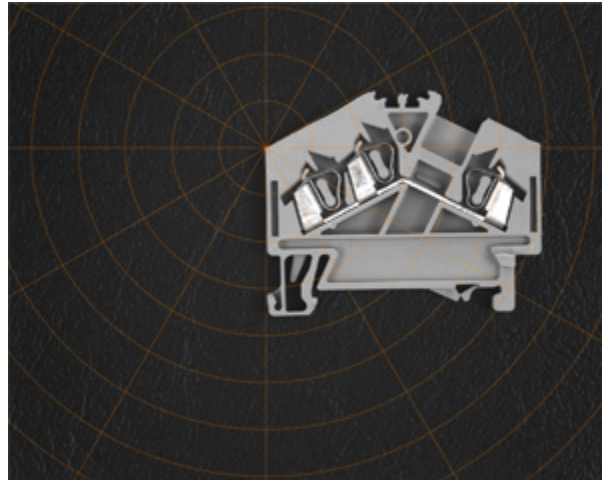
The **Counting Tool** helps with interactive counting. Click the left mouse button when the tool is selected to set a visual marker. If you hold the **Ctrl** key while clicking, a new group is started.

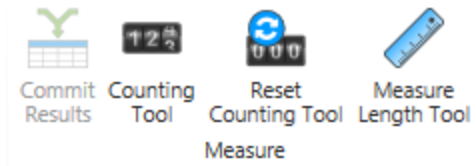
The **Reset Counting Tool** resets and counts and markers in all counting groups.

When done with counting, you can use the **Commit Results** command to write the measurement into a text file (CSV-File) that you can then store to disk.

The **Measure Length tool** displays a scale that you can align with a position to measure. Once the alignment is perfect (the two top corners are the hit spots for moving), you can use the **Commit Results** command (or the **space** key) to write the measurement into a text file.



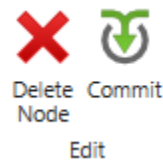




15.1.4 Locate

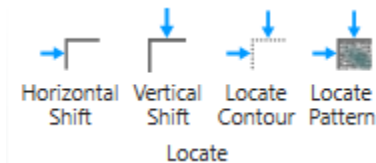
The **Locate** tab groups commands for location of parts.

The **Edit** group contains commands for pipeline editing.



The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Locate** group contains the tools for part location.



The **Horizontal Shift** and **Vertical Shift** tools detect the location of a part by using edge detection. They can be combined to find the horizontal and vertical position of a part.

The **Locate Contour** and **Locate Pattern** tools detect the position of a part with regards to translation and rotation using geometrical pattern matching (**Locate Contour**) or normalized grayscale correlation (**Locate Pattern**). In order to work with the tools, a region of interest is placed on some distinctive portion of the part. The **Teach** command is then used to teach this position.

Other tools respect this position and move their region of interest accordingly, so that the region of interest follows the part movement.

The tools are meant to be chained one after the other, where one of the location tools is often the first tool in a chain of tools.

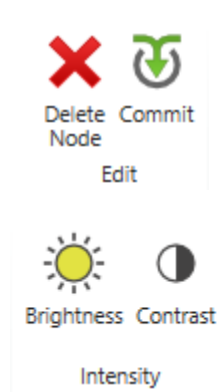
15.1.5 Measure

The **Measure** tab groups commands for measurements of parts.

The **Edit** group contains commands for pipeline editing.

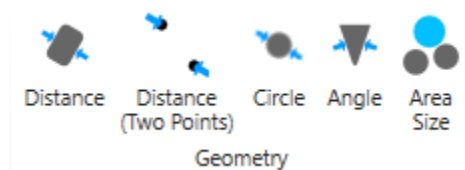
The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Intensity** group contains commands for intensity based measurements.



The two tools calculate brightness and contrast within a region of interest. Brightness and contrast are often sufficient to check for the presence of something.

The **Geometry** group contains commands to make geometric measurements.



The commands can calculate distances and angles, can fit circles and measure areas. All measurements can then be classified to return true (ok) when they are within an expected range, or false (nok), when they are outside this range.

The **Count** group contains commands for edge and object counting.

The first tool counts edges crossing a line of interest. The next tool counts the number of contour points within a region of interest (which is high, if the region is structured and low if the region is unstructured). The last command counts the number of distinct objects.

The tools are meant to be chained one after the other, where one of the location tools is often the first tool in a chain of tools.

15.1.6 Pipeline

The **Pipeline** tab groups pipeline related commands.

The **Edit** group contains commands for pipeline editing.

The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

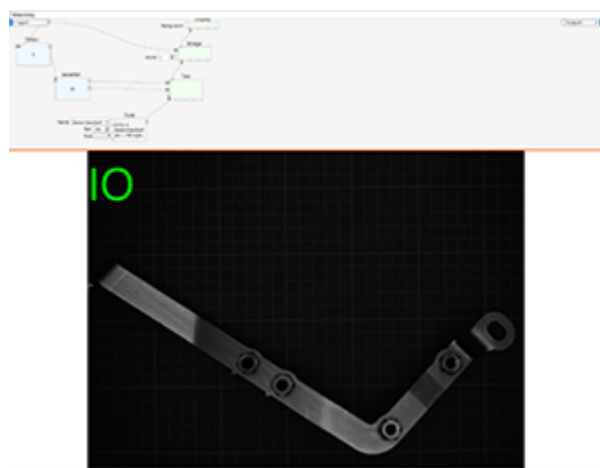
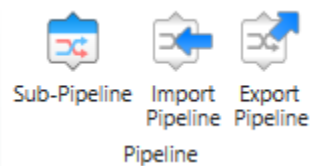
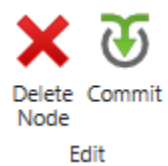
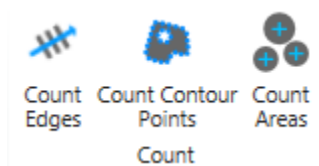
The **Pipeline** group has commands for sub-pipeline creation as well as import and export of pipelines.

The **Sub-Pipeline** command inserts a sub-pipeline into the linear pipeline.

This sub-pipeline step has a visualization in two panes, where the upper portion (above the blue splitter line, that you can drag down) is the editor for the sub-pipeline, and the lower portion is the output visualization of the sub-pipeline.

A sub-pipeline by default has one input and one output, as well as any number of nodes and connections between the nodes. Nodes are inserted by selecting them from a context menu (right mouse button click in the upper pane).

The visualization in the lower pane either shows the first output, or some other visualization that is composed within the sub-pipeline.



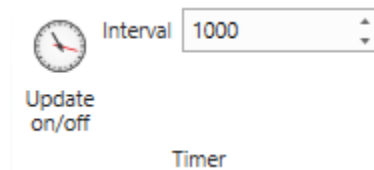
The **Import Pipeline** and **Export Pipeline** commands allow you to load and save pipelines from disk.

The **Control** group contains commands to apply the pipeline to a set of files on disk.



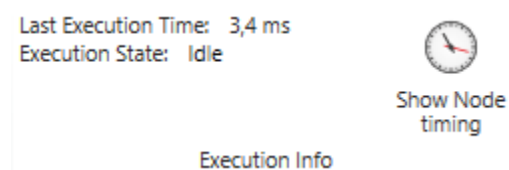
Some commands in this group are active if you have loaded a set of files with the **Open Folder** command. The commands allow you to step forth and back in the sequence of files, process all images in the folder and control how export nodes in the pipeline behave.

The **Timer** group contains commands for polling nodes.



Polling can be switched on an off, and the polling interval can be specified. Polling affects only one polling node (nodes that support polling have a little clock symbol, which must be checked to make such a node the polling node).

The **Execution Timing** group contains information about the execution time of a pipeline.



You can see the overall execution time of the pipeline, and with the **Show Node Timing** command you can see timing information broken down to a single node in the graphical editor.

The **Execution Statistics** group gives statistic information about pipeline execution.

The **Statistics Start/Stop** command enables or disables the accumulation of statistic information.

15.1.7 Process

The **Process** tab groups many commands for image processing.

The **Statistics** group contains commands for statistical analysis of images.


The **Histogram** command adds a step to the pipeline that calculates a histogram.

The **Horizontal Profile** command adds a step to the pipeline that calculates a horizontal profile.

The **Horizontal Profile Overlay** command adds a visual tool on top of the image display that displays the profile and allows you to inspect positions and values.

The **Crop Box** command adds an interactive tool that allows to crop a rectangular portion out of an image.

The rectangular area can be moved and sized interactively. Only the cropped portion is sent downstream into the following node.


 Average Execution Time: 0 ms (0/s)
 Execution Rate: 0/s
 Execution Count: 0

Statistics
Start/Stop

Execution Statistics



Histogram

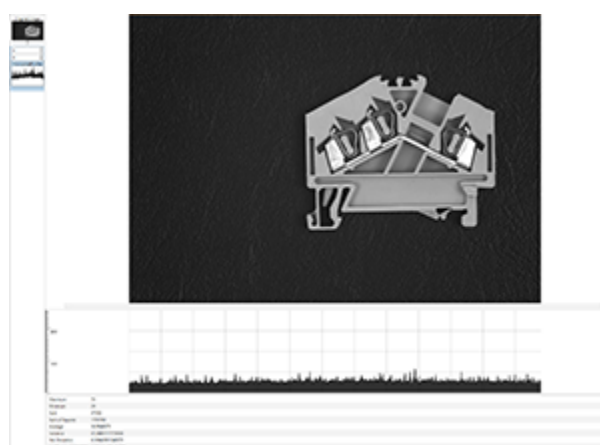
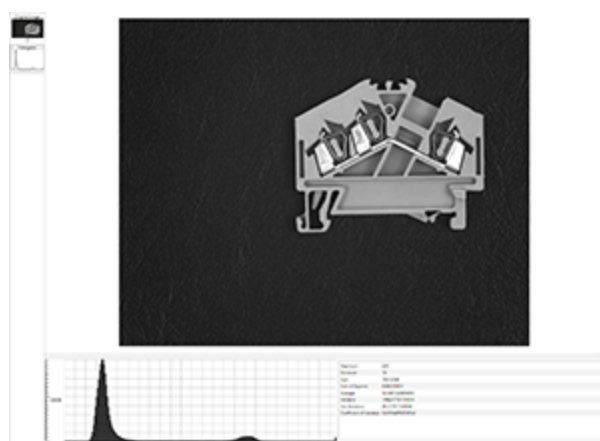


Horizontal
Profile

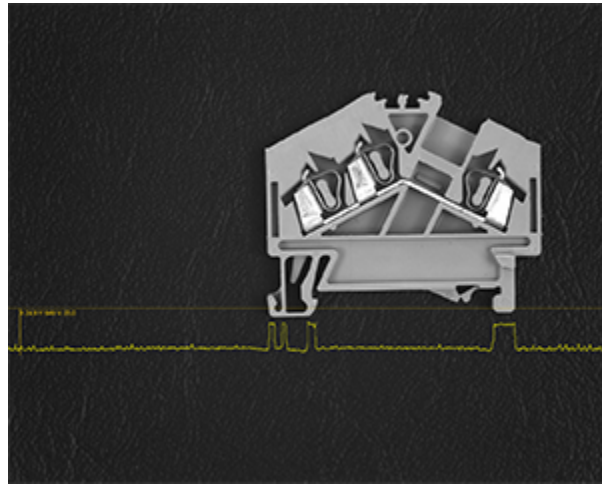


Horizontal
Profile Overlay

Statistics



Horizontal
Profile Overlay
Overlay



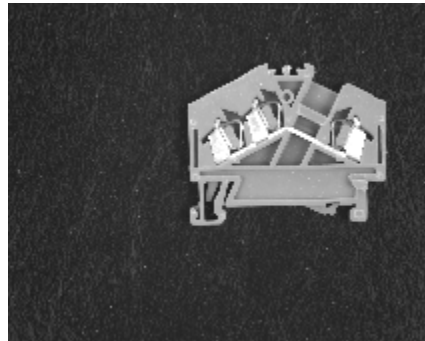
Crop
Box
Crop



Bin
Bin

The **Bin** command shrinks an image by an integer factor (horizontal and vertical can be different).

The command supports various binning modes: arithmetic, geometric, harmonic and quadratic mean, maximum and minimum, as well as median and midrange.



The **Point** group contains commands for pixel-wise operations on images.



Some commands (**Not**, **Negate**, **Increment**, **Decrement** and **Absolute**) do not need any parameters at all.

Other commands (arithmetic, logic and comparison) need one parameter, which can be supplied via the linear pipeline.

The point commands also exist between two images, and those variants can be used in the graphical pipeline editor.

The **Color** group contains functions for color space conversions.



Images can be transformed into different color spaces.

Rgb or monochrome are usually used. Hls (hue, luminance, saturation) and Hsi (hue, saturation, intensity) are sometimes advantageous, because they model human perception better than Rgb. Lab tries to deliver perceptual uniformity and is useful for color segmentation.

The **Frame** group contains the **Frame** command that puts a frame around an image.

The **Filter** group contains commands for neighborhood filtering operations on images.

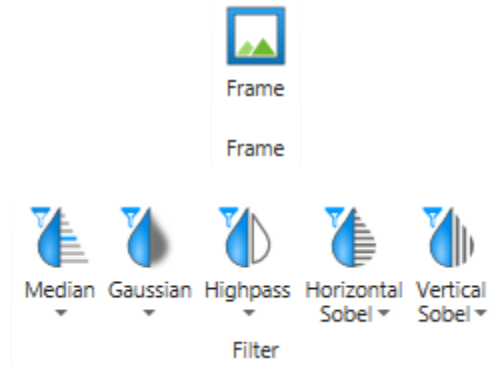
The **Filter** group contains commands for median, lowpass and highpass filtering.

Median filtering is often used for noise reduction. The **Median** and **Hybrid Median** filters are two variants of median filtering.

Lowpass filtering is another method used for noise reduction. The **Gaussian** (sigma), **Gaussian** (fixed 3x3 and 5x5 kernels) and **Lowpass** filters are variants of lowpass filtering.

The other filters are for edge emphasis and edge detection.

The **Morphology** group contains commands for morphological operations on images and regions.



The morphological filters can operate on images (blue icons) or on regions (red icons). Morphological filters on regions are much more performant and should be chosen if possible.

Dilate, **Erode**, **Open** and **Close** are the basic morphological functions. The morphological **Gradient** is a form on edge detection.

The **Inner Boundary** and **Outer Boundary** turn a region into the respective boundary.

Fill Holes fills holes in a region.

The **Geometry** tab groups commands for geometric operations on images.

The **Mirror** commands reverses right and left in an image.

The **Flip** command reverses top and bottom in an image.

The **Rotate Clockwise**, **Rotate 180°** and **Rotate Counter Clockwise** commands rotate an image.

15.1.8 Segmentation

The **Segmentation** tab groups commands for segmentation of images (separation into foreground and background) and blob analysis (particle analysis).

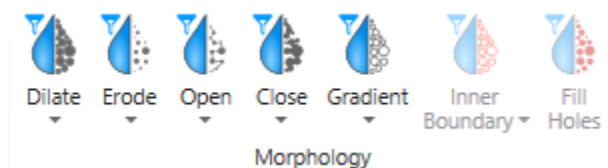
The **Edit** group contains commands for pipeline editing.

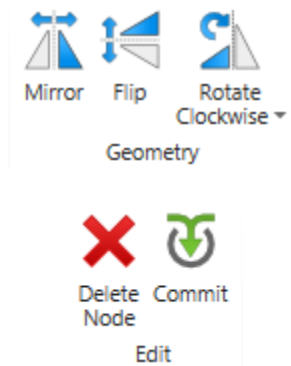
The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Threshold** command performs binary thresholding at a threshold value and creates a region as a result. The result is really only one region, despite it may consist of visually separated parts.

The **Connected Components** command takes the region and splits it into a list of regions based on the connectivity (or distance).

The **Filter Regions** commands allows you to remove those regions from a list of regions that do not satisfy a condition, such as Area > 200. You can use any region feature for the filtering.





The **Blob Analysis** command calculates features for each region. You can select the features to be calculated. The result is a grid of values: for each object or particle you will get a row of numeric results, that you can commit and save to a CSV file if needed.

The **Plot** command displays features graphically.

15.1.9 User Tools

The **User Tools** tab allows to create custom tools and group them on the ribbon.

The **Edit** group contains commands for pipeline editing.

The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Tools** group contains a button to save a tool, as well as the saved tools.

The first step is to actually create a tool. You do this by adding a sub-pipeline to the linear flow and name it appropriately, by editing its title in the linear flow. This name will become the name of the tool.

After doing that just click **Save as Tool** and it will appear on the ribbon.

Note that tool names must be unique. If a tool already exists, a dialog will pop up asking for permission to update it.

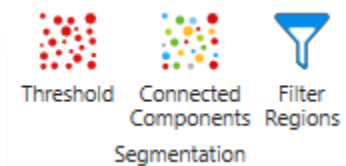
Tools are saved at: %USERPROFILE%/AppData/Local/Impuls/nVision

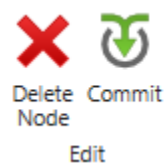
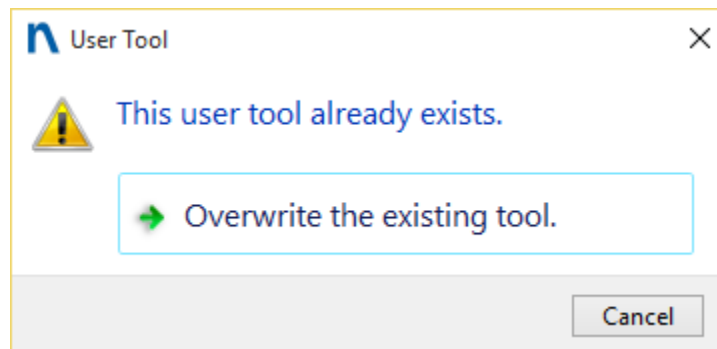
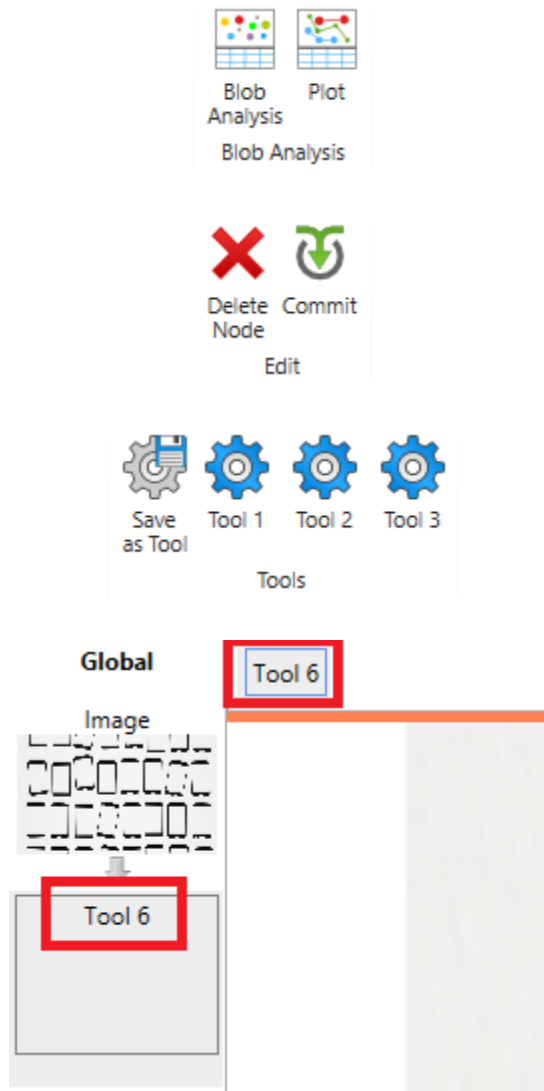
To load a tool just select the point of insertion on the linear flow and click on the desired tool ribbon button.

15.1.10 Verify

The **Verify** tab groups commands for pattern matching.

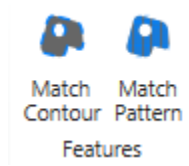
The **Edit** group contains commands for pipeline editing.





The first command deletes a node in the linear pipeline. The second command commits the result of the selected node to the browser, where it can be stored into the filesystem. These two commands are replicated on most tabs of the ribbon.

The **Features** group contains commands for pattern matching.



The commands use pattern matching with geometric or correlation methods to calculate the similarity. Both tools allow you to teach the look of a part and match the actual image against this template.

The tools are meant to be chained one after the other, where one of the location tools is often the first tool in a chain of tools.

15.2 Tools

15.2.1 Angle Tool



The **Angle** tool measures the angle between two lines.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays two boxes that are used to select regions, the boxes are labeled A and B. You can move the boxes around on the image by dragging their inside. You can also resize the boxes by picking them at their boundary lines (on the boundary line or just a bit outside) or at their corner points (on the corner point of just a bit outside). The arrows at the boxes visualize the search direction of the line detection (box A has horizontal arrows and should be used for vertical lines, box B has vertical arrows and should be used for horizontal lines).

Once the search box has been positioned over an edge in the image it displays the calculated edge points in green, and a resulting line in blue that it calculates by fitting a line through the points.

The tool has outlier detection. Outlier points are marked in red and they are not used for the line fit.

The search boxes move with the pose, that is their position is relative to the part position that has been determined by a location tool upstream.

If the tool finds both lines it calculates the angle between those lines. The angle is visualized graphically and its numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red, if the measured angle is not within the expected boundary values (between 89,9 ° and 90,1 °):

The angle tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum angles.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Spacing is the distance between the search lines of the edge detection (default = 5). The effect of the spacing can be seen by the density of the visualized edge points.

15.2.2 Area Size Tool



The **Area Size** tool measures the object area in a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The tool is meant to be used on regions where you have contrasting objects, either dark or bright. It automatically detects the objects. In a region without much contrast, using the tool is mostly meaningless because of noise.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red, if the measured area is not within the expected boundary values.

The area size tool has a configuration panel, which can be used to set parameters.

The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Polarity** can be set to **Dark Objects** or **Bright Objects**.

Objects smaller than a specified amount of pixels can be filtered out when **Filter Objects Smaller Than** is set.

Objects touching the border of the region of interest can be filtered out when **Filter Objects Touching Border** is set.

15.2.3 Barcode Tool

The **Barcode** tool decodes a barcode inside a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

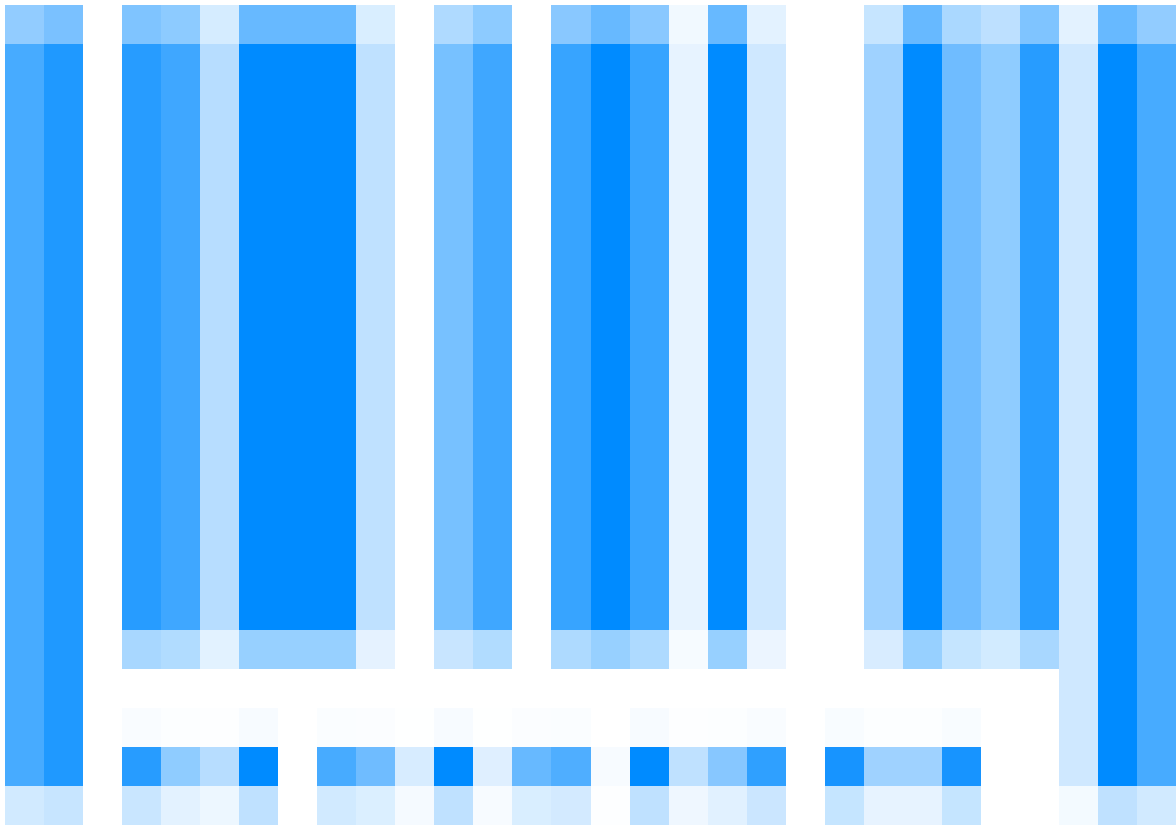
It displays a box that is used to select the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside).

Once the search box has been positioned over a code in the image it tries to decode the code. Its position and the decoded text is displayed in green, if it matches the expected text (anything in this case), or red, if the decoded text does not match the expectation:

The **Barcode** tool has a configuration panel, which can be used to set parameters.

The controls in **Pattern Matching** allow you to specify a pattern that the decoded result must follow in order to be correct.

Regular Expression: Defines the pattern that the decoded string should match. The pattern follows the rule of a regular expression, specifically the .NET variant of regular expressions.



The following table explains common cases. For full information, look at the [original documentation](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx) ([https://msdn.microsoft.com/en-us/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx)).

Pat-tern	Description		
.	Any character (except newline).	a.c	abc, aac, acc, ...
^	Start of a string.	^abc	abc, abcdefg, abc123,...
\$	End of a string.	abc\$	abc, endsinabc, 123abc,...
|	Alternation.	bill|ted	bill, ted
[. . .]	Explicit set of characters to match.	a[bB]c	abc, aBc
*	0 or more of previous expression.	ab*c	ac, abc, abbc, ...
+	1 or more of previous expression.	ab+c	abc, abbc, abbbc, ...
?	0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.	ab?c	ac, abc
{...}	Explicit quantifier notation.	ab{2}c	abbc
(...)	Logical grouping of part of an expression.	(abc){2}	abcabc
\\	Preceding one of the above, it makes it a literal instead of a special character.	a\\.b	a.b

Ignore Case: If checked the case of the characters is ignored, otherwise the case must match exactly.

The controls in **Enabled Symbolologies** allow you to enable or disable specific symbolologies. Supported symbolologies are Code 39, Code 30 Extended, Code 93, EAN 128, EAN 8, EAN 13, UPC A, UPC E, Databar and Databar Limited.

The controls in **Decoder Parameters** allow you to manipulate locator and decoder behavior.

Number of Scanning Directions: 0: vertical, 1: horizontal, 2: both, 3 and more: oblique, every 180/N degrees. The default setting of 4 means the locator scans every 45 degrees.

Scanning Density: The spacing between scanning-lines, in pixels; small values favor the decoding rate and increase the running time. 5 is the default.

Noise: The noise threshold. Threshold level to get rid of spurious bars caused by noise. 40 is the default.

Check Quiet Zone: Check the quiet zone. When checked, a quiet zone as large as the standard requires must be present; when set to false, it is advisable to disable other symbolologies to avoid false matches. The default is checked.

Minimum Bars: The minimum number of bars for a successful decode.

15.2.4 Brightness Tool



The **Brightness** tool measures the average brightness in a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The numerical value of the brightness is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red if it is outside the expected values.

The brightness tool has a configuration panel, which can be used to set parameters.

The **Min Brightness** parameter is used to set the minimal acceptable brightness (default = 127), the **Max Brightness** parameter is used to set the maximal acceptable brightness (default = 255).

15.2.5 Calibrate Tool



The **Calibrate** tool is used to establish a camera calibration. The calibration can correct a perspective distortion and still measure proper distances in the calibrated plane.

The tool detects the positions of dots on a calibration target and uses those to detect the perspective distortion. In addition to the detected dot positions the tool needs to know the distance between two dots in world units as well as the world unit.

Only the dots within the yellow region of interest turn blue and are used for calibration.

The **Calibrate** tool has a configuration panel where you can enter the distance between two dots as well as the world units.

Once you have a proper image of the calibration target, eventually aligned the region of interest properly and entered the distance between the dots as well as the unit, press the **Teach** button.

15.2.6 Circle Tool

The **Circle** tool measures a circle in a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a yellow box that specifies the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside). Inside the box is a little yellow point that marks the center of the box.

Once the box has been positioned over a circle (or part of a circle) in the image it displays the calculated circle, which is visualized graphically and its numerical diameter is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red, if the measured distance is not within the expected boundary values.

The tool displays its measurement in pixels, or in world units, such as mm, if the system has been calibrated.

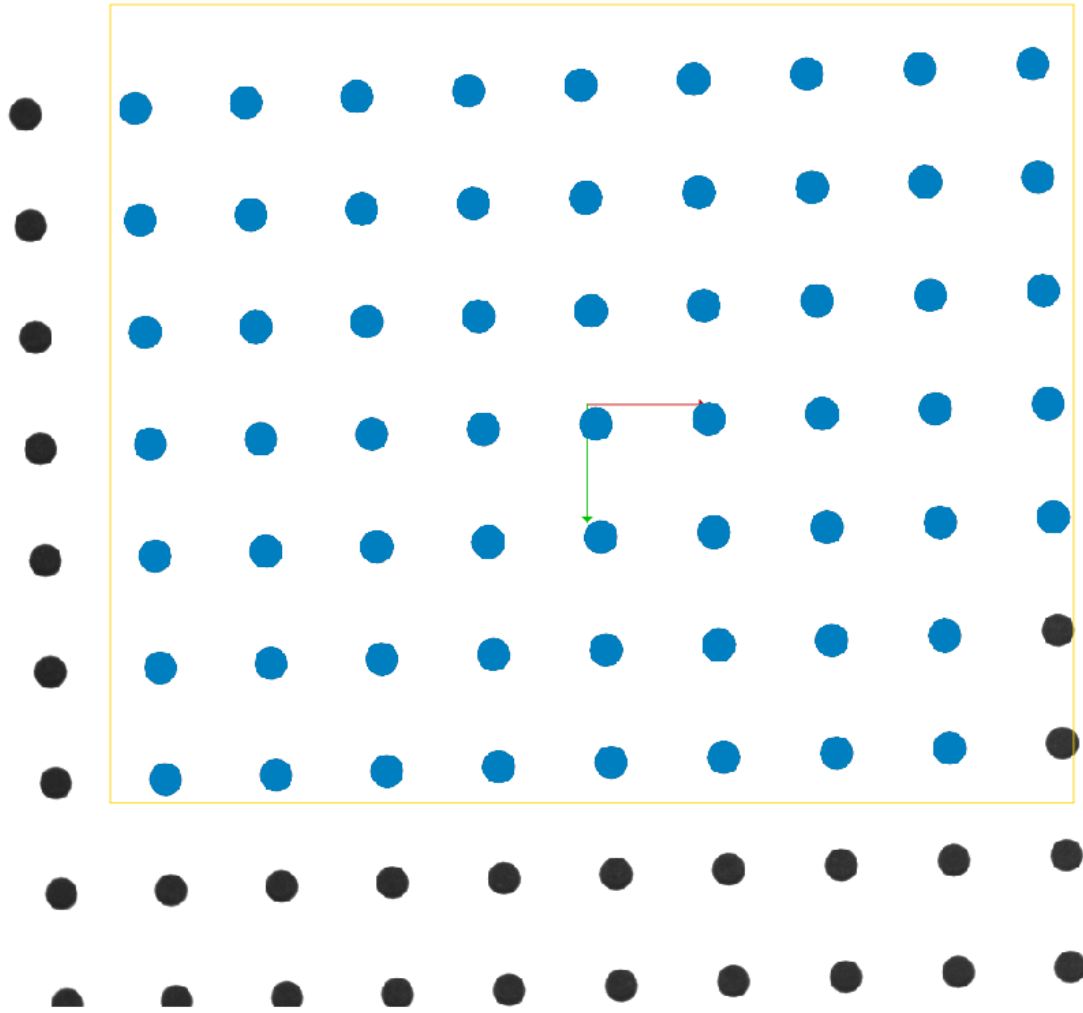
The circle tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum diameter.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

10 x 10
10 mm



Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Spacing is the angular spacing in degrees of the search lines.

15.2.7 Contrast Tool



The **Contrast** tool measures the contrast in a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red if it is outside expected values.

The contrast tool has a configuration panel, which can be used to set parameters.

The **Min Contrast** parameter is used to set the minimal acceptable contrast (default = 30), the **Max Contrast** parameter is used to set the maximal acceptable contrast (default = 255).

15.2.8 Count Areas Tool



The **Count Areas** tool counts the number of objects in a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

The tool is meant to be used on regions where you have contrasting objects, either dark or bright. It automatically detects the objects. In a region without much contrast, using the tool is mostly meaningless because of noise.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or red, if the area count is not within the expected boundary values.

The tool has a configuration panel, which can be used to set parameters.

The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Polarity** can be set to **Dark Objects** or **Bright Objects**.

Objects smaller than a specified amount of pixels can be filtered out when **Filter Objects Smaller Than** is set.

Objects touching the border of the region of interest can be filtered out when **Filter Objects Touching Border** is set.

15.2.9 Count Contour Points Tool



The **Count Contour Points** tool counts the number of high-contrasting pixels within a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a rectangle that is used to select a region. You can move the rectangle around on the image by dragging its inside. You can also resize the rectangle by picking it at its boundary lines (on the boundary line or just a bit outside). You can rotate the rectangle by picking it at its corner points (on the corner point of just a bit outside).

One use of the tool is to check for raffle or embossing on metal parts.

The numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or red, if the number of contour points is not within the expected boundary values.

The count contour points tool has a configuration panel, which can be used to set parameters.

The **Min** parameter is used to set the minimal acceptable area, the **Max** parameter is used to set the maximal acceptable area.

The **Sensitivity** can be set to a value between 0 (highly sensitive) and 255 (not sensitive).

15.2.10 Count Edges Tool



The **Count** tool counts the number of edges along a linear region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a yellow line that specifies the search line. You can move the line around on the image by dragging it. You can also pick either of the line endpoints to move just this endpoint.

Once the line has been positioned and finds edges, it visualizes them as blue points and it outputs their number. If the number is within expected bounds, the tool color is green, otherwise it is red.

The tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum diameter.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7). **Strength** is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

15.2.11 Define Scale Tool



The **Define Scale** tool is used to define an overall scale in the image. You can use it if your optical axis is orthogonal to the measurement plane, and if the desired measurement accuracy is not too high.

The tool calculates a calibration factor that is used to convert pixel measurements to world units, such as μm , mm, cm or m (or any other spatial unit that is needed in your application). Once a calibration has been established, other length measurement tools respect the calibration and output their measurements in world unit (as opposed to image units in pixels).

The **Define Scale** tool can use a line to establish a scale. Alternatively, it can use a circle to establish a scale. Alternatively, if the image has been acquired with a scanner, you can select the DPI setting to establish a scale.

The **Define Scale** tool has a configuration panel where you can select whether you want to use a line or a circle for setting the scale, or where you can select the DPI setting of the image.

Once you have aligned the line or circle with a known distance in your image, type the distance as well as the unit and press the **Teach** button.

15.2.12 Distance Tool



The **Distance** tool measures the distance between edges along a line.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a yellow line that specifies the search line. You can move the line around on the image by dragging it. You can also pick either of the line endpoints to prolong the line or move just this endpoint (move the mouse a little bit away from the endpoint, until the cursor changes). If you move the mouse in the vicinity of the line, a little bit outside, you can drag the thickness of the line. The thickness affects the edge detection, because it specifies the area that is averaged.

Once the search line has been positioned over two edges in the image it displays the calculated distance, which is visualized graphically and its numerical value is displayed in green, if it is in between the tool's minimum and maximum expected values, or in red, if the measured distance is not within the expected boundary values.

The tool displays its measurement in pixels, or in world units, such as mm, if the system has been calibrated.

The distance tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum distance.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

15.2.13 Distance Two Points Tool



The **Distance (Two Points)** measures the distance between two points.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays two yellow lines labelled A and B that specify the search linees. You can move the lines around on the image by dragging them. You can also pick either of the line endpoints to prolong the line or move just this endpoint (move the mouse a little bit away from the endpoint, until the cursor changes). If you move the mouse in the vicinity of the line, a little bit outside, you can drag the thickness of the line. The thickness affects the edge detection, because it specifies the area that is averaged.

Once the search lines have been positioned over two edges in the image the calculated distance, which is visualized graphically and its numerical value are displayed.

The tool displays its measurement in pixels, or in world units, such as mm, if the system has been calibrated.

The **Distance (Two Points)** tool has a configuration panel, which can be used to set parameters.

The **Min** and **Max** parameters are used to set the expected minimum and maximum distance.

The distance type between the two points can be the straight, horizontal or vertical distances.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

15.2.14 Exposure Tool



The **Exposure** tool is used to set the exposure time of a camera. It works with **GenICam** cameras that have an exposure time parameter.

The tool displays the live image and shows over-exposed pixels (> 250 greyvalues by default) in yellow color and under-exposed pixels < 5 greyvalues by default) in blue color. This gives you a quick visual impression of the illumination/exposure of an image. In addition, the percentage of underflow and overflow pixels are displayed, as well as a histogram that shows the overall brightness distribution.

The **Exposure** tool has a configuration panel, which can be used to set parameters.

The **Exposure Time (ms)** is the exposure time in ms (milliseconds). The tool uses GenICam to write the exposure time to the camera, which has to be connected to the global **Camera** port in the system globals.

Underflow and **Overflow** show the respective percentages of the underflow and overflow pixels.

When the exposure time is changed, the blue and yellow image overlays, the percentage values and the histogram graphic are all affected and change. The graphic and the numeric outputs help you to properly set illumination or exposure interactively.

The **Parameters** group has seldomly used additional parameters:

The **Underflow Threshold** sets the threshold for detection of underflow pixels (by default this is set to 5).

The **Overflow Threshold** sets the threshold for detection of overflow pixels (by default this is set to 250).

The **GenICam Parameter Name** sets the parameter name of the GenICam property that is used to set the exposure time. Usually that is `ExposureTime`, but some cameras use a different name, such as `ExposureTimeAbs`. You can find out the proper name by inspecting the **Camera Parameters** window (you can open it with the **Show Camera Parameters** command on the **Home** tab of the ribbon in the **Tool Windows** section).



15.2.15 Horizontal Shift Tool

The **Horizontal Shift** tool uses horizontal edge detection to find a horizontal location.

The tool displays a region of interest that is used to detect the shift. The region of interest can be moved by the user. The parameters in the **Edge Detection Parameters** affect the edge detection.

Orientation can be used to set the direction of the edge detection: **Left to Right** or **Right to Left** are available.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

The graphic below the controls shows the image that corresponds to the region of interest. The filled blue curve is the averaged brightness and the yellow curve is the gradient, both after the smoothing is applied. You can observe how the shape of the curves is affected by changing the **Smoothing**.

The two horizontal lines visualize the effect of the edge strength. An edge is only found if the yellow curve crosses the lines. You can observe how the edge detection is affected by changing the **Strength**.

The vertical line shows where in the curve and image the edge is detected.

15.2.16 Locate Contour Tool



The locate contour tool locates a part using geometric contour matching.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point or just a bit outside).

Make sure that you select a characteristic portion when you define the pattern. Once the box has been positioned, make sure that the filename in the **Template File** edit control is correct and press **Teach**. This saves the pattern.

If the tool finds the pattern, it displays a green box where it found the pattern and also moves the coordinate axes to the middle of the box to visualize the new origin and rotation (the pose).

Subsequent tools use the pose to position their ROIs, such that the ROIs are always in the correct position, even if the part moves considerably.

The locate shape tool has a configuration panel, which can be used to set parameters.

The **Template File** parameter specifies the match template, which is loaded from disk. If the template file exists, it is displayed here.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.6). Matches below this score are not considered.

The **Teach** button saves the pattern to the disk.

15.2.17 Locate Pattern Tool



The locate pattern tool locates a part using pattern matching.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).

Make sure that you select a characteristic portion when you define the pattern. Once the box has been positioned, make sure that the filename in the **Template File** edit control is correct and press **Teach**. This saves the pattern.

If the tool finds the pattern, it displays a green box where it found the pattern and also moves the coordinate axes to the middle of the box to visualize the new origin and rotation (the pose).

Subsequent tools use the pose to position their ROIs, such that the ROIs are always in the correct position, even if the part moves considerably.

The locate shape tool has a configuration panel, which can be used to set parameters.

The **Template File** parameter specifies the match template, which is loaded from disk. If the template file exists, it is displayed here.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.6). Matches below this score are not considered.

The **Teach** button saves the pattern to the disk.

15.2.18 Match Contour Tool



The **Match Contour** tool locates a part using geometric pattern matching.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify both the pattern as well as the search region of the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).

Usually, the box is put tightly around an area where you expect a specific pattern. Once the box has been positioned, make sure that the filename in the **Template File** edit control is correct and press **Teach**. This saves the pattern.

When the pattern has been taught, it may be beneficial to make the search region bigger, so that there is some tolerance in finding the pattern.

If the tool finds the pattern, it displays the match score (between 0 and 1, where 1 is a perfect match). Scores below 0.5 are mostly meaningless. Please note that the characteristic of the score is different to the one used in the **Match Pattern** tool.

If the tool cannot find the pattern at all, or if the score is too low, it outputs the text “No Match” in red.

The tool has a configuration panel, which can be used to set parameters.

The **Template File** parameter specifies the match template, which is loaded from disk. If the template exists, it is displayed here.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.3). Matches below this score are not considered.

The **Teach** button saves the pattern to the disk.

15.2.19 Match Pattern Tool



The **Match Pattern** tool locates a part using geometric pattern matching.

The tool displays graphical elements to specify the region of interest as well as to display the results.

The tool displays a yellow region of interest that you can drag around to specify both the pattern as well as the search region of the pattern. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or at its corner points (on the corner point of just a bit outside).

Usually, the box is put tightly around an area where you expect a specific pattern. Once the box has been positioned, make sure that the filename in the **Template File** edit control is correct and press **Teach**. This saves the pattern.

When the pattern has been taught, it may be beneficial to make the search region bigger, so that there is some tolerance in finding the pattern.

If the tool finds the pattern, it displays the match score (between 0 and 1, where 1 is a perfect match). Scores below 0.5 are mostly meaningless. Please note that the characteristic of the score is different to the one used in the **Match Contour** tool.

If the tool cannot find the pattern at all, or if the score is too low, it outputs the text “No Match” in red.

The tool has a configuration panel, which can be used to set parameters.

The **Template File** parameter specifies the match template, which is loaded from disk. If the template exists, it is displayed here.

The **Min Score** parameter is used to set the minimum score for an acceptable match (default = 0.5). Matches below this score are not considered.

The **Teach** button saves the pattern to the disk.

15.2.20 Matrixcode Tool



The **Matrixcode** tool decodes a two-dimensional barcode inside a region of interest.

The tool displays graphical elements to specify the region of interest as well as to display the results.

It displays a box that is used to select the region of interest. You can move the box around on the image by dragging its inside. You can also resize the box by picking its boundary lines (on the boundary line or just a bit outside) or its corner points (on the corner point of just a bit outside).

Once the search box has been positioned over a code in the image it tries to decode the code. Its position and the decoded text is displayed in green, if it matches the expected text, or red, if the decoded text does not match the expectation:

The **Matrixcode** tool has a configuration panel, which can be used to set parameters.

The controls in **Pattern Matching** allow you to specify a pattern that the decoded result must follow in order to be correct.

Regular Expression: Defines the pattern that the decoded string should match. The pattern follows the rule of a regular expression, specifically the .NET variant of regular expressions.

The following table explains common cases. For full information, look at the [original documentation](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx) ([https://msdn.microsoft.com/en-us/library/az24scfc\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx)).

Pat-tern	Description		
.	Any character (except newline).	a.c	abc, aac, acc, ...
^	Start of a string.	^abc	abc, abcdefg, abc123,...
\$	End of a string.	abc\$	abc, endsinabc, 123abc,...
|	Alternation.	bill|ted	bill,ted
[. . .]	Explicit set of characters to match.	a[bB]c	abc, aBc
*	0 or more of previous expression.	ab*c	ac, abc, abbc, ...
+	1 or more of previous expression.	ab+c	abc, abbc, abbbc, ...
?	0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.	ab?c	ac, abc
{...}	Explicit quantifier notation.	ab{2}c	abbc
(...)	Logical grouping of part of an expression.	(abc){2}	abcabc
\\	Preceding one of the above, it makes it a literal instead of a special character.	a\\.b	a.b

Ignore Case: If checked the case of the characters is ignored, otherwise the case must match exactly.

The controls in **Enabled Symbolologies** allow you to enable or disable specific symbolologies. Supported symbolologies are Data Matrix and QR Code.

The controls in **Decoder Parameters** allow you to manipulate locator and decoder behavior.

Polarities: Selects the possible contrast of the marking. **Auto** (default), **Black on White** or **White on Black**.

Views: Selects the possible orientation of the marking. **Auto** (default), **Straight** or **Mirrored**.

15.2.21 Reference Pose Tool

The **Reference** tool modifies the pose established by any location tool upstream.



The tool displays graphical elements to specify the new reference pose.

The tool displays a yellow box that you can drag and rotate to set the new pose. You can move the box around on the image by dragging its inside. You can also resize the box by picking it at its boundary lines (on the boundary line or just a bit outside) or rotate it by picking it at its corner points (on the corner point of just a bit outside).

Both the original pose (in gray) and the new pose are displayed, as well as two dimmer lines to help with alignment.

Subsequent tools use the pose to position their ROIs, such that the ROIs are always in the correct position, even if the part moves considerably.

Stabilize Image keeps the image steady for downstream tools. It translates and rotates the image so that the selected pose becomes the new center.

15.2.22 Result Tool

The **Result** tool is used to communicate the result of an inspection task. It works with an Adam module from Advantec.

The result tool writes the result in the form of an electrical signal to the Adam module.

The result tool has a configuration panel, which can be used to set parameters.

The parameters in **Configure IO** specify the electrical lines that are switched on.

OK specifies the number of the output of the Adam module which is used to signal the **OK** state.

Not OK specifies the number of the output of the Adam module which is used to signal the **not OK** state.

15.2.23 Vertical Shift Tool

The **Vertical Shift** tool uses horizontal edge detection to find a horizontal location.

The tool displays a region of interest that is used to detect the shift. The region of interest can be moved by the user.

The **Vertical Shift** tool has a configuration panel, which can be used to set parameters.

The parameters in the **Edge Detection Parameters** affect the edge detection.

Orientation can be used to set the direction of the edge detection: **Top to Bottom** or **Bottom to Top** are available.

Polarity can be used to select the type of edge: **Both**, **Rising** and **Falling** can be selected (default = Both). Rising means an edge going from dark to bright along the search direction, falling means an edge from bright to dark, and both means both edge types.

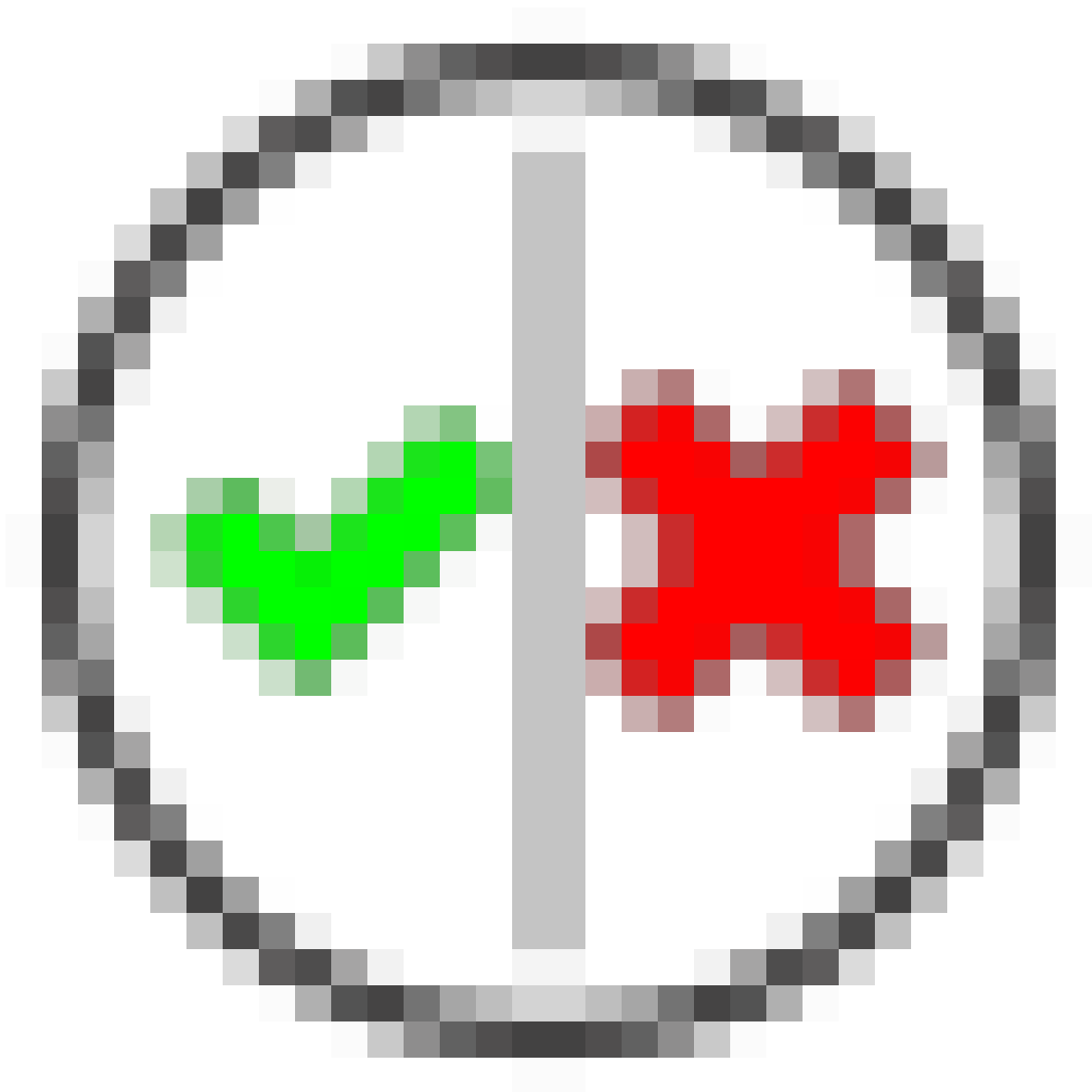
Smoothing is the amount of smoothing used in edge detection (default: 2.7).

Strength is the minimum strength of an edge (the gray scale slope of the edge, default = 15). Smoothing and Strength are interrelated: the more you smooth the more you lessen the strength of an edge and vice versa.

The graphic below the controls shows the image that corresponds to the region of interest. The filled blue curve is the averaged brightness and the yellow curve is the gradient, both after the smoothing is applied. You can observe how the shape of the curves is affected by changing the **Smoothing**.

The two vertical lines visualize the effect of the edge strength. An edge is only found if the yellow curve crosses the lines. You can observe how the edge detection is affected by changing the **Strength**.

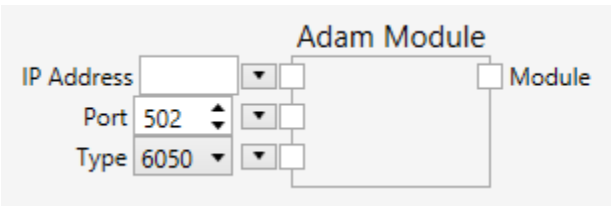
The horizontal line shows where in the curve and image the edge is detected.



15.3 Nodes

15.3.1 Adam Module

Establish a connection to an Adam-60xx Ethernet Module from Adavantech, which allows you to read and write digital signals.



This node initializes the module. If you connect its output to a **GetProperty** node, you can gather information about the device. Usually you would connect the output to an **Adam600 DigIn** node to read electrical input, or an **Adam6000 DigOut** node to write electrical output.



The Adam modules are connected via LAN or wireless LAN to a PC and support a variety of input/output lines. The modules may have additional features, such as counters, etc., but these are not yet supported.

module	bus	features
ADAM-6050	LAN	12 digital inputs, 6 digital outputs
ADAM 6050W	WLAN	12 digital inputs, 6 digital outputs
ADAM-6051	LAN	12 digital inputs, 2 digital outputs
ADAM-6051W	WLAN	12 digital inputs, 2 digital outputs
ADAM-6052	LAN	8 digital inputs, 8 digital outputs
ADAM 6060	LAN	6 digital inputs, 6 digital outputs
ADAM-6060W	WLAN	6 digital inputs, 6 digital outputs
ADAM-6066	LAN	6 digital outputs

Inputs

IP Address (Type: string)

The IP address of the Adam module (an IP V4 address such as 192.168.178.55).

Port (Type: Int32)

The port of the Adam module.

Type (Type: string)

The type of the Adam module (choose one of 6050, 6050W, 6051, 6051W, 6052, 6060, 6060W or 6066).

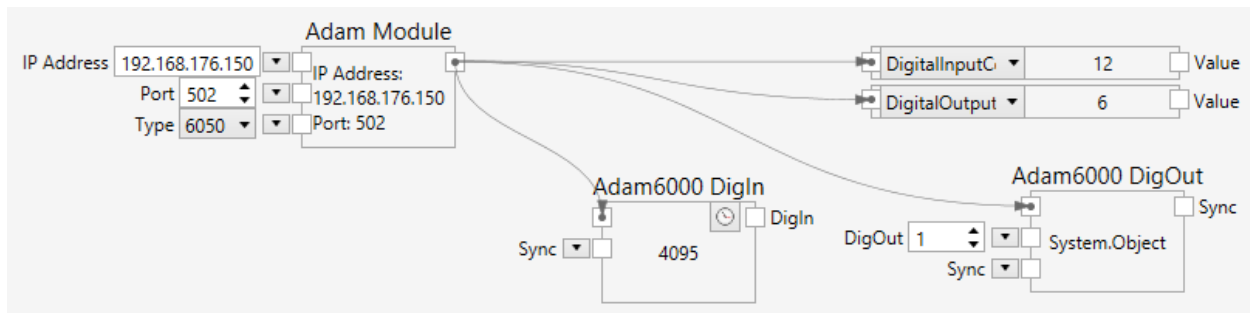
Outputs

Module (Type: Adam6000Module)

The Adam 60xx module instance.

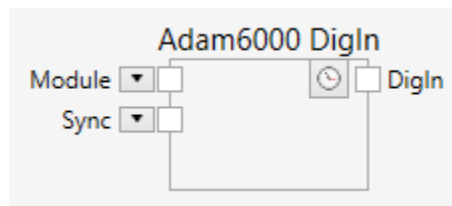
Sample

Here is an example:




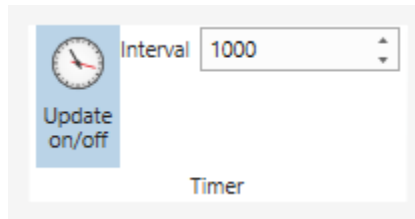
15.3.2 Adam 6000 DigIn


Reads digital electrical inputs from an Adam-60xx Ethernet Module from Adavantech.



This node reads digital signals from an Adam-60xx module.

If the clock button inside the node is pressed , the node polls the module regularly. The frequency of the polling is controlled with the timer interval setting on the Pipeline tab on the ribbon.



If the clock button inside the node is released , the module is not polled, and the node executes during regular pipeline updates only.

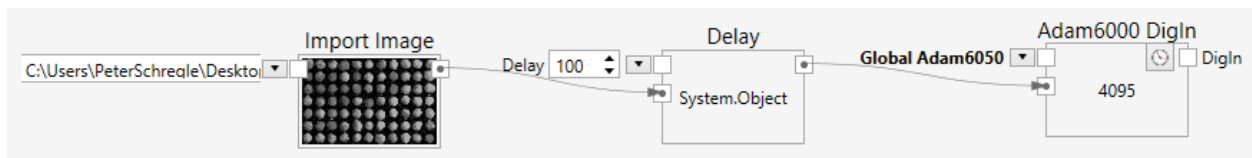
Inputs

Module (Type: Adam6000Module)

An Adam 60xx module instance.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.



Here, the digital input is read **after** the image has been imported.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

Outputs

DigIn (Type: Int32)

The digital input in the form of a 32 bit integer number.

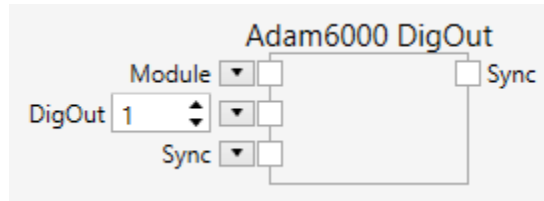
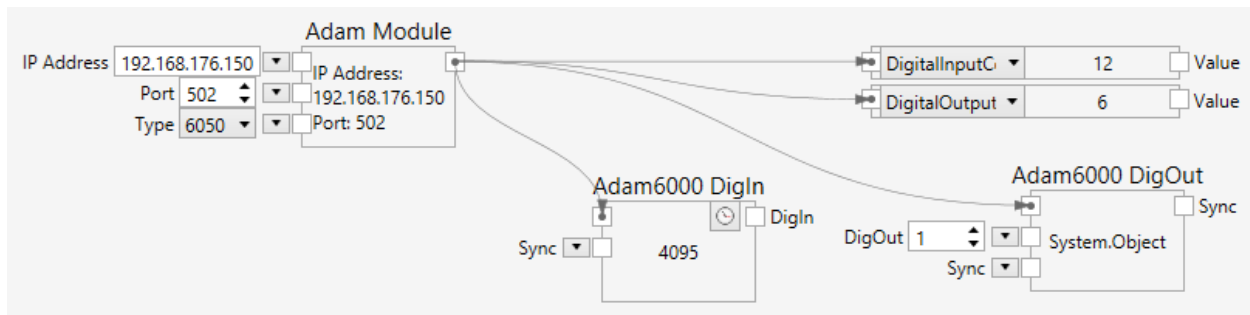
Sample

Here is an example:

15.3.3 Adam 6000 DigOut

Writes digital electrical outputs to an Adam-60xx ethernet module from Advantech.

This node writes digital signals to an Adam-60xx module.



Inputs

Module (Type: Adam6000Module)

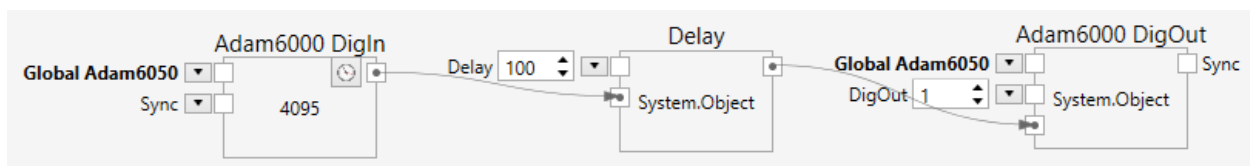
An Adam 60xx module instance.

DigOut (Type: Int32)

The numeric value that is written to the electrical outputs.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.



Here, the digital output is written **after** the digital input has been read.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

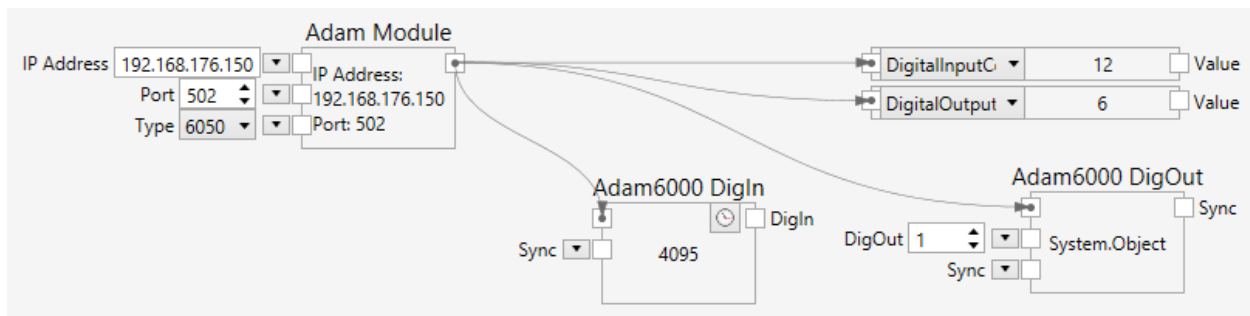
Outputs

Sync (Type: object)

Synchronizing object. It is used to establish an order of execution.

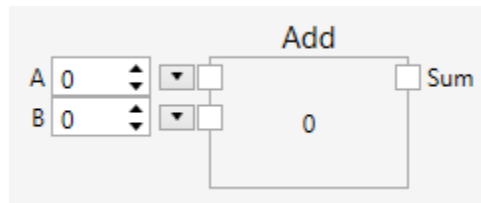
Sample

Here is an example:



15.3.4 Add

Adds two or more numbers.



Inputs

A (Type: Double)

The first operand.

B (Type: Double)

The second operand.

C...Z (Type: Double)

Additional operands, added on demand.

Outputs

Sum (Type: Double)

The sum of all operands.

Comments

This node calculates the sum of multiple operands.

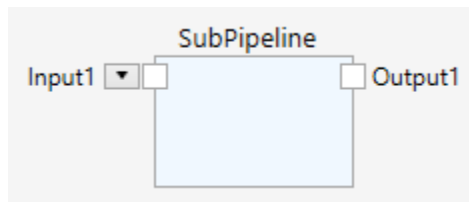
The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.5 Add Input

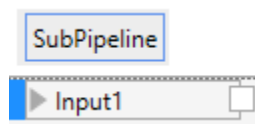
Input ports are the means by which data flows into a subpipeline. A subpipeline can have any number of input ports, including zero. Input ports must be named uniquely, two input ports of the same subpipeline cannot have the same name.

By default a sub-pipeline has one input, named **Input1**.

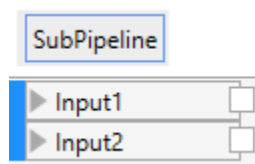
This is how this looks from the outside:



and here how it looks from the inside:



With the **Add Input (Ctrl-I)** command (or the **Ctrl-I** keyboard shortcut), you can add any number of additional inputs, that are automatically named as well.



To delete an input, right-click on the arrow in front of the name and choose the **Delete** command.

To rename an input, click on the name and type.

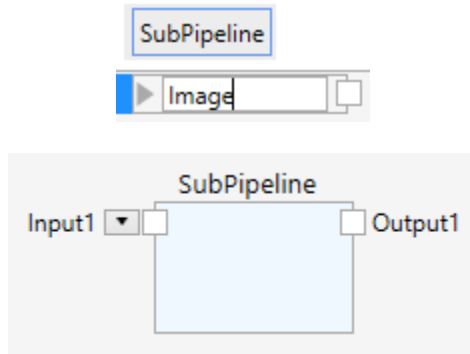
15.3.6 Add Output

Output ports are the means by which data flows out of a subpipeline. A subpipeline can have any number of output ports, including zero. Output ports must be named uniquely, two output ports of the same subpipeline cannot have the same name.

By default a sub-pipeline has one output, named **Output1**.

This is how this looks from the outside:

and here how it looks from the inside:



With the **Add Output (Ctrl-O)** command (or the `Ctrl-O` keyboard shortcut), you can add any number of additional outputs, that are automatically named as well.

To delete an ouotput, right-click on the arrow after of the name and choose the **Delete** command.

To rename an output, click on the name and type.

15.3.7 AffineTransform

Geometrically transforms an image by an affine transform.

Inputs

Image (Type: Image)

The input image.

Transform (Type: `NgImage.AffineMatrix`)

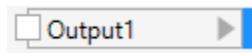
The affine transformation matrix.

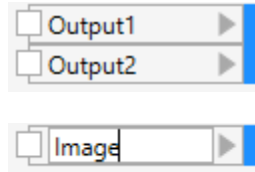
Filter (Type: `String`)

The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

Extend Size (Type: `bool`)

The geometric transformation may produce pixels that are outside of the input image bounds. If the parameter is set, the size of the output image is adapted to make room for these pixels. If the parameter is cleared, the size of the output image is kept the same as the input image.





Outputs

Transformed (Type: Image)

The output image.

15.3.8 And

Logical And of two or more boolean values.

Inputs

A (Type: Boolean)

The first operand.

B (Type: Boolean)

The second operand.

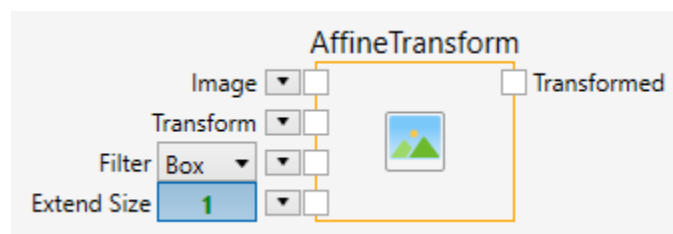
C...Z (Type: Boolean)

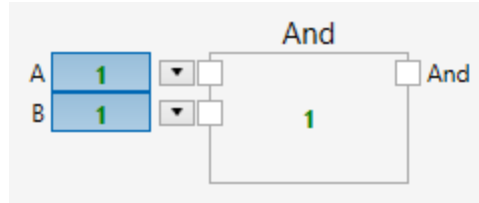
Additional operands, added on demand.

Outputs

And (Type: Boolean)

The logical And of all operands.





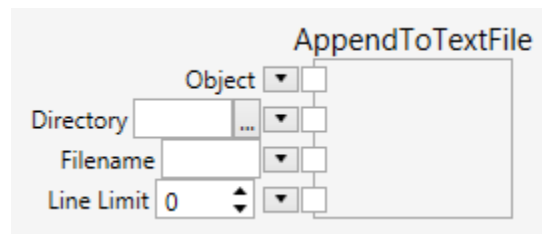
Comments

This node calculates the logical And of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.9 AppendToTextFile

Appends an line of text to a text file.



Inputs

Object (Type: Object)

The input element. The input element is converted to a text which is then appended as a line to the existing contents of the file.

Directory (Type: String)

A directory. This is preset with the user's documents folder.

Filename (Type: String)

A filename. This is preset with "data.txt".

LineLimit (Type: Int32)

The maximum number of lines the file is limited to. The default is 0 which means that there is no limit. Negative values will be treated the same as 0.

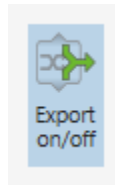
Comments

The **AppendToTextFile** node appends a line of text to a file. The input element can be any object, which is converted to a line of text.

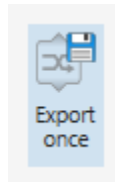
The directory and filename of the file can be specified. If no directory is used, the node uses the user's documents folder. If no filename is used, the node uses the name "data.txt". If the directory does not exist, it will be created.

The maximum number of lines that the file should have after the append can be specified. If the maximum number of lines would be exceeded, the system still appends a line to the file, but at the same time it also removes the line at the beginning of the file.

The node only executes, if the **Export On/Off** mode has been turned on,



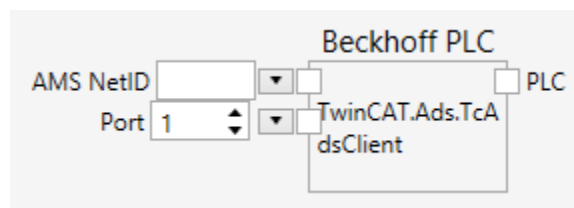
or if the **Export Once** button is clicked.



Otherwise, the node does not execute and has no effect.

15.3.10 Beckhoff PLC

Establish a connection to a Beckhoff TwinCAT PLC, which allows you to read and write to PLC symbols.



This node initializes the PLC. If you connect its output to a **GetProperty** node, you can gather information about the PLC. Usually you would connect the output to a **Read Symbol** node to read symbol data, or a **Write Symbol** node to write symbol data.

Inputs

AMS NetID (Type: string)

The identifier of the Beckhoff PLC.

Port (Type: Int32)

The port of the Beckhoff PLC.

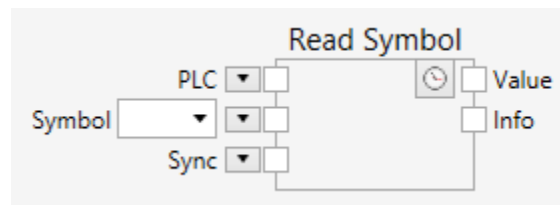
Outputs**Module (Type: TwinCAT.Ads.AdsClient)**


The Beckhoff PLC (ADS) instance.

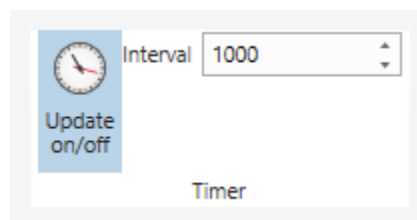
Since establishing a connection to a PLC is often an operation with a somewhat global character, the **Beckhoff PLC** node is sometimes put into the pipeline globals or the system globals.


15.3.11 Read Symbol

Reads symbol data from a Beckhoff PLC.



If the clock button inside the node is pressed , the node polls the PLC regularly. The frequency of the polling is controlled with the timer interval setting on the Pipeline tab on the ribbon.



If the clock button inside the node is released , the PLC is not polled, and the node executes during regular pipeline updates only.

Inputs**PLC (Type: TwinCAT.Ads.AdsClient)**

A Beckhoff PLC (ADS) instance.

Symbol (Type: string)

A PLC symbol name.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

Outputs**Value (Type: String)**

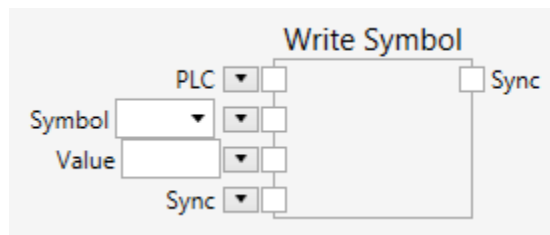
The symbol value.

Info (Type: TwinCAT.Ads.ITcAdsSymbol)

Additional symbol information.

15.3.12 Write Symbol

Writes symbol data to a Beckhoff PLC.



This node writes digital signals to an OpenLayers module.

Inputs**PLC (Type: TwinCAT.Ads.AdsClient)**

A Beckhoff PLC (ADS) instance.

Symbol (Type: string)

A PLC symbol name.

Value (Type: string)

The value that is written to the Beckhoff PLC symbol.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

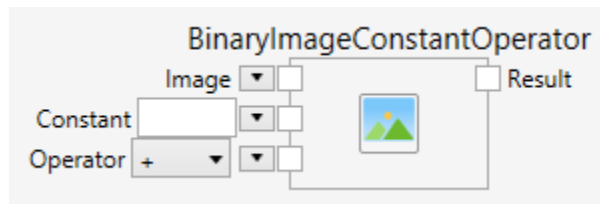
Outputs

Sync (Type: object)

Synchronizing object. It is used to establish an order of execution.








15.3.13 BinaryImageConstantOperator

Applies a binary point operation between an image and a constant. Possible operations can be grouped into arithmetic, logic and comparison fields. Binary point operations are applied pixel by pixel, the same way for every pixel.



Arithmetic Operators

The following operators are available:

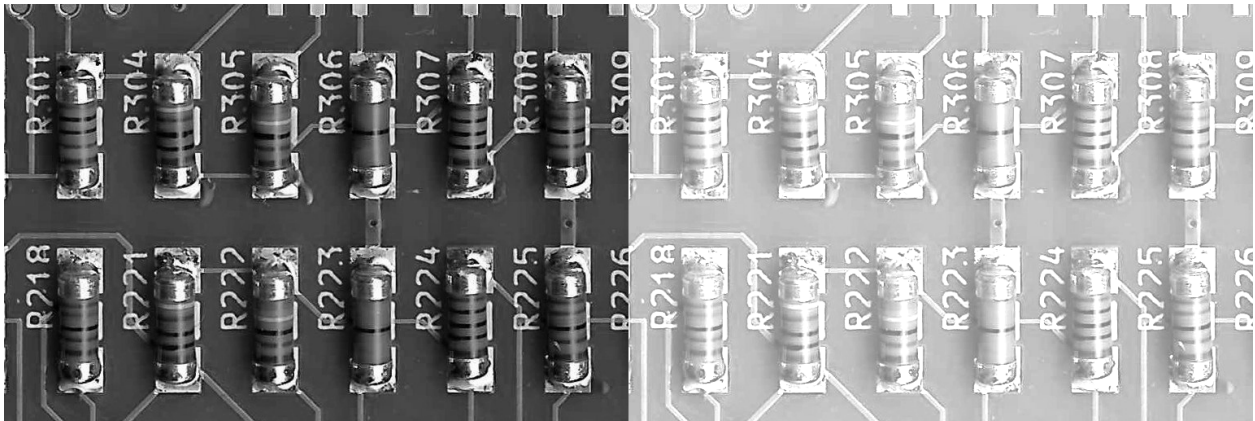
-  Add
-  Subtract
-  Difference
-  Multiply
-  Divide
-  Multiply (Blend)
-  Divide (Blend)

Add

Add with saturation.

Image + Constant -> Result

Subtract



Subtract with saturation.

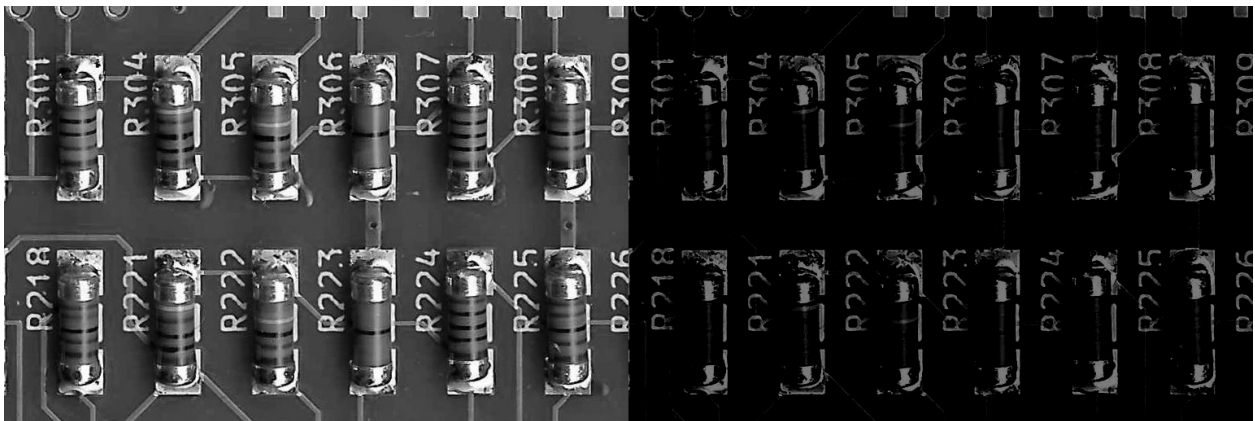


Image - Constant -> Result

Difference

Difference with saturation.

abs (Image - Constant) -> Result

Multiply

Multiply without saturation.

Image * Constant -> Result

Divide

Divide without saturation.

Image / Constant -> Result

Multiply (Blend)

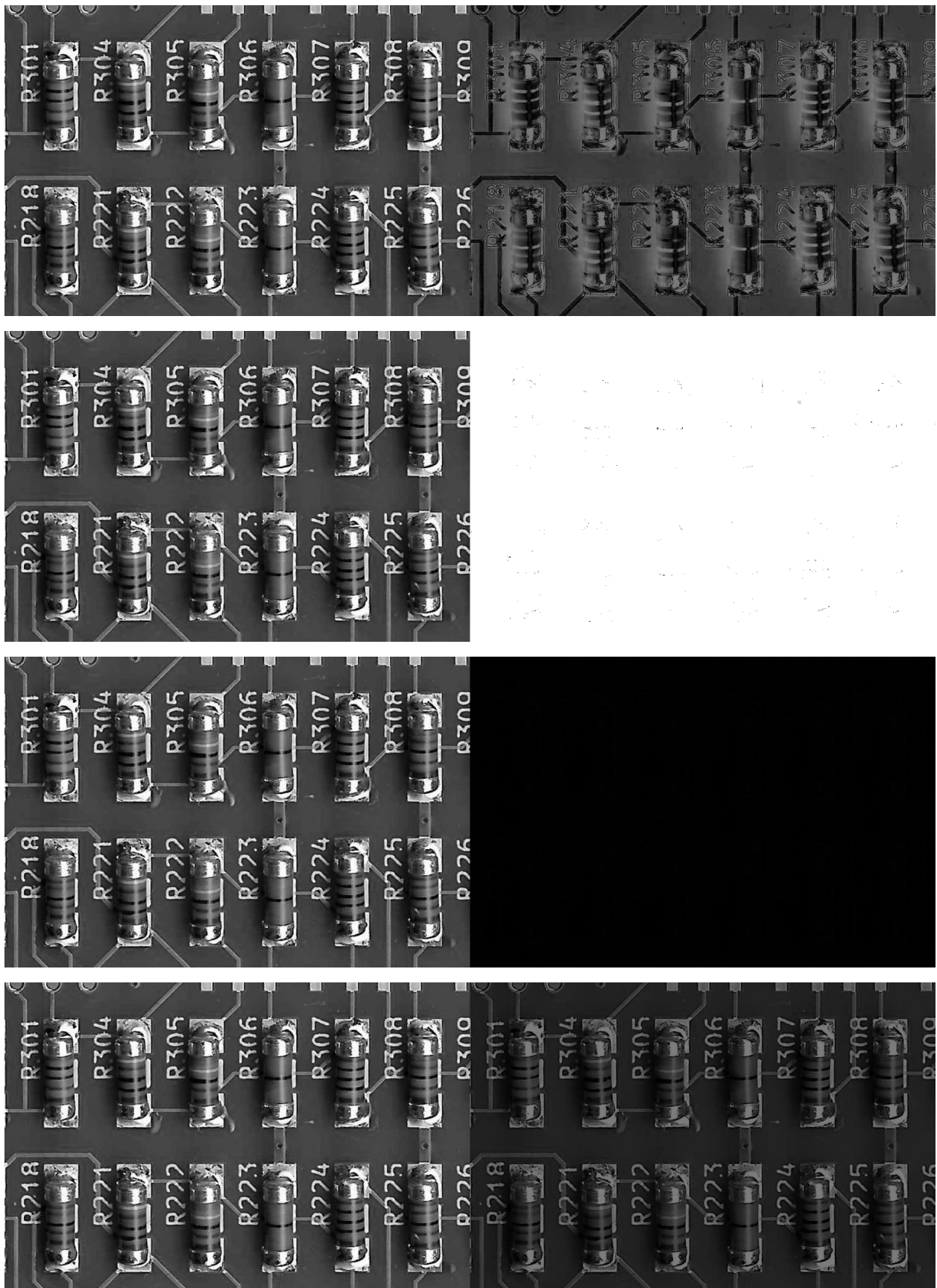
Multiply with saturation.

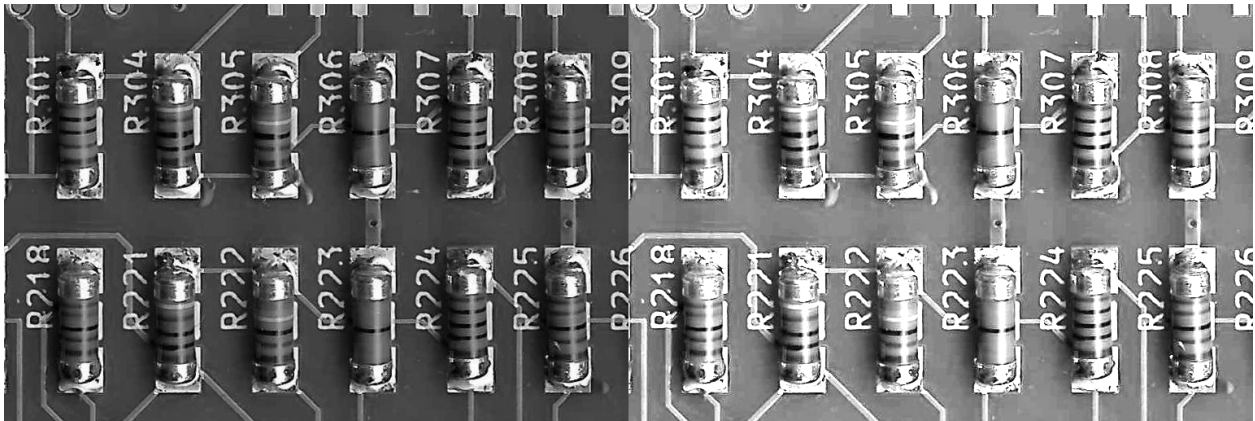
Image * Constant -> Result

Divide (Blend)

Divide with saturation.

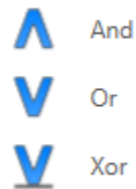
Image / Constant -> Result





Logic Operators

The following operators are available:



And

Logical And.

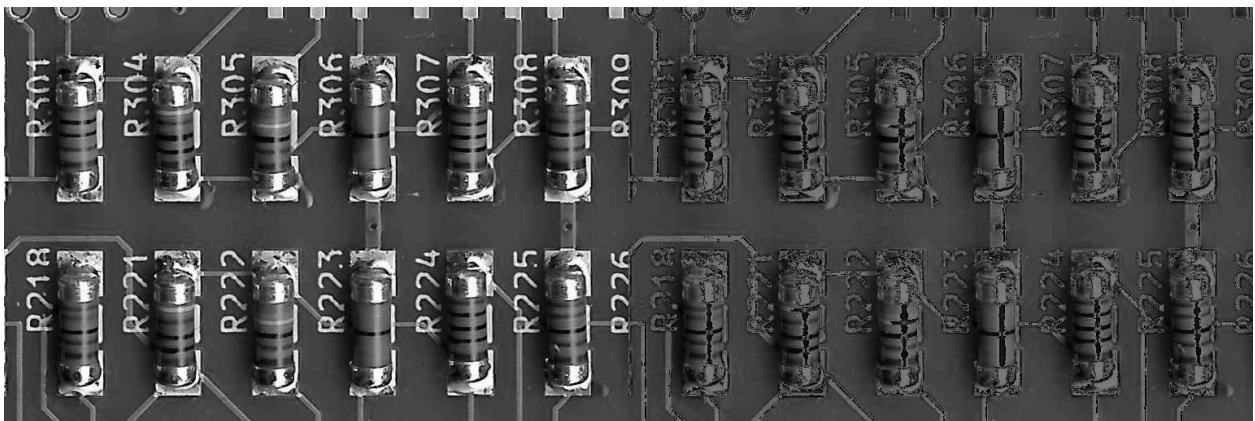


Image & Constant -> Result

Or

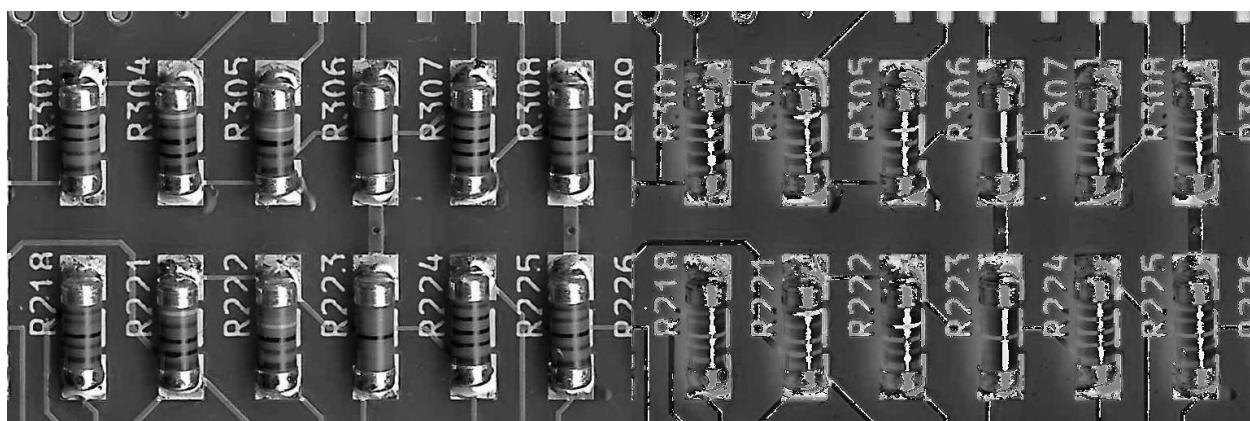
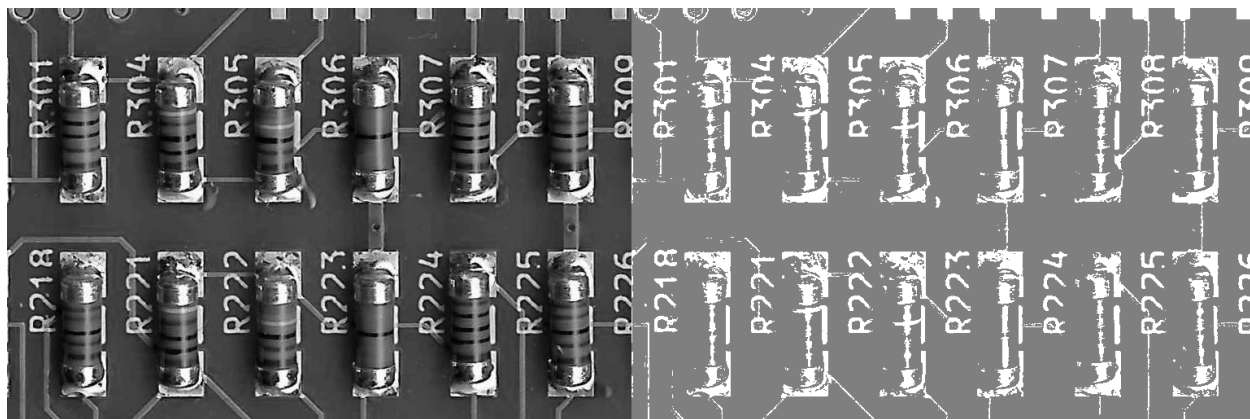
Logical Or.

Image | Constant -> Result

Xor






Logical Xor.

Image ^ Constant -> Result



Comparison Operators

The following operators are available:

	Bigger
	Bigger or Equal
	Equal
	Smaller
	Smaller or Equal

Smaller

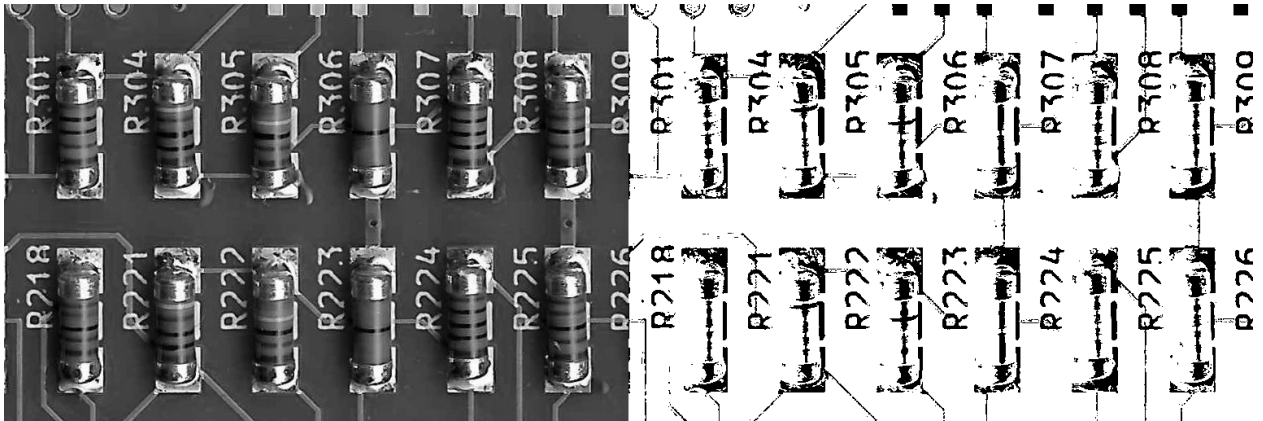
Compare smaller.

Image < Constant -> Result

Smaller or Equal

Compare smaller or equal.

Image <= Constant -> Result



Equal

Compare equal.

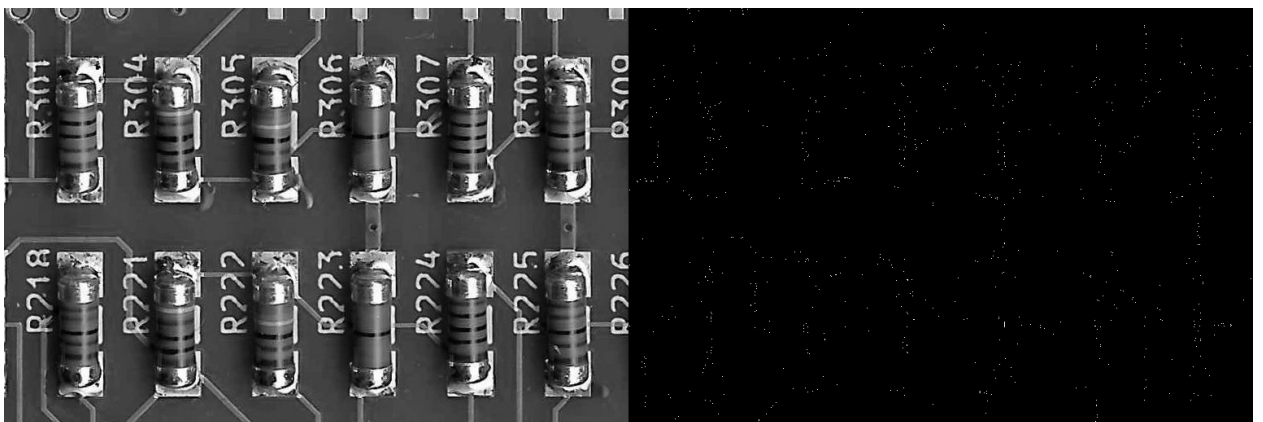


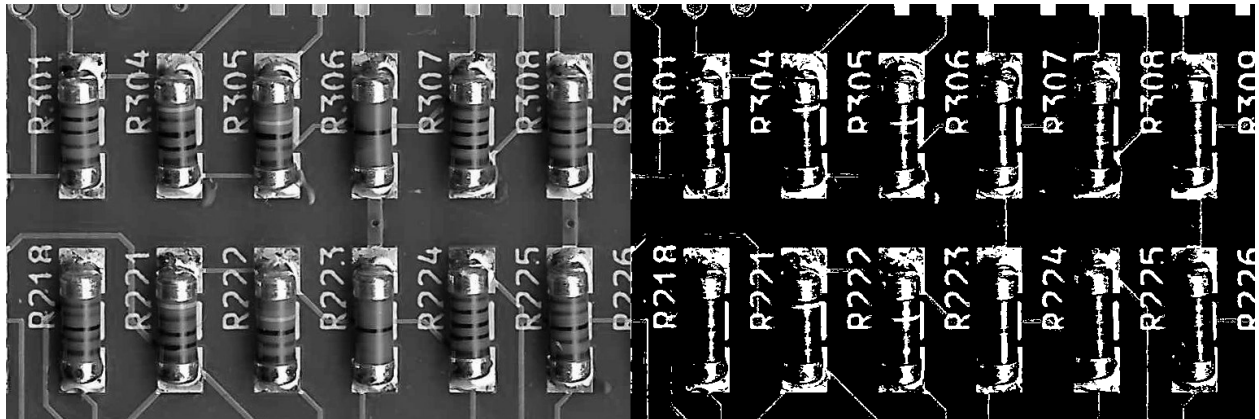
Image == Constant -> Result

Bigger or Equal

Compare bigger or equal.

Image >= Constant -> Result

Bigger



Compare bigger.

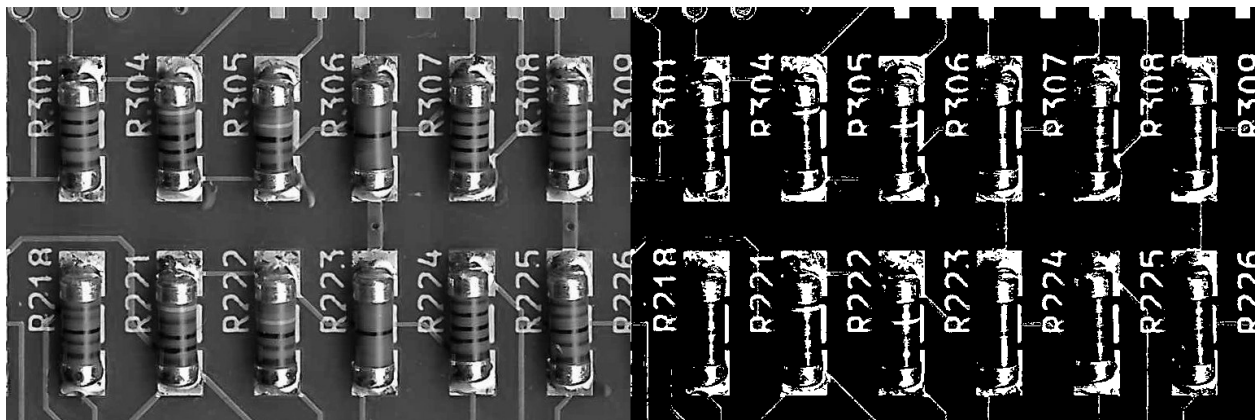


Image > Constant -> Result

Min/Max Operators

The following operators are available:



Min

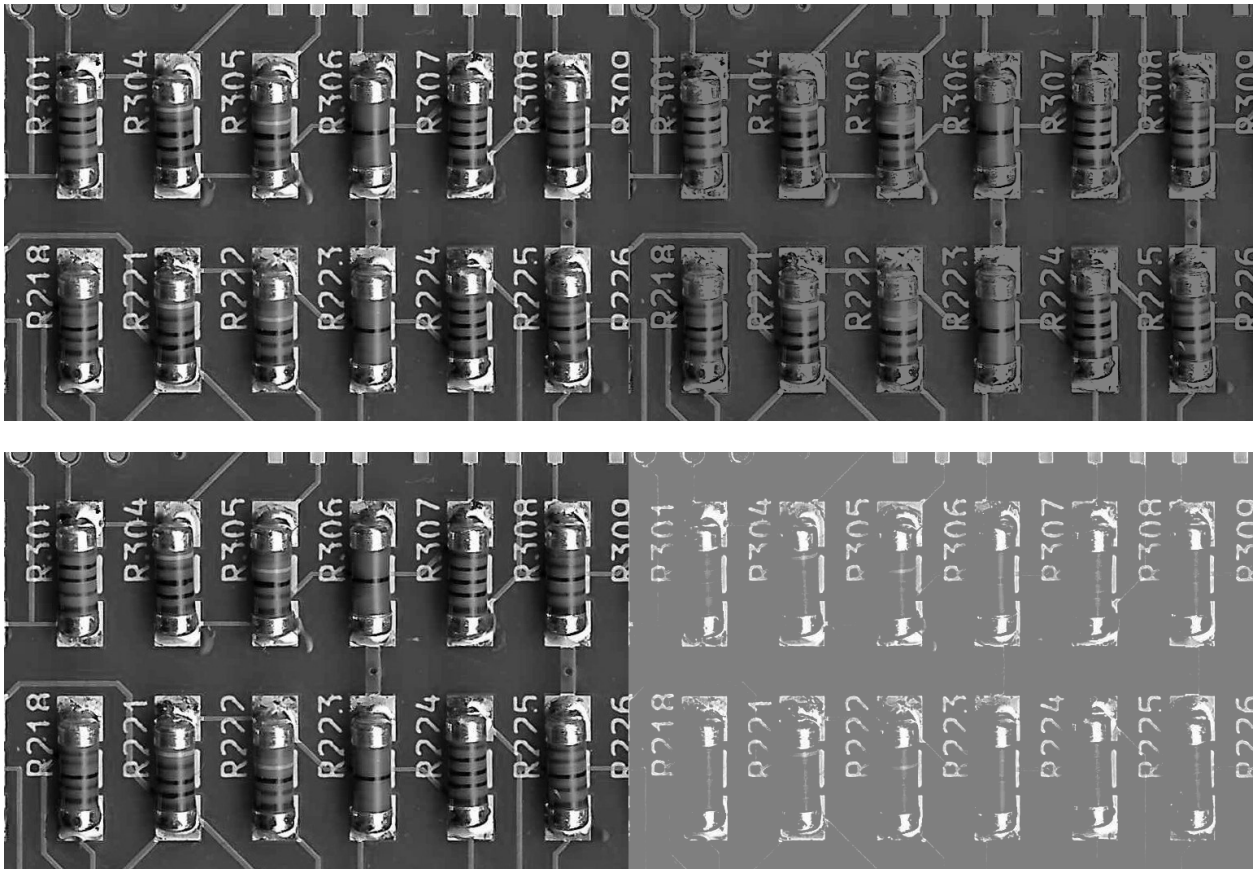
Minimum.

`min(Image, Constant) -> Result`

Max

Maximum.

`max(Image, Constant) -> Result`



Inputs

Image (Type: Image)

The input image.

Constant (Type: object)

The constant.

Operator (Type: string)

Specifies the operator.

operator	operation
+	add
-	subtract
diff	difference
*	multiply
/	divide
*_blend	multiply (blend)
/_blend	divide (blend)
&	and
	or
^	xor
<	smaller
<=	smaller or equal
==	equal
>=	bigger or equal
>	bigger
min	minimum
max	maximum

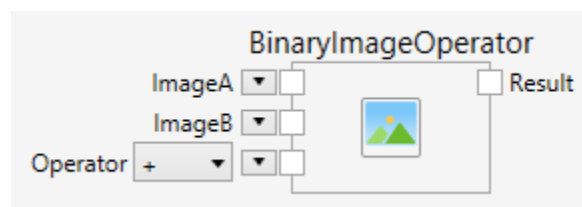
Outputs

Result (Type: Image)

The result image.

15.3.14 BinaryImageOperator

Applies a binary point operation between two images. Possible operations can be grouped into arithmetic, logic and comparison fields. Binary point operations are applied pixel by pixel, the same way for every pixel, for corresponding pixels of two images.



Arithmetic Operators

The following operators are available:








Add

Add with saturation.

Image + Gradient -> Result

Subtract

Subtract with saturation.

-  Add
-  Subtract
-  Difference
-  Multiply
-  Divide
-  Multiply (Blend)
-  Divide (Blend)

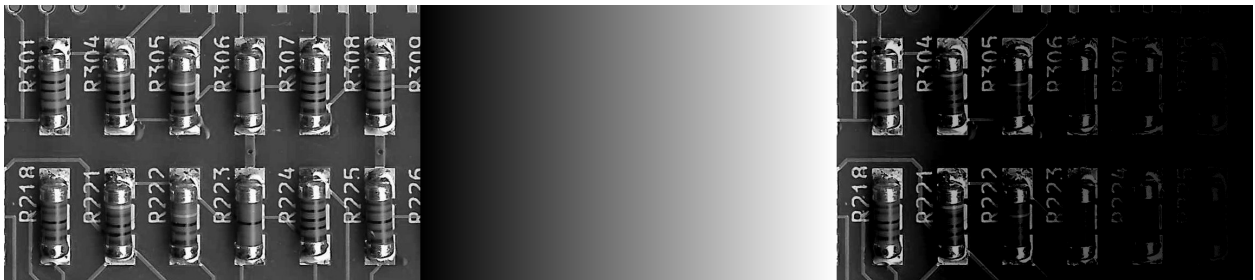
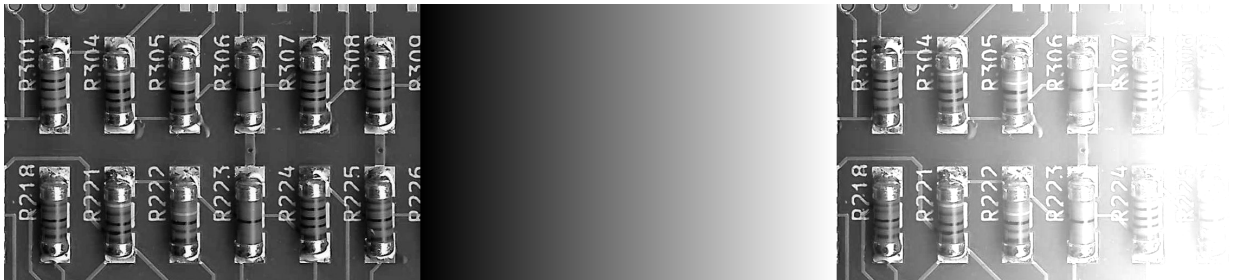
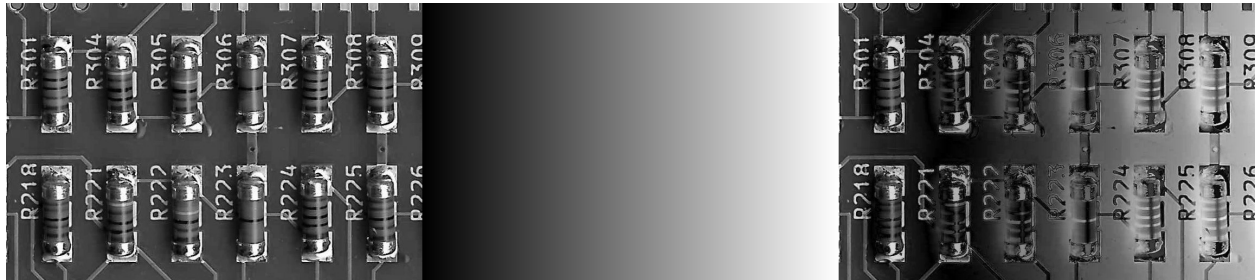


Image - Gradient -> Result

Difference

Difference with saturation.



abs (Image - Gradient) -> Result

Multiply

Multiply without saturation.

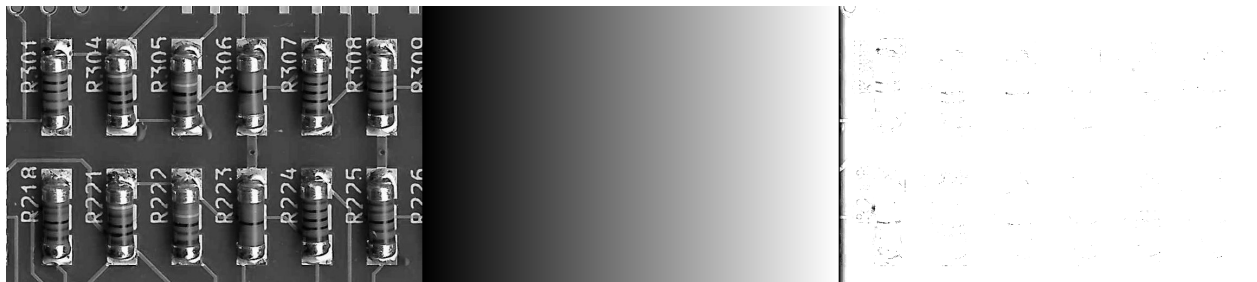


Image * Gradient -> Result

Divide

Divide without saturation.

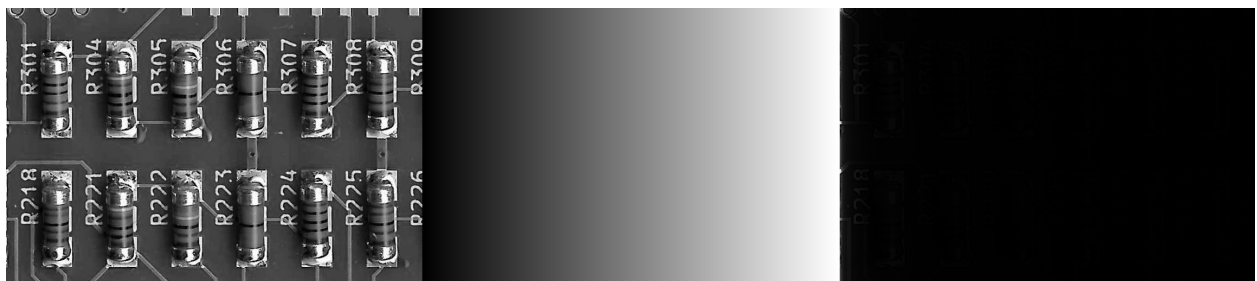


Image / Gradient -> Result

Multiply (Blend)

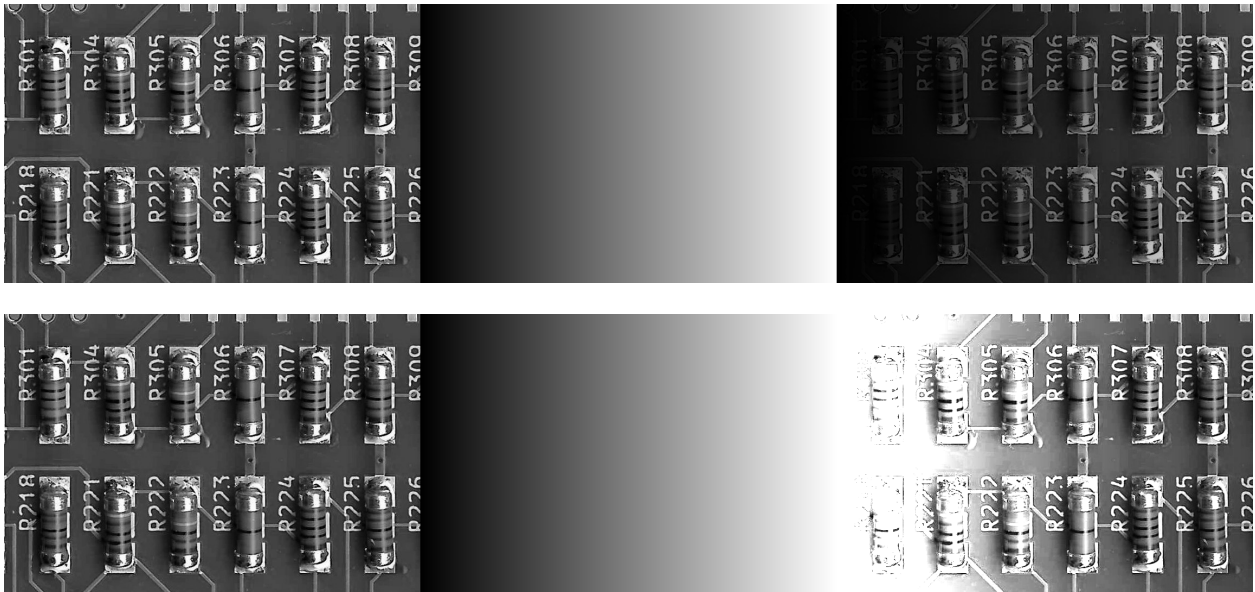
Multiply with saturation.

Image * Gradient -> Result

Divide (Blend)

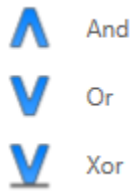
Divide with saturation.

Image / Gradient -> Result



Logic Operators

The following operators are available:



And

Logical And.

Image & Gradient -> Result

Or

Logical Or.

Image | Gradient -> Result

Xor

Logical Xor.

Image ^ Gradient -> Result

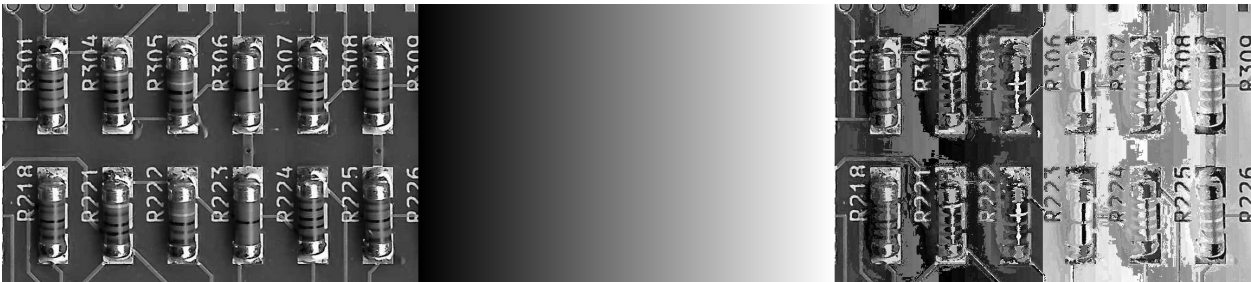
Comparison Operators






The following operators are available:

Smaller

Compare smaller.

Image < Gradient -> Result



-  Bigger
-  Bigger or Equal
-  Equal
-  Smaller
-  Smaller or Equal



Smaller or Equal

Compare smaller or equal.

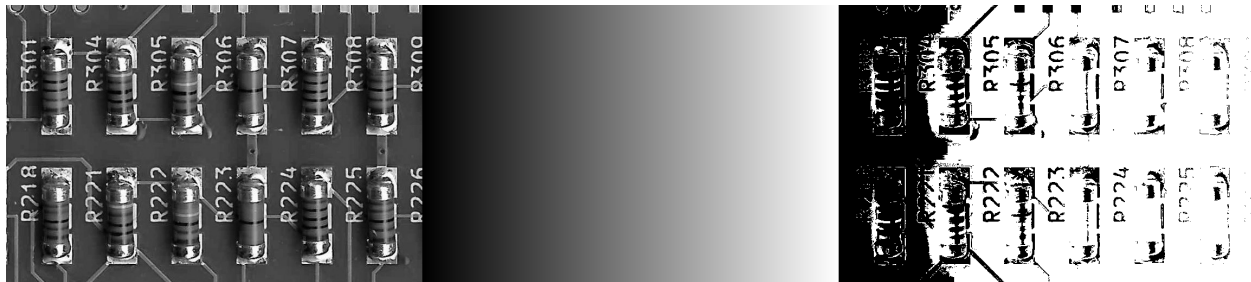


Image <= Gradient -> Result

Equal

Compare equal.

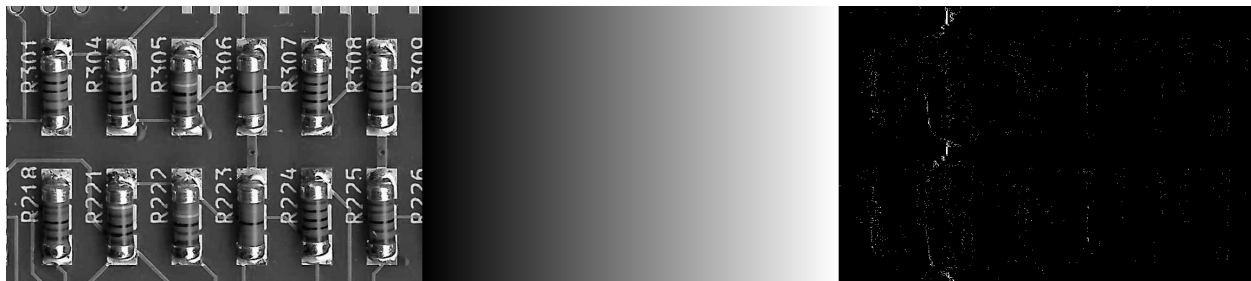


Image == Gradient -> Result

Bigger or Equal

Compare bigger or equal.

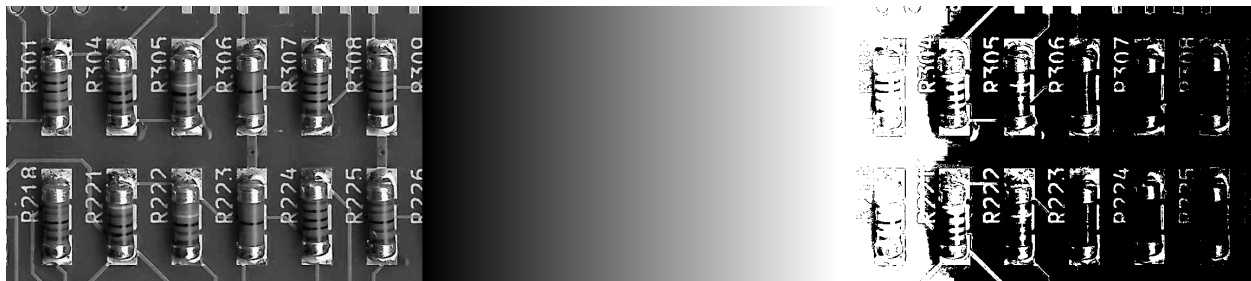


Image >= Gradient -> Result

Bigger

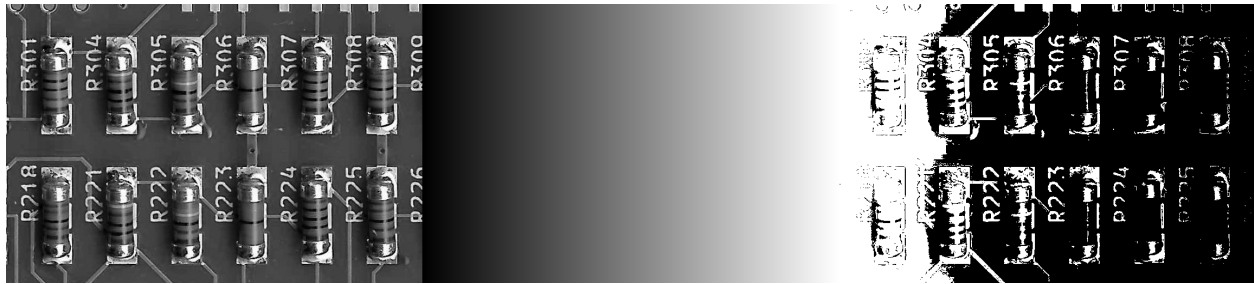
Compare bigger.

Image > Gradient -> Result

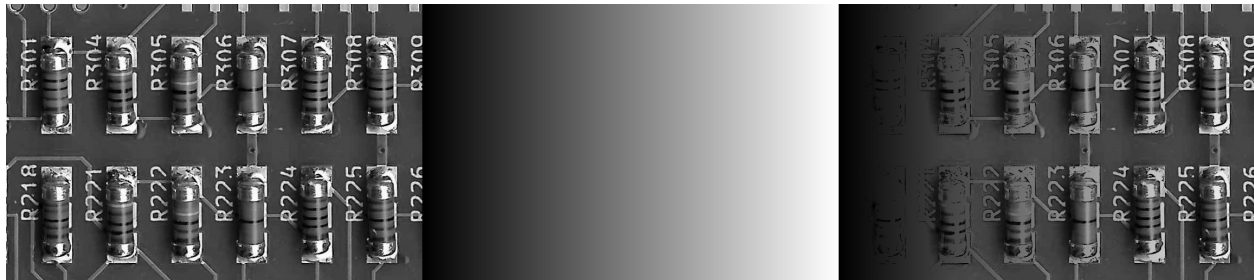
Min/Max Operators

The following operators are available:

Min



Minimum.



`min(Image, Gradient) -> Result`

Max

Maximum.

`max(Image, Gradient) -> Result`

Inputs

ImageA (Type: Image)

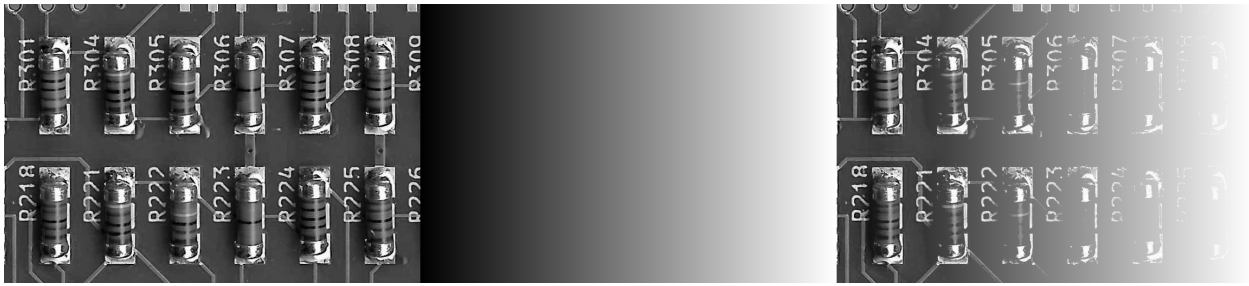
The first input image.

ImageB (Type: Image)

The second input image.

Operator (Type: string)

Specifies the operator.



operator	operation
+	add
-	subtract
diff	difference
*	multiply
/	divide
*_blend	multiply (blend)
/_blend	divide (blend)
&	and
	or
^	xor
<	smaller
<=	smaller or equal
==	equal
>=	bigger or equal
>	bigger
min	minimum
max	maximum

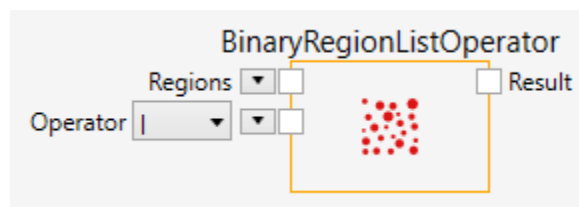
Outputs

Result (Type: Image)

The result image.

15.3.15 BinaryRegionListOperator

Applies a binary operation between the regions in a list.



The following operators are available: union and intersection.

Union (|)

Calculates the union of all regions in the list.

Intersection (&)

Calculates the intersection of all regions in the list.

Inputs

Regions (Type: RegionList)

The list of input regions.

Operator (Type: string)

Specifies the operator.

operator	operation
	union
&	intersection

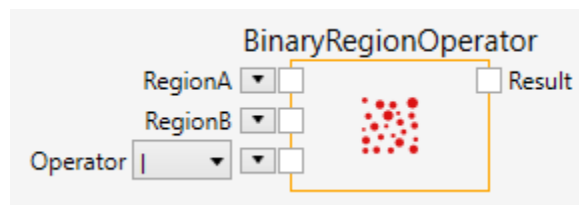
Outputs

Result (Type: Region)

The result region.

15.3.16 BinaryRegionOperator

Applies a binary operation between two regions.



The following operators are available:

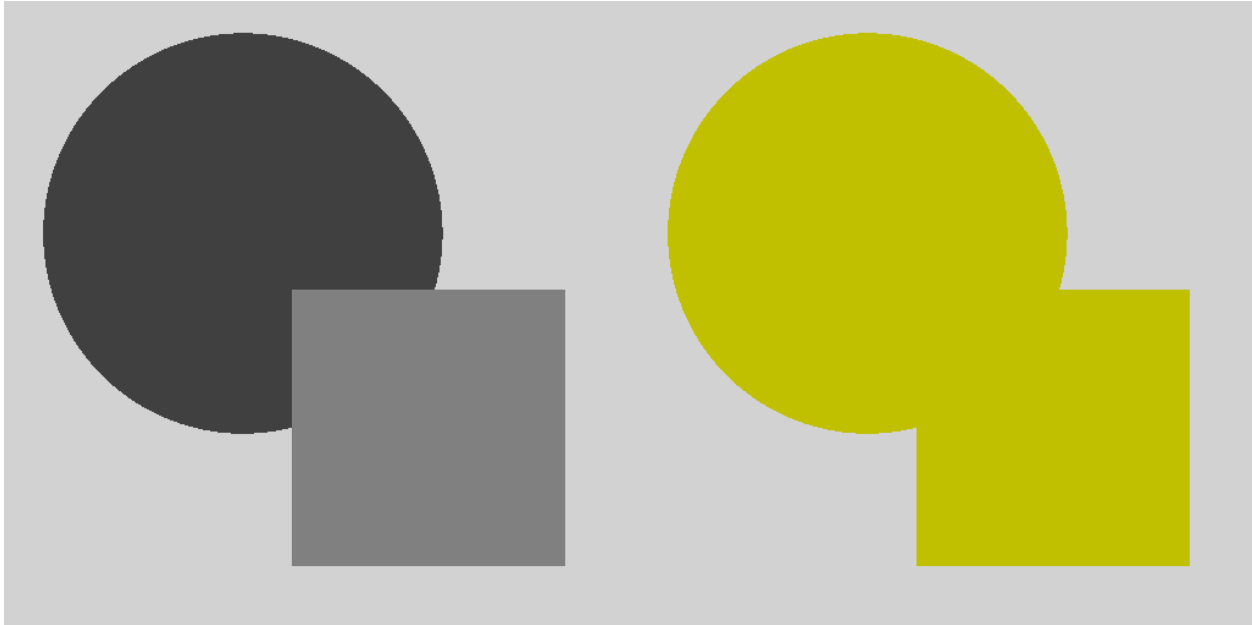


Union (|)

Calculates the union (yellow) of two regions A (dark), B (bright).

$A \cup B \rightarrow \text{Union}$

Intersection (&)



Calculates the intersection (yellow) of two regions A (dark), B (bright).



A & B -> Intersection

Difference (\)

Calculates the difference (yellow) of two regions A (dark), B (bright).

A B -> Difference

B A -> Difference



Inputs

RegionA (Type: Region)

The first input region.

RegionB (Type: Region)

The second input region.

Operator (Type: string)

Specifies the operator.

operator	operation
	union
&	intersection
\	difference

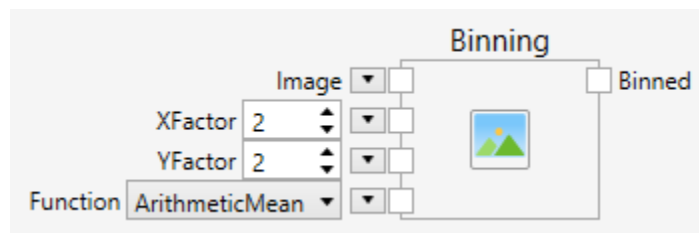
Outputs

Result (Type: Region)

The result region.

15.3.17 Binning

Reduces an image in size by combining several neighboring pixels.



Inputs

Image (Type: Image)

The input image.

XFactor (Type: Int32)

The horizontal binning factor.

YFactor (Type: Int32)

The vertical binning factor.

Function (Type: String)

The binning function.

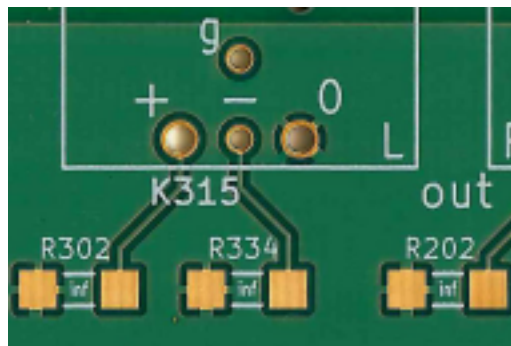
Outputs**Binned (Type: Image)**

The output image.

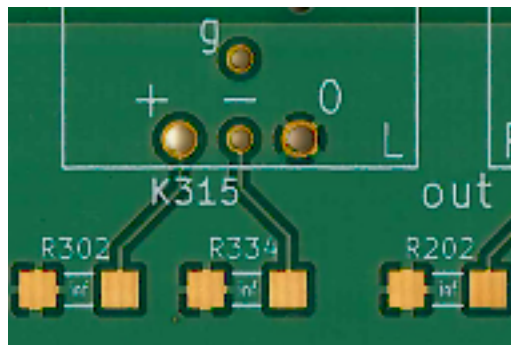
Comments

This function reduces an image in size by combining several neighboring pixels. The binning factor in horizontal and vertical direction can be different. Different mathematical methods for combination can be selected with the binning function.

ArithmeticMean: the arithmetic mean of the neighboring pixels is used.



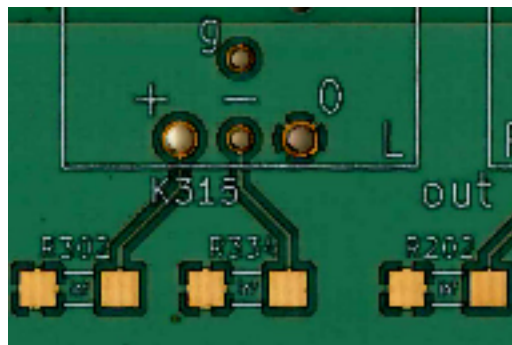
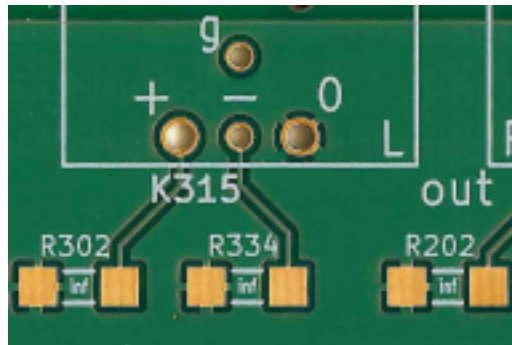
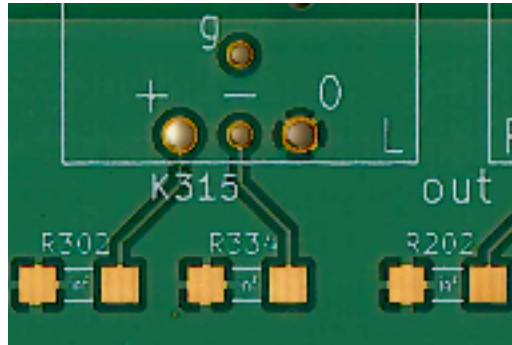
GeometricMean: the geometric mean of the neighboring pixels is used.



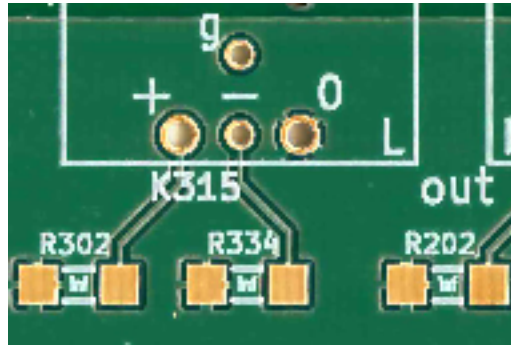
HarmonicMean: the harmonic mean of the neighboring pixels is used.

QuadraticMean: the quadratic mean of the neighboring pixels is used.

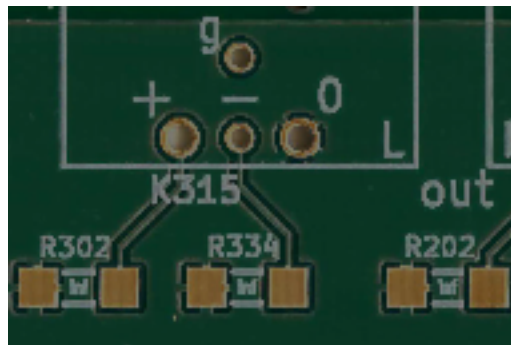
Minimum: the minimum of the neighboring pixels is used.



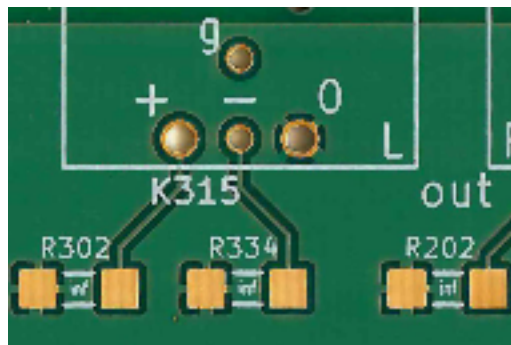
Maximum: the maximum of the neighboring pixels is used.



Midrange: the midrange of the neighboring pixels is used.



Median: the median of the neighboring pixels is used.

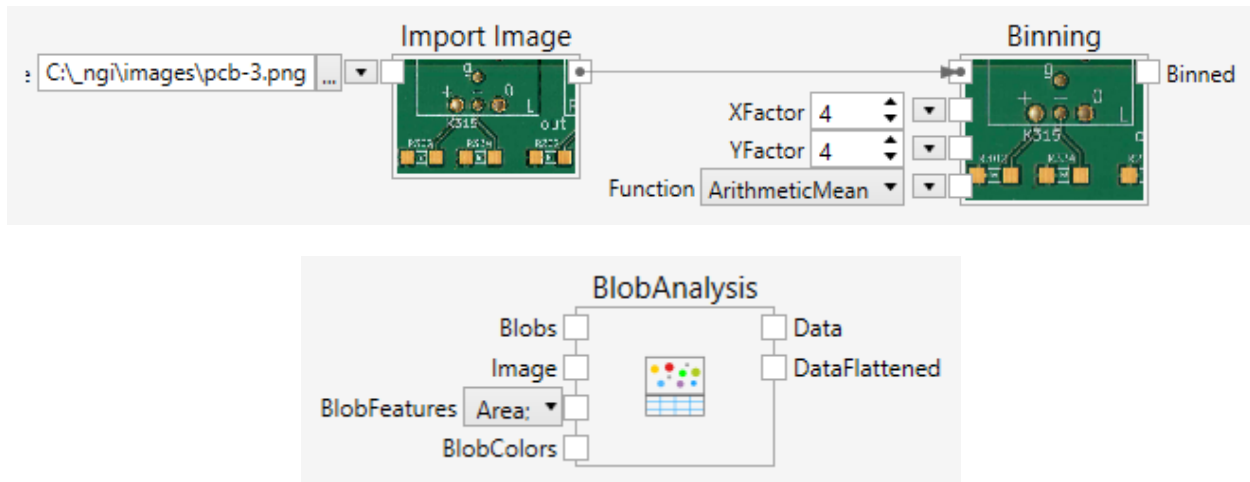


Sample

Here is an example that bins an image by a factor of 4 in both horizontal and vertical directions.

15.3.18 Blob Analysis

Measures features of segmented blobs of pixels.



Inputs

Blobs (Type: RegionList)

The regions that define the blobs..

Image (Type: Image)

An optional image. This is only needed, if densitometric features should be calculated, because they need pixel values.

BlobFeatures (Type: List [String])

A list of strings which defines the features that should be calculated. The full set of features is described in the Blob Analysis chapter of the nVision User Guide.

BlobColors (Type: “)

An optional list of colors that define the colors that should be used to color the different regions, so that they can be visually distinguished.

Outputs

Data (Type: ‘System.Data.DataTable’)

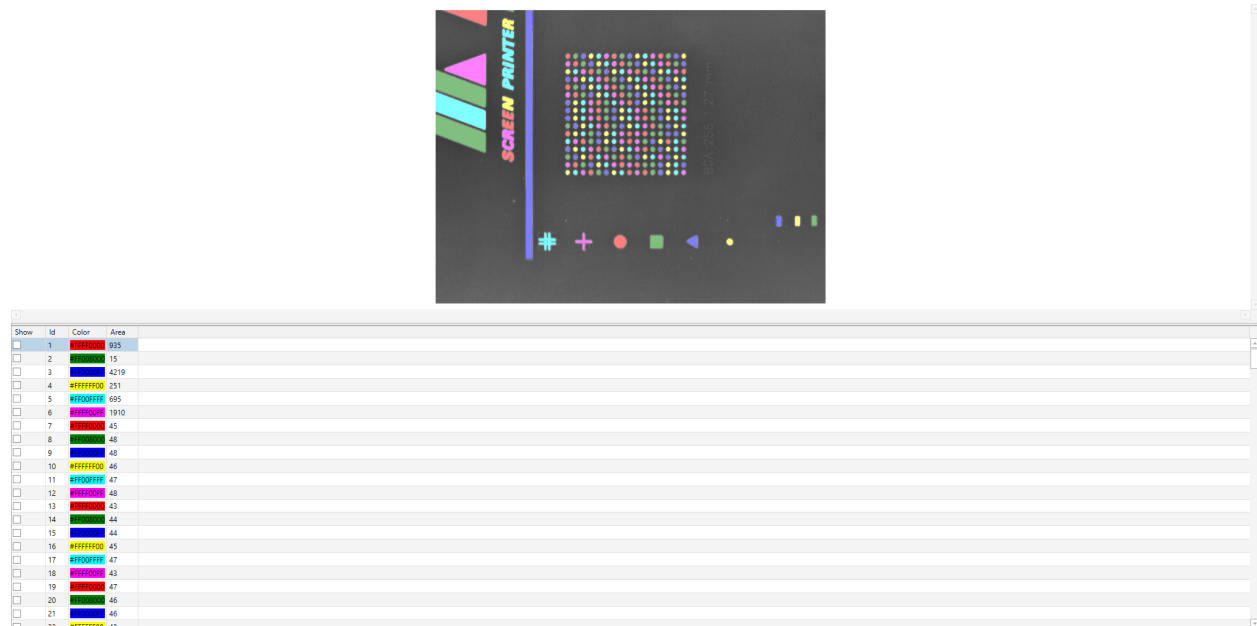
The blob analysis result data.

Comments

The **Blob Analysis** node calculates the features for a list of blobs.

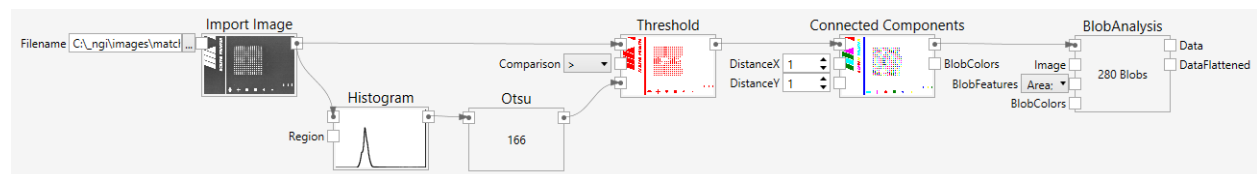
The set of features can be selected with the BlobFeatures input.

The results of the blob analysis are displayed by default (if connected to the output of a sup-pipeline) as a colored overlay on top of an image in the upper half and a table in the lower half. The table shows the numeric values of the calculated features. Colored regions in the image can be clicked with the mouse to find the values in the table (the row will be highlighted). Vice versa, if the checkboxes in the Show column are marked and geometric features (like Bounds and/or Equivalent Ellipse) are calculated, the blob is highlighted by the geometric feature (i.e. a bounding box or the equivalent ellipse).



Sample

Here is an example that calculates the average of an image.



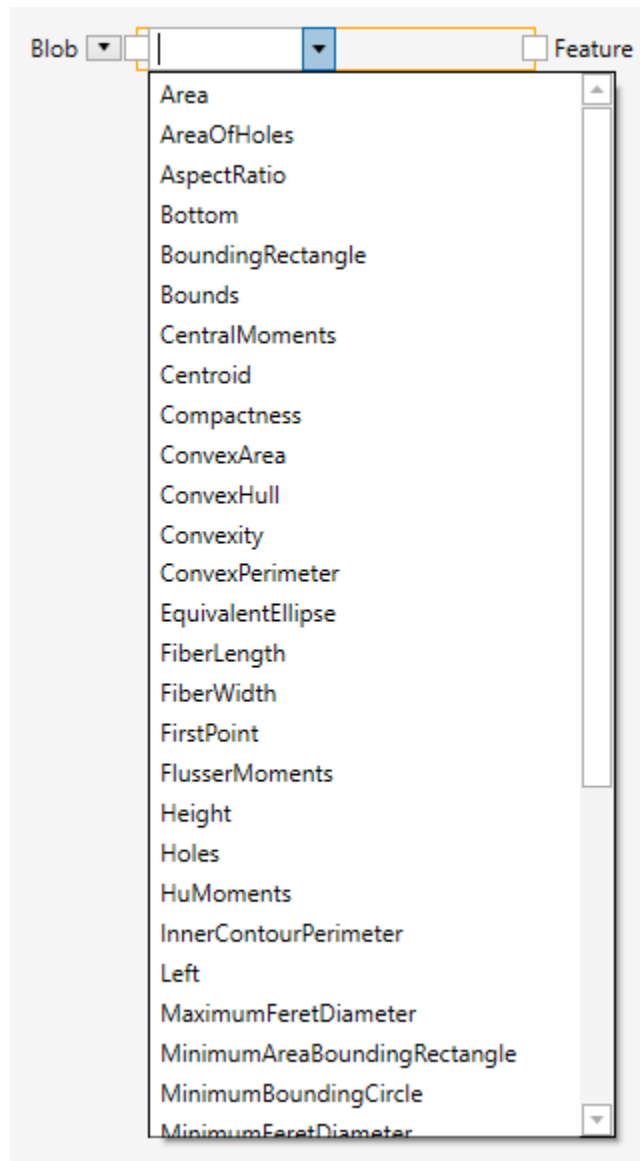
15.3.19 Blob Feature

Measures features of a region.

Inputs

Blob (Type: Region)

The region that defines the blob.



Outputs

Feature

The blob analysis result data. The type of the result depends on the selected feature.

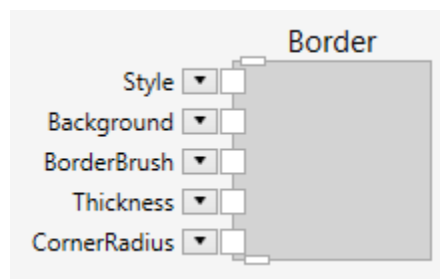
Comments

The **Blob Feature** node calculates the feature for a single region.

The features can be selected inside the node. The full set of features is described in the Blob Analysis chapter of the nVision User Guide.

15.3.20 HMI Border

The **Border** puts a border around its child.



At the bottom of the **Border** node is a pin that allows it to connect one child.

Inputs

Style (Type: `Style`)

Optional styling of the **Border**.

The **Border** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background* styles.

Background (Type: `SolidColorBrushByte`)

The background of the **Border**.

BorderBrush (Type: `SolidColorBrushByte`)

The brush of the **Border** frame.

Thickness (Type: `Thickness`)

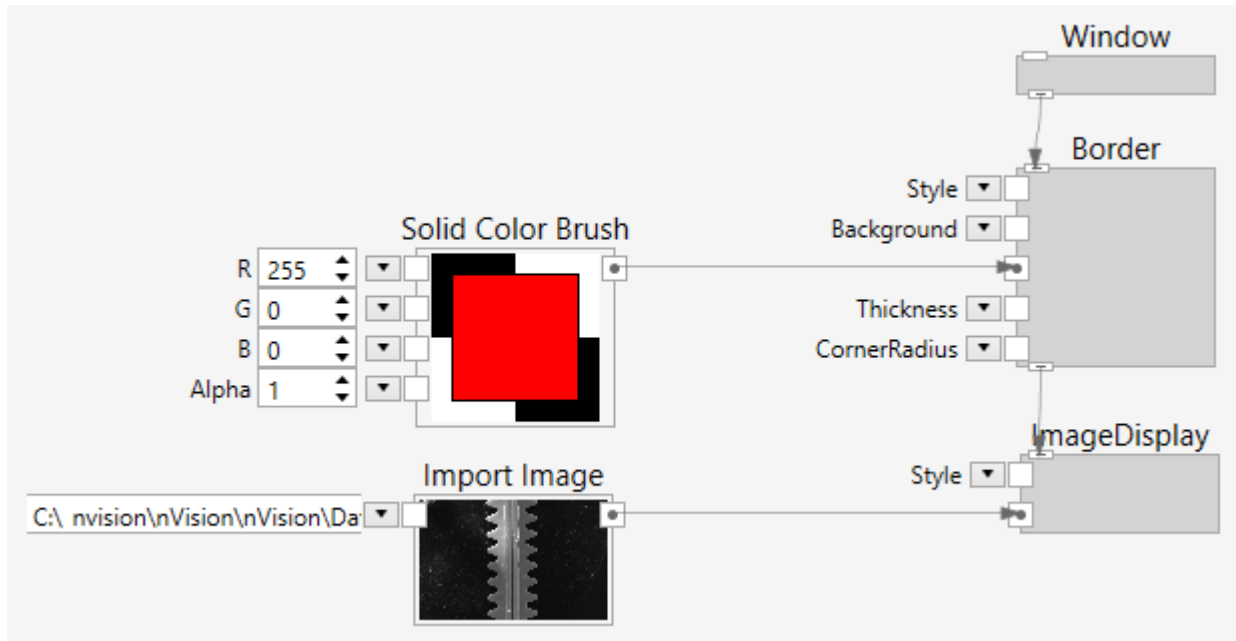
The thickness of the **Border** frame.

CornerRadius (Type: 'CornerRadius')

The corner radius of the **Border** frame.

Example

Here is an example that puts a border around an image. This definition:



creates the following user interface:

15.3.21 Average

Calculates the average value of all pixels in a buffer.

Inputs**Buffer (Type: Buffer)**

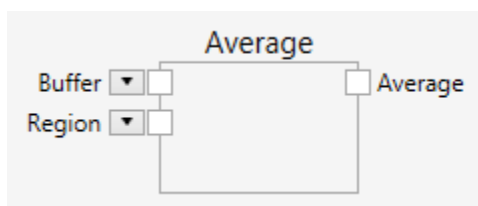
The input buffer.

Region (Type: Region)

An optional region.

Outputs**Average (Type: Object)**

The average value.



Comments

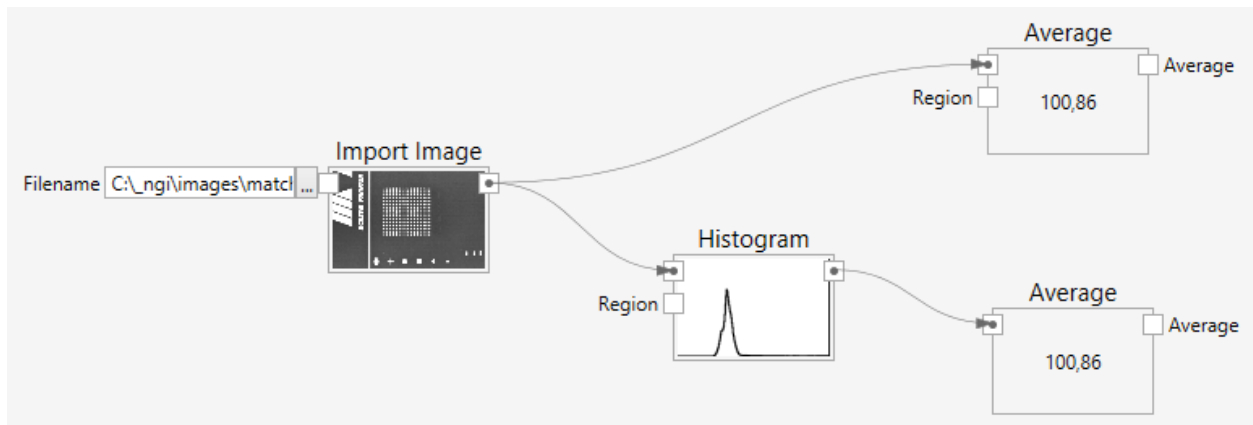
The **Average** node calculates the average value of a buffer. It sums all pixels and divides by the number of pixels.

If the *Region* input is connected, the calculation of the average is constrained to within the region only, otherwise the whole buffer is used.

If a color buffer is used, the average is calculated channel-wise.

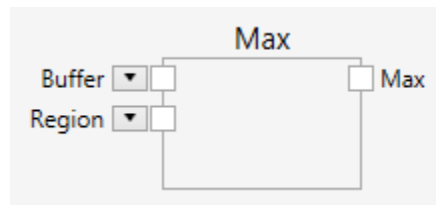
Sample

Here is an example that calculates the average of an image.



15.3.22 Maximum

Calculates the maximum value of a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

Maximum (Type: Object)

The maximum value.

Comments

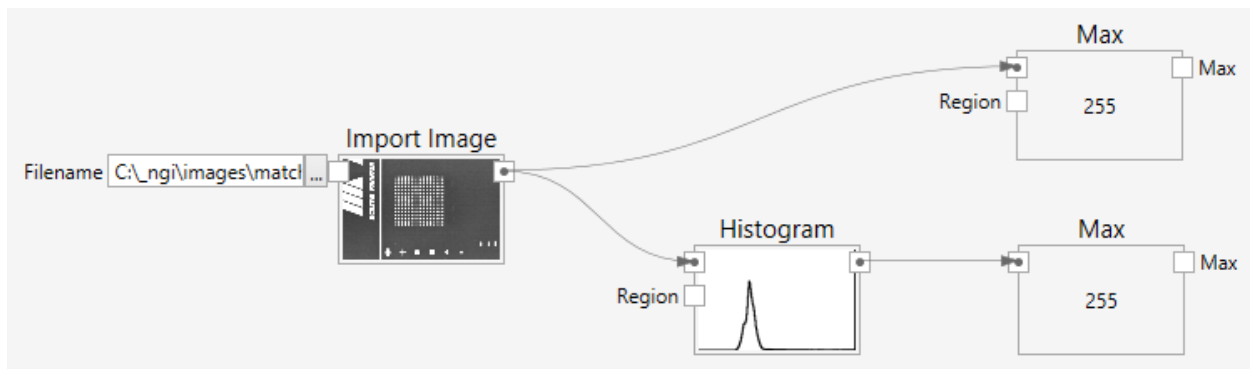
The **Maximum** node calculates the maximum value of a buffer.

If the *Region* input is connected, the calculation of the maximum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the maximum length color vector is calculated.

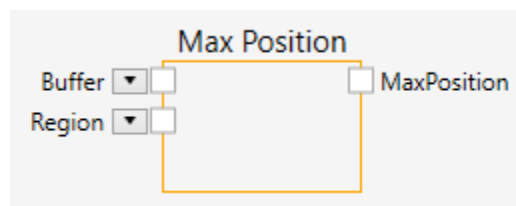
Sample

Here is an example that calculates the maximum of an image.



15.3.23 Max Position

Calculates the position of the maximum value of a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs**MaxPosition (Type: Index3d)**

The position of the maximum value.

Comments

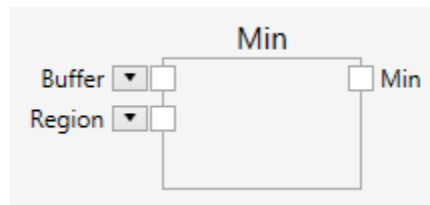
The **Max Position** node calculates the (first occurrence of the) position of the maximum value of a buffer.

If the *Region* input is connected, the calculation of the maximum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the maximum length color vector is calculated.

15.3.24 Minimum

Calculates the minimum value of a buffer.

**Inputs****Buffer (Type: Buffer)**

The input buffer.

Region (Type: Region)

An optional region.

Outputs**Minimum (Type: Object)**

The minimum value.

Comments

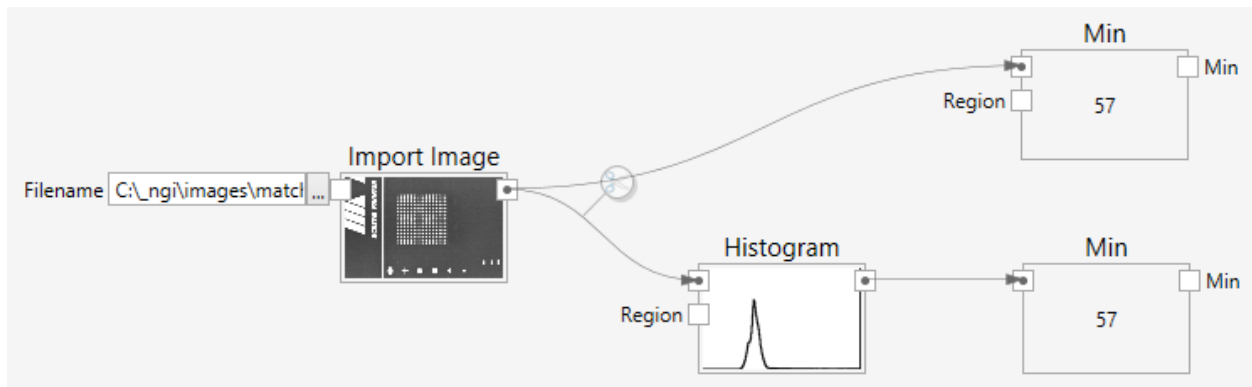
The **Minimum** node calculates the minimum value of a buffer.

If the *Region* input is connected, the calculation of the minimum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the minimum length color vector is calculated.

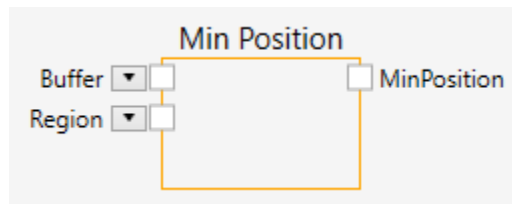
Sample

Here is an example that calculates the minimum of an image.



15.3.25 Min Position

Calculates the position of the minimum value of a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

MinPosition (Type: Index3d)

The position of the minimum value.

Comments

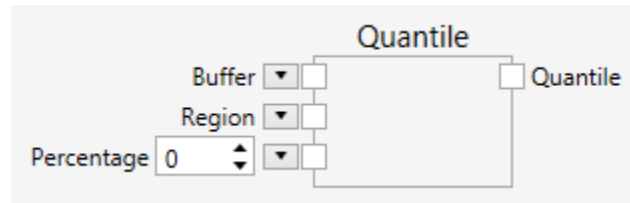
The **Min Position** node calculates the (first occurrence of the) position of the minimum value of a buffer.

If the *Region* input is connected, the calculation of the minimum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the minimum length color vector is calculated.

15.3.26 Quantile

Calculates the quantile of a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Percentage (Type: Double)

Specifies the quantile.

Outputs

Quantile (Type: Object)

The quantile value.

Comments

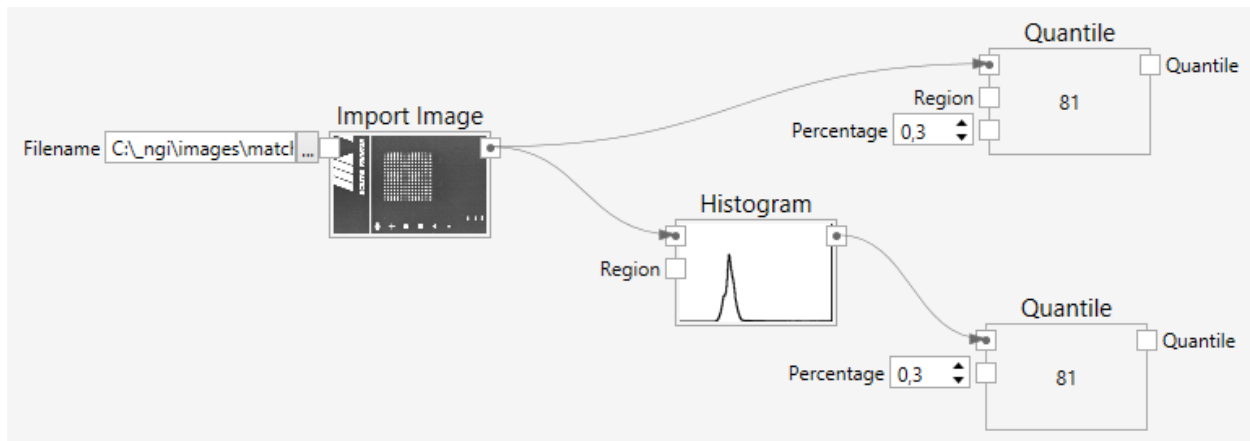
The **Quantile** node calculates a quantile of a buffer.

If the *Region* input is connected, the calculation of the quantile is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the quantile is calculated channel-wise.

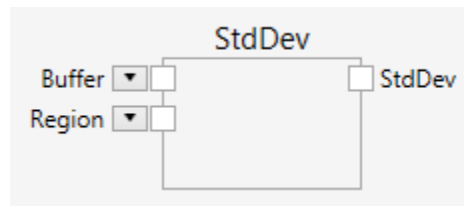
Sample

Here is an example that calculates the 50% quantile of an image.



15.3.27 StdDev

Calculates the standard deviation of all pixels in a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

StdDev (Type: Object)

The standard deviation.

Comments

The **StdDev** node calculates the standard deviation of all pixel values of a buffer.

If the *Region* input is connected, the calculation of the standard deviation is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the standard deviation is calculated channel-wise.

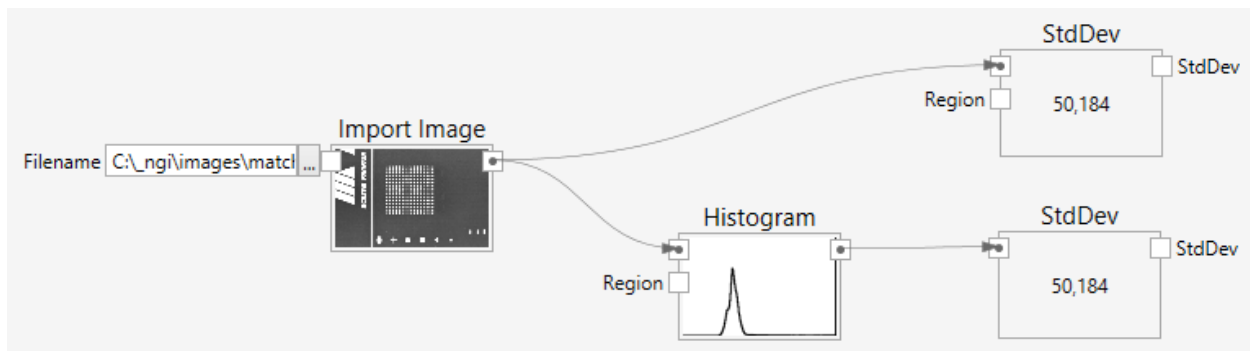
The standard deviation is a measure of statistical dispersion, averaging the squared distance of values from the mean value. The standard deviation is a measure of how much a statistical distribution is spread out. The standard deviation is the square root of the variance.

The standard deviation is calculated according to this formula:

$$\sigma = \sqrt{\frac{1}{n} \sum x^2 - \mu^2}$$

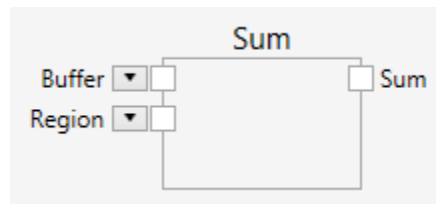
Sample

Here is an example that calculates the standard deviation of an image.



15.3.28 Sum

Calculates the sum of all pixels in a buffer.



Inputs

Buffer (Type: *Buffer*)

The input buffer.

Region (Type: *Region*)

An optional region.

Outputs

Sum (Type: *Object*)

The sum.

Comments

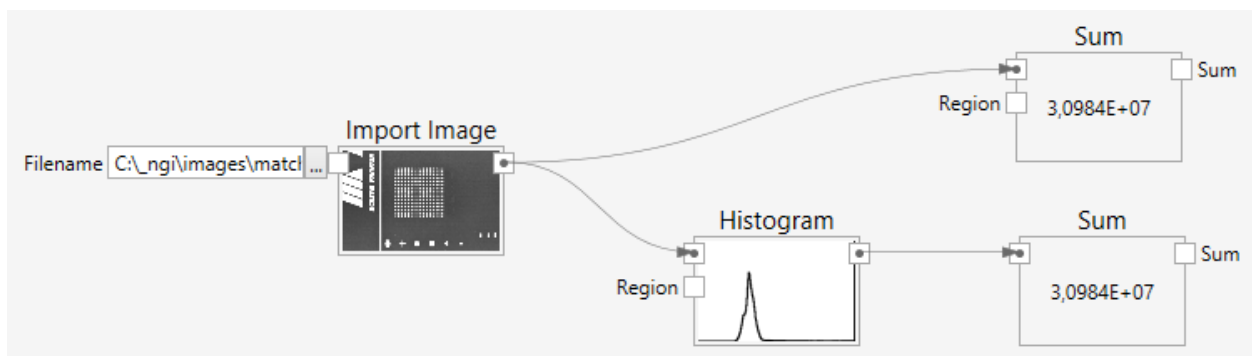
The **Sum** node calculates the sum of all pixel values of a buffer.

If the *Region* input is connected, the calculation of the sum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the sum is calculated channel-wise.

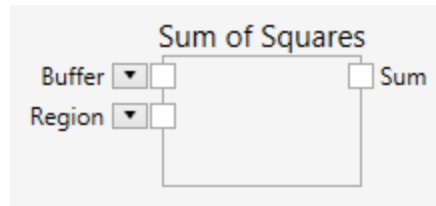
Sample

Here is an example that calculates the sum of an image.



15.3.29 Sum of Squares

Calculates the sum of squares of all pixels in a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

Sum (Type: Object)

The sum of squares.

Comments

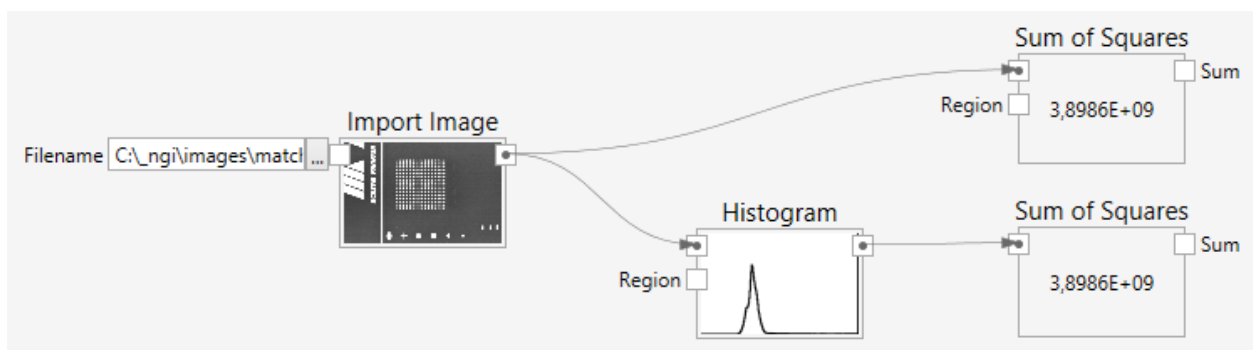
The **Sum** node calculates the sum of squares of all pixel values of a buffer.

If the *Region* input is connected, the calculation of the sum is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the sum of squares is calculated channel-wise.

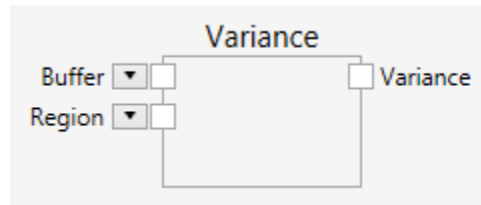
Sample

Here is an example that calculates the sum of squares of an image.



15.3.30 Variance

Calculates the variance of all pixels in a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

Variance (Type: Object)

The variance.

Comments

The **Variance** node calculates the variance of all pixel values of a buffer.

If the *Region* input is connected, the calculation of the variance is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the variance is calculated channel-wise.

The variance is a measure of statistical dispersion, averaging the squared distance of values from the mean value. The variance is a measure of how much a statistical distribution is spread out.

The variance is calculated according to this formula:

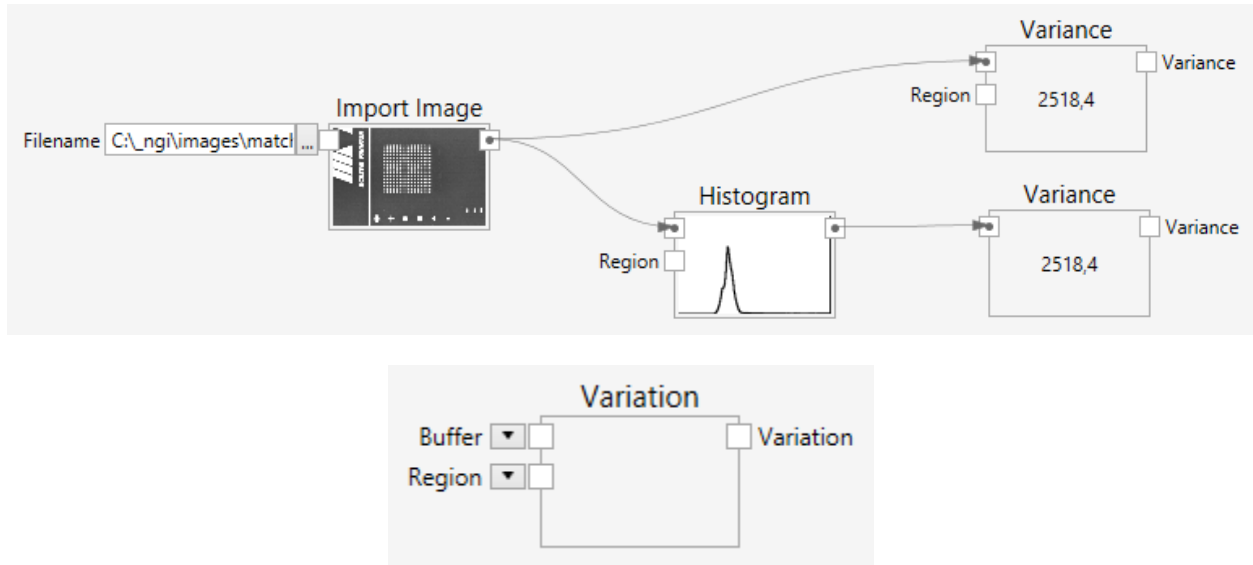
$$\sigma^2 = \frac{1}{n} \sum x^2 - \mu^2$$

Sample

Here is an example that calculates the variance of an image.

15.3.31 Variation

Calculates the coefficient of variation of all pixels in a buffer.



Inputs

Buffer (Type: Buffer)

The input buffer.

Region (Type: Region)

An optional region.

Outputs

Variation (Type: Object)

The coefficient of variation.

Comments

The **Variation** node calculates the coefficient of variation of all pixel values of a buffer.

If the *Region* input is connected, the calculation of the coefficient of variation is constrained to the pixels within the region only, otherwise the whole buffer is used.

If a color buffer is used, the coefficient of variation is calculated channel-wise.

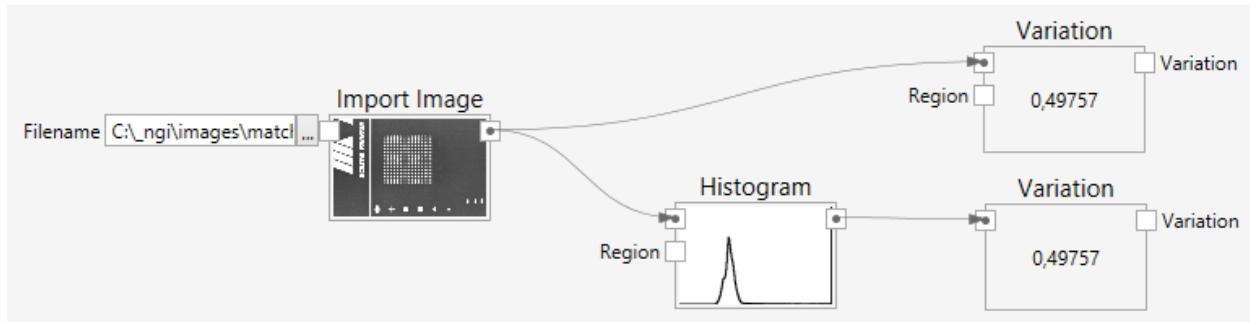
The coefficient of variation is the standard deviation divided by the average.

The coefficient of variation is calculated according to this formula:

$$c_v = \frac{\sigma}{\mu}$$

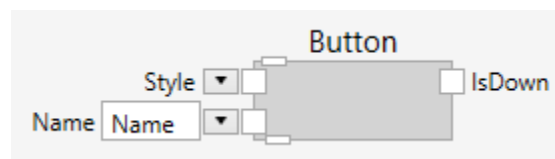
Sample

Here is an example that calculates the coefficient of variation of an image.



15.3.32 HMI Button

Buttons are used to switch or visualize state, as well as to trigger actions. The **Button** is a push-button, which is released when you no longer push it.



A **Button** is down as long as it is held down with the mouse. When it is no longer held down, it releases itself to the up state.

At the bottom of the **Button** node is a pin that allows it to connect one child.

Inputs

Style (Type: `Style`)

Optional styling of the **Button**.

The **Border** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Name (Type: `string`)

The text that is displayed within the button. This text is only displayed, when no child is connected. If a child is connected, the visual definition of the child is used.

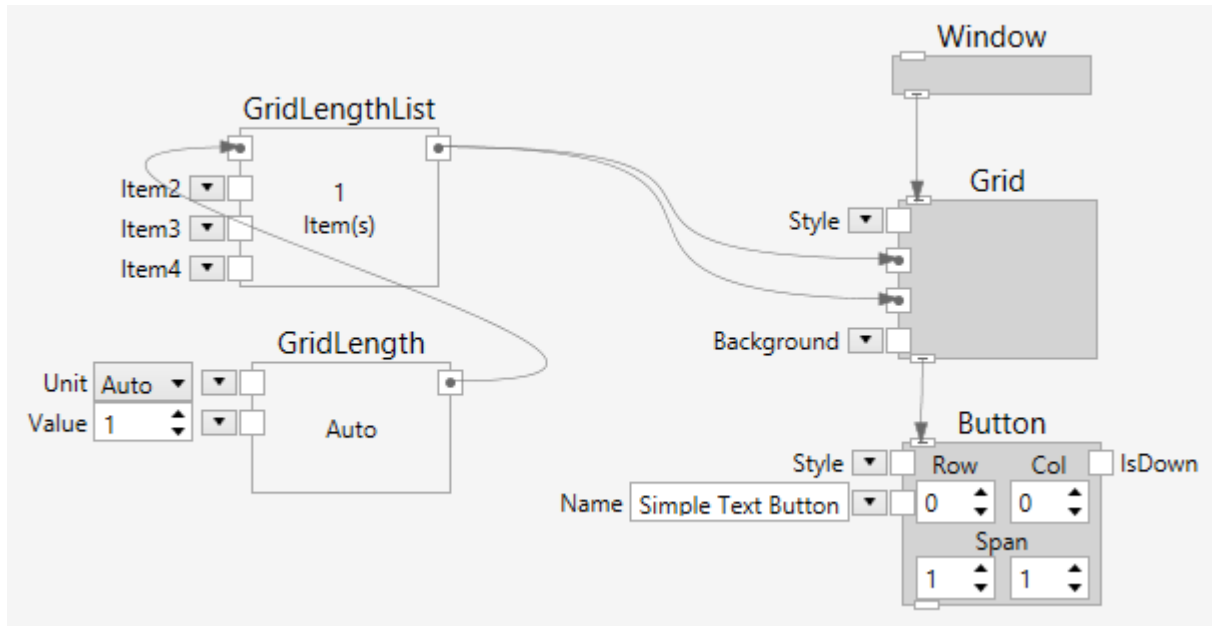
Outputs

IsDown (Type: `boolean`)

True if the button is down, False otherwise.

Example

Here is an example that shows a simple text button. This definition:



creates the following user interface:

Simple Text Button

And this more complicated example shows a button with an image and a text. This definition:

creates the following user interface:

15.3.33 Execute Command

Executes a GenICam command of a camera.

Inputs

Camera (Type: CameraInfo)

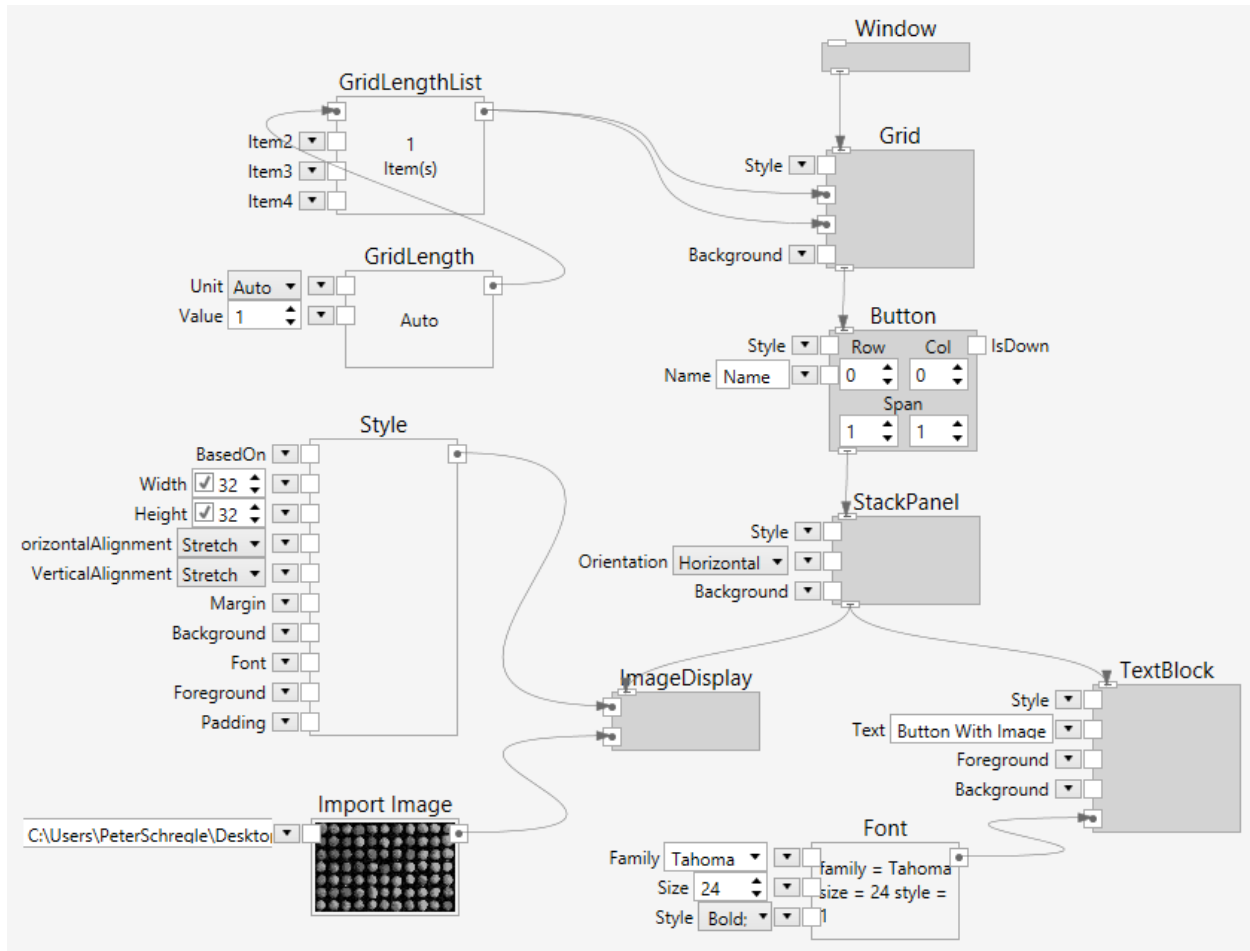
Specifies the camera.

Sync (Type: System.Object)

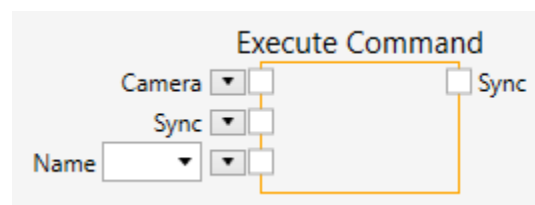
This can be used to establish a temporal order, so that you can specify, what has to occur before the parameter will be written.

Name (Type: String)

The name of the GenICam command. These names are determined by the camera vendor.



Button With Image



Outputs

Sync (Type: `System.Object`)

This can be used to establish a temporal order, so that you can specify, what has to occur after the parameter has been written.

Comments

The names of the available camera commands are the same as those in the camera parameters window that you can open with the **Show Camera Parameters** on the **Home** ribbon bar.

15.3.34 Get Parameter

Reads a GenICam parameter value from a camera.

Inputs

Camera (Type: `CameraInfo`)

Specifies the camera.

Sync (Type: `System.Object`)

This can be used to establish a temporal order, so that you can specify, what has to occur before the parameter will be read.

Name (Type: `String`)

The name of the GenICam parameter. These names are determined by the camera vendor.

Outputs

Value (Type: `String`)

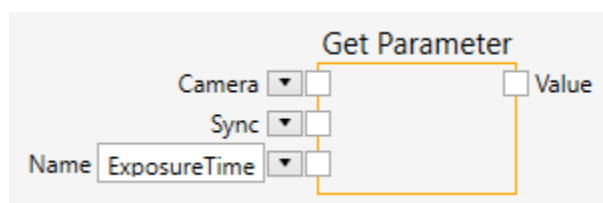
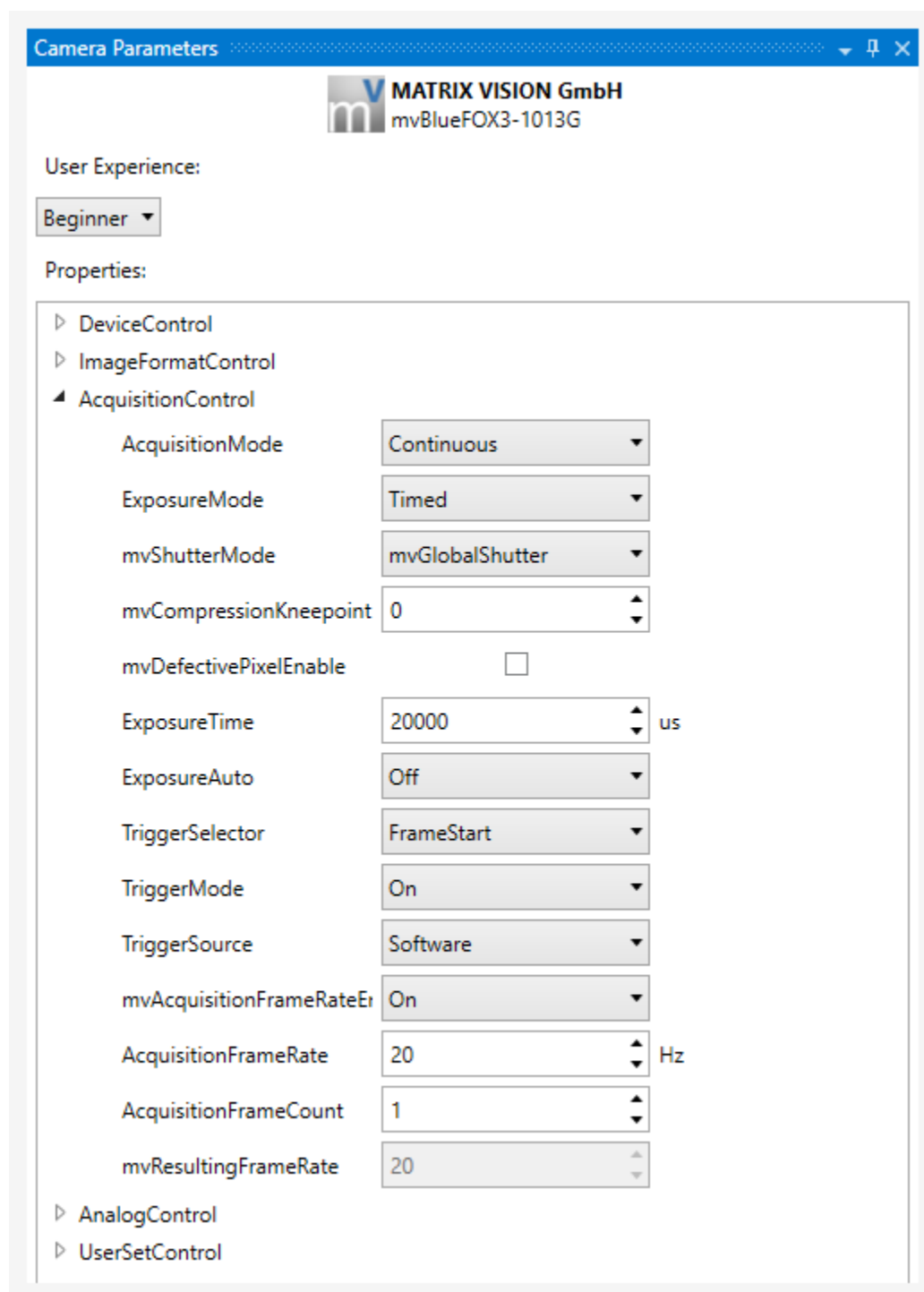
The value read from the camera parameter.

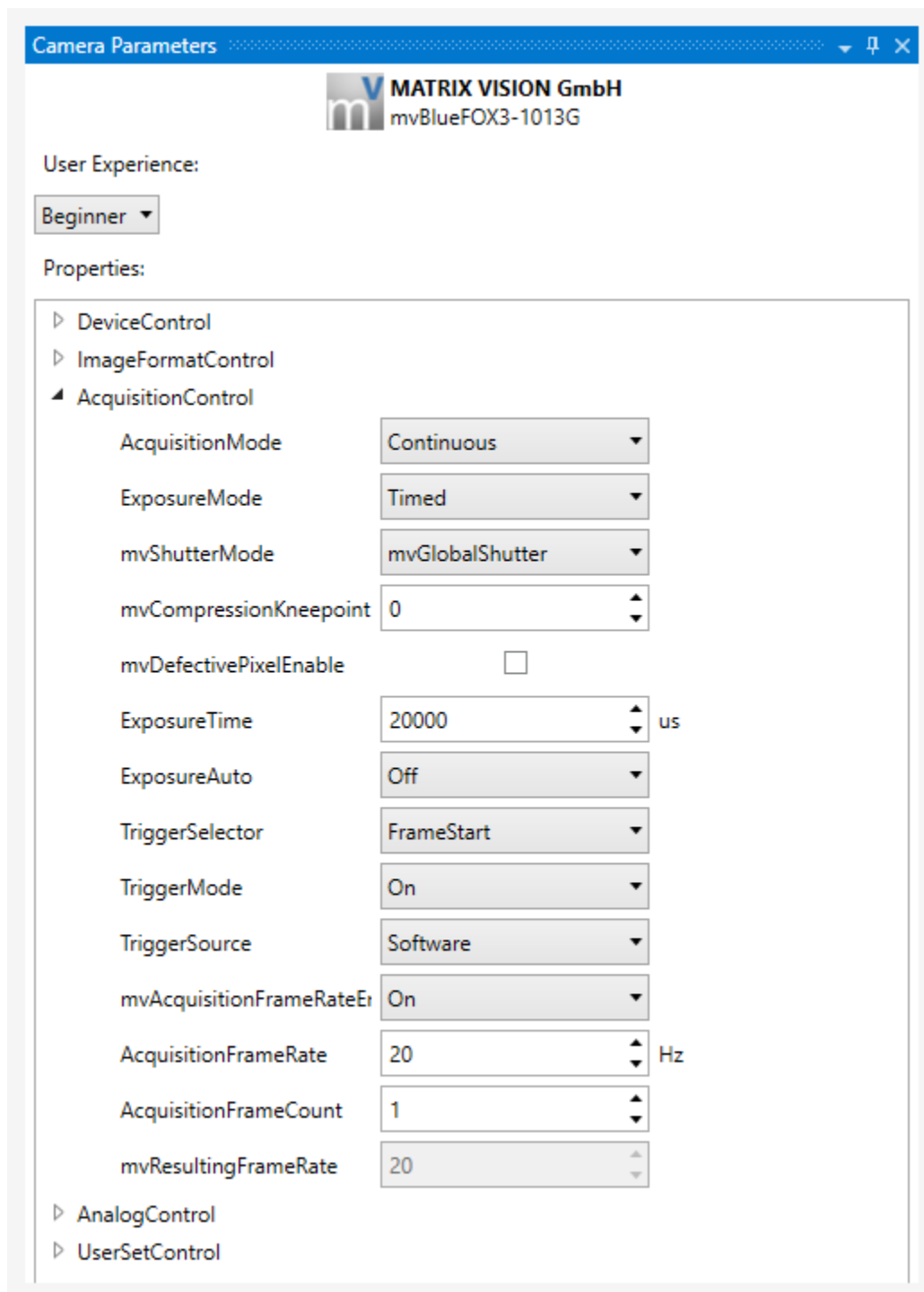
Comments

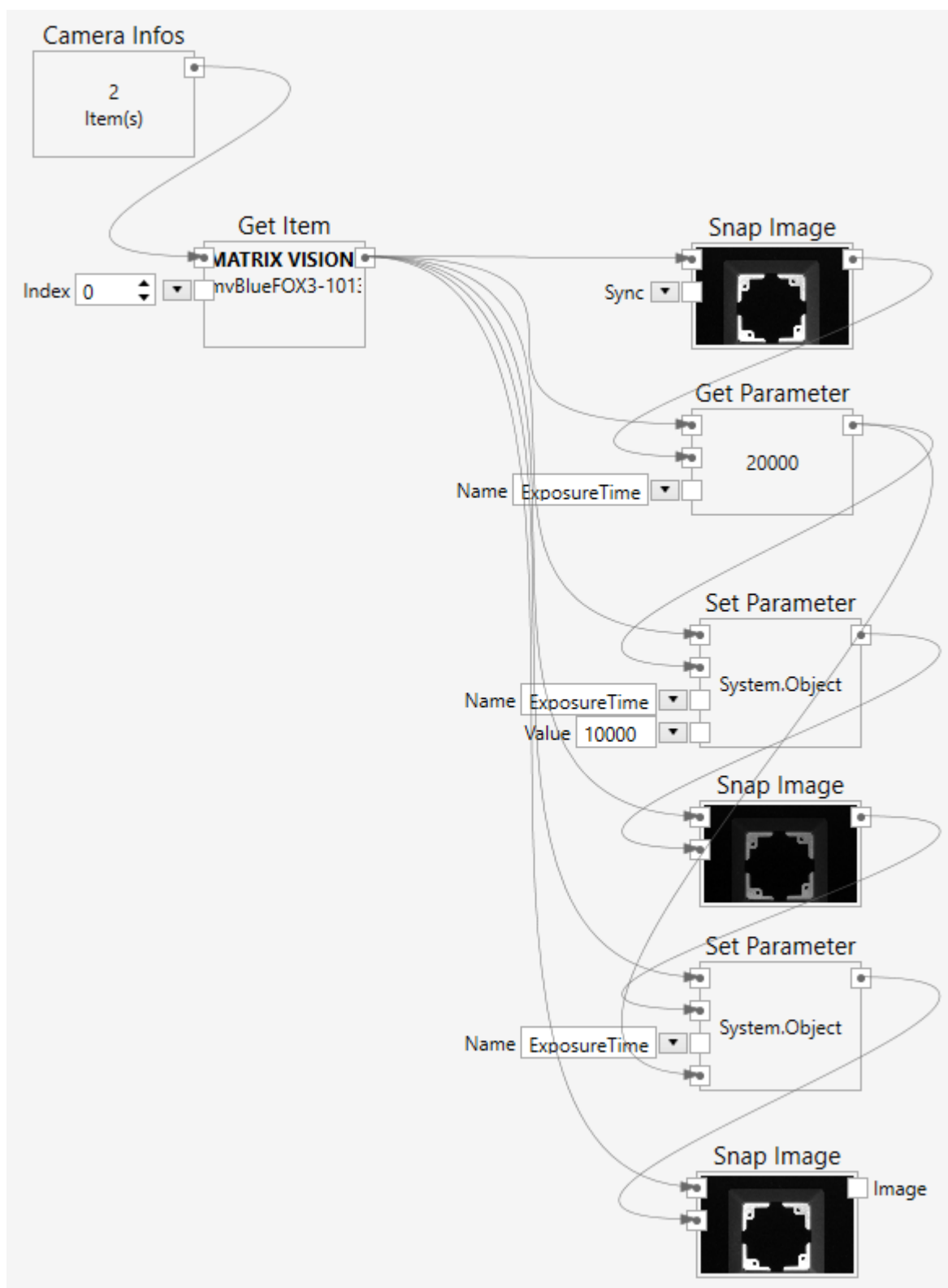
The names of the available camera parameters can be found by opening the camera parameters window with the **Show Camera Parameters** on the **Home** ribbon bar.

Example

Here is an example that snaps one image, reads the current ExposureTime (20 ms), sets the ExposureTime to 10 ms, takes a second image, sets the ExposureTime back to its original value and takes a third image. The connections of the Sync outputs and inputs establish the temporal order.

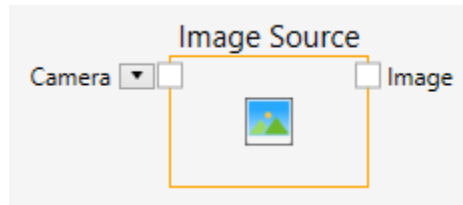






15.3.35 Image Source

Delivers images from a camera in live acquisition mode.



Inputs

Camera (Type: CameraInfo)

Specifies the camera.

Outputs

Image (Type: Image)

Delivers an image from the camera.

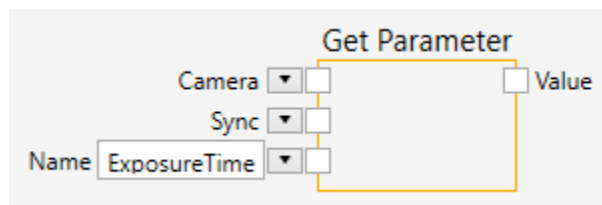
Comments

The **Start Acquisition** node starts the acquisition on the specified camera. Once the acquisition is running, an **Image Source** node will deliver images from the camera. The **Stop Acquisition** node stops the acquisition on the specified camera.

This **Image Source** node runs asynchronously. When a new image from a camera arrives, the pipeline executor is notified and executes the pipeline.

15.3.36 Set Parameter

Writes a GenICam parameter value to a camera.



Inputs

Camera (Type: CameraInfo)

Specifies the camera.

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, what has to occur before the parameter will be written.

Name (Type: String)

The name of the GenICam parameter. These names are determined by the camera vendor.

Value (Type: String)

The value to write to the camera parameter.

Outputs

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, what has to occur after the parameter has been written.

Comments

The names of the available camera parameters can be found by opening the camera parameters window with the **Show Camera Parameters** on the **Home** ribbon bar.

Example

Here is an example that snaps one image, reads the current ExposureTime (20 ms), sets the ExposureTime to 10 ms, takes a second image, sets the ExposureTime back to its original value and takes a third image. The connections of the Sync outputs and inputs establish the temporal order.

15.3.37 Snap Image

Snaps an image from a camera.

The preview shows the number of available cameras.

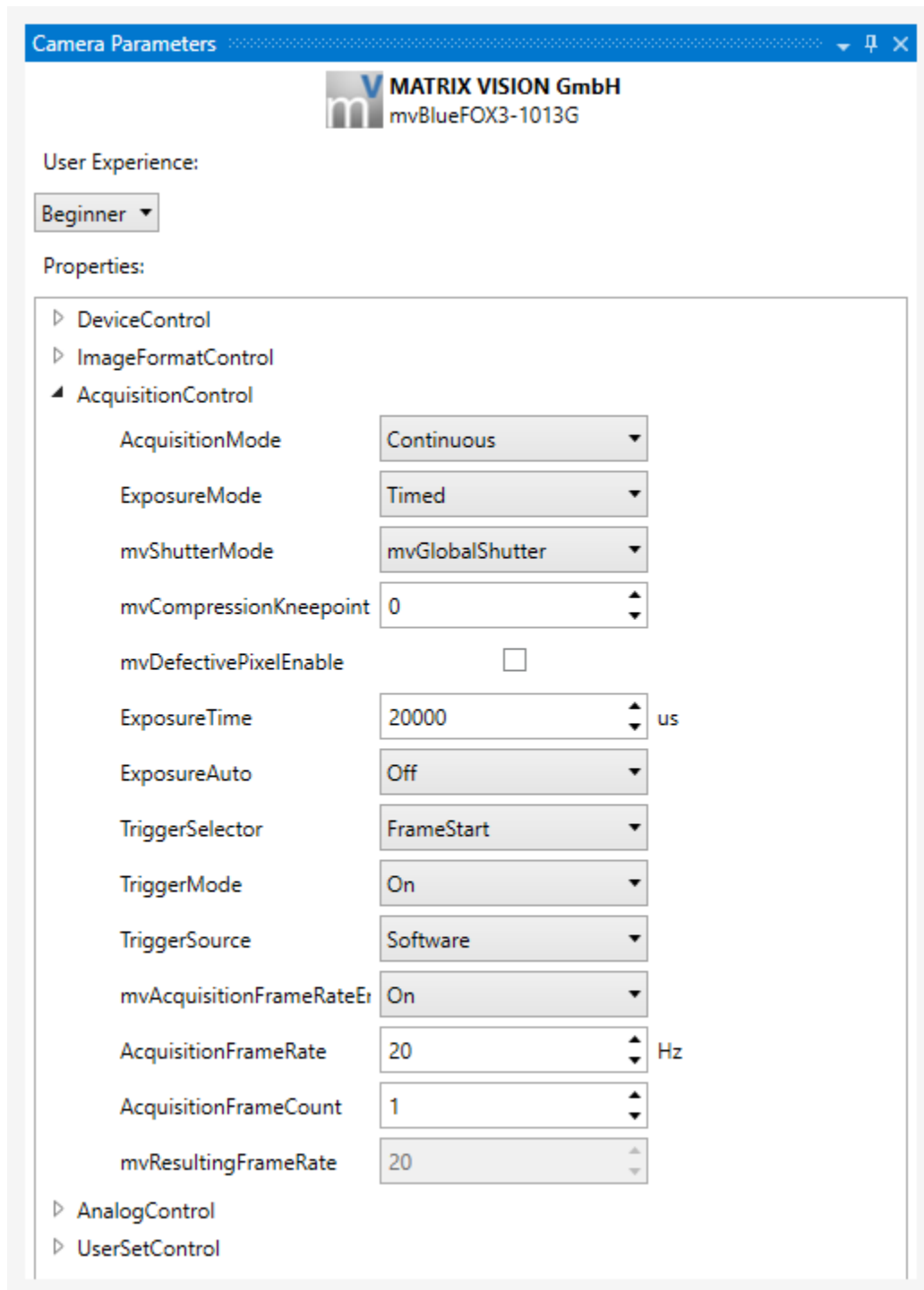
Inputs

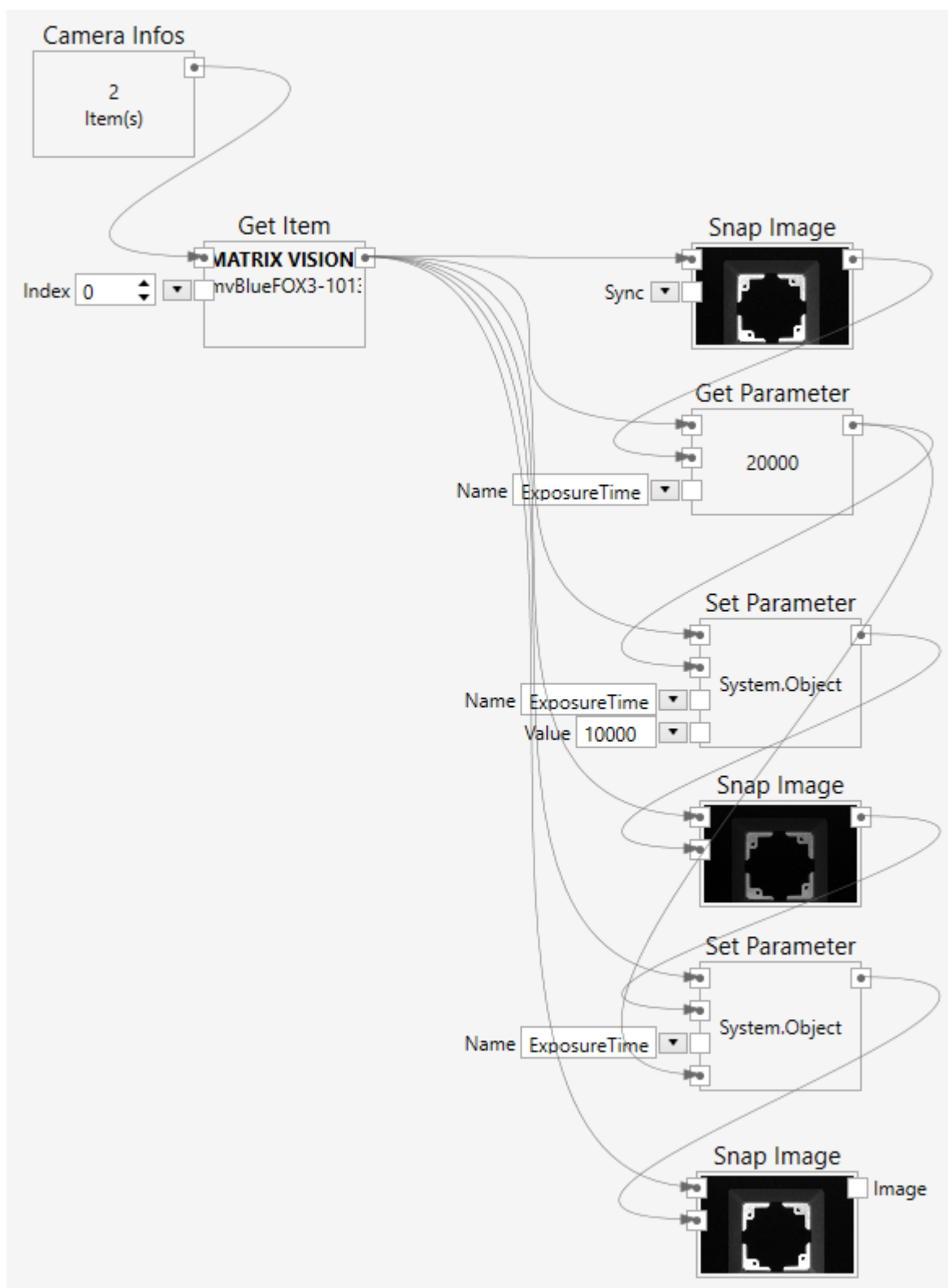
Camera (Type: CameraInfo)

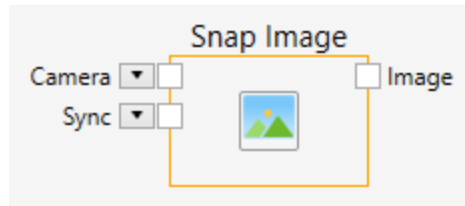
Specifies the camera.

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, when the image snapshot occurs.







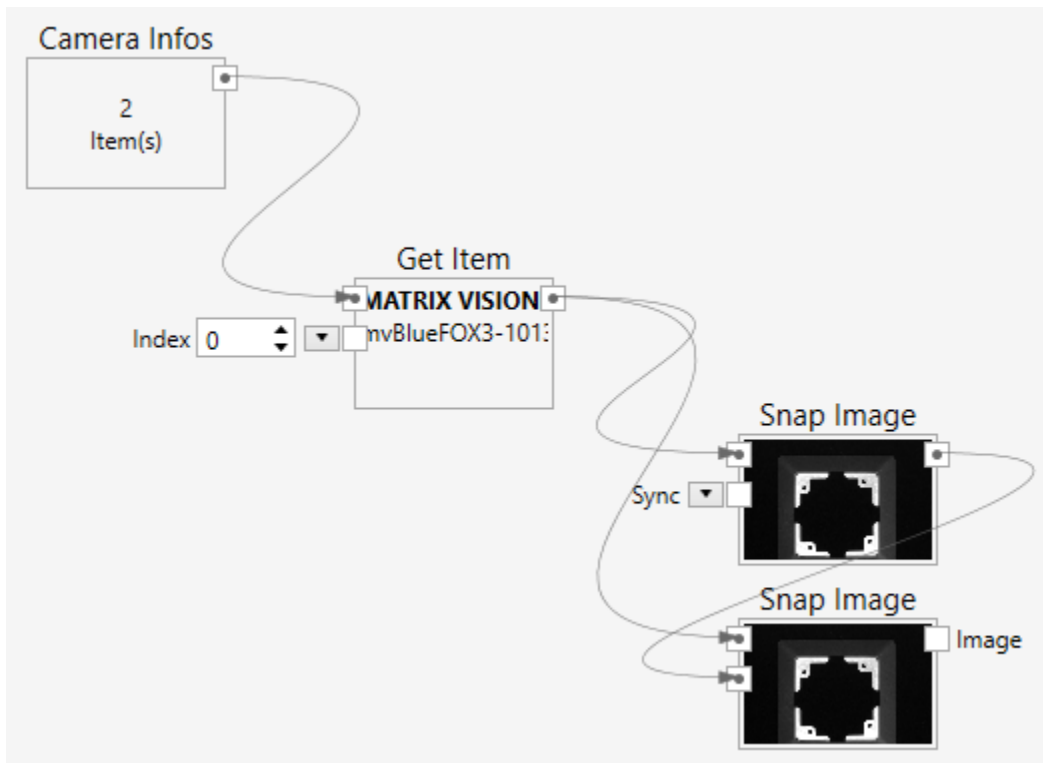
Outputs

Image (Type: Image)

The acquired image.

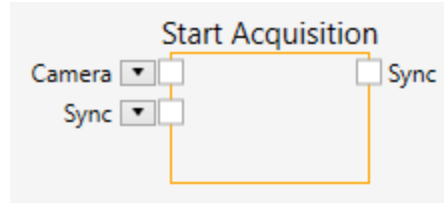
Example

Here is an example that snaps two images in fast succession:



15.3.38 Start Acquisition

Starts the acquisition of a camera.



Inputs

Camera (Type: CameraInfo)

Specifies the camera.

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, what has to occur before the parameter will be written.

Outputs

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, what has to occur after the parameter has been written.

Comments

The **Start Acquisition** node starts the acquisition on the specified camera. Once the acquisition is running, an **Image Source** node will deliver images from the camera. The **Stop Acquisition** node stops the acquisition on the specified camera.

15.3.39 Stop Acquisition

Stops the acquisition of a camera.



Inputs

Camera (Type: CameraInfo)

Specifies the camera.

Sync (Type: System.Object)

This can be used to establish a temporal order, so that you can specify, what has to occur before the parameter will be written.

Outputs**Sync (Type: System.Object)**

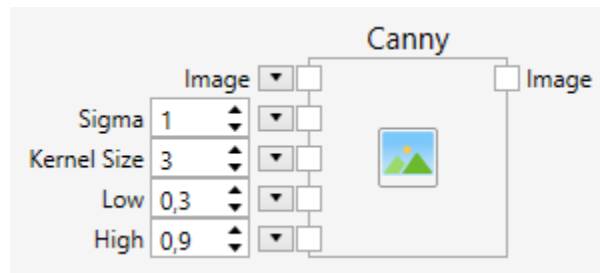
This can be used to establish a temporal order, so that you can specify, what has to occur after the parameter has been written.

Comments

The **Start Acquisition** node starts the acquisition on the specified camera. Once the acquisition is running, an **Image Source** node will deliver images from the camera. The **Stop Acquisition** node stops the acquisition on the specified camera.

15.3.40 Canny

Applies Canny edge detection to an image.



According to Canny's edge detection algorithm, the filter performs Gaussian smoothing, edge detection with a Sobel filter and then uses hysteresis thresholding and non maximum suppression to end up with a binary image of the edges.

You can follow this node with a Threshold node to convert the binary image to a region.

Inputs**Image (Type: Image)**

The input image.

Sigma (Type: Double)

Determines the size of the Gaussian filter kernel.

SobelSize (Type: Int32)

Determines the size of the Sobel filter kernel.

Low (Type: Double)

The low threshold of the hysteresis thresholding expressed as a percentage.

High (Type: Double)

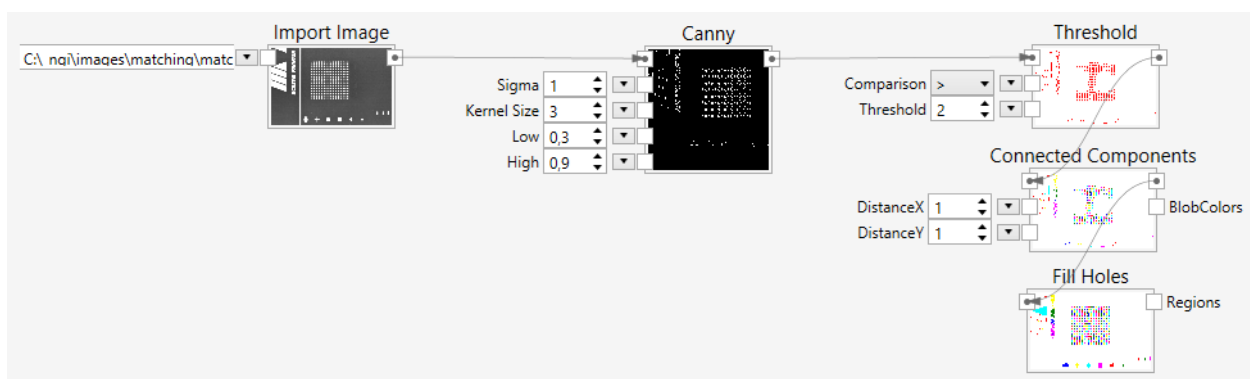
The high threshold of the hysteresis thresholding expressed as a percentage.

Outputs**Image (Type: Image)**

The output image of the edges.

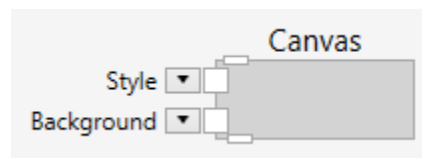
Sample

Here is an example:



15.3.41 HMI Canvas

The **Canvas** allows you to put controls at specific positions on the canvas.



At the bottom of the **Canvas** node is a pin that allows it to connect several children.

Inputs**Style (Type: style)**

Optional styling of the **Canvas**.

The **Canvas** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background* styles.

Background (Type: `SolidColorBrushByte`)

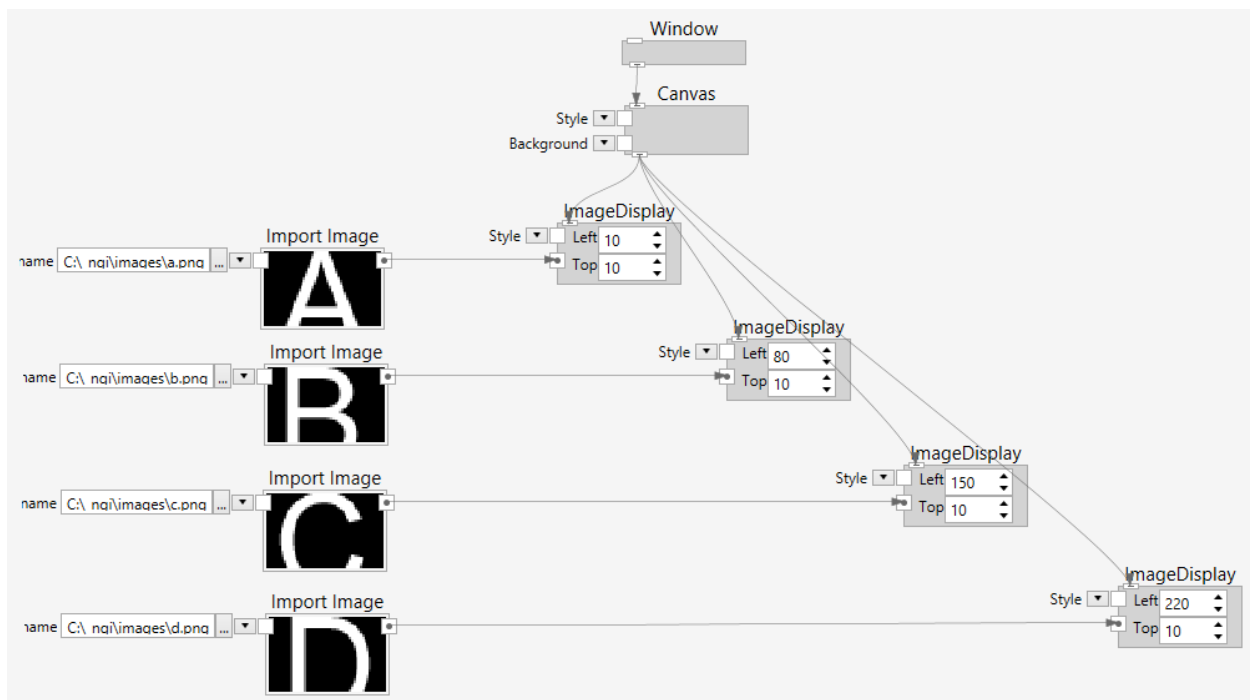
The background of the **Canvas**.

Comments

Children of the **Canvas** have the attached properties **Left** and **Top**. These properties are pixel positions and define where exactly the respective child will be put on the canvas.

Example

Here is an example that puts four images on a canvas. This definition:

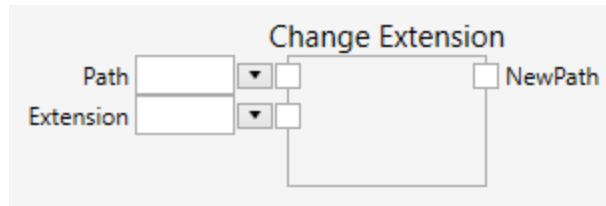


creates the following user interface:



15.3.42 Change Extension

Changes the extension of a path string.



Inputs

Path (Type: String)

The path information to modify.

Extension (Type: String)

The new extension (with or without a leading period). Specify an empty string to remove an existing extension from path.

Outputs

NewPath (Type: Ngi.Path)

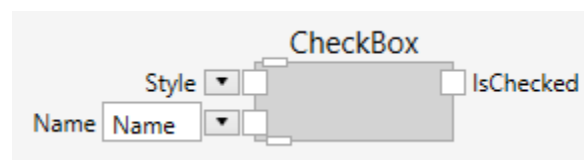
The modified path information

Comments

If path is an empty string, the path information is returned unmodified. If extension is an empty string, the returned string contains the specified path with its extension removed. If path has no extension, and extension is not the empty string, the returned path string contains extension appended to the end of path.

15.3.43 HMI CheckBox

Buttons are used to switch or visualize state, as well as to trigger actions. The **CheckBox** has two states, checked or not.



A **CheckBox** toggles between the checked and unchecked states when it is clicked.

At the bottom of the **CheckBox** node is a pin that allows it to connect one child.

Inputs

Style (Type: `style`)

Optional styling of the **CheckBox**.

The **CheckBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Name (Type: `string`)

The text that is displayed within the button. This text is only displayed, when no child is connected. If a child is connected, the visual definition of the child is used.

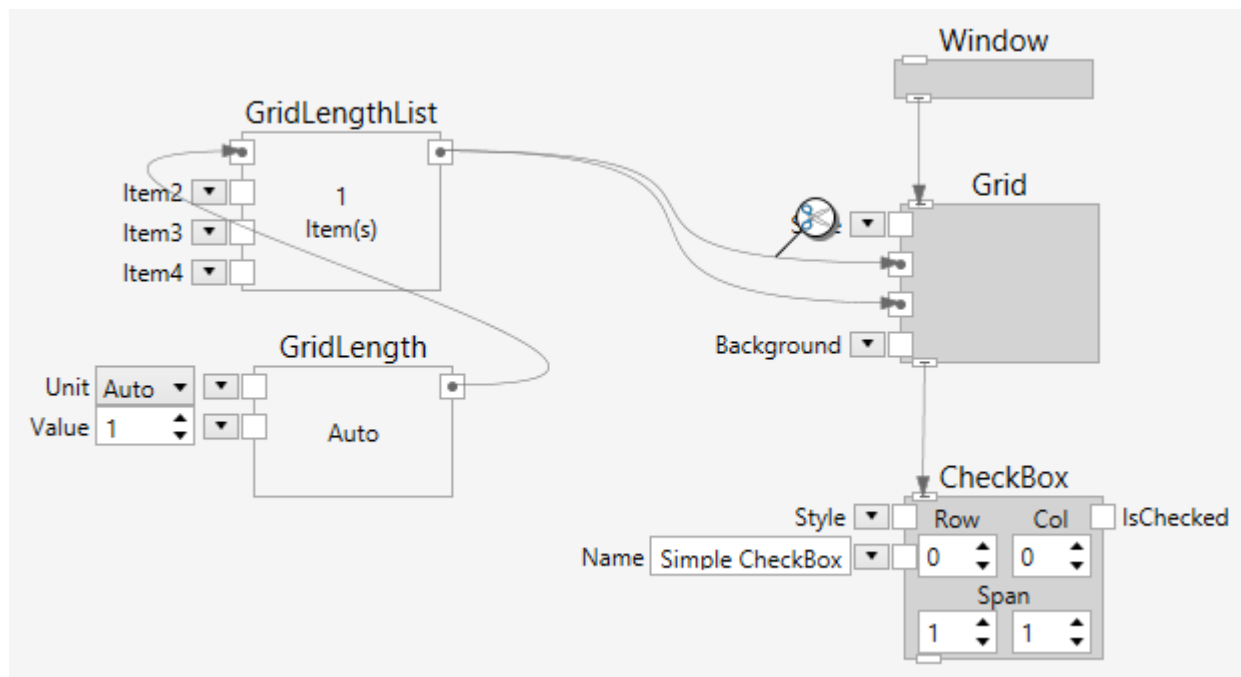
Outputs

IsChecked (Type: `boolean`)

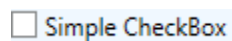
True if the checkbox is checked, False otherwise.

Example

Here is an example that shows a simple **CheckBox**. This definition:

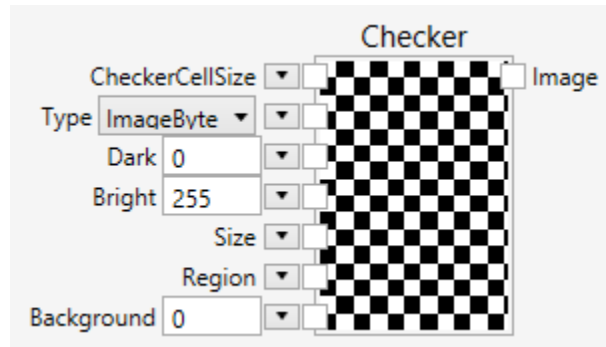


creates the following user interface:



15.3.44 Checker

Presets an image with a checkerboard pattern.



Inputs

CheckerCellSize (Type: VectorInt32)

The cell size of one checkerboard field. The default is 64 x 64 pixels.

Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble

Dark (Type: String)

The value of a dark checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: String)

The value of a bright checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs**Image (Type: Image)**

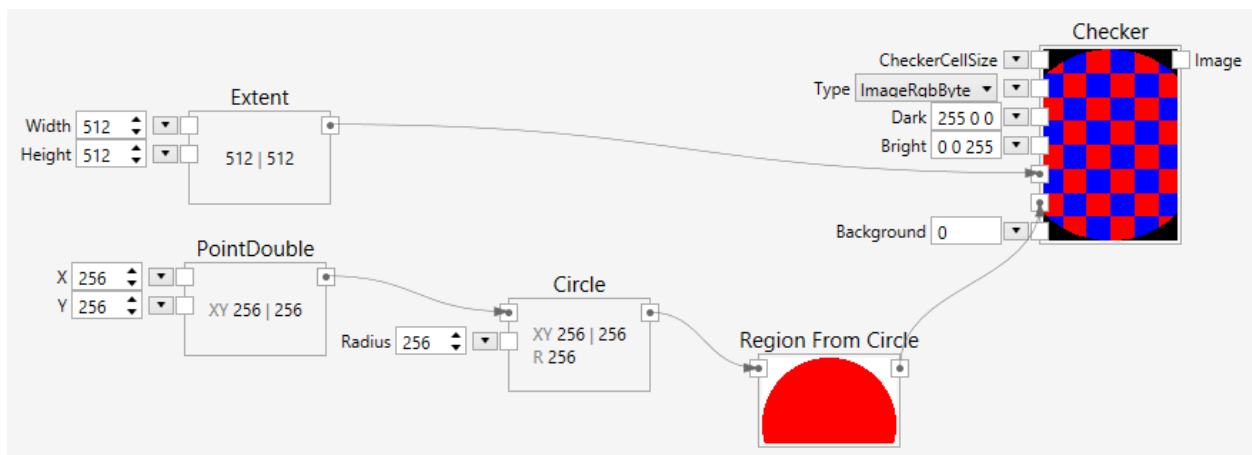
The image preset with a checkerboard pattern.

Comments

The **Checker** node presets an image with a checkerboard pattern. The size of the created image, the colors for dark and bright checkerboard fields, the region of interest and the background color can all be specified.

Sample

Here is an example that shows how to used the **Checker** node.



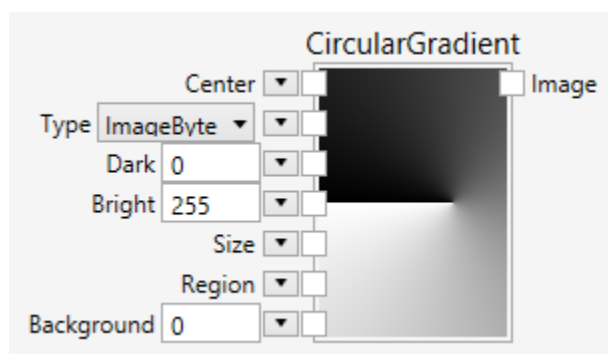
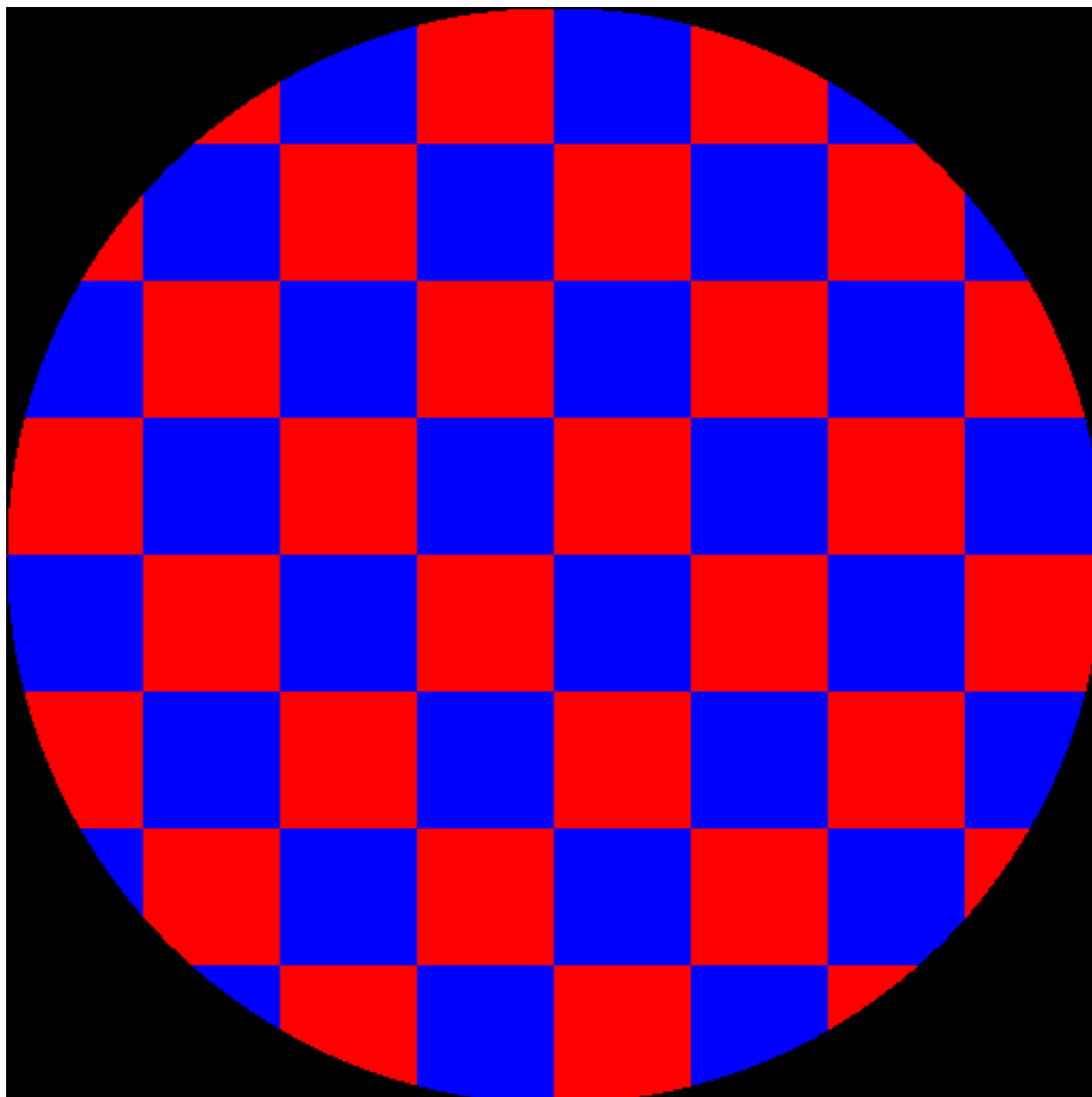
Here is the image created with the sample:

15.3.45 CircularGradient

Presets an image with a circular gradient pattern.

Inputs**Center (Type: PointDouble)**

The center of the gradient. The default is 512, 512.



Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble

Dark (Type: String)

The dark value. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: String)

The bright value. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

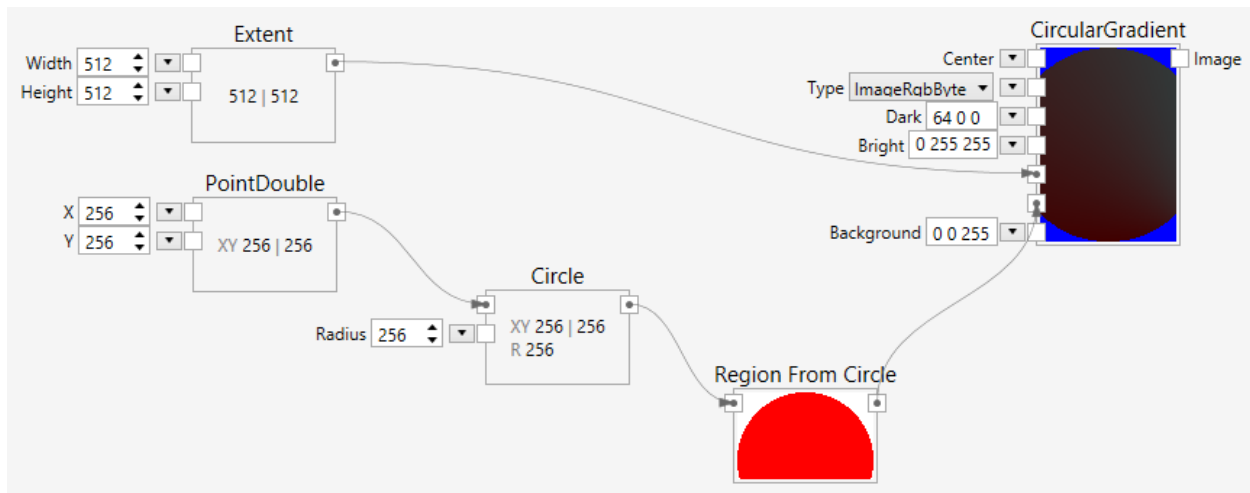
The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs**Image (Type: Image)**

The image preset with a circular pattern.

Comments

The **CircularGradient** node presets an image with a circular gradient pattern. The size of the created image, the colors for dark and bright checkerboard fields, the region of interest and the background color can all be specified.



Sample

Here is an example that shows how to use the **CircularGradient** node.

Here is the image created with the sample:

15.3.46 Combine

Combines two images to form a combined image.

Inputs

A (Type: Image)

The first input image.

B (Type: Image)

The second input image.

Direction (Type: String)

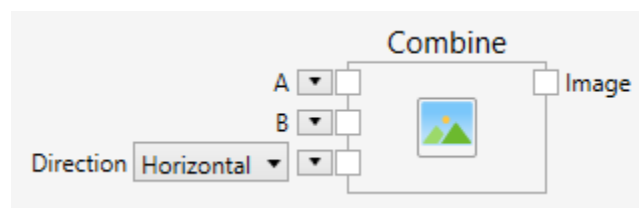
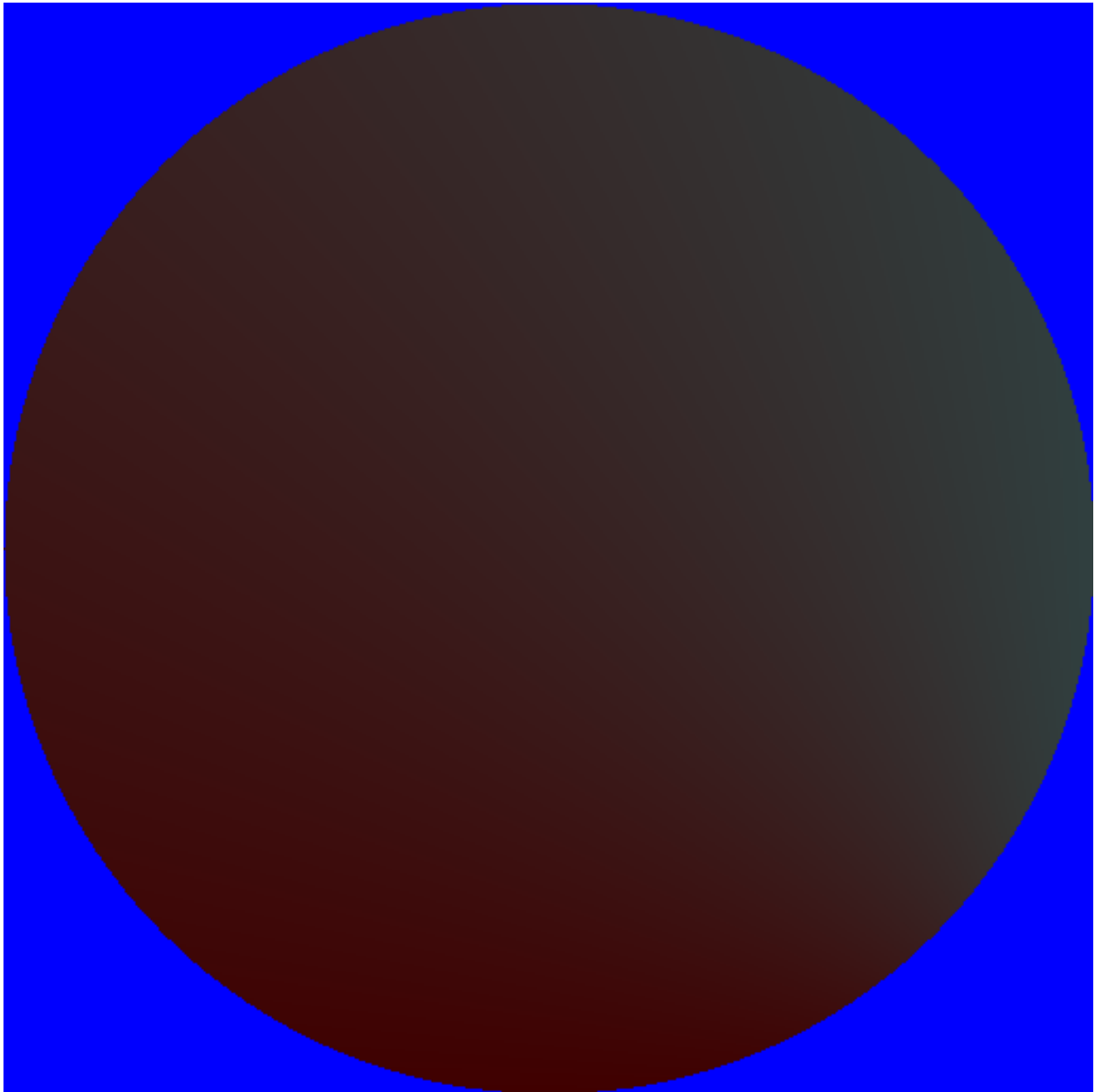
The direction to combine the image. This can be `Horizontal`, `Vertical` or `Planar` to combine in the X, Y, or Z directions, respectively.

The other dimensions of the image must match, otherwise the combine node issues an error message.

Outputs

Image (Type: Image)

The combined output image.

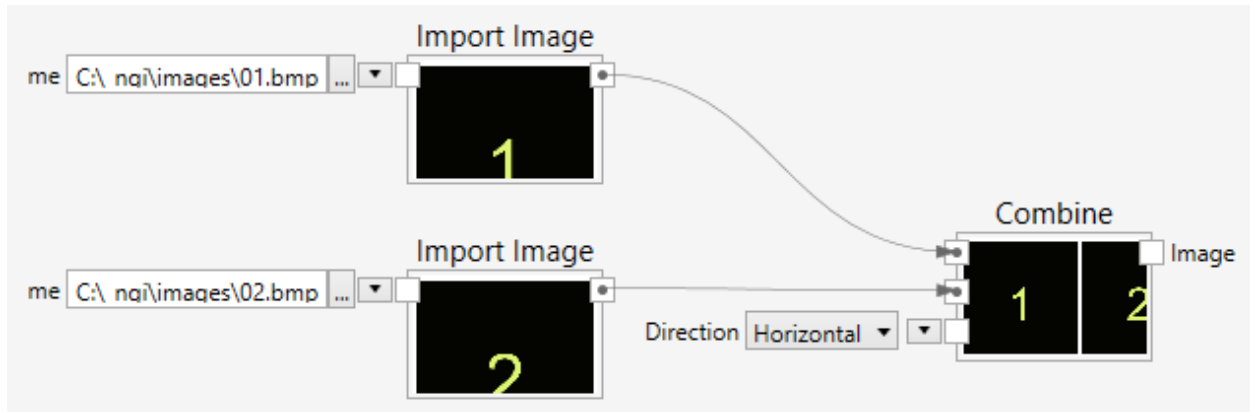


Comments

This function combines an image in two parts, in either the horizontal, vertical or planar directions.

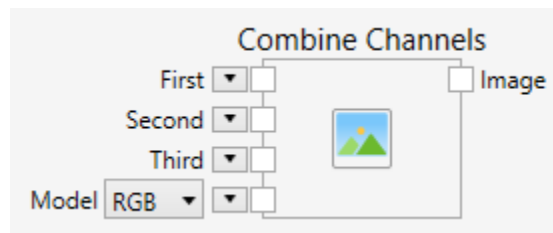
Sample

Here is an example:



15.3.47 Combine Channels

Combines three channels to create a color image.



Inputs

First (Type: Image)

The first channel image.

Second (Type: Image)

The second channel image.

Third (Type: Image)

The third channel image.

Model (Type: String)

The color model of the resulting image.

Outputs**Image (Type: Image)**

The output image.

Comments

This function combines three channels to a color image.

The color model specifies the color space of the resulting image and also the meaning of the channels. The following color models are available: **RGB**, **HLS**, **HSI** and **LAB**.

Given the following channels (red, green and blue from left to right):



Here are the channels again visualized in their proper colors:



The result of the combination as an RGB image is:



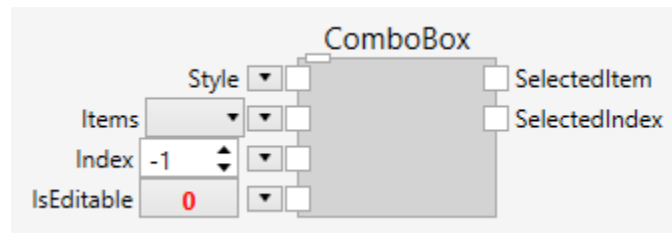
Sample

Here is an example that combines three channels to form a color image.



15.3.48 HMI ComboBox

A **ComboBox** can be used to select one from a number of entries. It has a collapsed and expanded state.



Inputs

Style (Type: Style)

Optional styling of the **ComboBox**.

The **ComboBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Items (Type: List<string>)

A list of text items that is displayed in the **ComboBox**.

Index (Type: Int32)

The initial selection of the **ComboBox**. This is a zero-based index. You can use *-1* to specify *no selection*. This value is used only when the *Items* pin is connected.

IsEditable (Type: Boolean)

If false, the **ComboBox** can be used to select items only. If true, you can also type into the edit field of the **ComboBox**. Typing either selects between one of the items in the list or can be used to enter a text which is not in the list.

Outputs**SelectedItem (Type: string)**

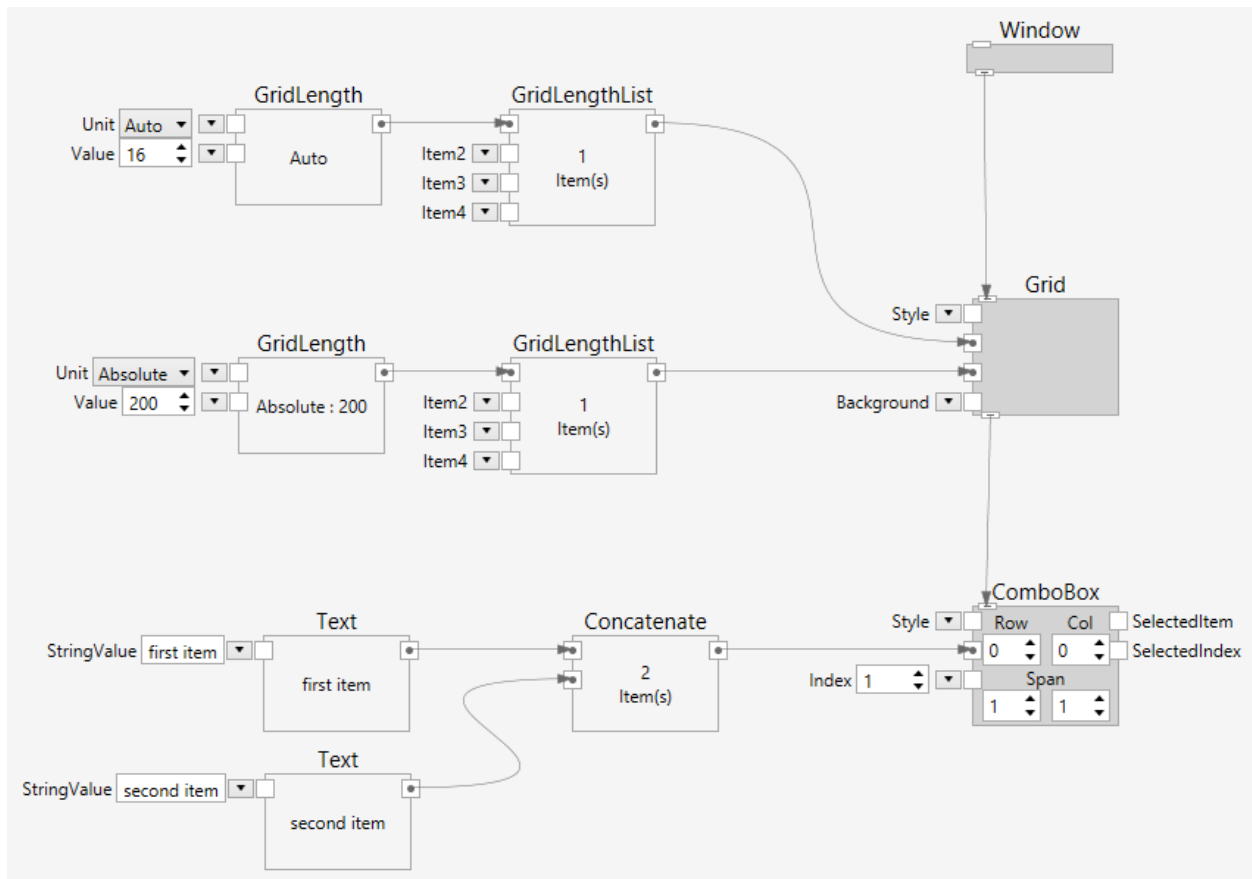
The selected item.

SelectedIndex (Type: Int32)

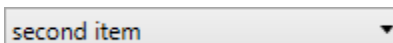
The zero-based index of the selected item, or -1 for *no selection*.

Example

Here is an example that shows a **ComboBox**. This definition:



creates the following user interface:

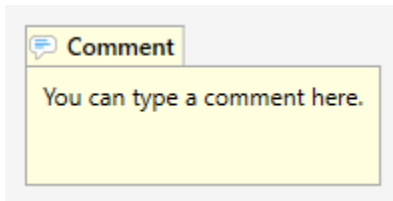


and in expanded state:



15.3.49 Comment

Comments a pipeline.



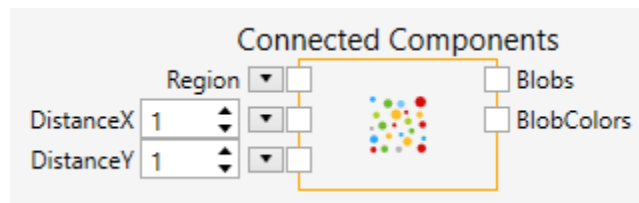
A **Comment** can be used to add text to a pipeline. It does not participate in pipeline execution, but it serves as documentation for a pipeline.

A **Comment** can be moved around freely on the pipeline canvas. The size of a comment is determined by the text included in the comment.

Text can be entered by clicking into the comment area and typing.

15.3.50 Connected Components

Splits a region into a list of multiple spatially disconnected regions.



Inputs

Region (Type: Region)

The input region.

DistanceX (Type: Int32)

A horizontal distance. Pixels within this horizontal distance are considered to be connected.

DistanceY (Type: Int32)

A vertical distance. Pixels within this horizontal distance are considered to be connected.

Outputs

Blobs (Type: RegionList)

The resulting list of disconnected regions.

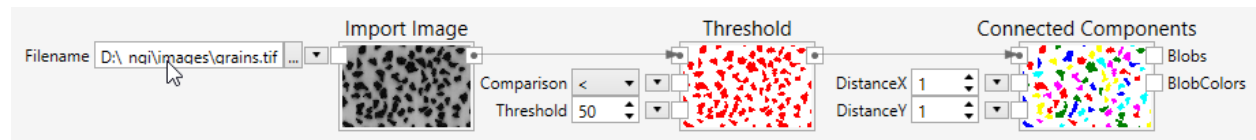
BlobColors

A list of colors that can be used to color the regions in a way that expresses their separated nature by using different colors (the list contains six distinct colors that are repeated).

Comments

Usually, connectedness is specified using the terms 4-connected and 8-connected in the literature. 4-connected means that objects touching only at the diagonal pixels are separated, 8-connected means that such objects are not separated. The two distance parameters can be used to specify 4-connectedness (DistanceX == 0, DistanceY == 0) or 8-connectedness (DistanceX == 1, DistanceY == 1). Bigger values of the distance parameters can jump over gaps and still treat objects as being connected.

Using this pipeline:



the following region has been created by thresholding:

and the following list of regions has been created by separating objects with the Connected Components node:

15.3.51 Constant

Presets an image with a constant color.

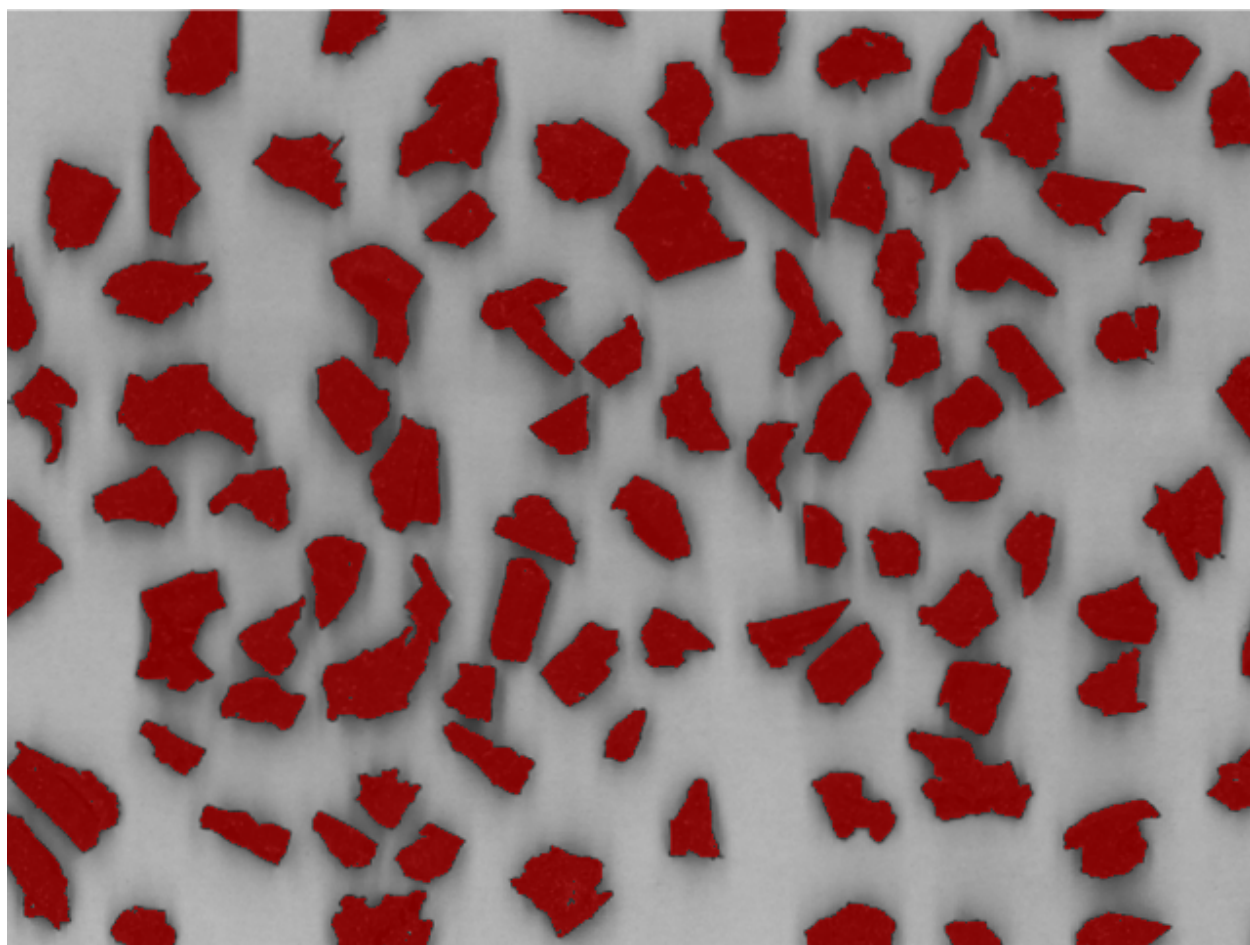
Inputs

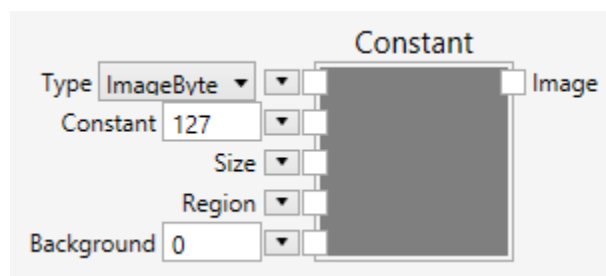
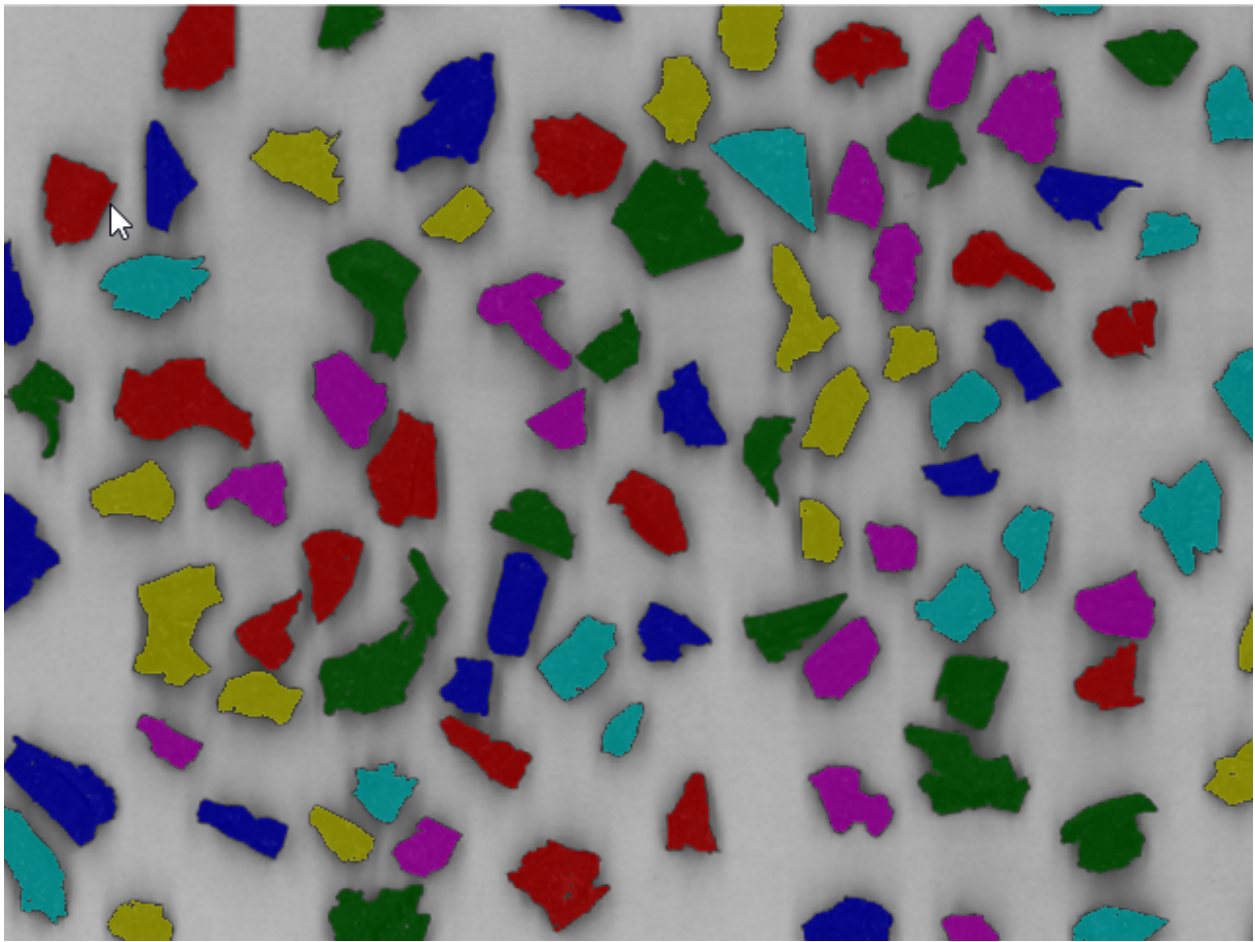
Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble

Constant (Type: String)

The constant value. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.





Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs**Image (Type: Image)**

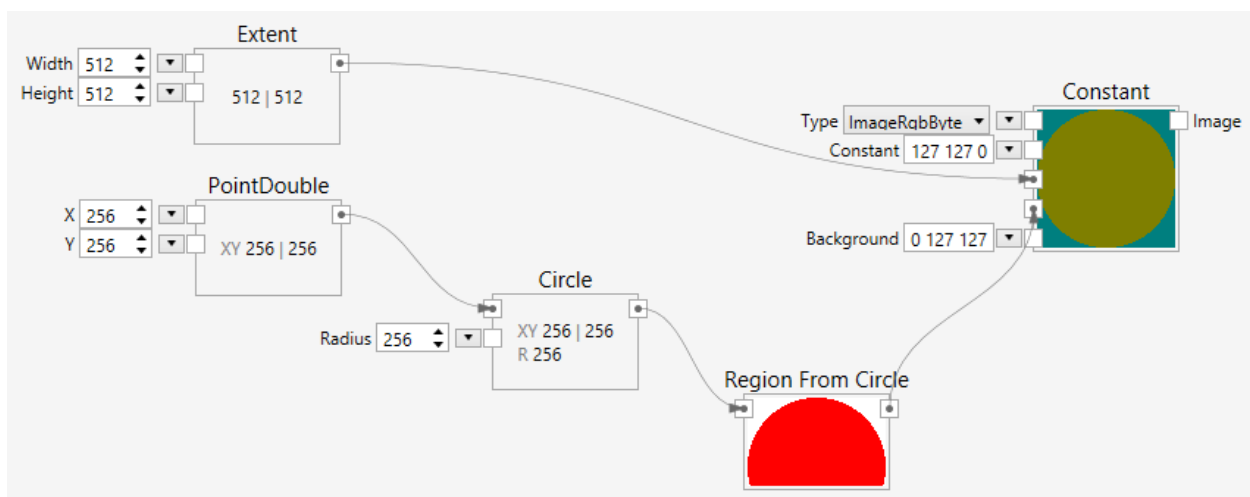
The image preset with a constant pattern.

Comments

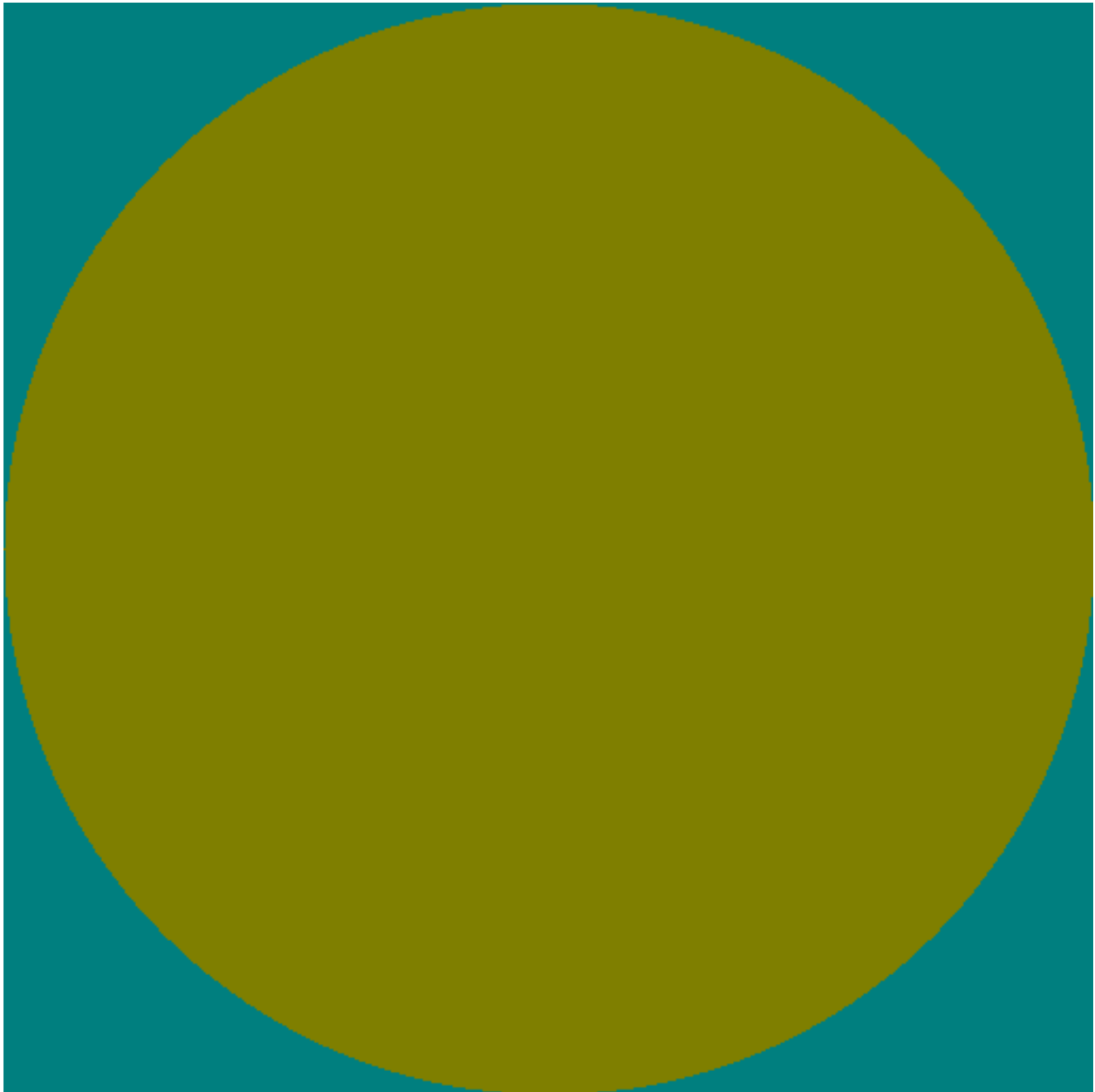
The **Constant** node presets an image with a constant color. The size of the created image, the constant color, the region of interest and the background color can all be specified.

Sample

Here is an example that shows how to used the **Constant** node.

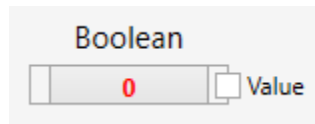


Here is the image created with the sample:



15.3.52 Boolean

Create and set a boolean value.



Outputs

Value (Type: Boolean)

The boolean value.

Comments

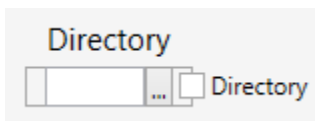
This node provides the means to create and set a boolean value.

The value can be changed by clicking on the button inside the node.

The title of the node can be edited.

15.3.53 Directory

Create and set a directory name.



Outputs

Directory (Type: String)

The directory name.

Comments

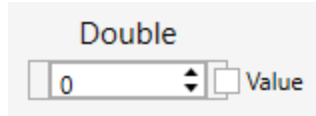
This node provides the means to create and set a directory name.

The value can be changed by typing the text inside the edit field in the node, or by selecting a directory using the [...] button.

The title of the node can be edited.

15.3.54 Double

Create and set a floating point value.



Outputs

Value (Type: Double)

The floating point value.

Comments

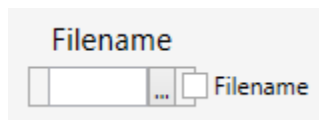
This node provides the means to create and set a floating point value.

The value can be changed by typing the floating point value inside the edit field in the node. Also, the spinner buttons can be used to increase or decrease the value.

The title of the node can be edited.

15.3.55 Filename

Create and set a filename.



Outputs

Filename (Type: String)

The filename.

Comments

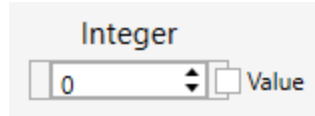
This node provides the means to create and set a filename.

The value can be changed by typing the text inside the edit field in the node, or by selecting a file using the [...] button.

The title of the node can be edited.

15.3.56 Integer

Create and set an integer value.



Outputs

Value (Type: `Int32`)

The integer value.

Comments

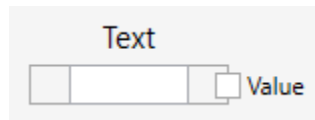
This node provides the means to create and set an integer value.

The value can be changed by typing the integer value inside the edit field in the node. Also, the spinner buttons can be used to increase or decrease the value.

The title of the node can be edited.

15.3.57 Text

Create and set a text value.



Outputs

Value (Type: `String`)

The text value.

Comments

This node provides the means to create and set a text value.

The value can be changed by typing the text inside the edit field in the node.

You can insert a few escape sequences to enter non-printable characters into the string.

escape sequence	non-printable character
<code>\t</code>	tab
<code>\n</code>	newline
<code>\r</code>	carriage return

The title of the node can be edited.

15.3.58 Context Menu

The context menu can be opened from the pipeline editor by clicking the right mouse button.

It shows commands to enter nodes and add additional input or output ports.

You can browse through the menu, and selecting an item will usually place the respective node in the pipeline editor canvas, at the position where the mouse was when the menu opened.

The only exceptions to this rule are the following entries:

entry	description
Add Input (Ctrl-I)	add an input port
Add Output (Ctrl-O)	add an output port
Copy (Ctrl-C)	copy the selected nodes
Paste (Ctrl-V)	paste the selected nodes

The search box at the top of the menu allows you to narrow down the menu. For example, if you are looking for commands that have something to do with histogram, start typing `his`, and the context menu will show only those commands and groups that have something to do with histograms.

Note that you do not need to click into the search edit box at the top of the menu, you can just start typing.

15.3.59 Convert Type

Converts the type of an image.

Inputs

Image (Type: Image)

The input image.

Type (Type: String)

The type of the resulting image.

Outputs

Image (Type: Image)

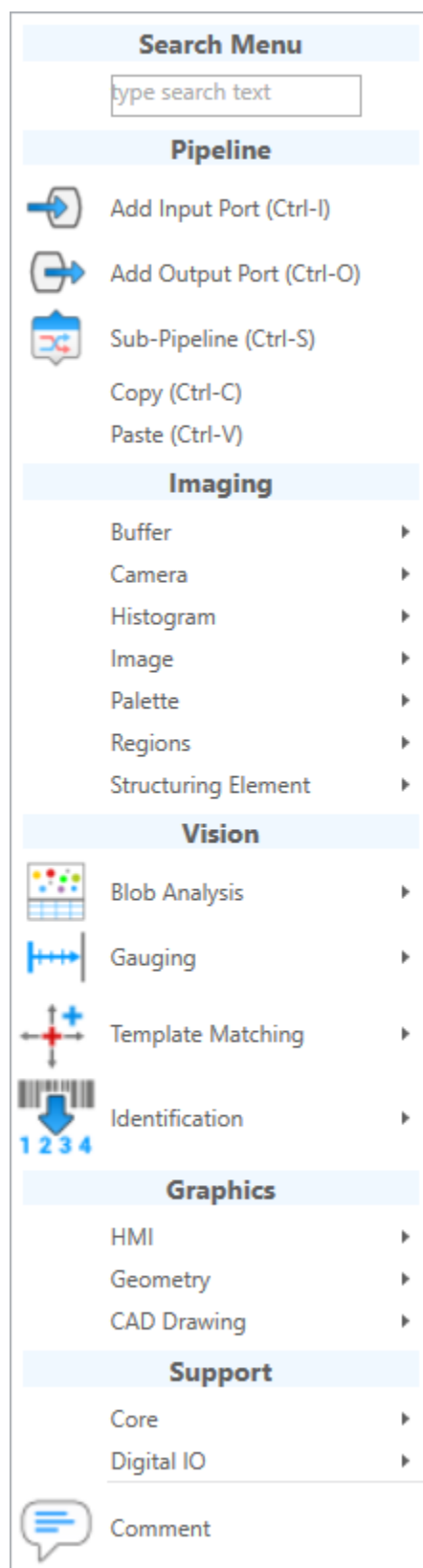
The output image.

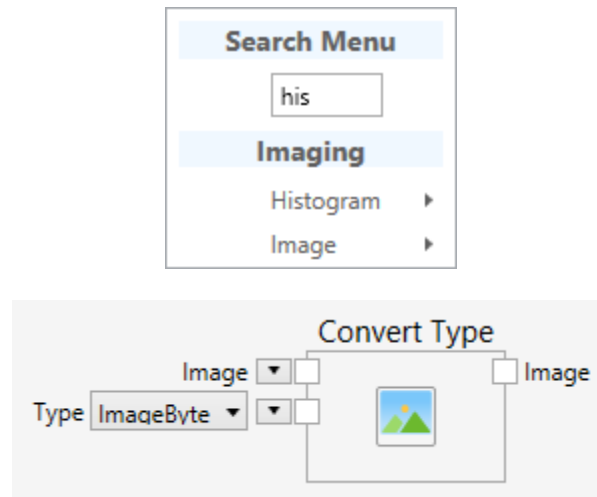
Comments

This function takes an image and converts it to the specified type.

The following types are available: `ImageByte`, `ImageUInt16`, `ImageUInt32`, `ImageDouble`, `ImageRgbByte`, `ImageRgbUInt32`, `ImageRgbUInt32` and `ImageRgbDouble`.

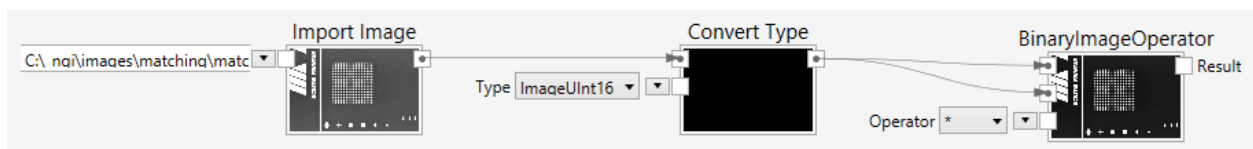
Usually, **nVision** works with variant image types and treats any specific image type transparently. Sometimes, however, one needs a specific type. This conversion helps to convert to a specific type.





Sample

Here is an example:



15.3.60 RGB -> Monochrome

Converts an image in the RGB color model to monochrome.

Inputs

Image (Type: Image)

The input image.

Outputs

Monochrome (Type: Image)

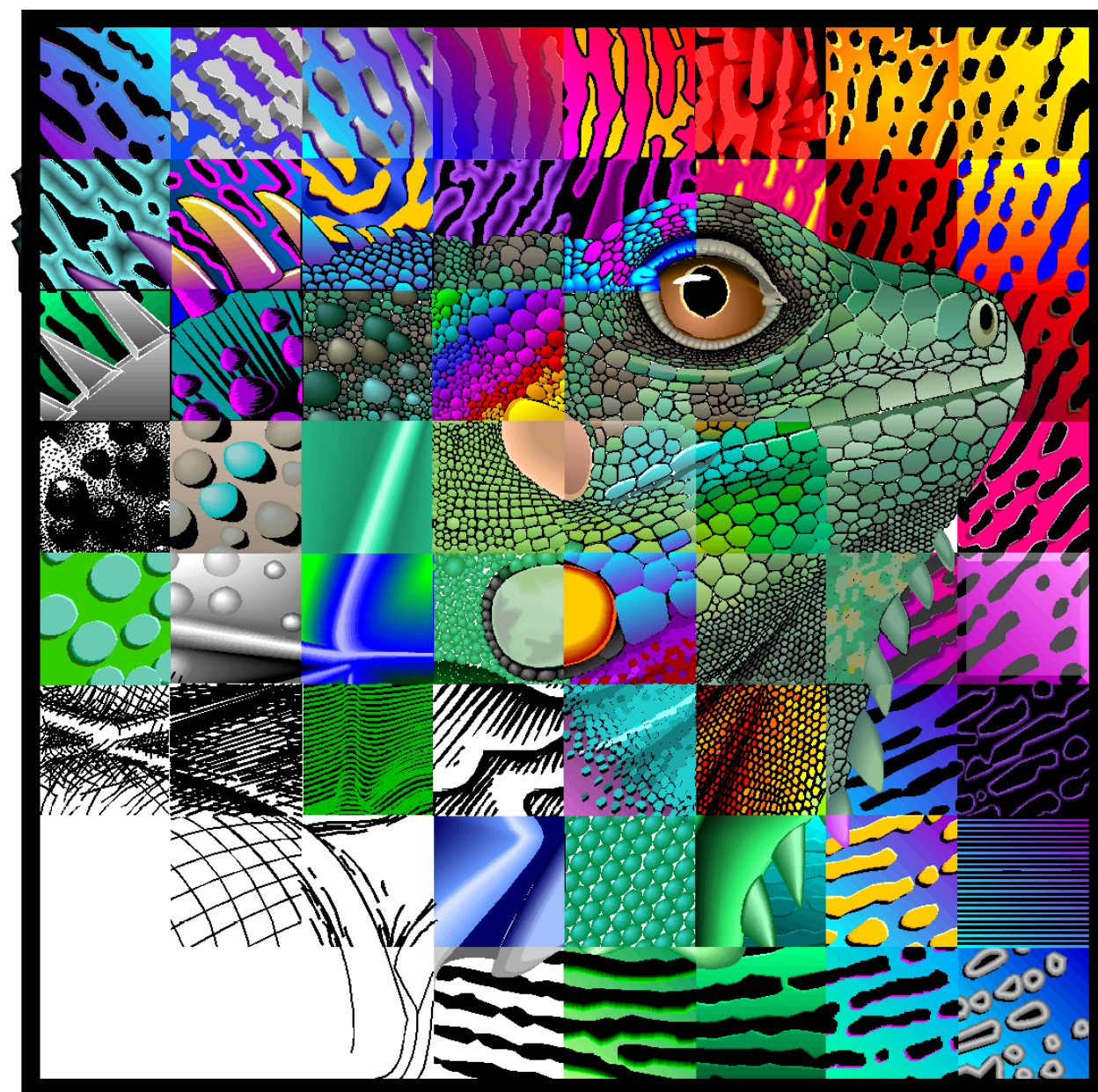
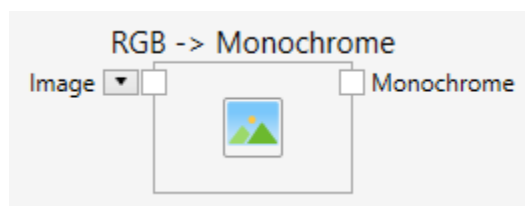
The output image.

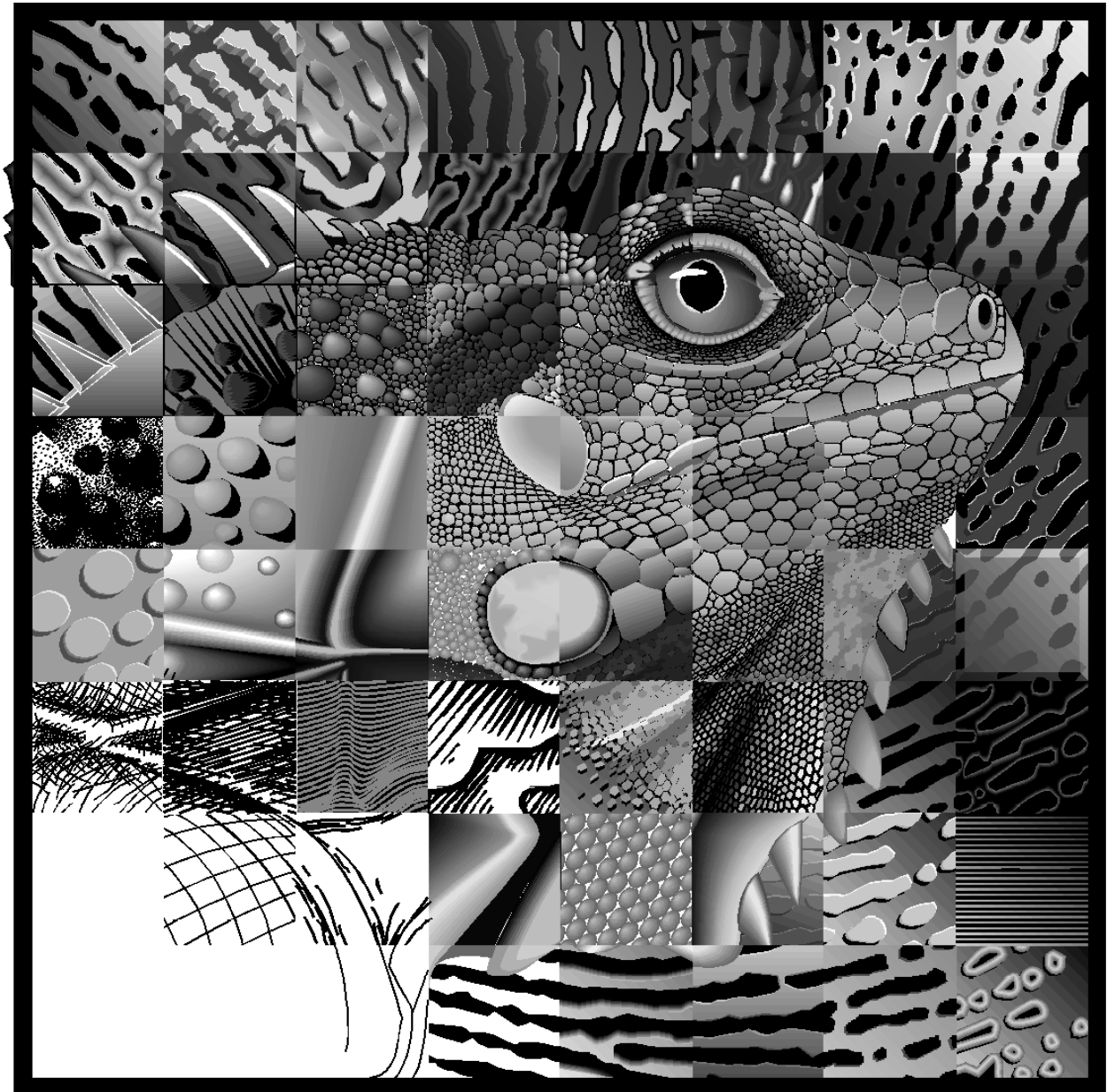
Comments

This function converts an image in the RGB color model to a monochrome image.

Here is an example of the original color image:

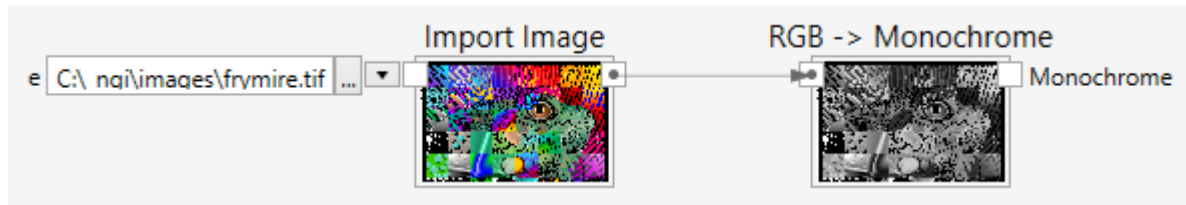
And here is the same image converted to monochrome:





Sample

Here is an example that converts a color image to monochrome.



15.3.61 Text -> CornerRadius

Converts text to a CornerRadius.

Inputs

Text (Type: `string`)

The text that is converted to a CornerRadius.

Outputs

CornerRadius (Type: `CornerRadius`)

The output CornerRadius.

Comments

A CornerRadius encodes radius values of four corners, for *TopLeft*, *TopRight*, *BottomRight* and *BottomLeft*.

You can specify one number, which is then used for all four corners, i.e. 50.

Alternatively, you can specify four numbers separated with semicolons, that are used for *TopLeft*, *TopRight*, *BottomRight* and *BottomLeft*, i.e. 5, 10, 20, 30.

15.3.62 Text -> Color

Converts text to a color.

Inputs

Text (Type: `string`)

The text that is converted to a color.

Outputs

Brush (Type: Rgba)















The output color.

Comments

There are various ways how you can specify the color.

By Name

Use the color name, such as Red or Orange. The name is case insensite, that is you can use red as well. These colors are available:

	AliceBlue	#FFF0F8FF		DarkTurquoise	#FF00CED1		LightSeaGreen	#FF20B2AA		PapayaWhip	#FFFFEFD5
	AntiqueWhite	#FFFAEBD7		DarkViolet	#FF9400D3		LightSkyBlue	#FF87CEFA		PeachPuff	#FFFDAB9
	Aqua	#FF00FFFF		DeepPink	#FFFF1493		LightSlateGray	#FF778899		Peru	#FFCD853F
	Aquamarine	#FF7FFFD4		DeepSkyBlue	#FF00BFFF		LightSteelBlue	#FFB0C4DE		Pink	#FFFC00CB
	Azure	#FFF0FFFF		DimGray	#FF696969		LightYellow	#FFFFFFE0		Plum	#FFDDA0DD
	Beige	#FFF5F5DC		DodgerBlue	#FF1E90FF		Lime	#FF00FF00		PowderBlue	#FFB0E0E6
	Bisque	#FFFFE4C4		Firebrick	#FFB22222		LimeGreen	#FF32CD32		Purple	#FF800080
	Black	#FF000000		FloralWhite	#FFFFFFA0		Linen	#FFFAF0E6		Red	#FFFF0000
	BlanchedAlmond	#FFFFEBCD		ForestGreen	#FF228B22		Magenta	#FFFF00FF		RosyBrown	#FFBC8F8F
	Blue	#FF0000FF		Fuchsia	#FFFF00FF		Maroon	#FF800000		RoyalBlue	#FF4169E1
	BlueViolet	#FF8A2BE2		Gainsboro	#FFDCDCDC		MediumAquamarine	#FF66CDAA		SaddleBrown	#FF8B4513
	Brown	#FFA52A2A		GhostWhite	#FFF8F8FF		MediumBlue	#FF0000CD		Salmon	#FFFA8072
	BurlyWood	#FFDEB887		Gold	#FFFD7000		MediumOrchid	#FFBA55D3		SandyBrown	#FFFA4A60
	CadetBlue	#FF5F9EA0		Goldenrod	#FFDAA520		MediumPurple	#FF9370DB		SeaGreen	#FF2E8B57
	Chartreuse	#FF7FFF00		Gray	#FF808080		MediumSeaGreen	#FF3CB371		SeaShell	#FFFFFFF5EE
	Chocolate	#FFD2691E		Green	#FF008000		MediumSlateBlue	#FF7B68EE		Sienna	#FFA0522D
	Coral	#FFF77F00		GreenYellow	#FFADFF2F		MediumSpringGreen	#FF00FA9A		Silver	#FFC0C0C0
	CornflowerBlue	#FF6495ED		Honeydew	#FFF0FFF0		MediumTurquoise	#FF48D1CC		SkyBlue	#FF87CEEB
	Cornsilk	#FFFFF8DC		HotPink	#FFF69B4		MediumVioletRed	#FFC71585		SlateBlue	#FF6A5ACD
	Crimson	#FFDC143C		IndianRed	#FFCD5C5C		MidnightBlue	#FF191970		SlateGray	#FF708090
	Cyan	#FF00FFFF		Indigo	#FF4B0082		MintCream	#FFF5FFFA		Snow	#FFFFFFFAFA
	DarkBlue	#FF00008B		Ivory	#FFFFFFF0F0		MistyRose	#FFFFE4E1		SpringGreen	#FF00FF7F
	DarkCyan	#FF008B8B		Khaki	#FFF0E68C		Moccasin	#FFFFE4B5		SteelBlue	#FF4682B4
	DarkGoldenrod	#FFB8860B		Lavender	#FFE6E6FA		NavajoWhite	#FFFFDEAD		Tan	#FFD2B48C
	DarkGray	#FFA9A9A9		LavenderBlush	#FFFFFFF0F5		Navy	#FF000080		Teal	#FF008080
	DarkGreen	#FF006400		LawnGreen	#FF7FCF00		OldLace	#FFFD5E6		Thistle	#FFD8BFD8
	DarkKhaki	#FFBDB76B		LemonChiffon	#FFFFFACD		Olive	#FF808000		Tomato	#FFFF6347
	DarkMagenta	#FF8B008B		LightBlue	#FFADD8E6		OliveDrab	#FF6B8E23		Transparent	#00FFFFFF
	DarkOliveGreen	#FF556B2F		LightCoral	#FFF08080		Orange	#FFFA5000		Turquoise	#FF40E0D0
	DarkOrange	#FF8C0000		LightCyan	#FFE0FFFF		OrangeRed	#FFFF4500		Violet	#FFEE82EE
	DarkOrchid	#FF9332CC		LightGoldenrodYellow	#FFFAFAD2		Orchid	#FFDA70D6		Wheat	#FFF5DEB3
	DarkRed	#FF8B0000		LightGray	#FFD3D3D3		PaleGoldenrod	#FFEEE8AA		White	#FFFFFFFFFF
	DarkSalmon	#FFE9967A		LightGreen	#FF90EE90		PaleGreen	#FF98FB98		WhiteSmoke	#FFF5F5F5
	DarkSeaGreen	#FF8FBC8F		LightPink	#FFF6B6C1		PaleTurquoise	#FFAFEEEE		Yellow	#FFFFFF00
	DarkSlateBlue	#FF483D8B		LightSalmon	#FFFA07A		PaleVioletRed	#FFDB7093		YellowGreen	#FF9ACD32
	DarkSlateGray	#FF2F4F4F									

By Color Components

Use a hash sign, followed by three (#rgb), four (#argb), six (#rrggbb) or eight (#aarrggbb) hexadecimal numbers. Examples are: #00F, #F00F, #0000FF, #FF0000FF.

15.3.63 Text -> Color

Converts text to a color.

Inputs

Text (Type: string)

The text that is converted to a color.

Outputs

Brush (Type: RgbaByte)

The output color.

Comments

There are various ways how you can specify the color.

By Name

Use the color name, such as Red or Orange. The name is case insensite, that is you can use red as well. These colors are available:

By Color Components

Use a hash sign, followed by three (#rgb), four (#argb), six (#rrggbb) or eight (#aarrggbb) hexadecimal numbers. Examples are: #00F, #F00F, #0000FF, #FF0000FF.

15.3.64 Text -> Brush

Converts text to a brush.

Inputs

Text (Type: string)

The text that is converted to a brush.

Outputs

Brush (Type: SolidColorBrush)

The output brush.










	AliceBlue	#FFF0F8FF		DarkTurquoise	#FF00CED1		LightSeaGreen	#FF20B2AA		PapayaWhip	#FFFEEFD5
	AntiqueWhite	#FFFAEBD7		DarkViolet	#FF9400D3		LightSkyBlue	#FF87CEFA		PeachPuff	#FFFDDAB9
	Aqua	#FF00FFFF		DeepPink	#FFFF1493		LightSlateGray	#FF778899		Peru	#FFCD853F
	Aquamarine	#FF7FFFD4		DeepSkyBlue	#FF00BFFF		LightSteelBlue	#FFB0C4DE		Pink	#FFFFC0CB
	Azure	#FFF0FFFF		DimGray	#FF696969		LightYellow	#FFFFFFF0		Plum	#FFDDA0DD
	Beige	#FFF5F5DC		DodgerBlue	#FF1E90FF		Lime	#FF00FF00		PowderBlue	#FFB0E0E6
	Bisque	#FFFFE4C4		Firebrick	#FFB22222		LimeGreen	#FF32CD32		Purple	#FF800080
	Black	#FF000000		FloralWhite	#FFFFFFAF0		Linen	#FFFAF0E6		Red	#FFFF0000
	BlanchedAlmond	#FFFEEBCD		ForestGreen	#FF228B22		Magenta	#FFFF00FF		RosyBrown	#FFBC8F8F
	Blue	#FF0000FF		Fuchsia	#FFFF00FF		Maroon	#FF800000		RoyalBlue	#FF4169E1
	BlueViolet	#FF8A2BE2		Gainsboro	#FFDCDCDC		MediumAquamarine	#FF66CDAA		SaddleBrown	#FF8B4513
	Brown	#FFA52A2A		GhostWhite	#FFF8F8FF		MediumBlue	#FF0000CD		Salmon	#FFFA8072
	BurlyWood	#FFDEB887		Gold	#FFFD7000		MediumOrchid	#FFBA55D3		SandyBrown	#FFFA4A60
	CadetBlue	#FF5F9EA0		Goldenrod	#FFDAA520		MediumPurple	#FF9370DB		SeaGreen	#FF2E8B57
	Chartreuse	#FF7FFF00		Gray	#FF808080		MediumSeaGreen	#FF3CB371		SeaShell	#FFFFFFF5EE
	Chocolate	#FFD2691E		Green	#FF008000		MediumSlateBlue	#FF7B68EE		Sienna	#FFA0522D
	Coral	#FFF77F50		GreenYellow	#FFADFF2F		MediumSpringGreen	#FF00FA9A		Silver	#FFC0C0C0
	CornflowerBlue	#FF6495ED		Honeydew	#FFF0FF00		MediumTurquoise	#FF48D1CC		SkyBlue	#FF87CEEB
	Cornsilk	#FFFFFF8DC		HotPink	#FFFF69B4		MediumVioletRed	#FFC71585		SlateBlue	#FF6A5ACD
	Crimson	#FFDC143C		IndianRed	#FFCD5C5C		MidnightBlue	#FF191970		SlateGray	#FF708090
	Cyan	#FF00FFFF		Indigo	#FF4B0082		MintCream	#FFF5FFFA		Snow	#FFFFFFFAFA
	DarkBlue	#FF00008B		Ivory	#FFFFFFF0F		MistyRose	#FFFFE4E1		SpringGreen	#FF00FF7F
	DarkCyan	#FF008B8B		Khaki	#FFF0E68C		Moccasin	#FFFFE4B5		SteelBlue	#FF4682B4
	DarkGoldenrod	#FFB8860B		Lavender	#FFE6E6FA		NavajoWhite	#FFFFDEAD		Tan	#FFD2B48C
	DarkGray	#FFA9A9A9		LavenderBlush	#FFFFFFF0F5		Navy	#FF000080		Teal	#FF008080
	DarkGreen	#FF006400		LawnGreen	#FF7CFC00		OldLace	#FFFD5E6		Thistle	#FFD8BFD8
	DarkKhaki	#FFBDB76B		LemonChiffon	#FFFFFACD		Olive	#FF808000		Tomato	#FFFF6347
	DarkMagenta	#FF8B008B		LightBlue	#FFADD8E6		OliveDrab	#FF6B8E23		Transparent	#00FFFFFF
	DarkOliveGreen	#FF556B2F		LightCoral	#FFF08080		Orange	#FFFA5000		Turquoise	#FF40E0D0
	DarkOrange	#FF8C0000		LightCyan	#FFE0FFFF		OrangeRed	#FF450000		Violet	#FFEE82EE
	DarkOrchid	#FF9332CC		LightGoldenrodYellow	#FFFAFAD2		Orchid	#FFDA70D6		Wheat	#FFF5DEB3
	DarkRed	#FF8B0000		LightGray	#FFD3D3D3		PaleGoldenrod	#FFEE8AA		White	#FFFFFFF000
	DarkSalmon	#FFE9967A		LightGreen	#FF90EE90		PaleGreen	#FF98FB98		WhiteSmoke	#FFF5F5F5
	DarkSeaGreen	#FF8FBC8F		LightPink	#FFF0F0F0		PaleTurquoise	#FFAFEEEE		Yellow	#FFFF0000
	DarkSlateBlue	#FF483D8B		LightSalmon	#FFFA07A		PaleVioletRed	#FFDB7093		YellowGreen	#FF9ACD32
	DarkSlateGray	#FF2F4F4F									

Comments

There are various ways how you can specify the brush color.

By Name

Use the color name, such as Red or Orange. The name is case insensite, that is you can use red as well. These colors are available:

 AliceBlue	#FFF0F8FF	 DarkTurquoise	#FF00CED1	 LightSeaGreen	#FF20B2AA	 PapayaWhip	#FFFFEFD5
 AntiqueWhite	#FFFAEBD7	 DarkViolet	#FF9400D3	 LightSkyBlue	#FF87CEFA	 PeachPuff	#FFFFDAB9
 Aqua	#FF00FFFF	 DeepPink	#FFFF1493	 LightSlateGray	#FF778899	 Peru	#FFCD853F
 Aquamarine	#FF7FFFD4	 DeepSkyBlue	#FF00BFFF	 LightSteelBlue	#FFB0C4DE	 Pink	#FFC0CB
 Azure	#FFF0FFFF	 DimGray	#FF696969	 LightYellow	#FFFFFFE0	 Plum	#FFDDA0DD
 Beige	#FFF5F5DC	 DodgerBlue	#FF1E90FF	 Lime	#FF00FF00	 PowderBlue	#FFB0E0E6
 Bisque	#FFFFE4C4	 Firebrick	#FFB22222	 LimeGreen	#FF32CD32	 Purple	#FF800080
 Black	#FF000000	 FloralWhite	#FFFFFFA0	 Linen	#FFFAF0E6	 Red	#FFFF0000
 BlanchedAlmond	#FFFEBECD	 ForestGreen	#FF228B22	 Magenta	#FFFF00FF	 RosyBrown	#FFBC8F8F
 Blue	#FF0000FF	 Fuchsia	#FFFF00FF	 Maroon	#FF800000	 RoyalBlue	#FF4169E1
 BlueViolet	#FF8A2BE2	 Gainsboro	#FFDCDCDC	 MediumAquamarine	#FF66CDAA	 SaddleBrown	#FF8B4513
 Brown	#FFA52A2A	 GhostWhite	#FFF8F8FF	 MediumBlue	#FF0000CD	 Salmon	#FFFA8072
 BurlyWood	#FFDEB887	 Gold	#FFFD7000	 MediumOrchid	#FFBA55D3	 SandyBrown	#FFFA4A60
 CadetBlue	#FF5F9EA0	 Goldenrod	#FFDAA520	 MediumPurple	#FF9370DB	 SeaGreen	#FF2E8B57
 Chartreuse	#FF7FFF00	 Gray	#FF808080	 MediumSeaGreen	#FF3CB371	 SeaShell	#FFFFFFF5EE
 Chocolate	#FFD2691E	 Green	#FF008000	 MediumSlateBlue	#FF7B68EE	 Sienna	#FFA0522D
 Coral	#FFF7F750	 GreenYellow	#FFADFF2F	 MediumSpringGreen	#FF00FA9A	 Silver	#FFC0C0C0
 CornflowerBlue	#FF6495ED	 Honeydew	#FFF0FF00	 MediumTurquoise	#FF48D1CC	 SkyBlue	#FF87CEEB
 Cornsilk	#FFFFF8DC	 HotPink	#FFFF69B4	 MediumVioletRed	#FFC71585	 SlateBlue	#FF6A5ACD
 Crimson	#FFDC143C	 IndianRed	#FFCD5C5C	 MidnightBlue	#FF191970	 SlateGray	#FF708090
 Cyan	#FF00FFFF	 Indigo	#FF4B0082	 MintCream	#FFF5FFFA	 Snow	#FFFFFFFAFA
 DarkBlue	#FF00008B	 Ivory	#FFFFFFF0	 MistyRose	#FFFFE4E1	 SpringGreen	#FF00FF7F
 DarkCyan	#FF008B8B	 Khaki	#FF0E68C	 Moccasin	#FFFFE4B5	 SteelBlue	#FF4682B4
 DarkGoldenrod	#FFB8860B	 Lavender	#FFE6E6FA	 NavajoWhite	#FFFFDEAD	 Tan	#FFD2B48C
 DarkGray	#FFA9A9A9	 LavenderBlush	#FFFFFF0F5	 Navy	#FF000080	 Teal	#FF008080
 DarkGreen	#FF006400	 LawnGreen	#FF7CFC00	 OldLace	#FFFD5E6	 Thistle	#FFD8BFD8
 DarkKhaki	#FFBDB76B	 LemonChiffon	#FFFFFACD	 Olive	#FF808000	 Tomato	#FFFF6347
 DarkMagenta	#FF8B008B	 LightBlue	#FFADD8E6	 OliveDrab	#FF6B8E23	 Transparent	#00FFFFFF
 DarkOliveGreen	#FF556B2F	 LightCoral	#FFF08080	 Orange	#FFFFA500	 Turquoise	#FF40E0D0
 DarkOrange	#FFF8C000	 LightCyan	#FFE0FFFF	 OrangeRed	#FFFF4500	 Violet	#FFEE82EE
 DarkOrchid	#FF932CC	 LightGoldenrodYellow	#FFFAFAD2	 Orchid	#FFDA70D6	 Wheat	#FFF5DEB3
 DarkRed	#FF8B0000	 LightGray	#FFD3D3D3	 PaleGoldenrod	#FFEE8AA	 White	#FFFFFFF
 DarkSalmon	#FFE9967A	 LightGreen	#FF90EE90	 PaleGreen	#FF98FB98	 WhiteSmoke	#FFF5F5F5
 DarkSeaGreen	#FF8FBC8F	 LightPink	#FFFFB6C1	 PaleTurquoise	#FFAFEEEE	 Yellow	#FFFFF000
 DarkSlateBlue	#FF483D8B	 LightSalmon	#FFFA0A7A	 PaleVioletRed	#FFDB7093	 YellowGreen	#FF9ACD32
 DarkSlateGray	#FF2F4F4F						

By Color Components

Use a hash sign, followed by three (#rgb), four (#argb), six (#rrggbb) or eight (#aarrggbb) hexadecimal numbers. Examples are: #00F, #F00F, #0000FF, #FF0000FF.

15.3.65 Text -> Brush

Converts text to a brush.

Inputs

Text (Type: `string`)

The text that is converted to a brush.

Outputs

Brush (Type: `SolidColorBrushByte`)

The output brush.

Comments

There are various ways how you can specify the brush color.

By Name

Use the color name, such as `Red` or `Orange`. The name is case insensite, that is you can use `red` as well. These colors are available:

By Color Components

Use a hash sign, followed by three (`#rgb`), four (`#argb`), six (`#rrggbb`) or eight (`#aarrggbb`) hexadecimal numbers. Examples are: `#00F`, `#F00F`, `#0000FF`, `#FF0000FF`.

15.3.66 Text -> Thickness

Converts text to a Thickness.

Inputs


































































Text (Type: `string`)

The text that is converted to a Thickness.

Outputs

Thickness (Type: `Thickness`)

The output Thickness.

	AliceBlue	#FFF0F8FF		DarkTurquoise	#FF00CED1		LightSeaGreen	#FF20B2AA		PapayaWhip	#FFFEEFD5
	AntiqueWhite	#FFFAEBD7		DarkViolet	#FF9400D3		LightSkyBlue	#FF87CEFA		PeachPuff	#FFFFDAB9
	Aqua	#FF00FFFF		DeepPink	#FFFF1493		LightSlateGray	#FF778899		Peru	#FFCD853F
	Aquamarine	#FF7FFFD4		DeepSkyBlue	#FF00BFFF		LightSteelBlue	#FFB0C4DE		Pink	#FFFFC0CB
	Azure	#FFF0FFFF		DimGray	#FF696969		LightYellow	#FFFFFFF0		Plum	#FFDDA0DD
	Beige	#FFF5F5DC		DodgerBlue	#FF1E90FF		Lime	#FF00FF00		PowderBlue	#FFB0E0E6
	Bisque	#FFFFE4C4		Firebrick	#FFB22222		LimeGreen	#FF32CD32		Purple	#FF800080
	Black	#FF000000		FloralWhite	#FFFFFFAF0		Linen	#FFFAF0E6		Red	#FFFF0000
	BlanchedAlmond	#FFFEBBCD		ForestGreen	#FF228B22		Magenta	#FFFF00FF		RosyBrown	#FFBC8F8F
	Blue	#FF0000FF		Fuchsia	#FFFF00FF		Maroon	#FF800000		RoyalBlue	#FF4169E1
	BlueViolet	#FF8A2BE2		Gainsboro	#FFDCDCDC		MediumAquamarine	#FF66CDAA		SaddleBrown	#FF8B4513
	Brown	#FFA52A2A		GhostWhite	#FFF8F8FF		MediumBlue	#FF0000CD		Salmon	#FFFA8072
	BurlyWood	#FFDEB887		Gold	#FFFD7000		MediumOrchid	#FFBA55D3		SandyBrown	#FFFA4A60
	CadetBlue	#FF5F9EA0		Goldenrod	#FFDAA520		MediumPurple	#FF9370DB		SeaGreen	#FF2E8B57
	Chartreuse	#FF7FFF00		Gray	#FF808080		MediumSeaGreen	#FF3CB371		SeaShell	#FFFFFFF5EE
	Chocolate	#FFD2691E		Green	#FF008000		MediumSlateBlue	#FF7B68EE		Sienna	#FFA0522D
	Coral	#FFF77F50		GreenYellow	#FFADFF2F		MediumSpringGreen	#FF00FA9A		Silver	#FFC0C0C0
	CornflowerBlue	#FF6495ED		Honeydew	#FFF0FF00		MediumTurquoise	#FF48D1CC		SkyBlue	#FF87CEEB
	Cornsilk	#FFFFFF8DC		HotPink	#FFFF69B4		MediumVioletRed	#FFC71585		SlateBlue	#FF6A5ACD
	Crimson	#FFDC143C		IndianRed	#FFCD5C5C		MidnightBlue	#FF191970		SlateGray	#FF708090
	Cyan	#FF00FFFF		Indigo	#FF4B0082		MintCream	#FFF5FFFA		Snow	#FFFFFFFAFA
	DarkBlue	#FF00008B		Ivory	#FFFFFFF0F		MistyRose	#FFFE4E1		SpringGreen	#FF00FF7F
	DarkCyan	#FF008B8B		Khaki	#FFF0E68C		Moccasin	#FFFE4B5		SteelBlue	#FF4682B4
	DarkGoldenrod	#FFB8860B		Lavender	#FFE6E6FA		NavajoWhite	#FFFFDEAD		Tan	#FFD2B48C
	DarkGray	#FFA9A9A9		LavenderBlush	#FFFFF0F5		Navy	#FF000080		Teal	#FF008080
	DarkGreen	#FF006400		LawnGreen	#FF7CFC00		OldLace	#FFFD5E6		Thistle	#FFD8BFD8
	DarkKhaki	#FFBDB76B		LemonChiffon	#FFFFFACD		Olive	#FF808000		Tomato	#FFFF6347
	DarkMagenta	#FF8B008B		LightBlue	#FFADD8E6		OliveDrab	#FF6B8E23		Transparent	#00FFFFFF
	DarkOliveGreen	#FF556B2F		LightCoral	#FFF08080		Orange	#FFFA5000		Turquoise	#FF40E0D0
	DarkOrange	#FF8C0000		LightCyan	#FFE0FFFF		OrangeRed	#FF450000		Violet	#FFEE82EE
	DarkOrchid	#FF9332CC		LightGoldenrodYellow	#FFFAFAD2		Orchid	#FFDA70D6		Wheat	#FFF5DEB3
	DarkRed	#FF8B0000		LightGray	#FFD3D3D3		PaleGoldenrod	#FFEE8AA		White	#FFFFFFF
	DarkSalmon	#FFE9967A		LightGreen	#FF90EE90		PaleGreen	#FF98FB98		WhiteSmoke	#FFF5F5F5
	DarkSeaGreen	#FF8FBC8F		LightPink	#FFFB6C1		PaleTurquoise	#FFAFEEEE		Yellow	#FFFFF000
	DarkSlateBlue	#FF483D8B		LightSalmon	#FFFA07A		PaleVioletRed	#FFDB7093		YellowGreen	#FF9ACD32
	DarkSlateGray	#FF2F4F4F									

Comments

A Thickness encodes values of four sides, for *Left*, *Top*, *Right* and *Bottom*.

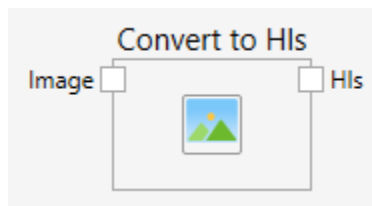
You can specify one number, which is then used for all four sides, i.e. 50.

Alternatively, you can specify two numbers separated with a semicolon, where the first is used for *Left* and *Right*, and the second for *Top* and *Bottom*, which creates a symmetric thickness, i.e. 5; 10.

Alternatively, you can specify four numbers separated with semicolons, that are used for *Left*, *Top*, *Right* and *Bottom*, i.e. 5, 10, 20, 30.

15.3.67 Convert to HLS

Converts an image to the HLS color model.



Inputs

Image (Type: Image)

The input image. This image can be of any color model.

Outputs

Hls (Type: Image)

The output image in the HLS color model.

Comments

This function converts an image to the HLS color model, regardless of the color space of the input image.

Here is an example of the original color image:

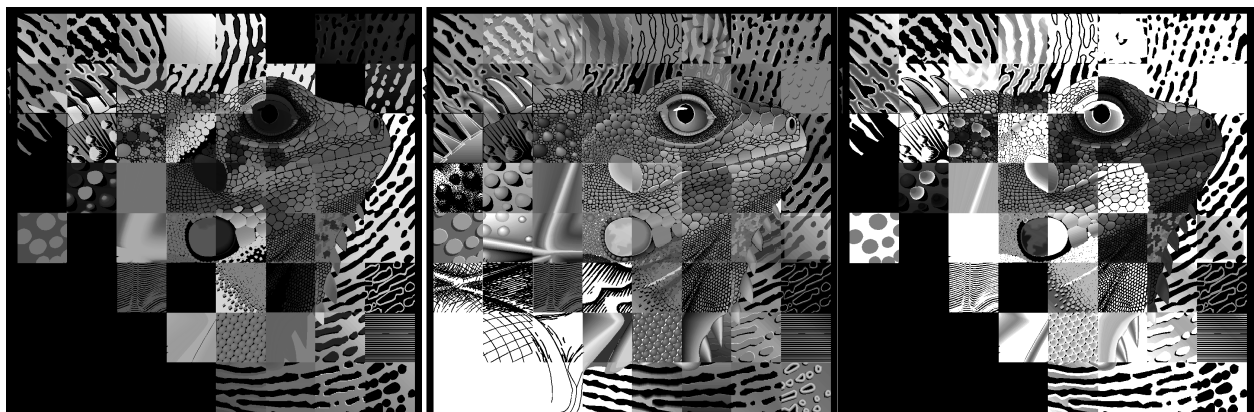
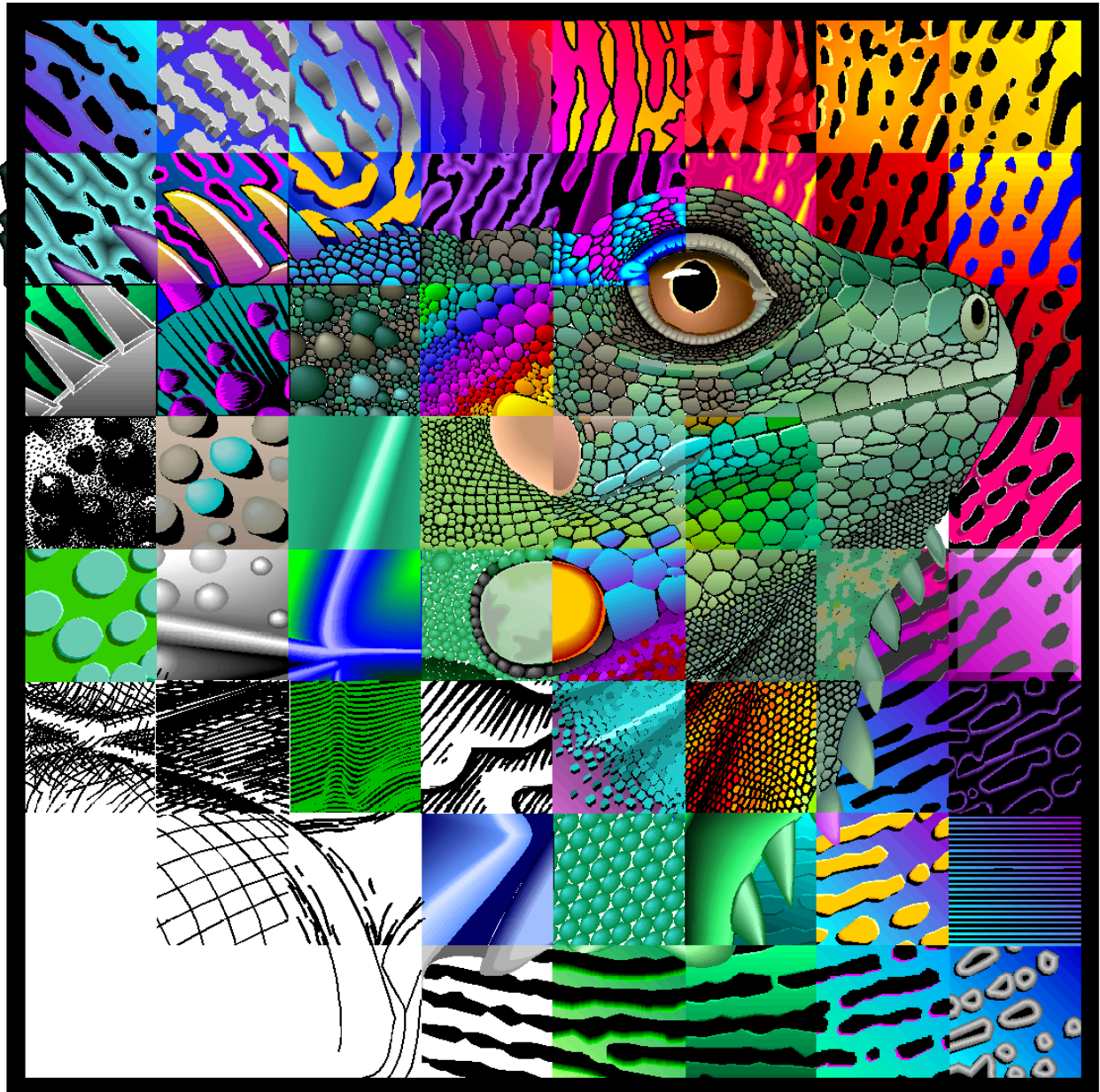
And here is how the hue, lightness and saturation channels look side by side.

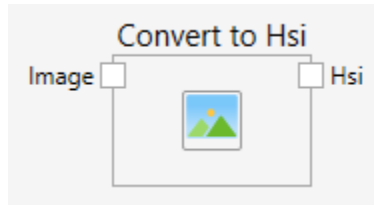
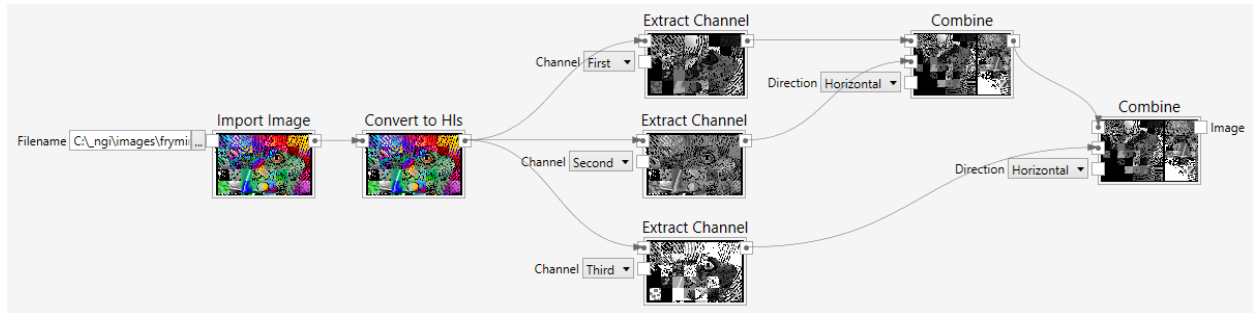
Sample

Here is an example:

15.3.68 Convert to HSI

Converts an image to the HSI color model.





Inputs

Image (Type: Image)

The input image. This image can be of any color model.

Outputs

Hsi (Type: Image)

The output image in the HSI color model.

Comments

This function converts an image to the HSI color model, regardless of the color space of the input image.

Here is an example of the original color image:

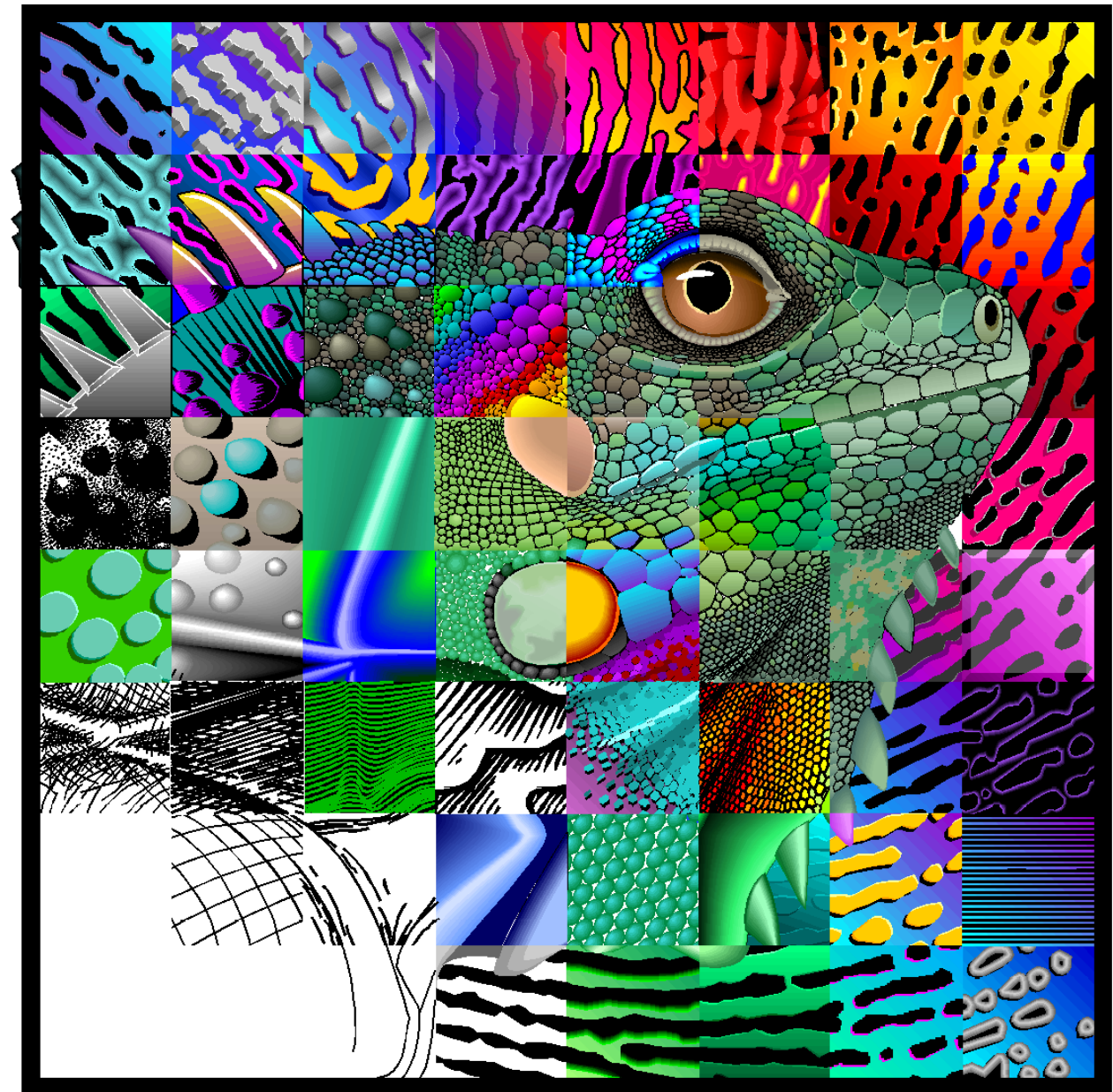
And here is how the hue, saturation and intensity channels look side by side.

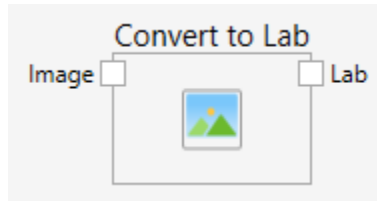
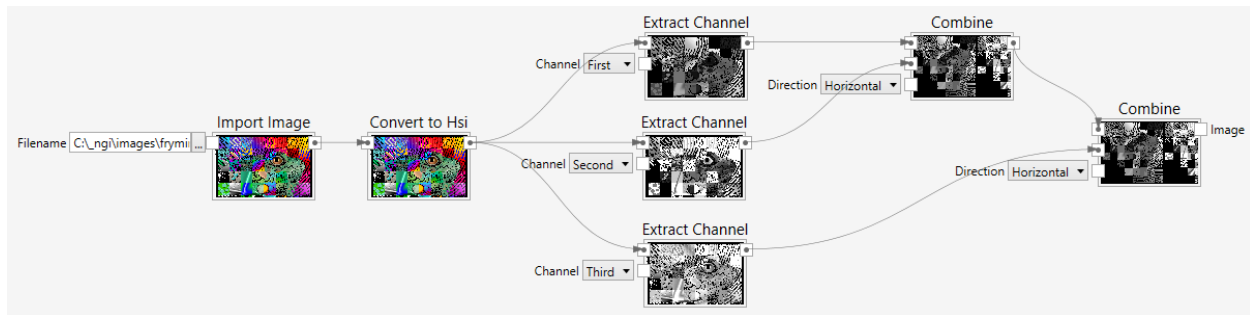
Sample

Here is an example:

15.3.69 Convert to LAB

Converts an image to the LAB color model.





Inputs

Image (Type: Image)

The input image. This image can be of any color model.

Outputs

Lab (Type: Image)

The output image in the LAB color model.

Comments

This function converts an image to the LAB color model, regardless of the color space of the input image.

Here is an example of the original color image:

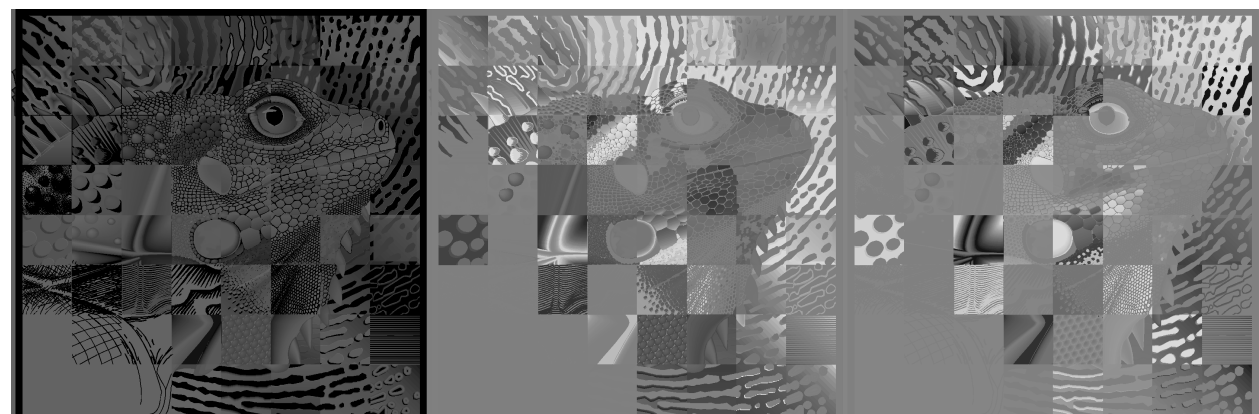
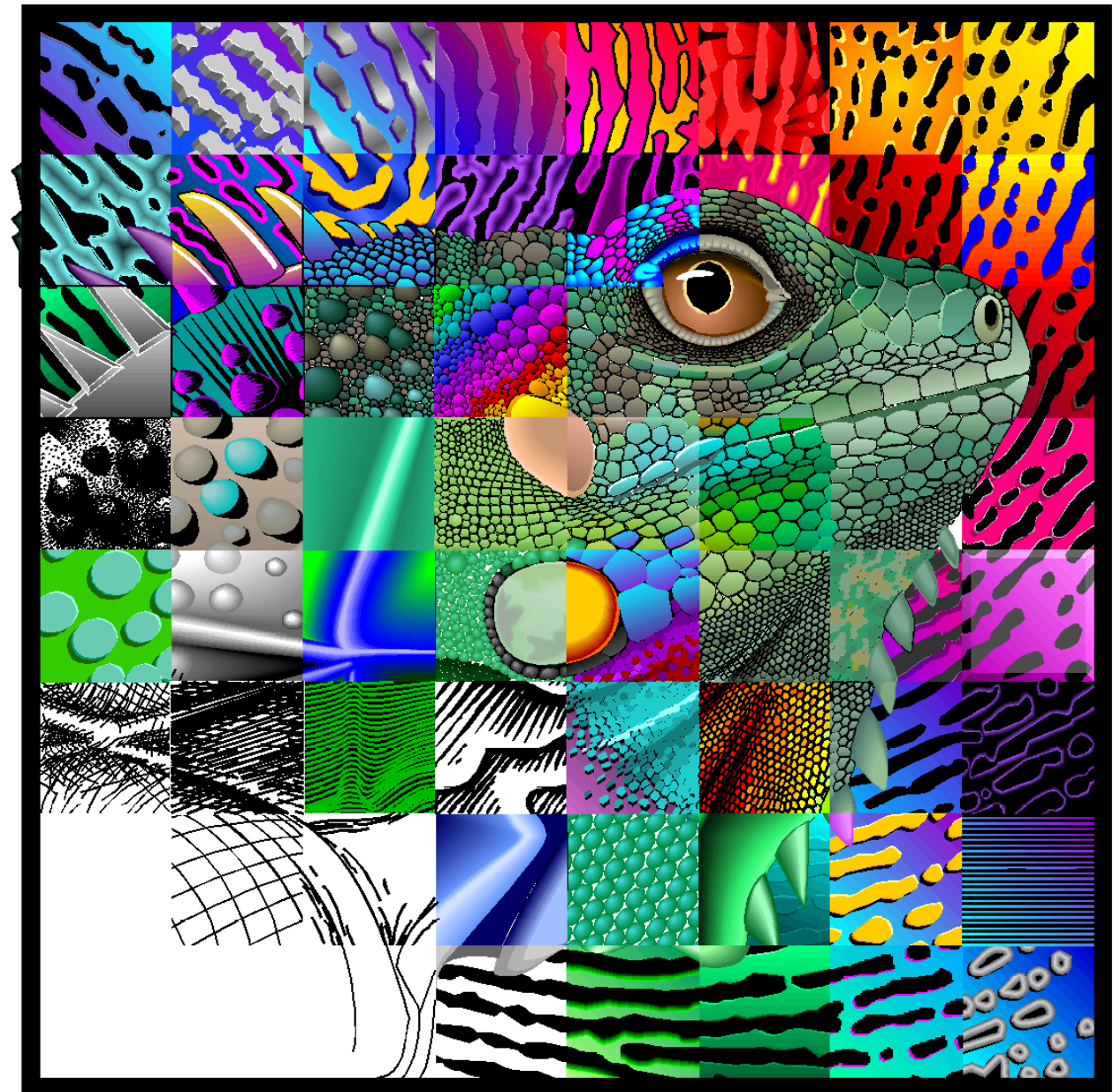
And here is how the L, a and b channels look side by side.

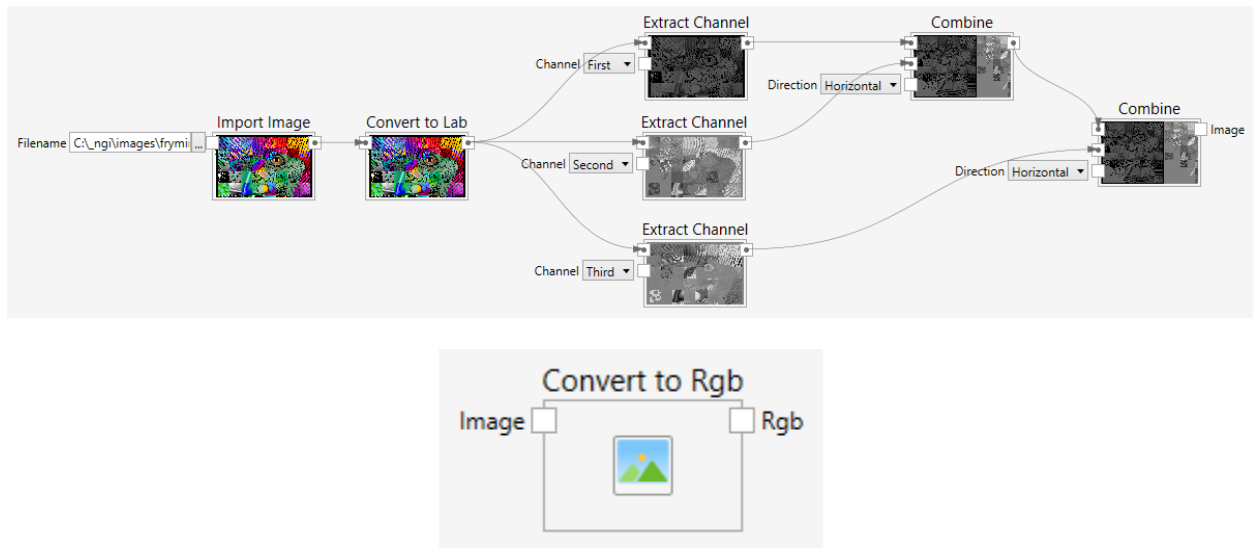
Sample

Here is an example:

15.3.70 Convert to Rgb

Converts an image to the RGB color model.





Inputs

Image (Type: Image)

The input image. This image can be of any color model.

Outputs

Rgb (Type: Image)

The output image in the RGB color model.

Comments

This function converts an image to the RGB color model, regardless of the color space of the input image.

15.3.71 Copy

You can use `Ctrl-C` or the **Copy (Ctrl-C)** command from the pipeline menu to copy the selected nodes and their inside connections to the clipboard.

You can select nodes by clicking them with the left mouse button. Hold down the `Ctrl` key if you want to add to (or subtract from) the selection.

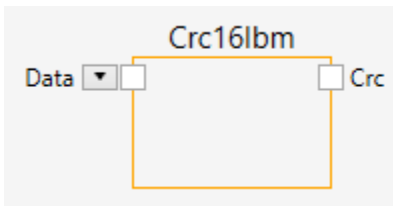
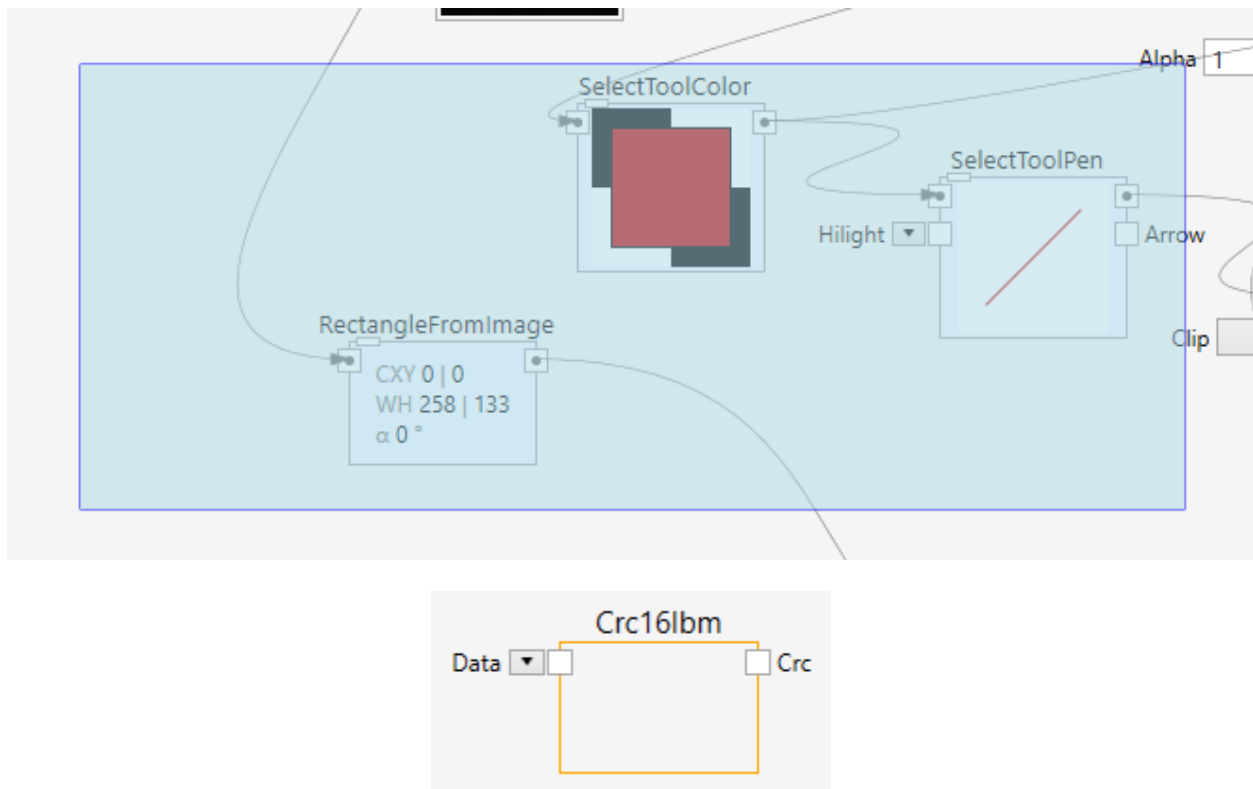
You can also select a set of nodes inside a rectangle that you drag while holding down the `Ctrl` key.

Once you have copied a set of nodes to the clipboard, you can paste them somewhere else.

15.3.72 Crc16Ibm

The CRC16 IBM cyclic redundancy check.

Calculates the CRC using the CRC16 IBM algorithm.



Inputs

Data (Type: DataList)

The data to write in the form of a DataList (a list of bytes).

Outputs

Crc (Type: Byte)

The result of the cyclic redundancy check.

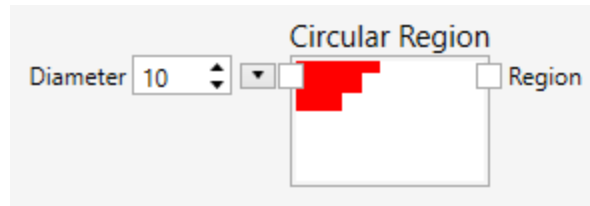
Comments

The CRC16 IBM algorithm is one implementation of a cyclic redundancy check, used for error detection.

For more information, have a look at https://en.wikipedia.org/wiki/Cyclic_redundancy_check.

15.3.73 Circular Region

Creates a circular region.



Inputs

Diameter (Type: Double)

The diameter of the circular region.

Outputs

Region (Type: Region)

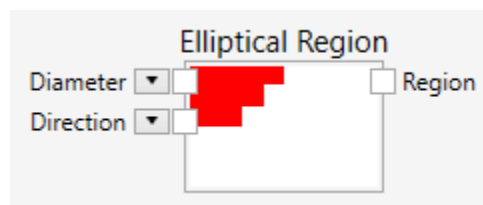
The circular region.

Comments

The region is centered on the origin. Regions centered on the origin are suitable for morphological operations, such as erosion, dilation, etc. The region is used as a structuring element and the shape of the region affects the morphological operation.

15.3.74 Elliptical Region

Creates an elliptical region.



Inputs

Diameter (Type: VectorDouble)

The diameter of the elliptical region.

Direction (Type: Direction)

The direction of the elliptical region.

Outputs

Region (Type: Region)

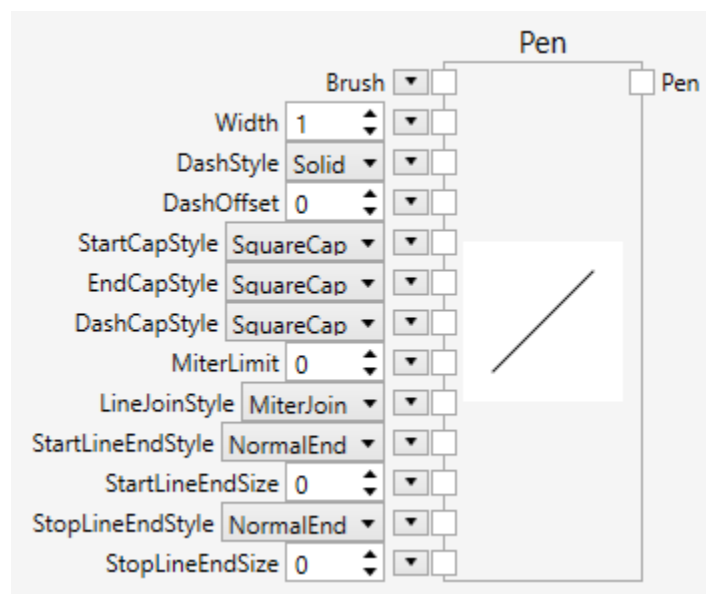
The elliptical region.

Comments

The region is centered on the origin. Regions centered on the origin are suitable for morphological operations, such as erosion, dilation, etc. The region is used as a structuring element and the shape of the region affects the morphological operation.

15.3.75 Pen

Creates a pen. A pen is used to specify outlines.



Inputs

Brush (Type: SolidColorBrushByte)

The brush.

Width (Type: Int32)

The width of the pen. A positive width scales with the zoom factor, a negative width does not scale.

DashStyle (Type: String)

The dash style, Solid, Dashed, Dotted, Dashdotted or Dashdotdotted.

DashOffset (Type : Double)

The offse into the dash pattern.

StartCapStyle (Type: String)

The cap style of the start point, FlatCap, SquareCap, RoundCap or TriangleCap.

EndCapStyle (Type: String)

The cap style of the end point, FlatCap, SquareCap, RoundCap or TriangleCap.

DashCapStyle (Type: String)

The dash cap style, FlatCap, SquareCap, RoundCap or TriangleCap.

MiterLimit (Type: Double)

The miter limit.

LineJoinStyle (Type: String)

The line join style, MiterJoin, BevelJoin, RoundJoin or MiterOrBevelCap.

StartLineEndStyle (Type: String)

The line end style of the start point, NormalEnd, FilledDiskEnd, FilledSquareEnd, FilledDiamondEnd, FilledArrowOutEnd, FilledArrowInEnd, BarEnd, BackslashEnd, SlashEnd, ArrowOutEnd, ArrowInEnd, FilledSlimArrowOutEnd, FilledSlimArrowInEnd, SlimArrowOutEnd or SlimArrowInEnd.

StartLineEndSize (Type: Double)

The size of the start line end.

StopLineEndStyle (Type: String)

The line end style of the stop point, NormalEnd, FilledDiskEnd, FilledSquareEnd, FilledDiamondEnd, FilledArrowOutEnd, FilledArrowInEnd, BarEnd, BackslashEnd, SlashEnd, ArrowOutEnd, ArrowInEnd, FilledSlimArrowOutEnd, FilledSlimArrowInEnd, SlimArrowOutEnd or SlimArrowInEnd.

StopLineEndSize (Type: Double)

The size of the stop line end.

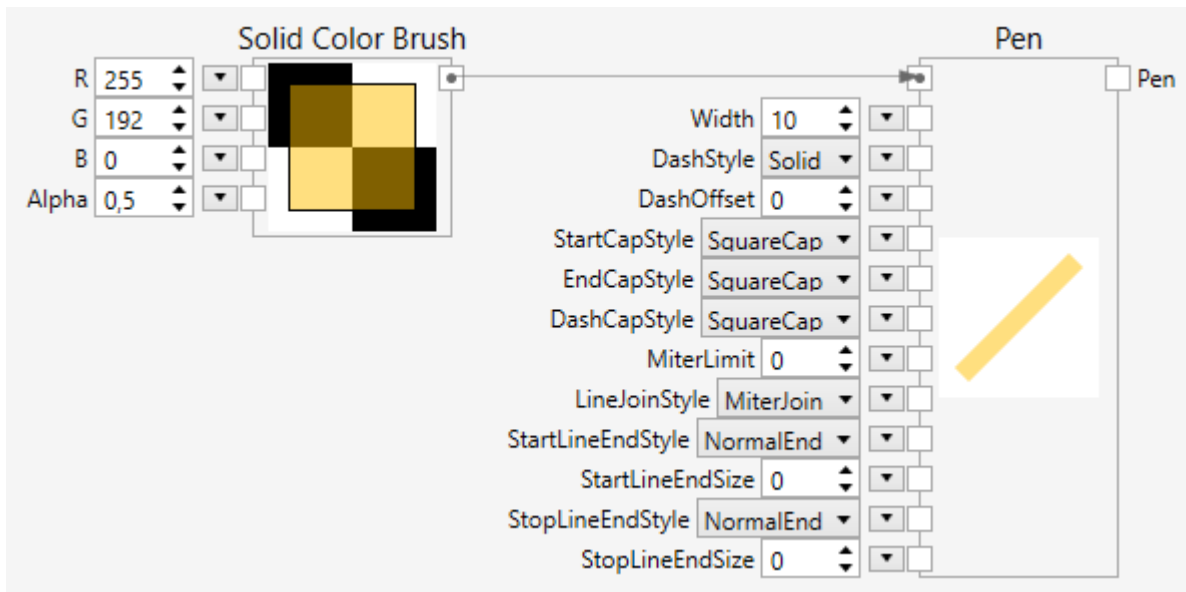
Outputs

Pen (Type: PenByte)

The pen.

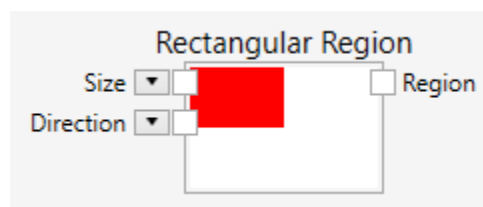
Sample

Here is an example that creates a yellowish half transparent brush and a wide pen using this brush.



15.3.76 Rectangular Region

Creates a rectangular region.



Inputs

Size (Type: VectorDouble)

The size of the rectangular region.

Direction (Type: Direction)

The direction of the rectangular region.

Outputs**Region (Type: Region)**

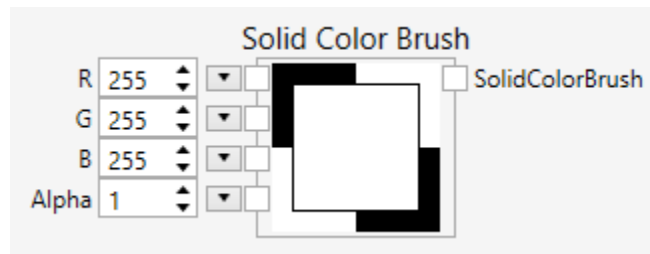
The rectangular region.

Comments

The region is centered on the origin. Regions centered on the origin are suitable for morphological operations, such as erosion, dilation, etc. The region is used as a structuring element and the shape of the region affects the morphological operation.

15.3.77 Solid Color Brush

Creates a solid color brush. A brush is used to specify fills.

**Inputs****R (Type: Byte)**

The red primary.

G (Type: Byte)

The green primary.

B (Type: Byte)

The blue primary.

Alpha (Type: Double)

The opacity.

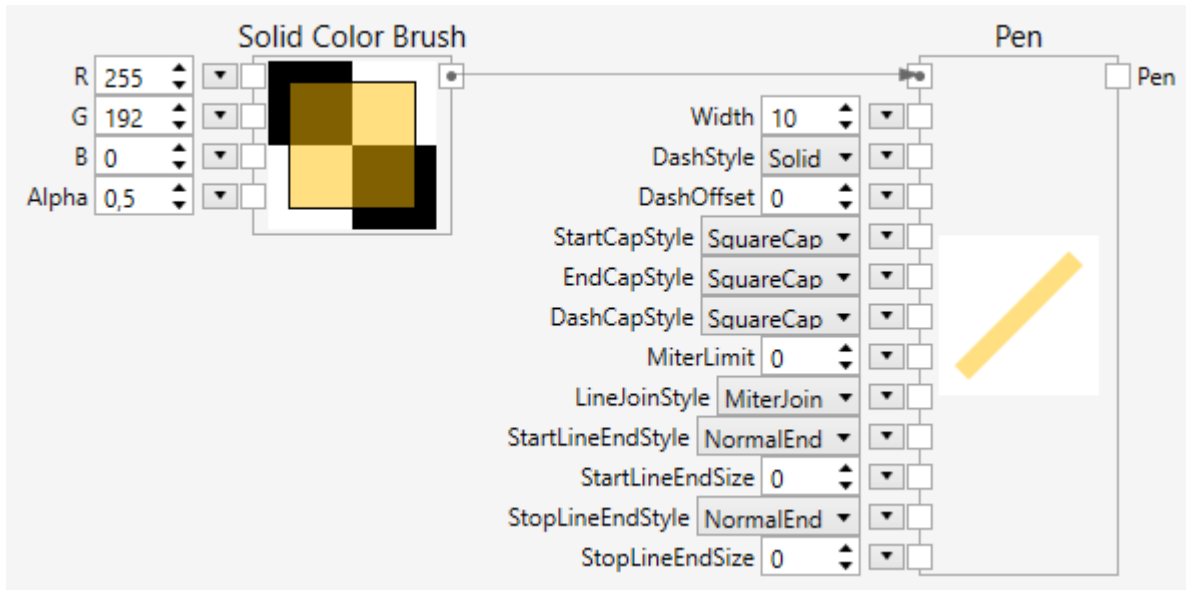
Outputs

SolidColorBrush (Type: SolidColorBrushByte)

The brush.

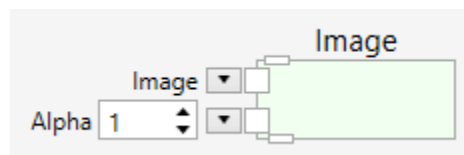
Sample

Here is an example that creates a yellowish half transparent brush and a wide pen using this brush.



15.3.78 Image

The **Image** node is used to display an image.



At the top of the **Image** node is a pin that allows it to connect to a parent widget (usually a **Display** widget node, but other widget nodes can be used as well). The image will be displayed only, if a parent connection is made.

At the bottom of the **Image** node is a pin that allows it to connect children to the image widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

Inputs

Image (Type: Image)

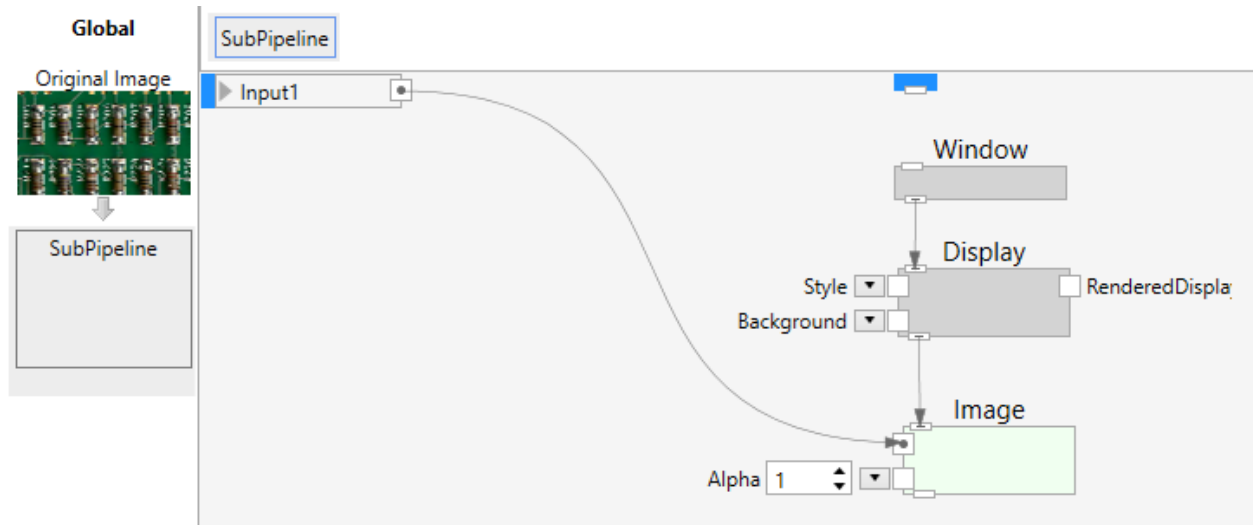
The image to display inside the widget.

Alpha (Type: Double)

The opacity of the image display.

Example

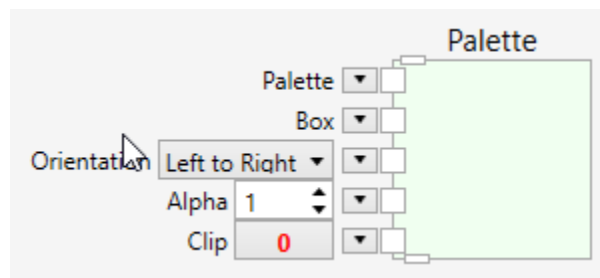
Here is an example that displays an image inside a display:



Below the **Window** node is a **Display** and below the **Display** is an **Image**.

15.3.79 Palette

The **Palette** node is used to display a palette.



At the top of the **Palette** node is a pin that allows it to connect to a parent widget. The palette will be displayed only, if a parent connection is made.

At the bottom of the **Palette** node is a pin that allows it to connect children to the palette widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

Inputs

Palette (Type: Palette)

The palette to display inside the widget.

Box (Type: BoxDouble)

A position for the palette display.

Orientation (Type: String)

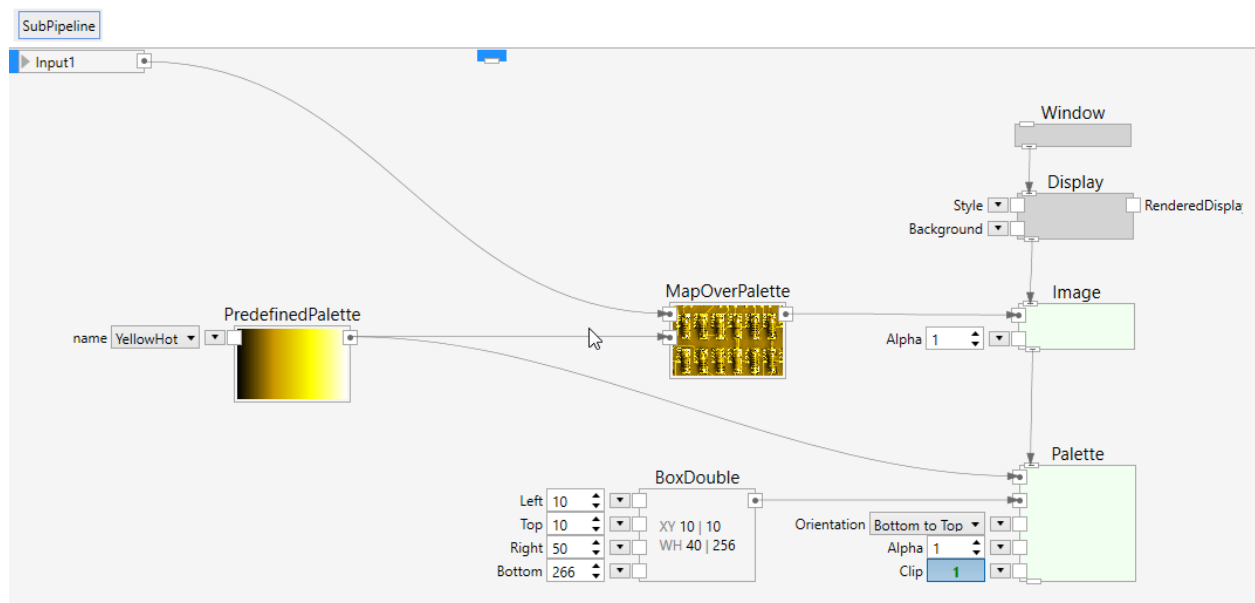
Orientation of the palette display: *LeftToRight*, *RightToLeft*, *TopToBottom* or *BottomToTop*.

Alpha (Type: Double)

The opacity of the palette display.

Example

Here is an example that displays an palette on top of an image. This pipeline

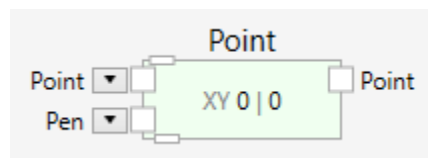
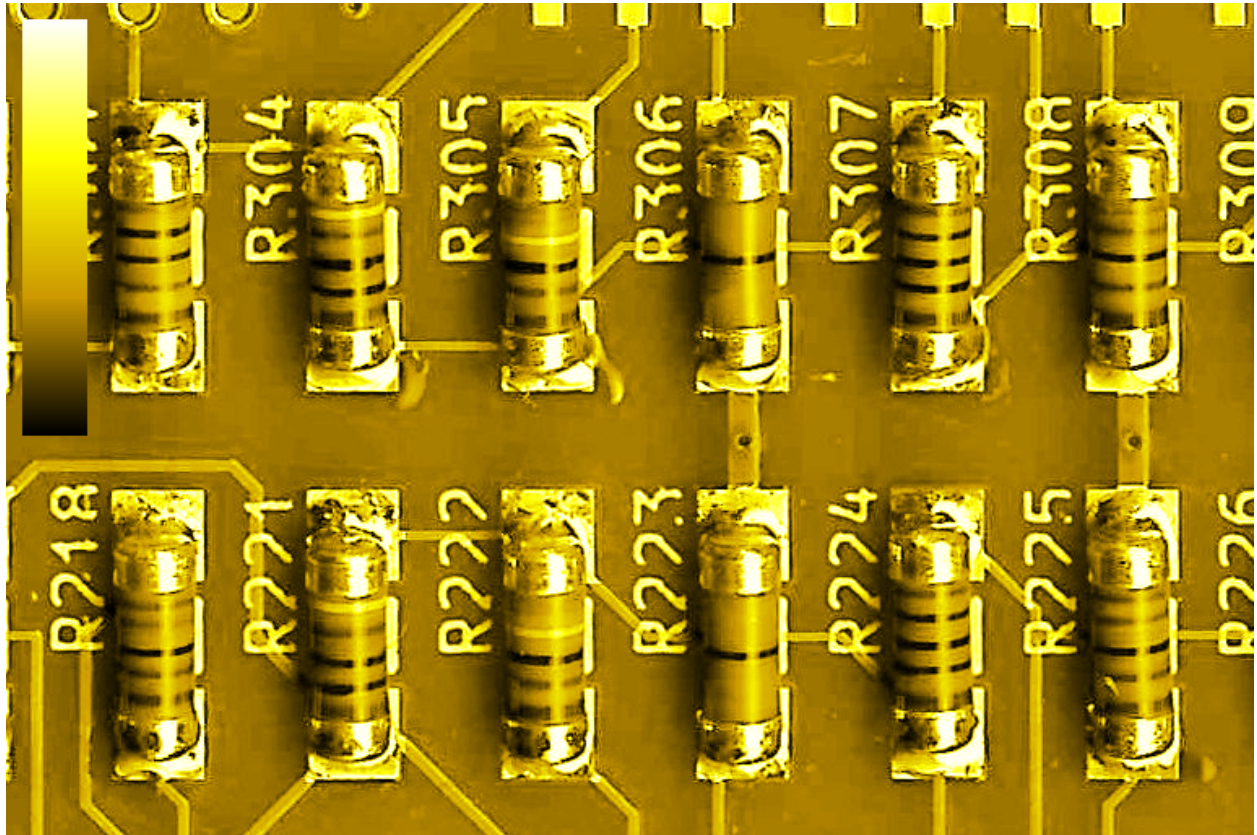


shows the following result:

15.3.80 Point

The **Point** node is used to display a point.

At the top of the **Point** node is a pin that allows it to connect to a parent widget. The point will be displayed only, if a parent connection is made.



At the bottom of the **Point** node is a pin that allows it to connect children to the point widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

Inputs

Point (Type: `Point`)

The position of the point. If this pin is connected, the incoming position is used and the point is fixed at this position. If this pin is left open (not connected), at the beginning the default position of (0, 0) is used and the point can be dragged around by pointing at it with the mouse, selecting it with a press of the left mouse button, and moving the mouse while holding down the left mouse button.

Pen (Type: `Pen`)

The pen used to draw the point.

Outputs

Point (Type: `Point`)

The position of the point.

Example

Here is an example that shows how a **Point** widget is used as an anchor to position children.

15.3.81 Region

The **Region** node is used to display a region.

At the top of the **Region** node is a pin that allows it to connect to a parent widget. The region will be displayed only, if a parent connection is made.

At the bottom of the **Region** node is a pin that allows it to connect children to the region widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

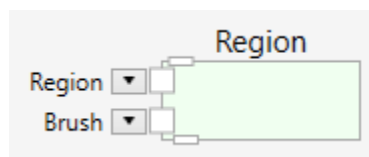
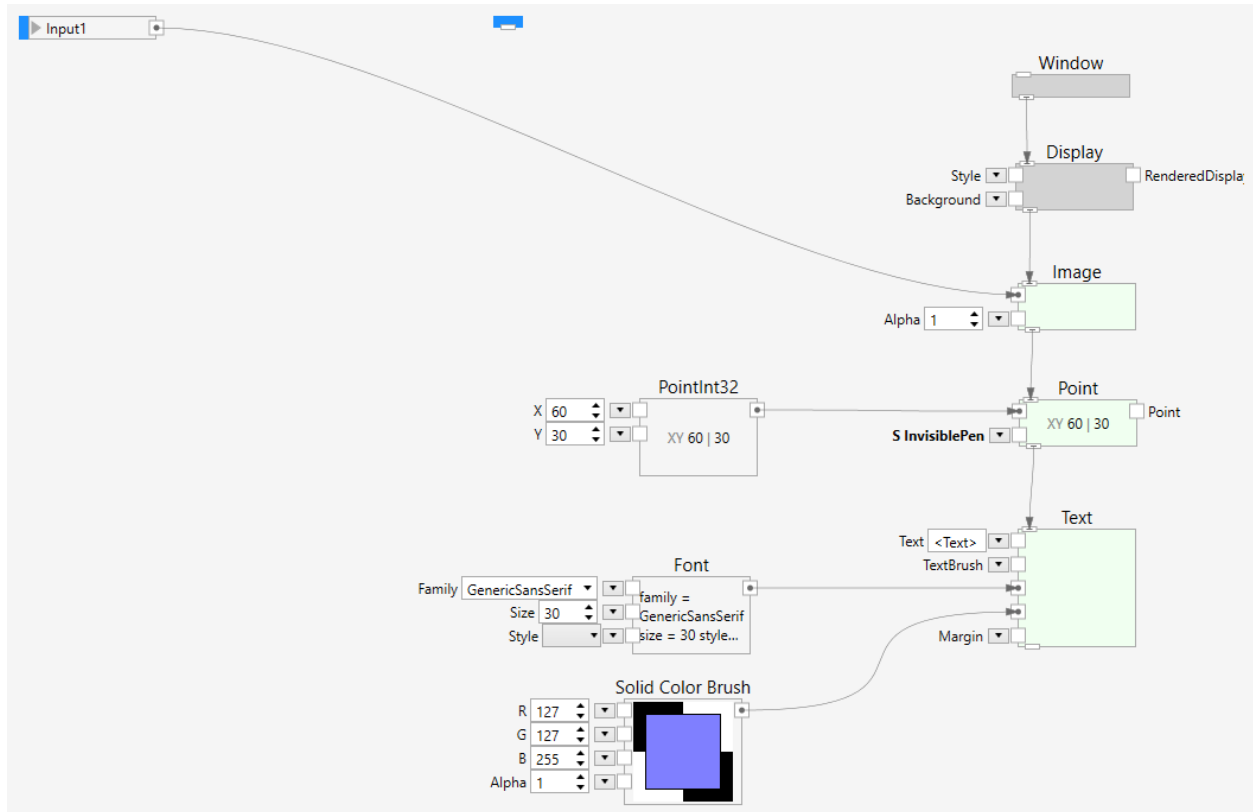
Inputs

Region (Type: `Region`)

The region to display inside the widget.

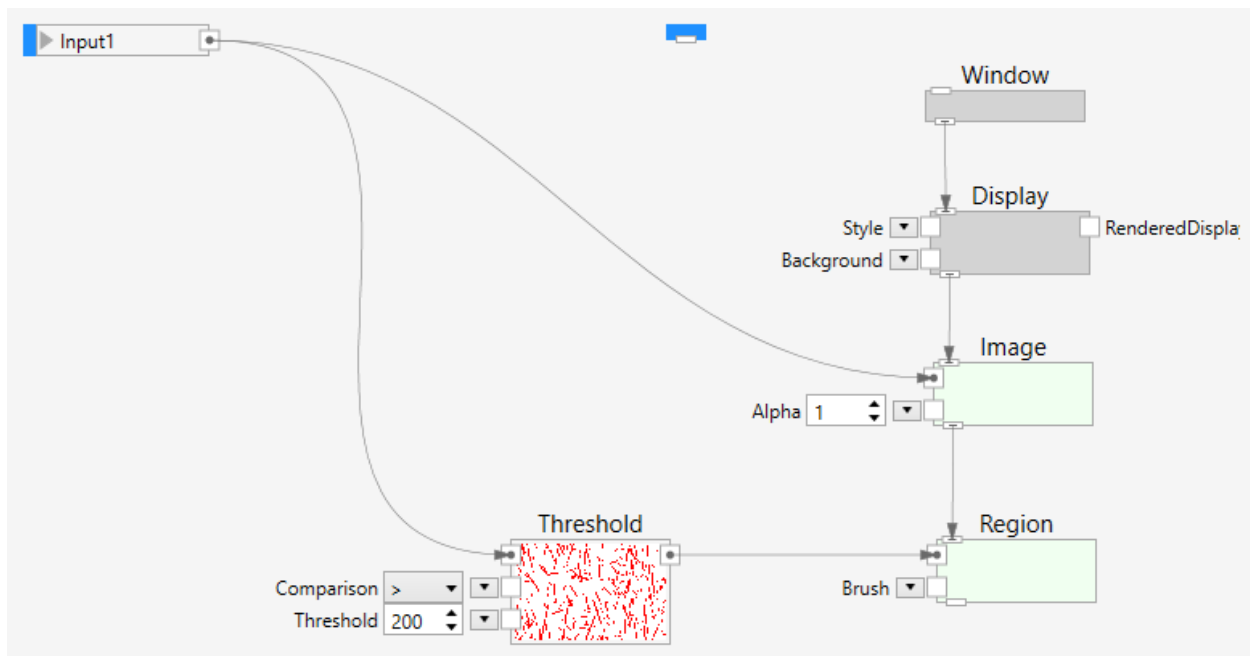
Brush (Type: `SolidColorBrush`)

A brush for the region display.



Example

Here is an example that displays a region on top of an image. This pipeline



shows the following result:

15.3.82 Regions

The **Regions** node is used to display a list of regions.

At the top of the **Regions** node is a pin that allows it to connect to a parent widget. The region will be displayed only, if a parent connection is made.

At the bottom of the **Regions** node is a pin that allows it to connect children to the regions widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

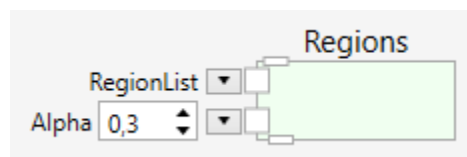
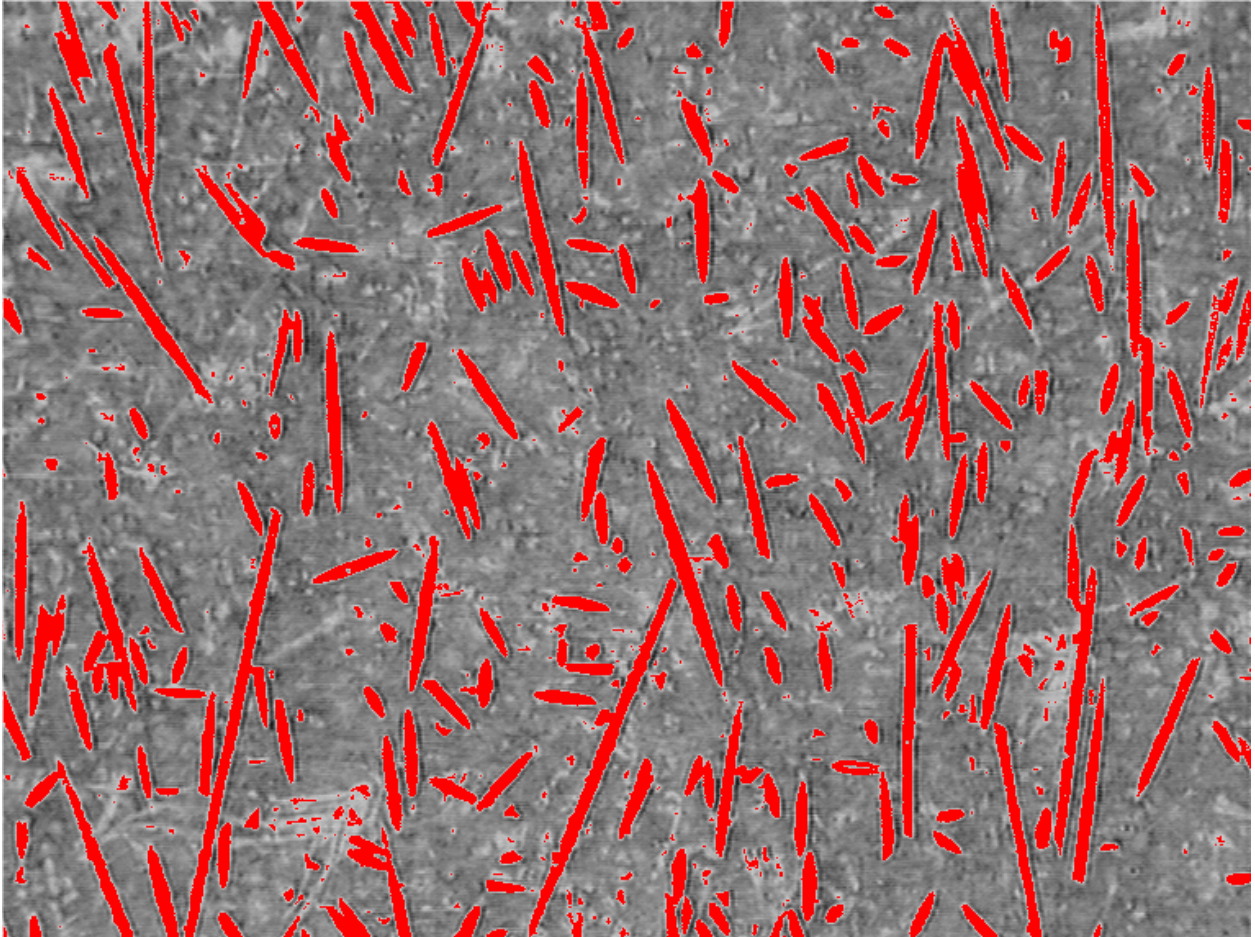
Inputs

RegionList (Type: RegionList)

The list of regions to display inside the widget. The different regions are colored with a total of six colors: red, green, blue, yellow, cyan, magenta. These colors are used in turn, and recycled as needed.

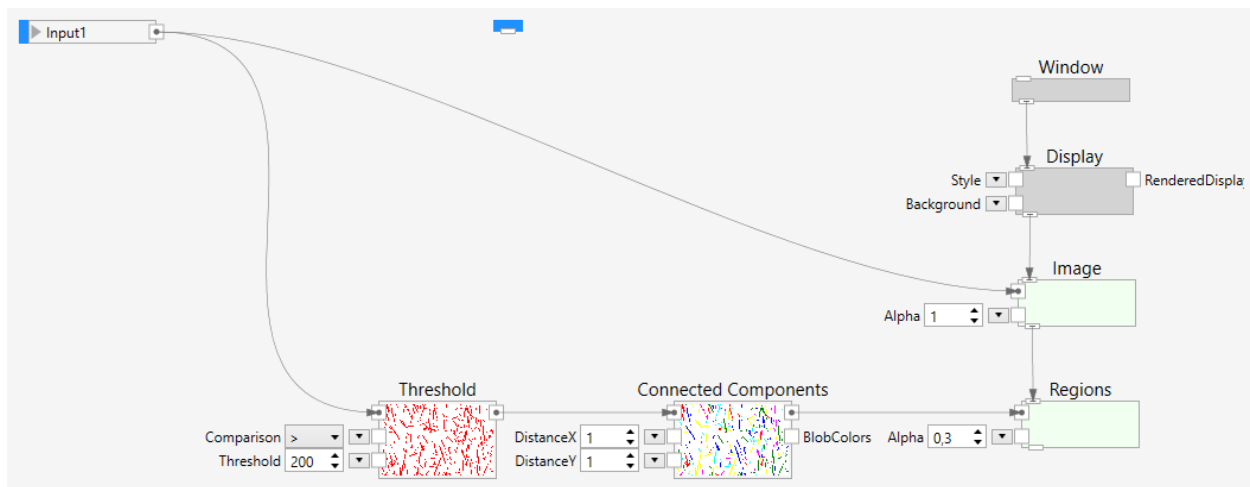
Alpha (Type: Double)

The opacity for the regions display: 0 is fully transparent, 1 is fully opaque.



Example

Here is an example that displays a list of regions on top of an image. This pipeline



shows the following result:

15.3.83 Text

The **Text** node is used to display a text.

At the top of the **Text** node is a pin that allows it to connect to a parent widget. The region will be displayed only, if a parent connection is made.

At the bottom of the **Text** node is a pin that allows it to connect children to the text widget. The parent, grandparent, children and grandchildren define the layering of the widgets and the graphical outcome.

Children are layered on top of the parent.

Inputs

Text (Type: String)

The text to be displayed.

TextBrush (Type: SolidColorBrush)

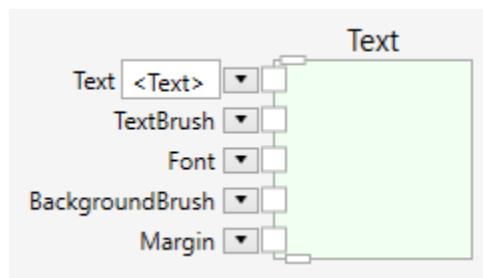
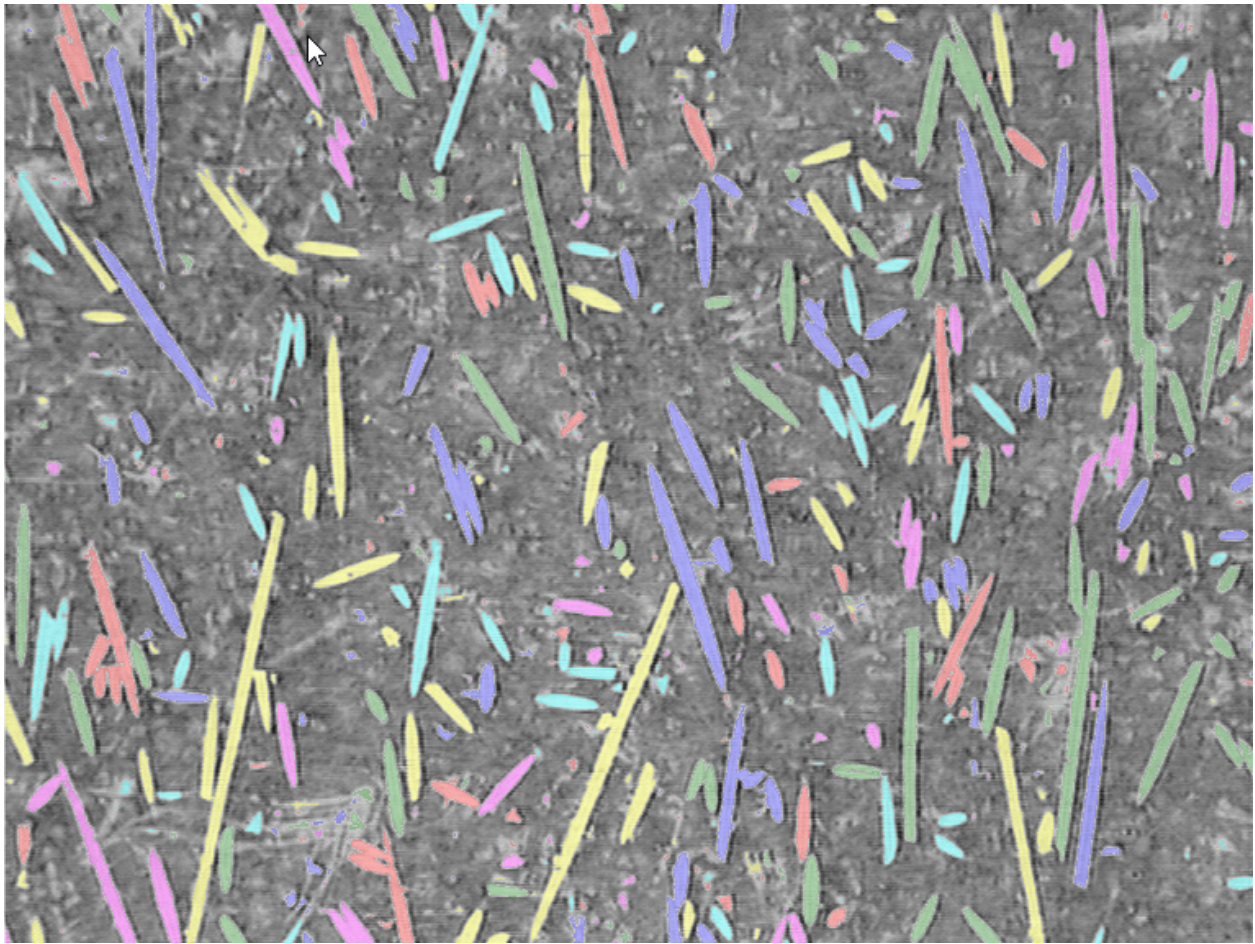
The brush used to draw the text.

Font (Type: Font)

The font used to draw the text.

BackgroundBrush (Type: SolidColorBrush)

The brush used to draw the background.

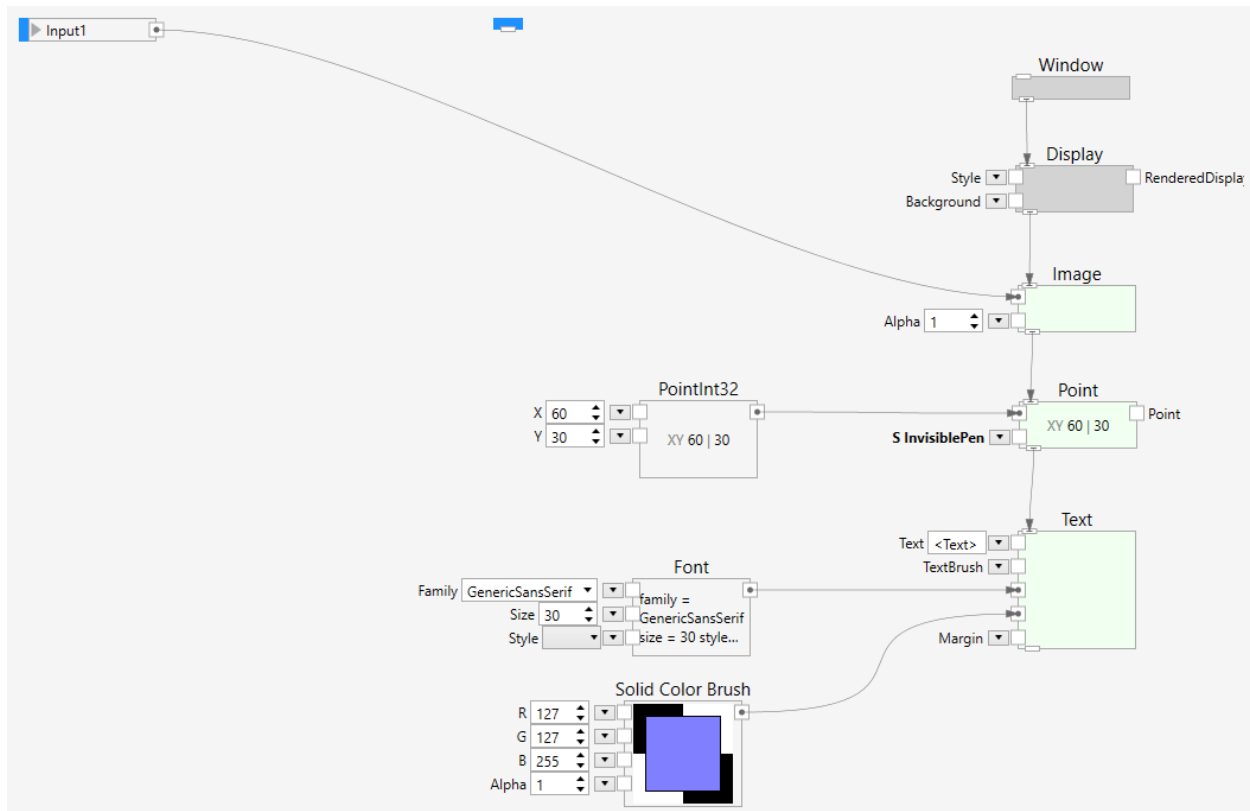


Margin (Type: Thickness)

The margin around the text.

Comments

The position of the text is at the origin of the parent widget. If the text is displayed on top of an image, it appears in the top-left position of the image. In order to move the text to some other position, you can use another widget to position it, such as a **Point** widget.



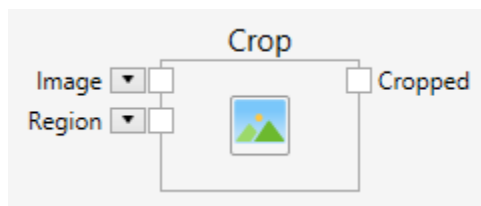
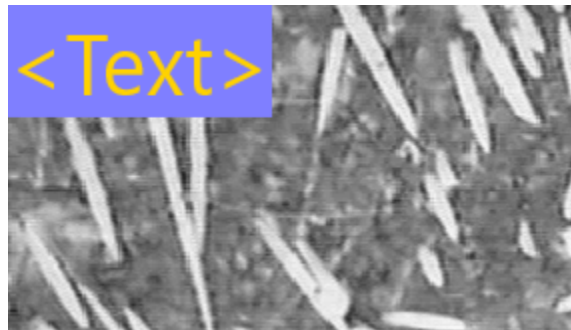
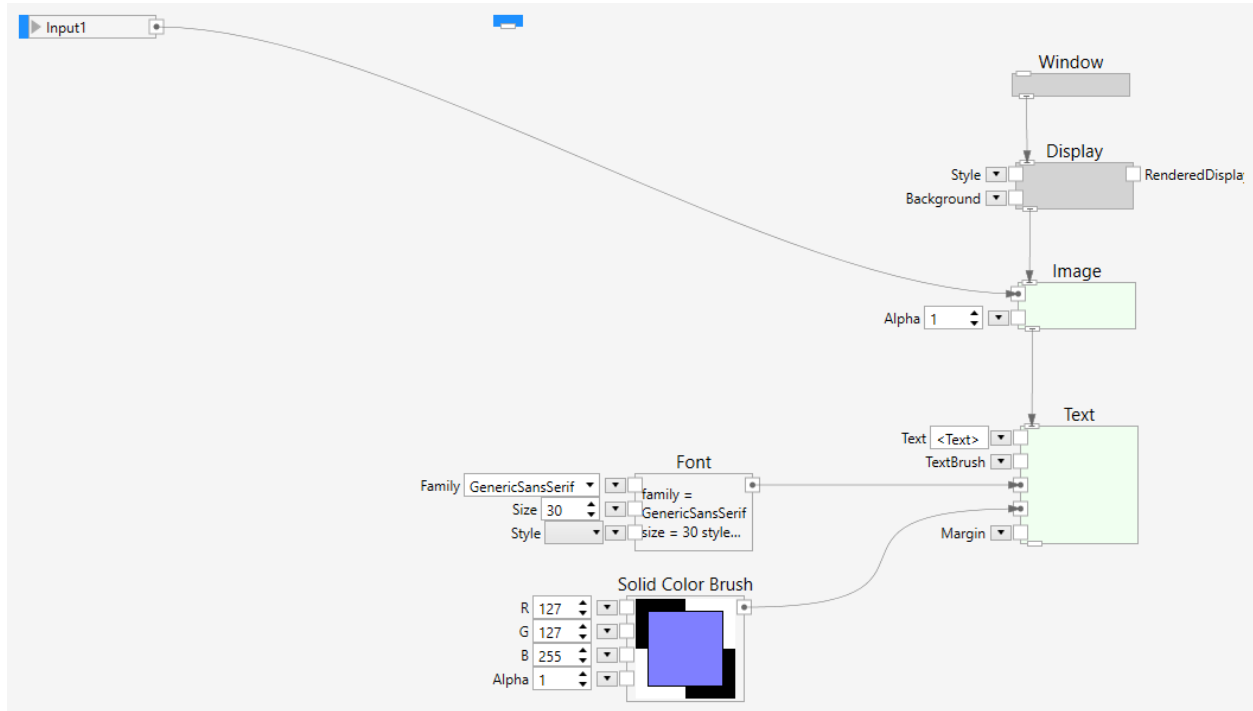
If you connect the *Point* input of the **Point** widget, the connected position is used. Otherwise, you can drag the point around interactively to specify the position.

Example

Here is an example that displays a text on top of an image. This pipeline shows the following result:

15.3.84 Crop

Crops a an arbitrary region out of an image.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

The region.

Outputs

Cropped (Type: Image)

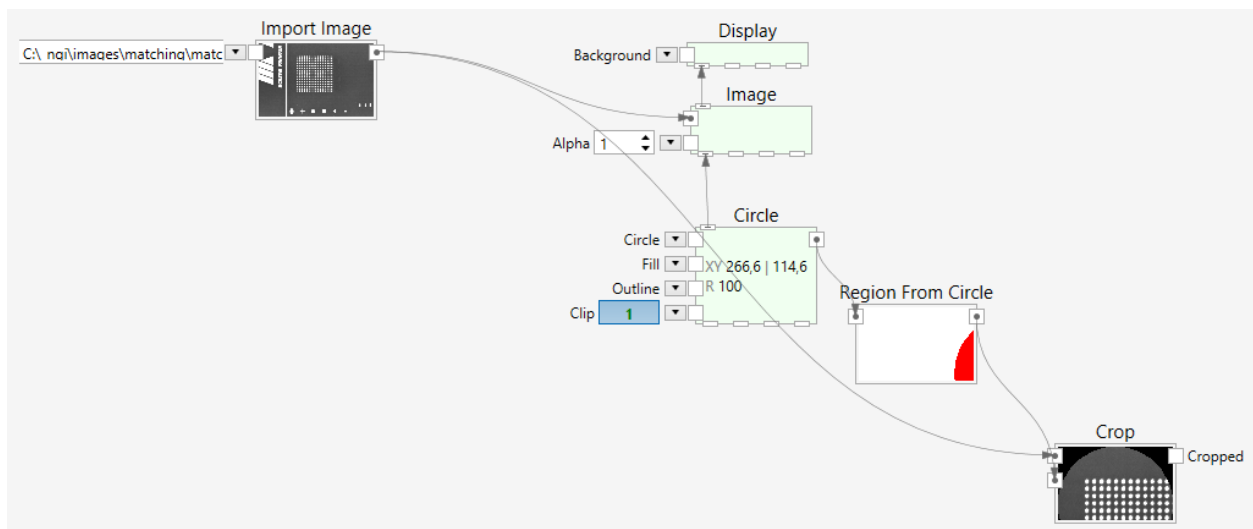
The portion of the original image that is specified by the crop region.

Comments

This function cuts out an arbitrary portion of an image. Often this is used to constrain a subsequent processing operation to a region of interest.

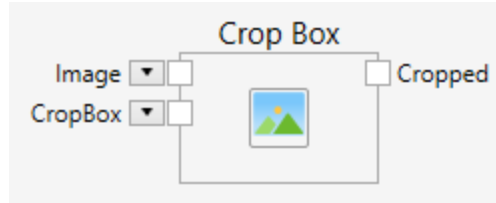
Sample

Here is an example, where the region is specified interactively with a circle widget:



15.3.85 Crop Box

Crops a rectangular portion out of an image.



Inputs

Image (Type: Image)

The input image.

CropBox (Type: Box)

The box geometry.

Outputs

Cropped (Type: Image)

The rectangular portion of the original image that is specified by the crop box.

Comments

This function cuts out a rectangular portion of an image. Often this is used to constrain a subsequent processing operation to an area of interest.

Sample

Here is an example, where the box is specified interactively with a box widget:

15.3.86 C# Expression

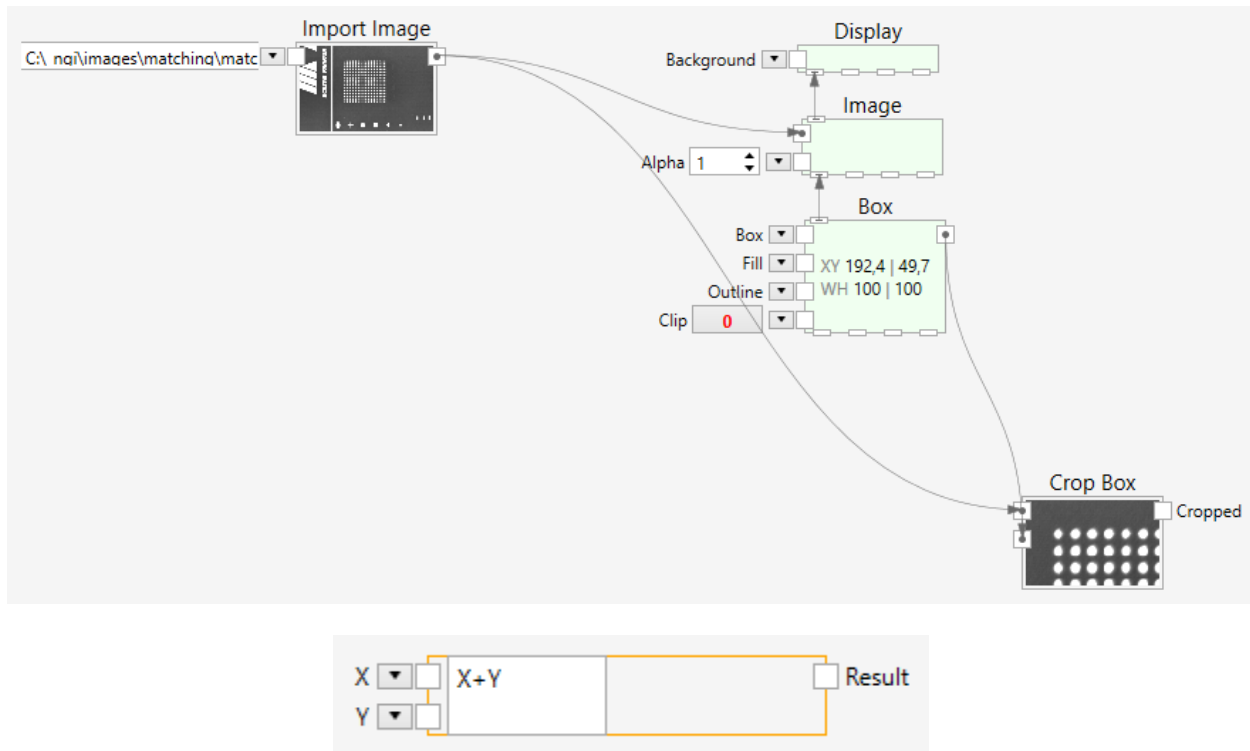
This node interprets simple C# statements. Statements are written using a subset of C# language specifications. The expression can be edited as long as no connections have been made to the inputs and outputs.

Inputs

The number and names of the inputs are determined when the expression has been specified. They are determined by parsing the expression.

Outputs

The output type is determined when the expression has been specified. It is determined by parsing the expression.



Comments

The expression parser is case sensitive.

Statements can be written using a subset of the C# syntax. Here you can find a list of the supported expressions:

Operators

Category

Operators

Primary

x.y f(x) a[x]

Unary

- - ! (T)x

Multiplicative

* / %

Additive

- -

Relational and type testing

< > <= >= is as

Equality

== !=

Conditional AND

&&

Conditional OR

||

Conditional

?:

The assignment operator is not available.

Operators precedence is respected following [C# rules \(Operator precedence and associativity\)](#).

Types

From an expression alone, the types cannot be deduced reliably. For example, the following expression

```
x + y
```

could add two numbers together, or concatenate two strings. Thus, additional syntax allows you to specify the type of the parameters of an expression. The expression

```
((int))x + ((int))y
```

ties the two parameters to the `int` type. If a type specification is missing, the parameter will be of type `double` by default.

The C# built in types can be used in their short or expanded form:

Short

Expanded

`bool`

`System.Boolean`

`byte`

`System.Byte`

`sbyte`

`System.SByte`

`char`

`System.Char`

`decimal`

`System.Decimal`

`double`

`System.Double`

`float`

`System.Single`

`int`

System.Int32
uint
System.UInt32
long
System.Int64
ulong
System.UInt64
object
System.Object
short
System.Int16
ushort
System.UInt16
string
System.String

In addition, any types available inside nVision can be used, such as in the following expression:

```
((Ngi.Image)) img.Width
```

Literals

Category

Operators

Constants

true false null

Numeric

f m

String/char

“” ‘’

The following character escape sequences are supported inside string or char literals:

- \ ' - single quote, needed for character literals
- \ " - double quote, needed for string literals
- \\ - backslash
- \0 - Unicode character 0
- \a - Alert (character 7)
- \b - Backspace (character 8)
- \f - Form feed (character 12)

- `\n` - New line (character 10)
- `\r` - Carriage return (character 13)
- `\t` - Horizontal tab (character 9)
- `\v` - Vertical quote (character 11)

License

The C# Expression node uses the **Dynamic Expresso** library from Davide Icardi.

[MIT License]

Copyright (c) 2015 Davide Icardi

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
- THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

15.3.87 Cut

You can use `Ctrl-X` or the **Cut (Ctrl-X)** command from the pipeline menu to cut the selected nodes and their inside connections to the clipboard.

You can select nodes by clicking them with the left mouse button. Hold down the `Ctrl` key if you want to add to (or subtract from) the selection.

You can also select a set of nodes inside a rectangle that you drag while holding down the `Ctrl` key.

Once you have cut a set of nodes to the clipboard, you can paste them somewhere else.

15.3.88 Direction

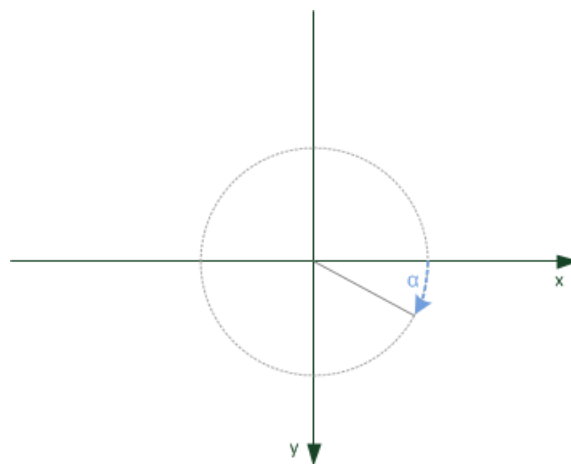
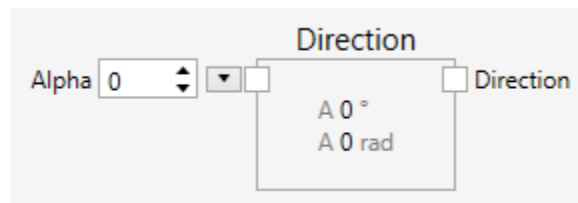
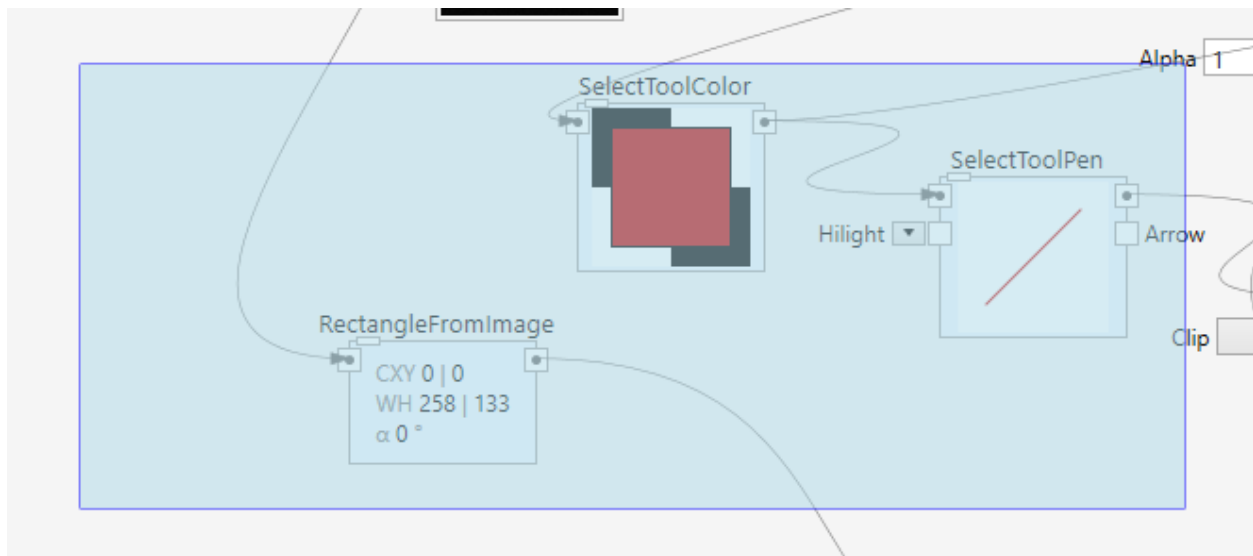
Creates a direction.

A direction is a geometric entity, representing a direction in the two-dimensional cartesian plane. A direction is similar to an angle, but is internally implemented as a direction vector (unit vector).

Inputs

Alpha (Type: Double)

The angle in radians.



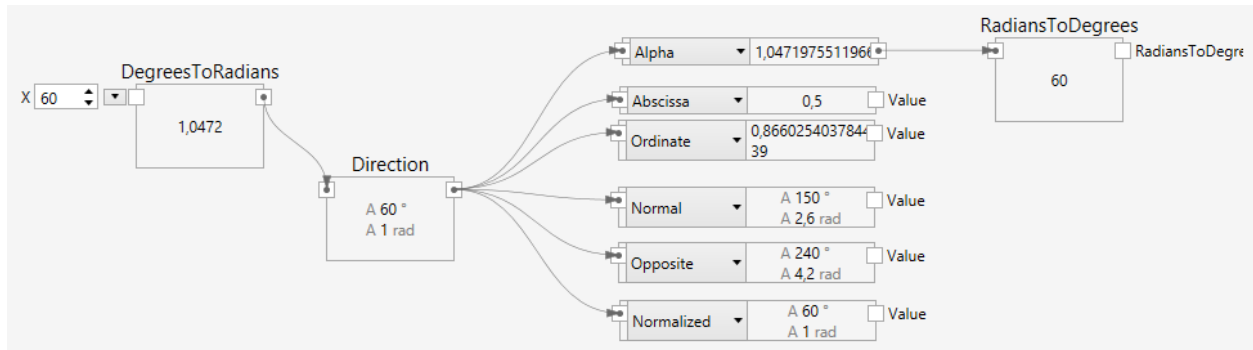
Outputs

Direction (Type: Direction)

The direction.

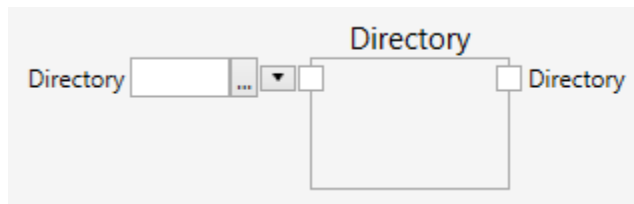
Sample

Here is an example:



15.3.89 Directory

Selects a directory.



Inputs

Directory (Type: String)

The directory (provided as a string, or selected via the ... button).

Outputs

Directory (Type: Ngi.Path)

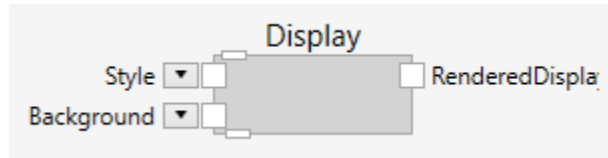
The directory as a path.

Comments

The purpose of this node is to make it easy for the user to select a directory from the filesystem.

15.3.90 Display

The **Display** node is the root of the hierarchical widget system, which is used to display images, regions, palettes, histograms, profiles and graphical items in a layered fashion.



The **Display** node creates the connection to the **nVision** workspace area.

If a **Display** node is available in the pipeline, it takes over the display. This means that the default of displaying the first pipeline output is overridden, and the display along with its children is displayed instead.

If a **Window** node is also available in the pipeline, it takes precedence over the **Display** node.

At the bottom of the **Display** node is a pin that allows it to connect children to the display. The children and grandchildren define the layering of the widgets and the graphical outcome.

Inputs

Style (Type: `Style`)

Optional styling of the **Display**.

The **Display** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Padding* styles.

Background (Type: `SolidColorBrushByte`)

The background of the **Display**.

Outputs

RenderedDisplay (Type: `Image`)

The rendering (with all the children) in the form of an image.

Example

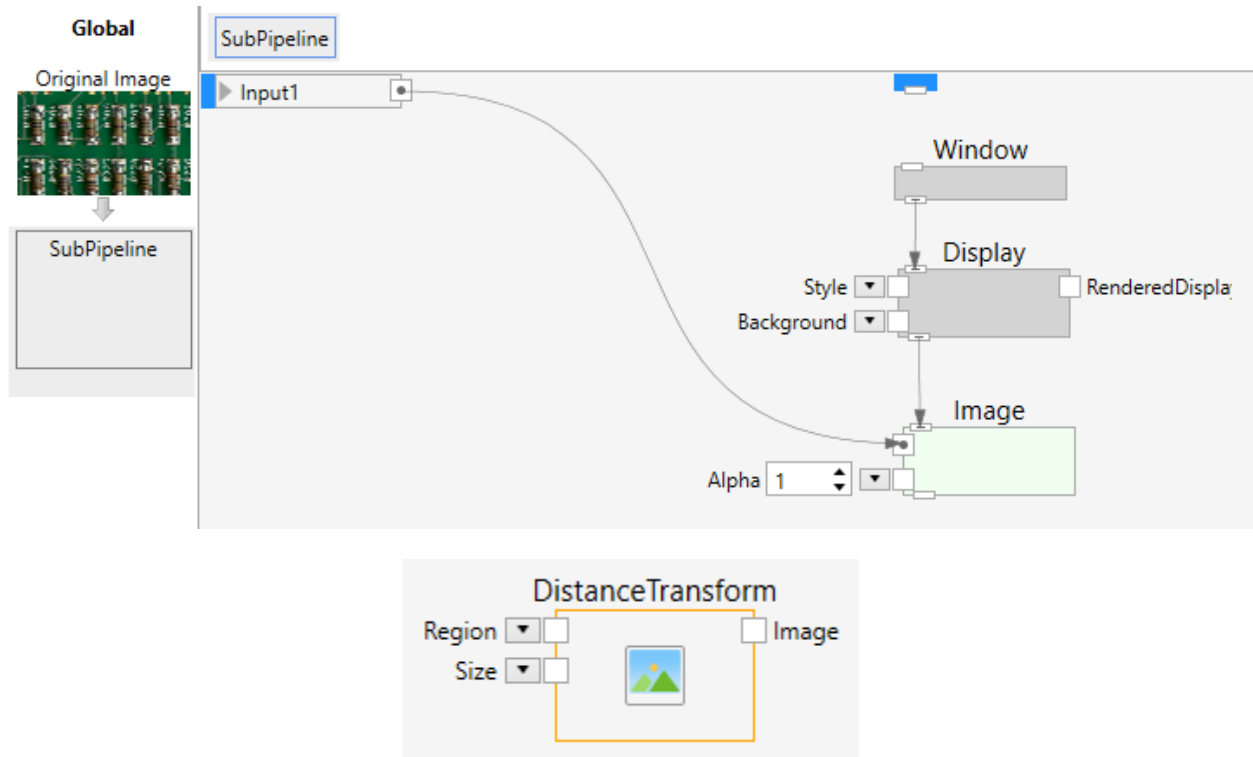
Here is an example that displays an image inside a display:

Below the **Window** node is a **Display** and below the **Display** is an **Image**.

Note: the **Window** node on top of the **Display** node is optional. The *Style* input is only respected, if the **Window** node has a **Display** node parent.

15.3.91 Distance Transform

Calculates the distance transform given a region.



Inputs

Region (Type: Region)

The input region.

Size (Type: Extent3d)

The size of the resulting image.

Outputs

Image (Type: Image)

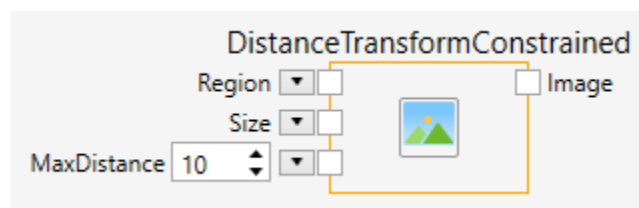
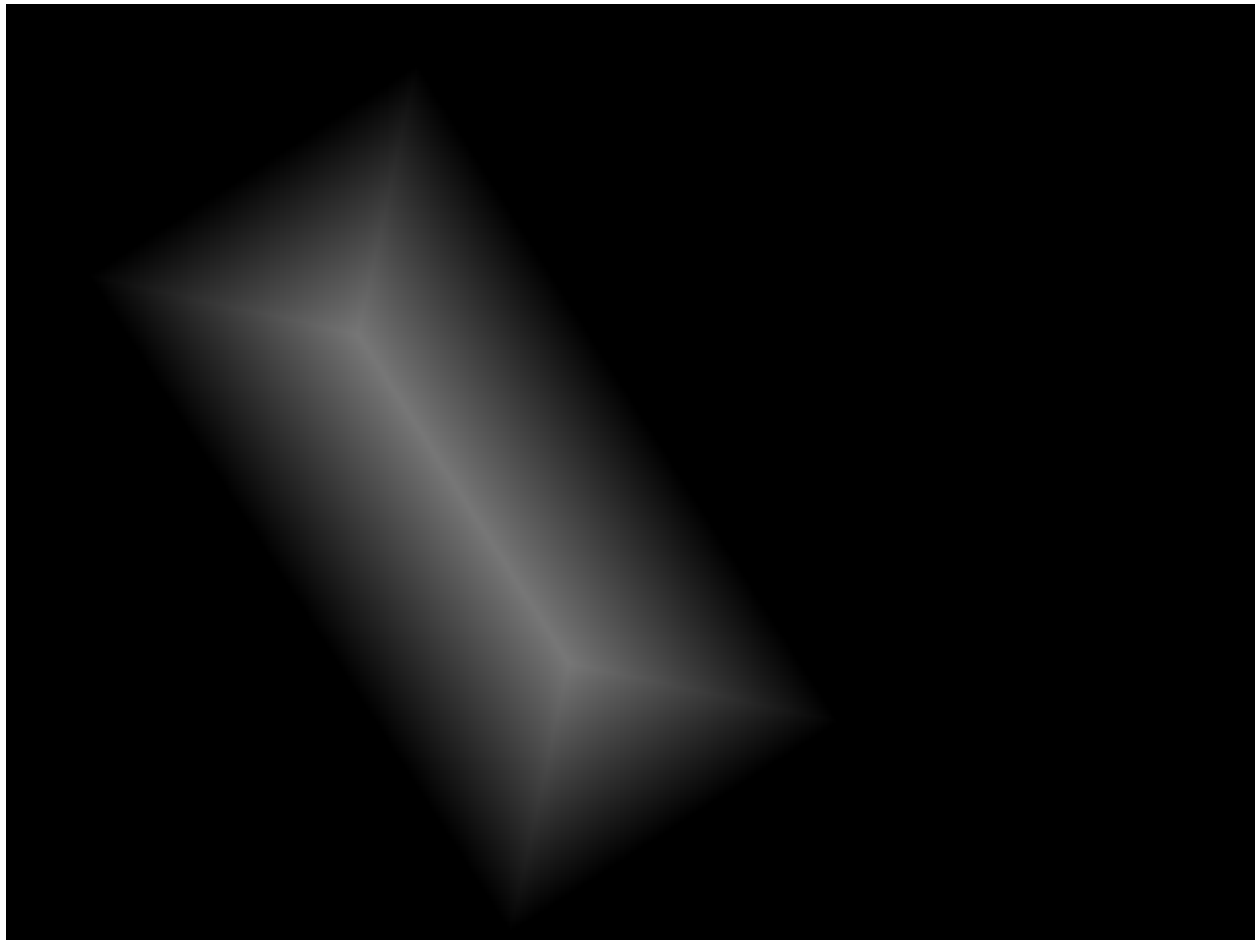
The resulting image.

Comments

Here is an example of a distance transform, given a rotated rectangular region on input:

15.3.92 Distance Transform Constrained

Calculates the constrained distance transform given a region.



Inputs

Region (Type: `Region`)

The input region.

Size (Type: `Extent3d`)

The size of the resulting image.

MaxDistance (Type: `Int32`)

The maximum distance.

Outputs

Image (Type: `Image`)

The resulting image.

Comments

Here is an example of a constrained distance transform, given a rotated rectangular region on input:

The resulting values of the transform are constrained to the maximum distance input.

15.3.93 HMI DockPanel

The **DockPanel** docks its child controls to one of the four sides *Left*, *Top*, *Right* or *Bottom*, where the respective child takes at much space as it needs.

At the bottom of the **DockPanel** node is a pin that allows it to connect several children.

Inputs

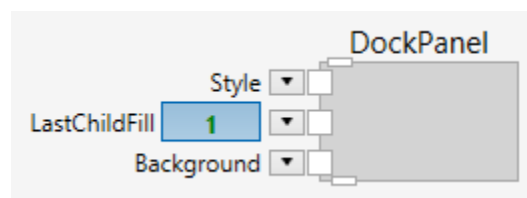
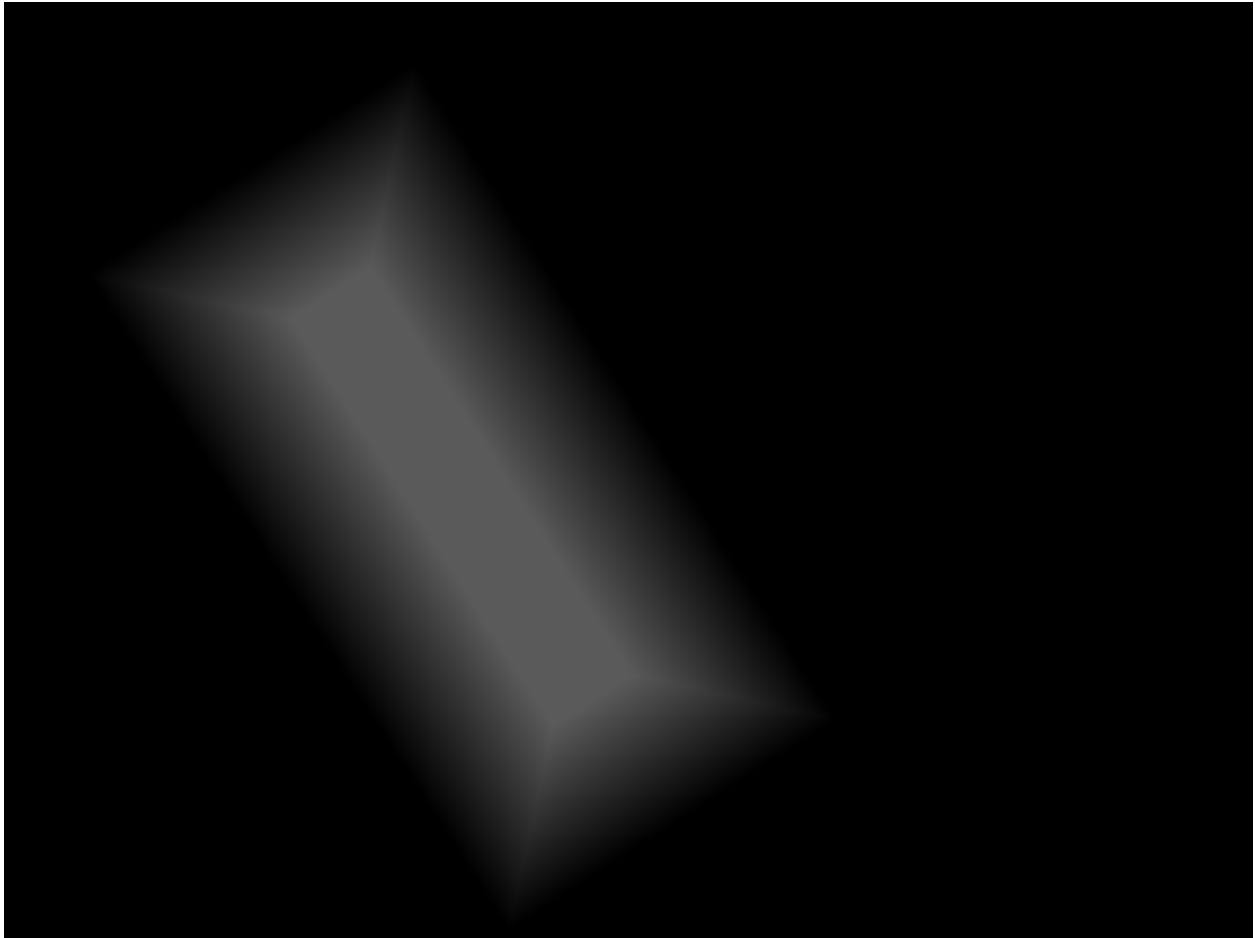
Style (Type: `Style`)

Optional styling of the **DockPanel**.

The **DockPanel** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Background* styles.

LastChildFill (Type: `boolean`)

If true, the last child added to the **DockPanel** takes all the available space. If false, the last child added takes only the space it needs.



Background (Type: SolidColorBrushByte)

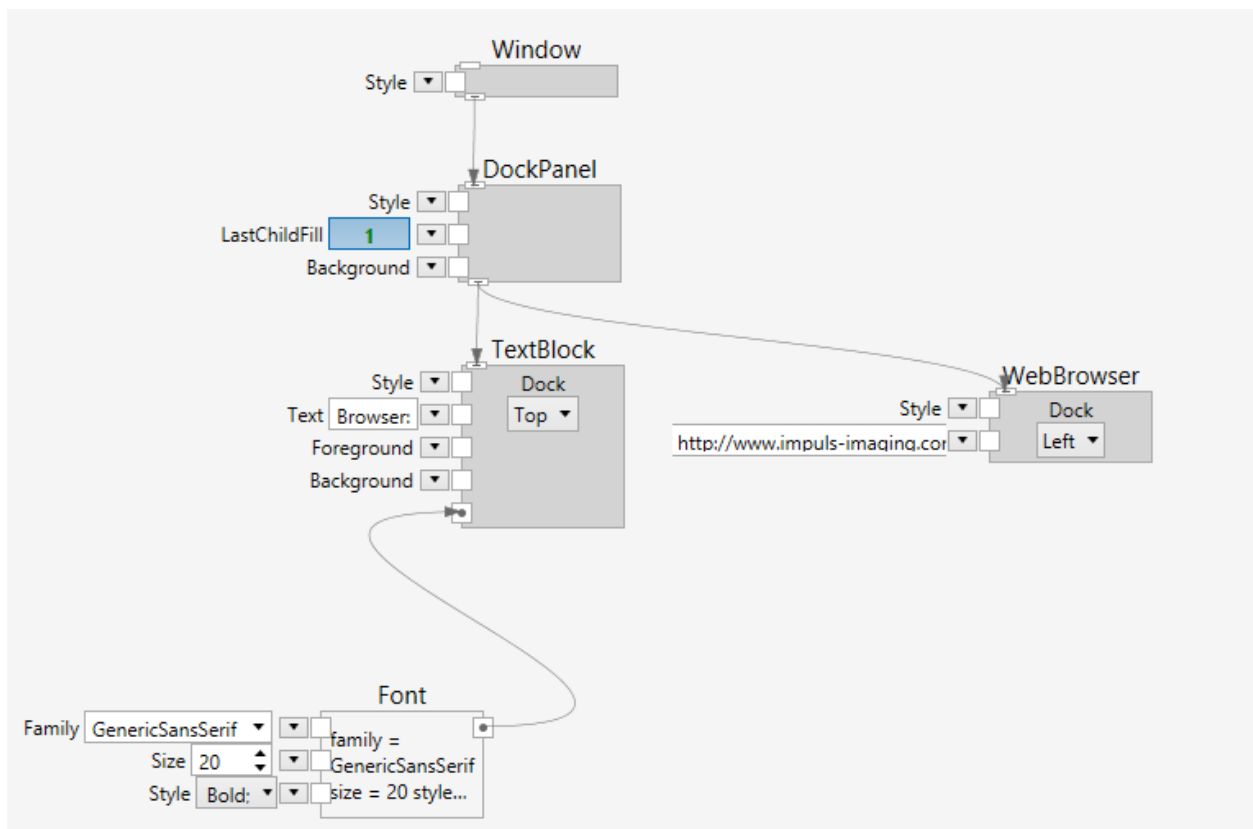
The background of the **DockPanel**.

Comments

Children of the **DockPanel** have the attached property **Dock**. This property can take the values *Left*, *Top*, *Right* and *Bottom* and specifies to which side the child will be docked.

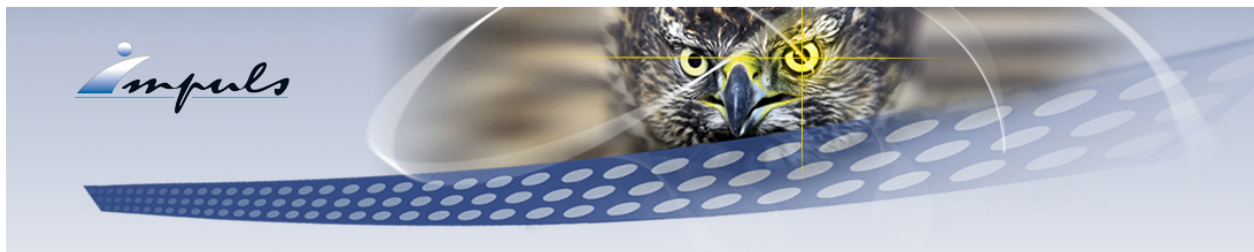
Example

Here is an example that docks a text label to the top and a webbrowser to the left, where the webbrowser fills the available space fully. This definition



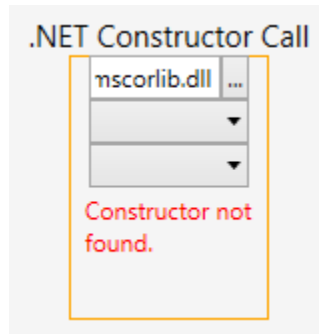
creates the following user interface:

Browser:



15.3.94 .NET Constructor Call

Call a constructor in a .Net assembly.



Inputs

The number and types of the inputs are determined when the assembly file name, the type name and the constructor name have been specified. They are determined by reflection and are the parameters of the constructor.

Outputs

The output type is determined when the assembly file name, the type name and the constructor name have been specified. It is determined by reflection and is of the selected type.

Comments

The parameters that specify which constructor to call are only editable when the ports of the node are not connected.

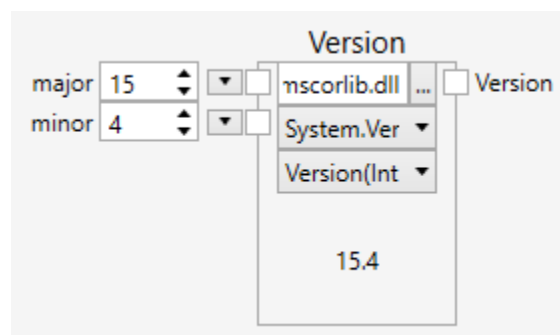
The first parameter is the filename of the assembly that contains the static method. You can use the browse button to browse to the desired assembly. An example of an assembly that you can try is: C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5.1\mscorlib.dll.

The second parameter is the class that contains the constructor, prefixed by its namespace, i.e. System.Version.

The third parameter is the constructor to call, i.e. GetEnvironmentVariable.

Once all parameters have been specified, and it is clear which constructor to call, the title of the node will adapt and show a more suitable name.

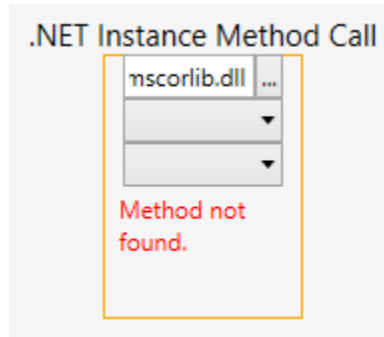
Here is an example that calls the constructor Version(Int32 major, Int32 minor) from the System.Version class in the mscorlib.dll assembly.



The example reads the value of the `windir` environment value and returns the string `C:\Windows`.

15.3.95 .NET Instance Method Call

Call an instance method in a .Net assembly.



Inputs

The number and types of the inputs are determined when the assembly file name, the type name and the method name have been specified. They are determined by reflection and are the parameters of the method..

The first input is the instance of the object, the other inputs are the parameters of the method.

Outputs

The output type is determined when the assembly file name, the type name and the method name have been specified. It is determined by reflection and is the return value of the method (if not void).

Comments

The parameters that specify which method to call are only editable when the ports of the node are not connected.

The first parameter is the filename of the assembly that contains the static method. You can use the browse button to browse to the desired assembly. An example of an assembly that you can try is: `C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5.1\mscorlib.dll`.

The second parameter is the class that contains the method, prefixed by its namespace, i.e. `System.String`.

Once all parameters have been specified, and it is clear which method to call, the title of the node will adapt and show a more suitable name.

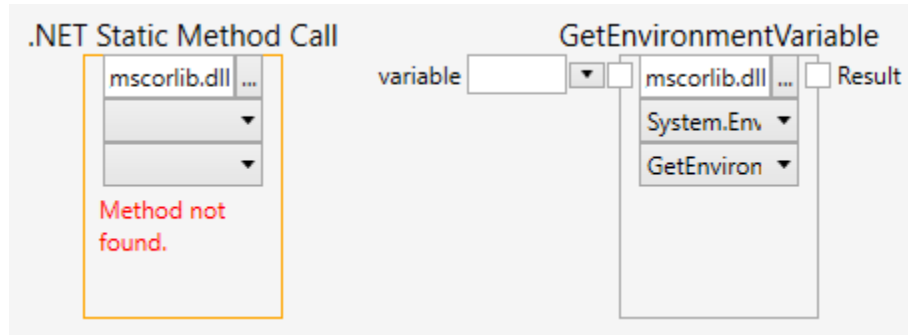
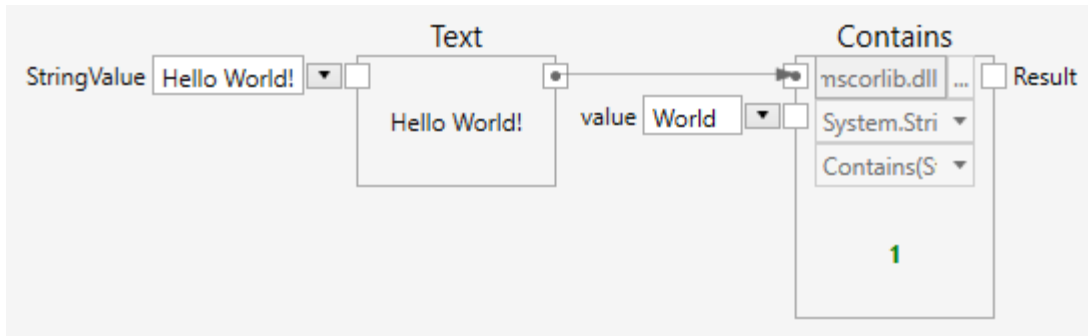
The third parameter is the instance method to call, i.e. `Contains(String value)`.

Here is an example that calls the method `Contains(String value)` from the `System.String` class in the `mscorlib.dll` assembly.

The example checks if the string `World` is contained in the string `Hello World!`.

15.3.96 .NET Static Method Call

Call a static method in a .Net assembly.



Inputs

The number and types of the inputs are determined when the assembly file name, the type name and the method name have been specified. They are determined by reflection and are the parameters of the method.

Outputs

The output type is determined when the assembly file name, the type name and the method name have been specified. It is determined by reflection and is the return value of the method (if not void).

Comments

The parameters that specify which method to call are only editable when the ports of the node are not connected.

The first parameter is the filename of the assembly that contains the static method. You can use the browse button to browse to the desired assembly. An example of an assembly that you can try is: `C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETCore\v4.5.1\mscorlib.dll`.

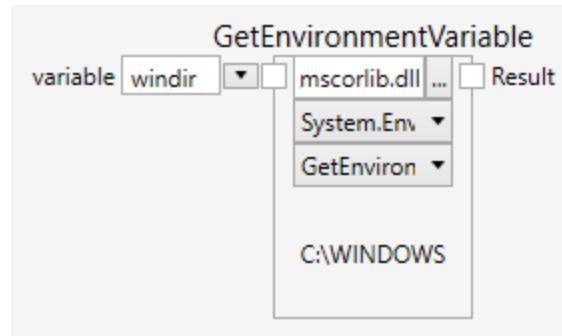
The second parameter is the class that contains the method, prefixed by its namespace, i.e. `System.Environment`.

The third parameter is the static method to call, i.e. `GetEnvironmentVariable(String variable)`.

Once all parameters have been specified, and it is clear which method to call, the title of the node will adapt and show a more suitable name.

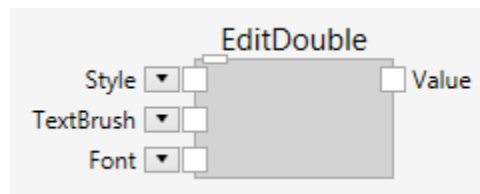
Here is an example that calls the method `GetEnvironmentVariable(String variable)` from the `System.Environment` class in the `mscorlib.dll` assembly.

The example reads the value of the `windir` environment value and returns the string `C:\Windows`.



15.3.97 EditDouble

The **EditDouble** is a HMI control that can be used to enter floating point numbers.



Inputs

Style (Type: Style)

Optional styling of the **EditDouble**.

The **EditDouble** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

TextBrush (Type: SolidColorBrush)

The text color.

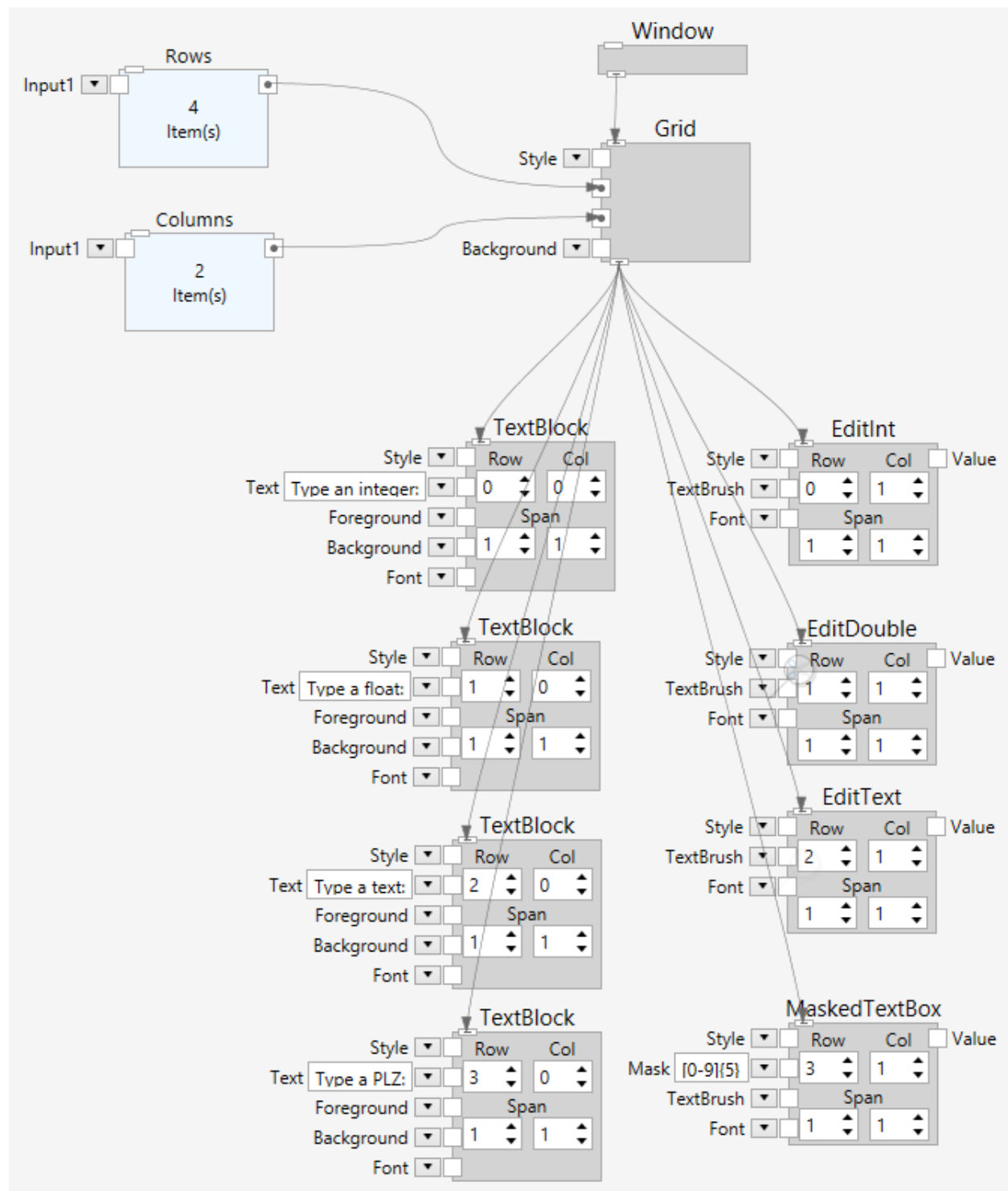
Font (Type: Font)

The font.

Outputs

Value (Type: Double)

The number entered by the user.



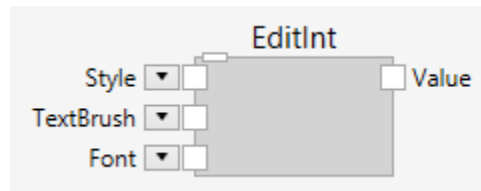
Example

Here is an example that shows the various edit controls. This definition:
creates the following user interface:

Type an integer:	2
Type a float:	0,5
Type a text:	Hello
Type a PLZ:	86842

15.3.98 EditInt

The **EditInt** is a HMI control that can be used to enter integer numbers.



Inputs

Style (Type: `Style`)

Optional styling of the **EditInt**.

The **EditInt** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

TextBrush (Type: `SolidColorBrush`)

The text color.

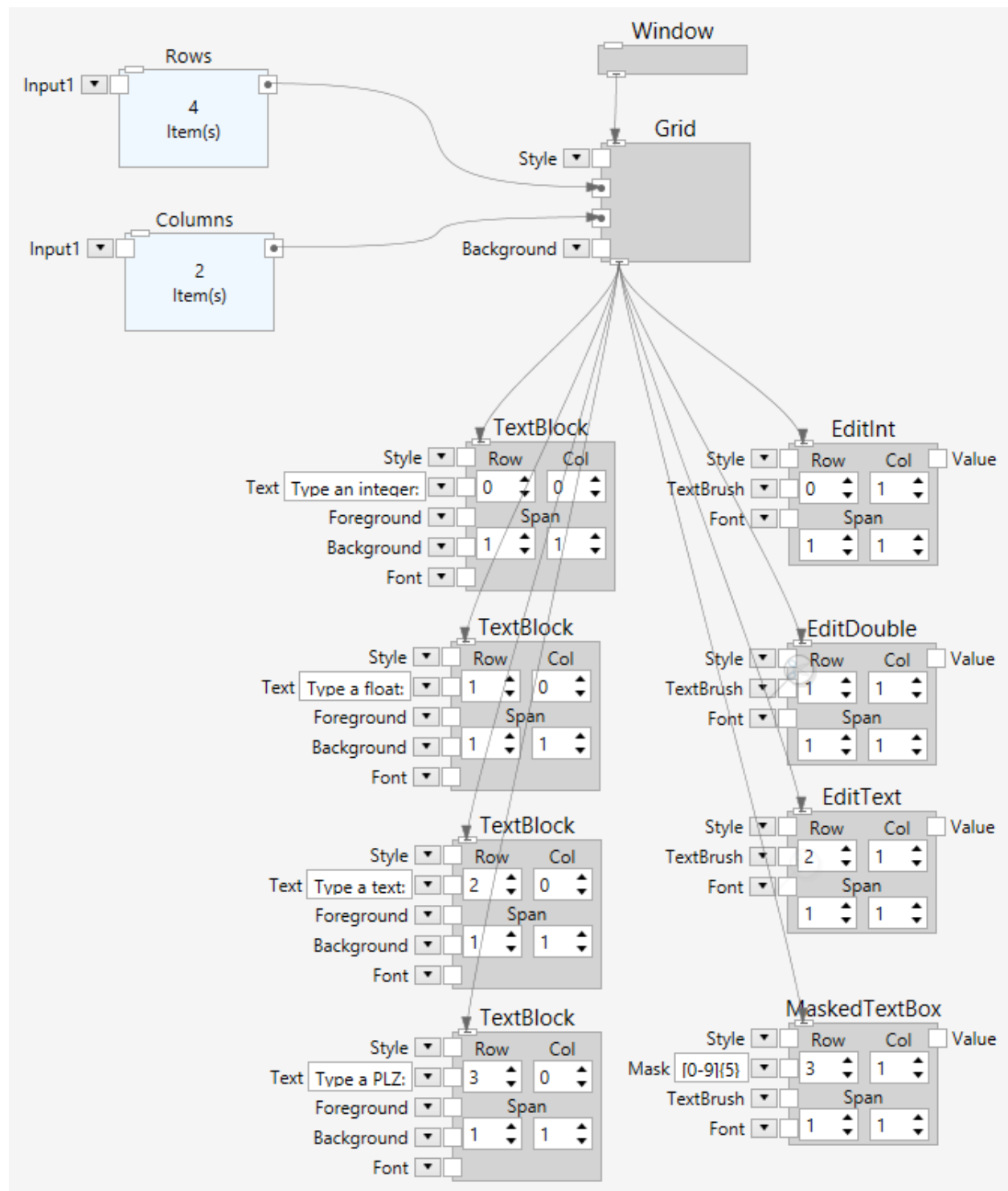
Font (Type: `Font`)

The font.

Outputs

Value (Type: `Int32`)

The number entered by the user.



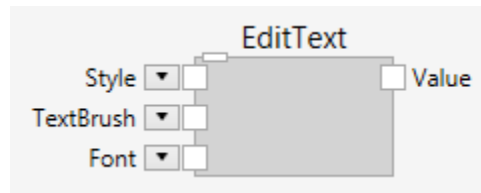
Example

Here is an example that shows the various edit controls. This definition:
creates the following user interface:

Type an integer:	2
Type a float:	0,5
Type a text:	Hello
Type a PLZ:	86842

15.3.99 EditText

The **EditText** is a HMI control that can be used to enter texts.



Inputs

Style (Type: `Style`)

Optional styling of the **EditText**.

The **EditText** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

TextBrush (Type: `SolidColorBrush`)

The text color.

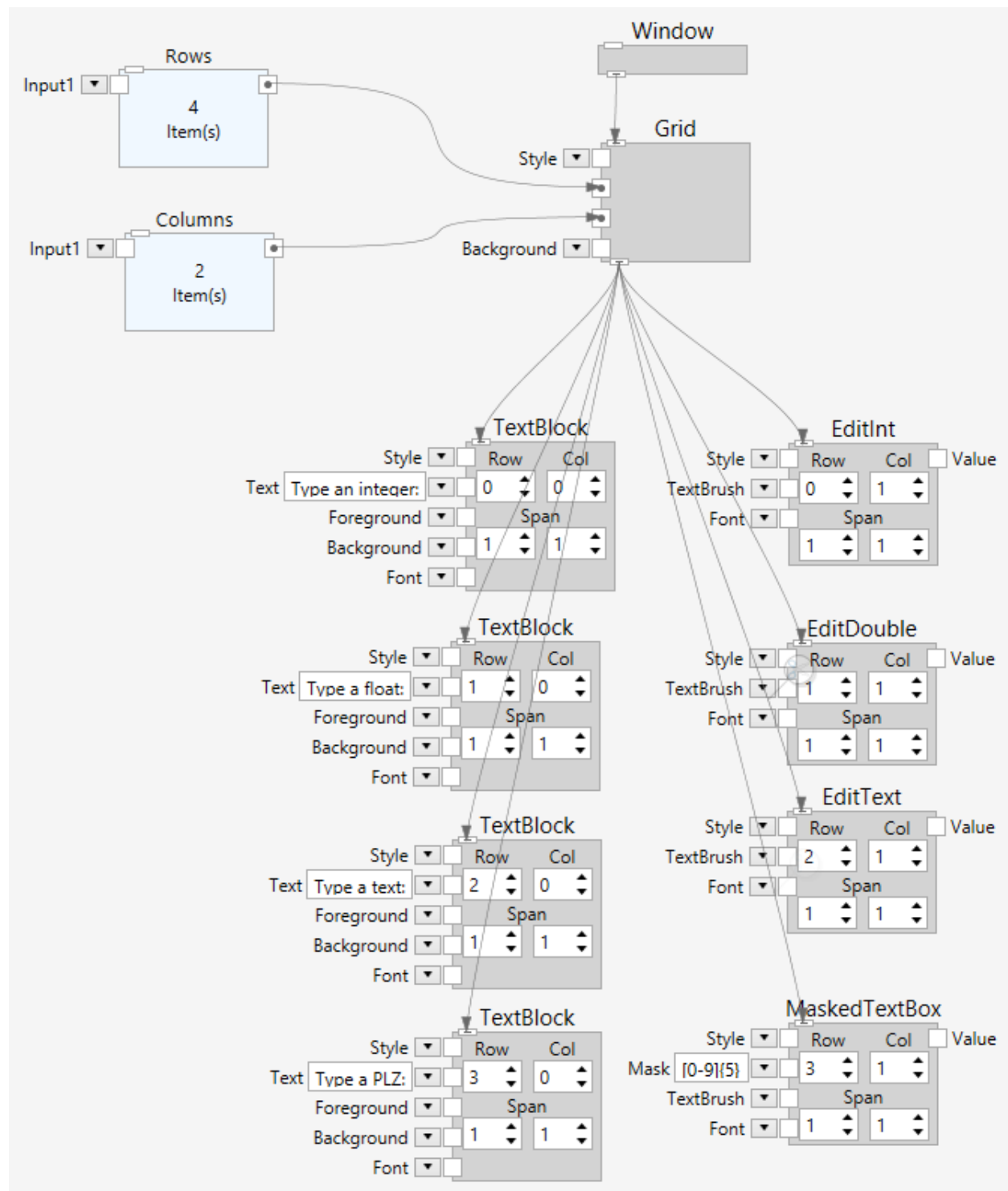
Font (Type: `Font`)

The font.

Outputs

Value (Type: `string`)

The text entered by the user.



Example

Here is an example that shows the various edit controls. This definition:
creates the following user interface:

Type an integer:	2	▲▼
Type a float:	0,5	▲▼
Type a text:	Hello	
Type a PLZ:	86842	

15.3.100 Environment

Provides information about the current environment and platform.



Outputs

Environment (Type: Environment)

The environment object. Use the environment object properties to get at data in the environment.

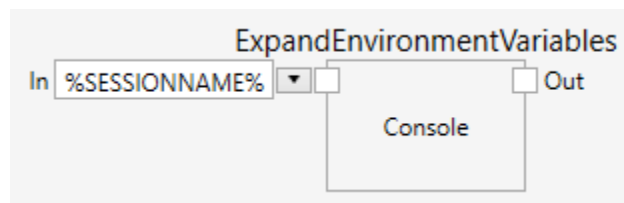
Comments

This node provides an environment object that has some properties which provide access to environmental information. The table shows the properties that can be accessed on an environment object.

property	description
Command-Line	Gets the command line for this process.
CurrentDirectory	Gets the fully qualified path of the current working directory.
EnvironmentVariableNames	Retrieves all environment variable names from the current process.
Is64BitOperatingSystem	Determines whether the current operating system is a 64-bit operating system.
Is64BitProcess	Determines whether the current process is a 64-bit process.
LogicalDriveNames	Returns a list of strings containing the names of the logical drives on the current computer, where each element contains the name of a logical drive. For example, if the computer's hard drive is the first logical drive, the first element returned is "C:".
MachineName	Gets the NetBIOS name of this local computer.
OSVersion	Gets an OperatingSystem object that contains the current platform identifier and version number.
ProcessorCount	Gets the number of processors on the current machine.
SystemDirectory	Gets the fully qualified path of the system directory.
UserDomainName	Gets the network domain name associated with the current user.
UserInteractive	Gets a value indicating whether the current process is running in user interactive mode.
UserName	Gets the user name of the person who is currently logged on to the Windows operating system.

15.3.101 ExpandEnvironmentVariables

Replaces the name of each environment variable embedded in the specified string with the string equivalent of the value of the variable, then returns the resulting string.



Inputs

In (Type: string)

A string containing the names of zero or more environment variables. Each environment variable is quoted with the percent sign character (%).

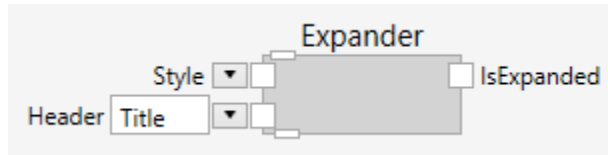
Outputs

Out (Type: `string`)

A string with each environment variable replaced by its value.

15.3.102 HMI Expander

The **Expander** puts an expandable and collapsible area around its child.



At the bottom of the **Expander** node is a pin that allows it to connect one child.

Inputs

Style (Type: `Style`)

Optional styling of the **Expander**.

The **Expander** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

Header (Type: `string`)

The title of the **Expander**.

Example

Here is an example that puts an expander around a profile. This definition:

creates the following user interface:

The profile display can be hidden or shown by clicking the little arrow button at the top-right corner of the expander.

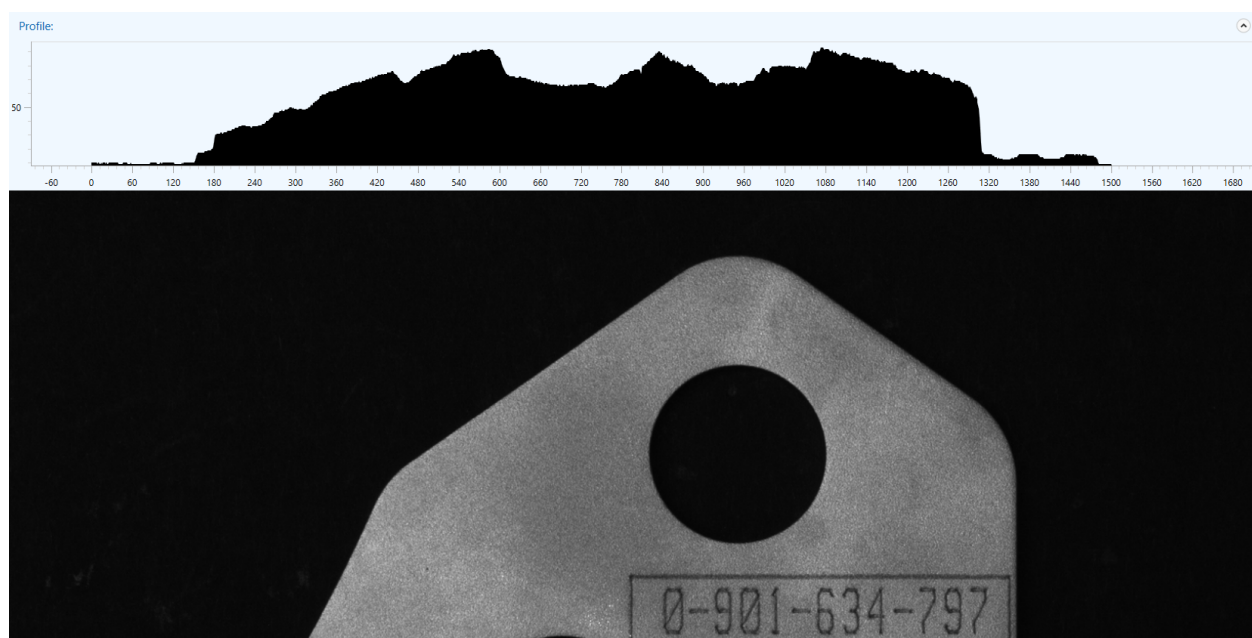
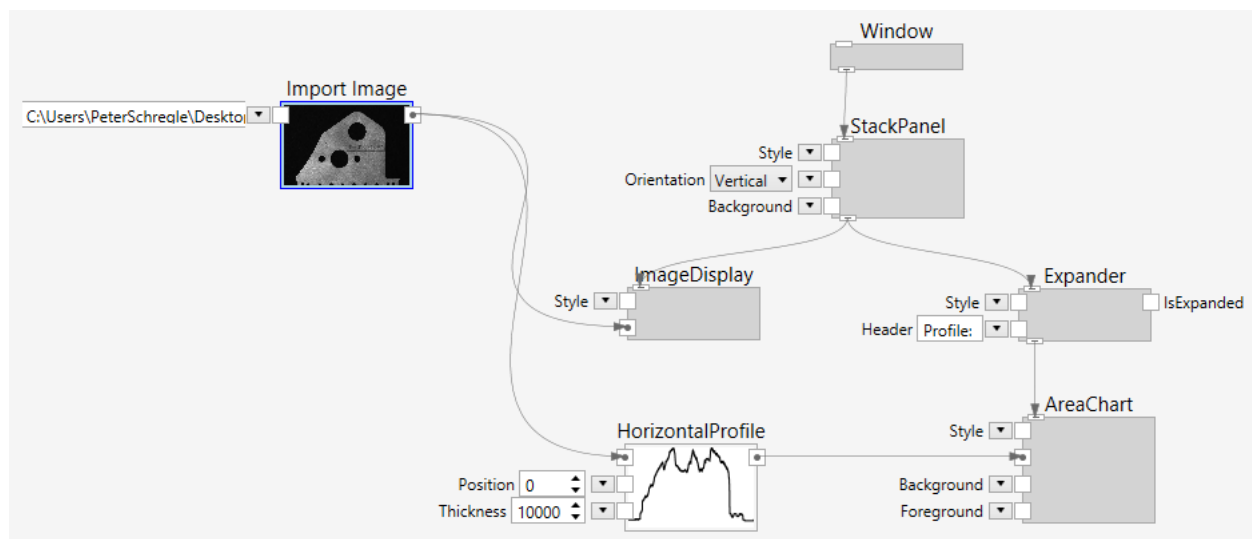
15.3.103 ExportDataTableToCsv

Exports the results of a blob analysis to a file with comma-separated values.

Inputs

Data (Type: `'System.Data.DataTable'`)

The blob analysis data.



The screenshot shows the **ExportDataTableToCsv** dialog box. It contains the following fields and controls:

- Data**: A dropdown menu with a checkmark icon.
- Directory**: A text input field with a browse button (three dots).
- Filename**: A text input field.
- ValueSeparator**: A dropdown menu with a semicolon (;) selected.
- QuoteCharacter**: A dropdown menu with a double quote (") selected.
- Culture**: A dropdown menu.

An orange line connects the **Data** dropdown to the **ValueSeparator**, **QuoteCharacter**, and **Culture** dropdowns.

Directory (Type: String)

A directory in the filesystem. This is the directory, where the saved files will be placed. If the directory does not exist, an error message will be shown.

Filename (Type: String)

A filename. You can either enter a filename (with extension) or leave this empty. If you leave it empty, the system will create a name in the form of “data_00000.csv”, where the number will be incremented automatically. If you enter a filename, the file will be overwritten in every execution of the pipeline node.

ValueSeparator (Type: String)

This separates single values in one line of the output file. This must be a single character.


QuoteCharacter (Type: String)

This character is used to quote values. This must be a single character.

Culture (Type: String)

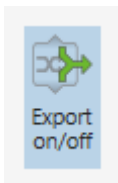
Cultural information used to format values.

Comments

The **ExportDataTableToCsv** node saves an csv file to the filesystem. The directory where the file will be saved can either be typed, or selected with a Sewarch Folder dialog - by clicking on the  button. If a file could not be saved for some reason, an appropriate error message is shown.

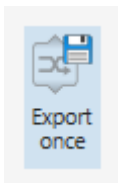
The **ExportDataTableToCsv** node saves data only in two cases:

1. The export mode is on (the **Export on/off** button is pressed).



If the **Export on/off** button is pressed, any time the pipeline runs the file will be saved.

2. The **Export once** button will be clicked.



If the **Export once** button is clicked, the file will be saved.

15.3.104 Export Image

Exports an image to a file.



Inputs

Image (Type: Image)

An image.


Directory (Type: String)

A directory in the filesystem. This is the directory, where the saved files will be placed. If the directory does not exist, an error message will be shown.

Filename (Type: String)

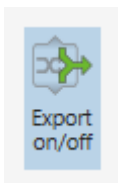
A filename. You can either enter a filename (with extension) or leave this empty. If you leave it empty, the system will create a name in the form of “image_00000.tif”, where the number will be incremented automatically. If you enter a filename, the file will be overwritten in every execution of the pipeline node.

Comments

The **ExportImage** node saves an image to the filesystem. The directory where the file will be saved can either be typed, or selected with a Search Folder dialog - by clicking on the  button. The node supports a variety of image file formats, such as TIF, PNG, JPG and BMP to name a few. If a file could not be saved for some reason, an appropriate error message is shown.

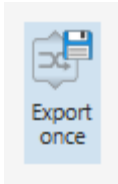
The **ExportImage** node saves images only in two cases:

1. The export mode is on (the **Export on/off** button is pressed).



If the **Export on/off** button is pressed, any time the pipeline runs the file will be saved.

2. The **Export once** button will be clicked.



If the **Export once** button is clicked, the file will be saved.

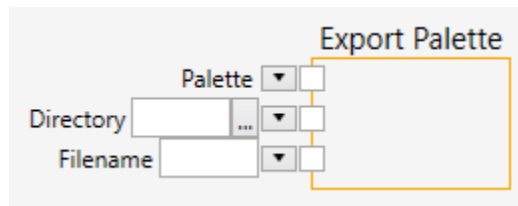
Sample

Here is an example that shows how to use the **ExportImage** node.



15.3.105 ExportPalette

Exports a palette to a file.



Inputs

Palette (Type: Palette)

A palette.


Directory (Type: String)

A directory in the filesystem. This is the directory, where the saved files will be placed. If the directory does not exist, an error message will be shown.

Filename (Type: String)

A filename. The file will be overwritten in every execution of the pipeline node.

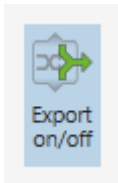
Comments

The **ExportImage** node saves a palette to the filesystem. The directory where the file will be saved can either be typed, or selected with a Sewarch Folder dialog - by clicking on the  button.

The supported format is a binary file consisting of either 256 bytes (for a monochrome palette) or 768 bytes (for a color palette).

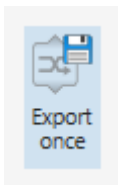
The **ExportImage** node saves palettes only in two cases:

1. The export mode is on (the **Export on/off** button is pressed).



If the **Export on/off** button is pressed, any time the pipeline runs the file will be saved.

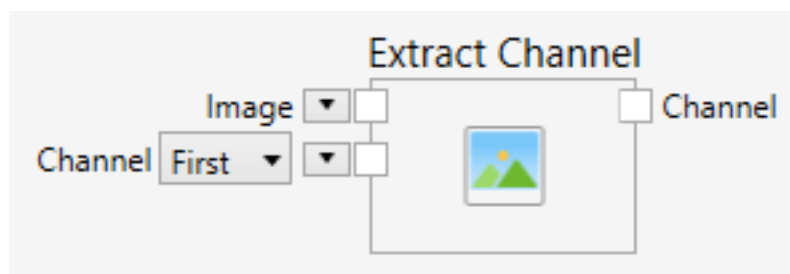
2. The **Export once** button will be clicked.



If the **Export once** button is clicked, the file will be saved.

15.3.106 Extract Channel

Extracts a specific color channel from a multi-channel image.



Inputs

Image (Type: Image)

The input image.

Channel (Type: String)

The channel that should be extracted.

Outputs

Channel (Type: Image)

The output image.

Comments

This function extracts a specific channel from a multi-channel color image.

The channel can be specified by meaning or by position.

Positional arguments are: `First`, `Second` and `Third`. The other arguments are `Red`, `Green`, `Blue`, `Hue`, `Lightness`, `Saturation`, `Intensity`, `L`, `A` and `B`. If a channel is not valid (such as when trying to extract the red channel from an HSI image), an error is shown.

Here is an example of the original color image:

And here are the three color channels red, green and blue from left to right:

The same color channels again with proper coloring:

Sample

Here is an example that extracts the three color channels from an RGB image.

15.3.107 Filename

Selects a file.

Inputs

Filename (Type: String)

The filename (provided as a string, or selected via the ... button).

Outputs

Filename (Type: Ngi.Path)

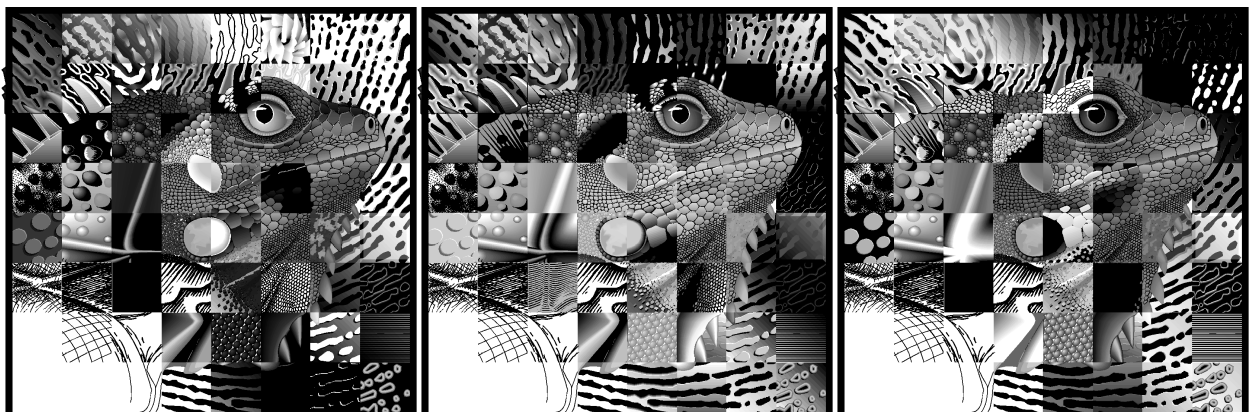
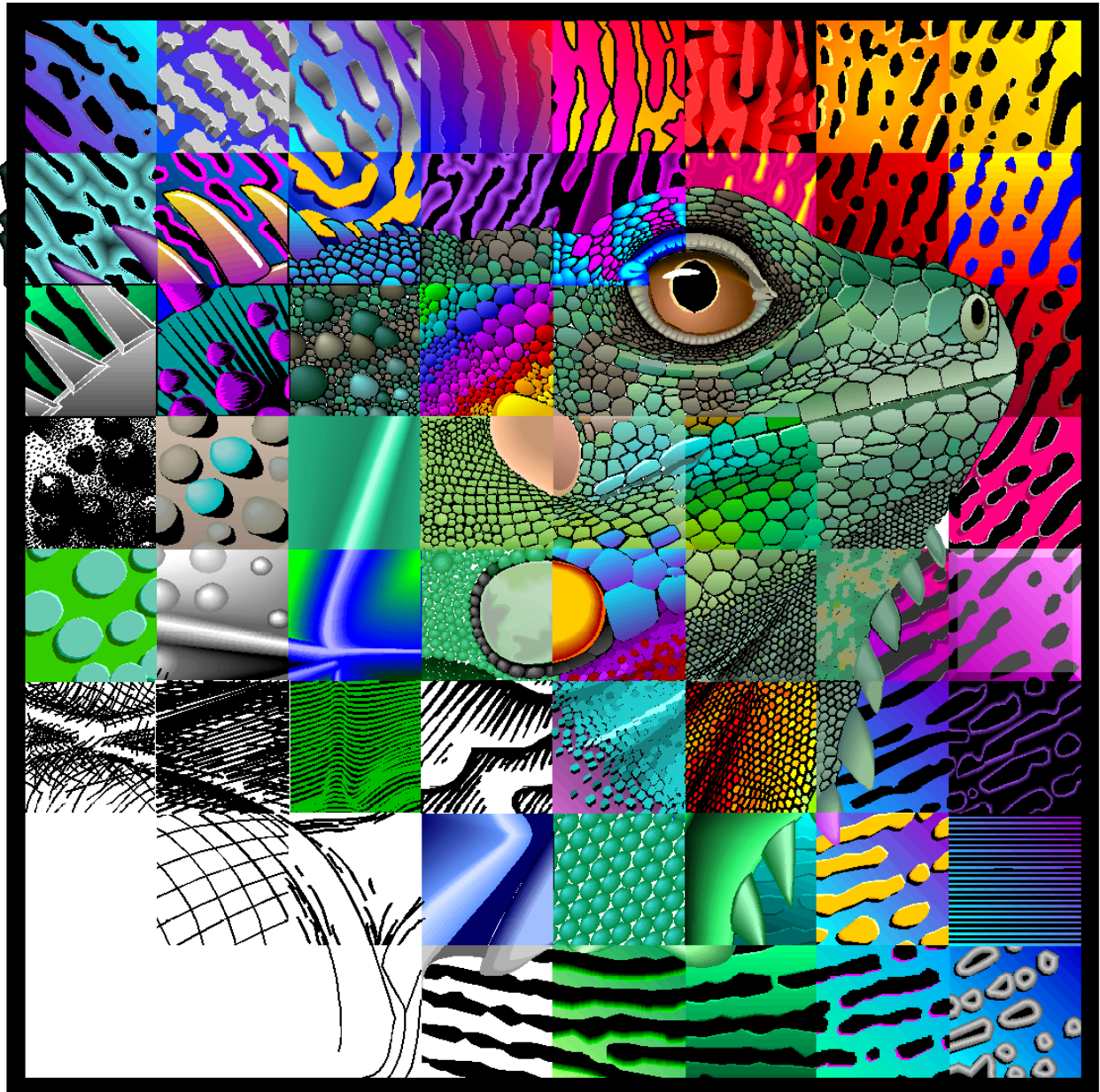
The filename as a path.

Comments

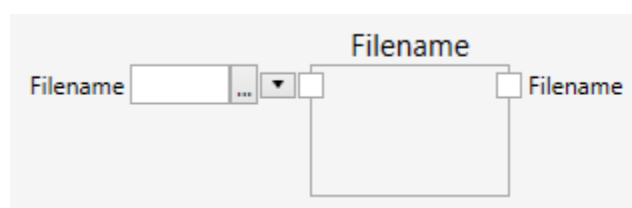
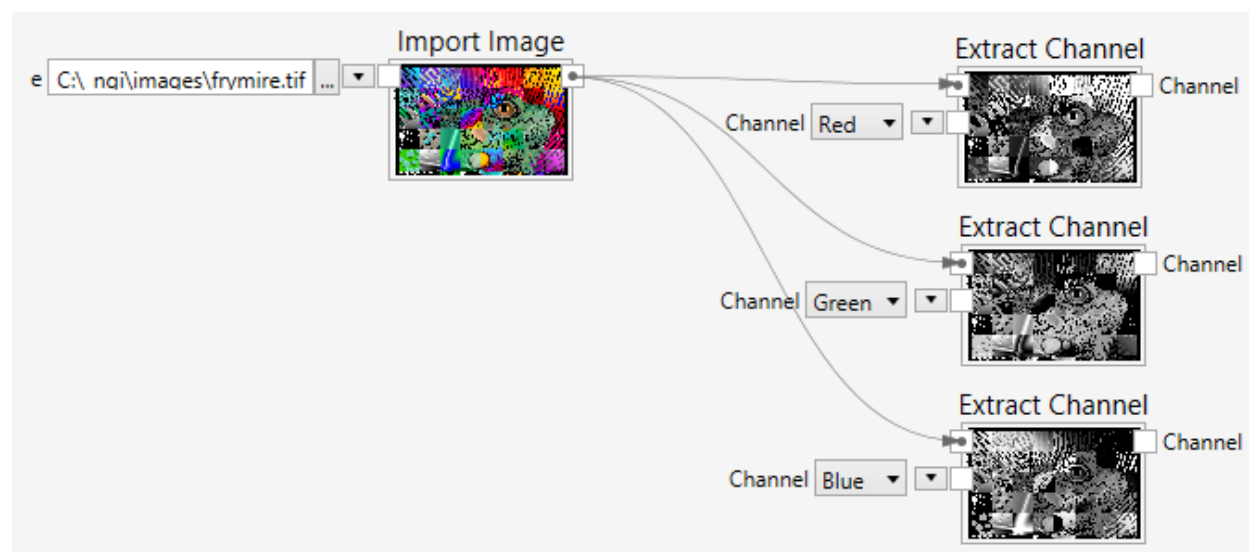
The purpose of this node is to make it easy for the user to select a file from the filesystem.

15.3.108 Fill Holes

Fills holes in a list of objects.







Inputs

Regions (Type: `RegionList`)

The list of regions.

Outputs

Regions (Type: `RegionList`)

The resulting list of regions, which have their holes filled.

Comments

Here is an example of a list of regions:

and here is the same list of regions with their holes filled:

15.3.109 Flip

Flips an image.

Inputs

Image (Type: `Image`)

The input image.

Outputs

Flipped (Type: `Image`)

The output image.

Comments

This function reverses an image in the vertical direction (turns it upside down).

Sample

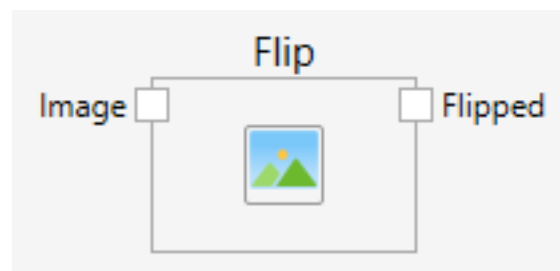
Here is an example that flips an image.

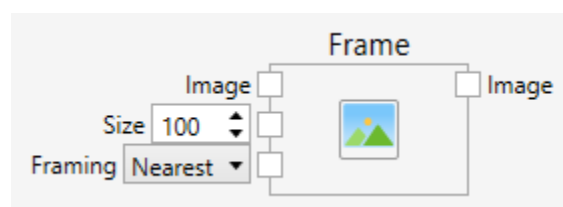
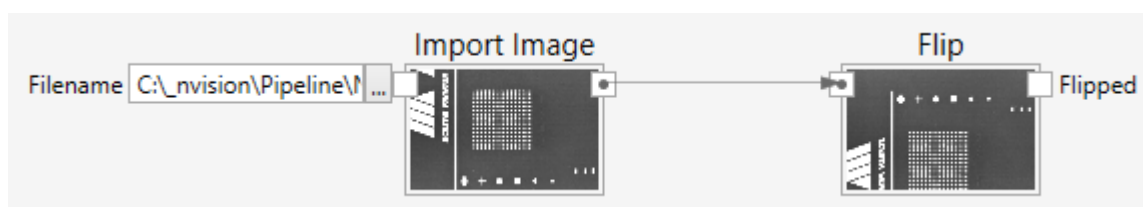
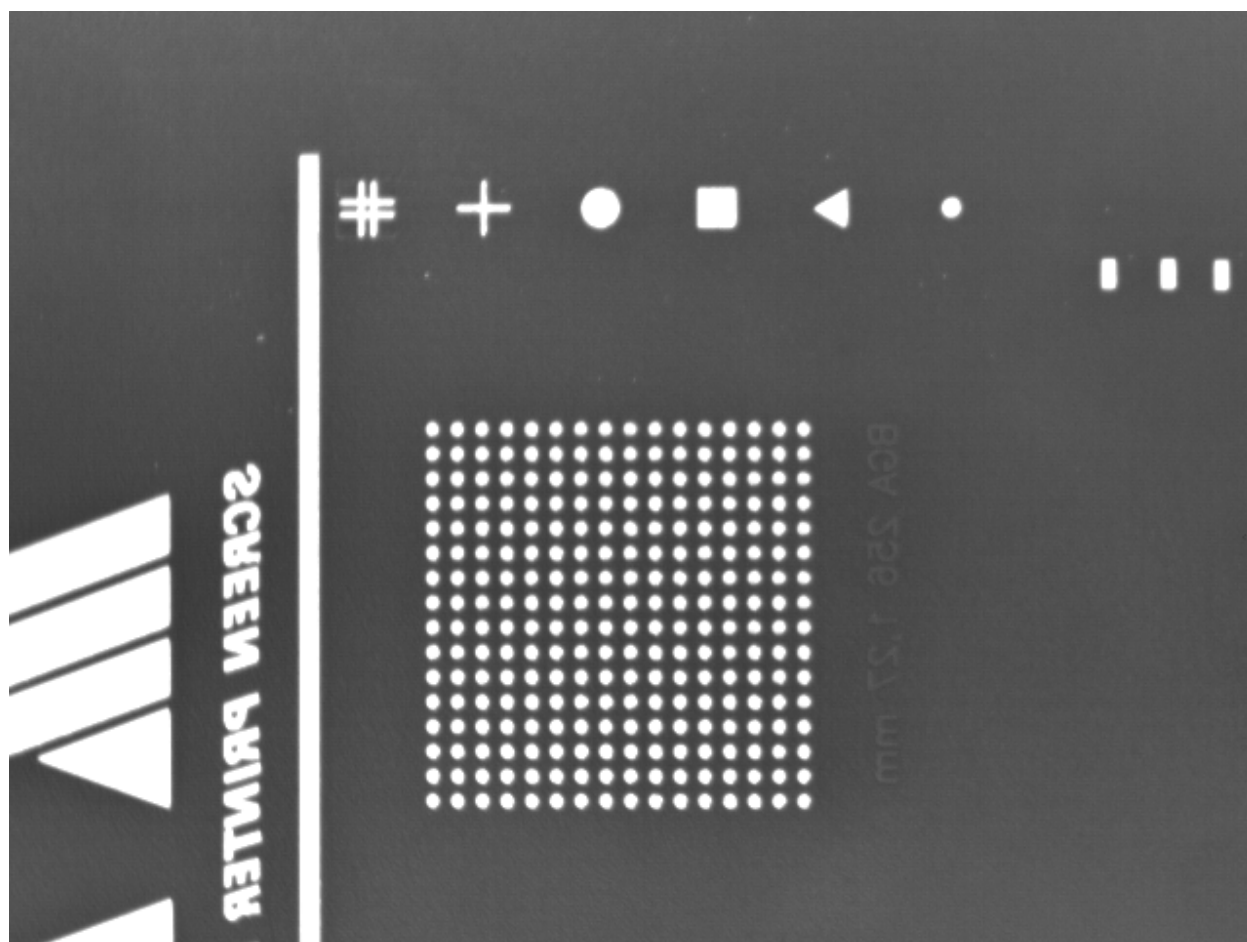
15.3.110 Frame

Puts a frame around an image.

[] ^ { | } € £ ¥
 . / : ; < = > ? @
 ! " # \$ % & ' () * + , -
 n o p q r s t u v w x y z
 a b c d e f g h i j k l m
 N O P Q R S T U V W X Y Z
 A B C D E F G H I J K L M
 0 1 2 3 4 5 6 7 8 9

[] ^ { | } € £ ¥
 . / : ; < = > ? @
 ! " # \$ % & ' () * + , -
 n o p q r s t u v w x y z
 a b c d e f g h i j k l m
 N O P Q R S T U V W X Y Z
 A B C D E F G H I J K L M
 0 1 2 3 4 5 6 7 8 9





Inputs

Image (Type: Image)

The input image.

Size (Type: Int32)

The frame size.

Framing (Type: String)

The framing method (Nearest, Reflective (Middle), Reflective (Border), Periodic).

Outputs

Image (Type: Image)

The output image.

Comments

The **Frame** node puts a border around the input image.

The border can be put around in various ways. *Nearest* takes the nearest pixel value to the image border and repeats it.

Reflective (Middle) reflects the image at the middle of the border pixel.

Reflective (Border) reflects the image at the border of the border pixel.

Periodic periodically repeats the image.

Sample

Here is an example that puts a symmetric frame around an image.

15.3.111 Frame (Extent)

Puts a frame around an image.

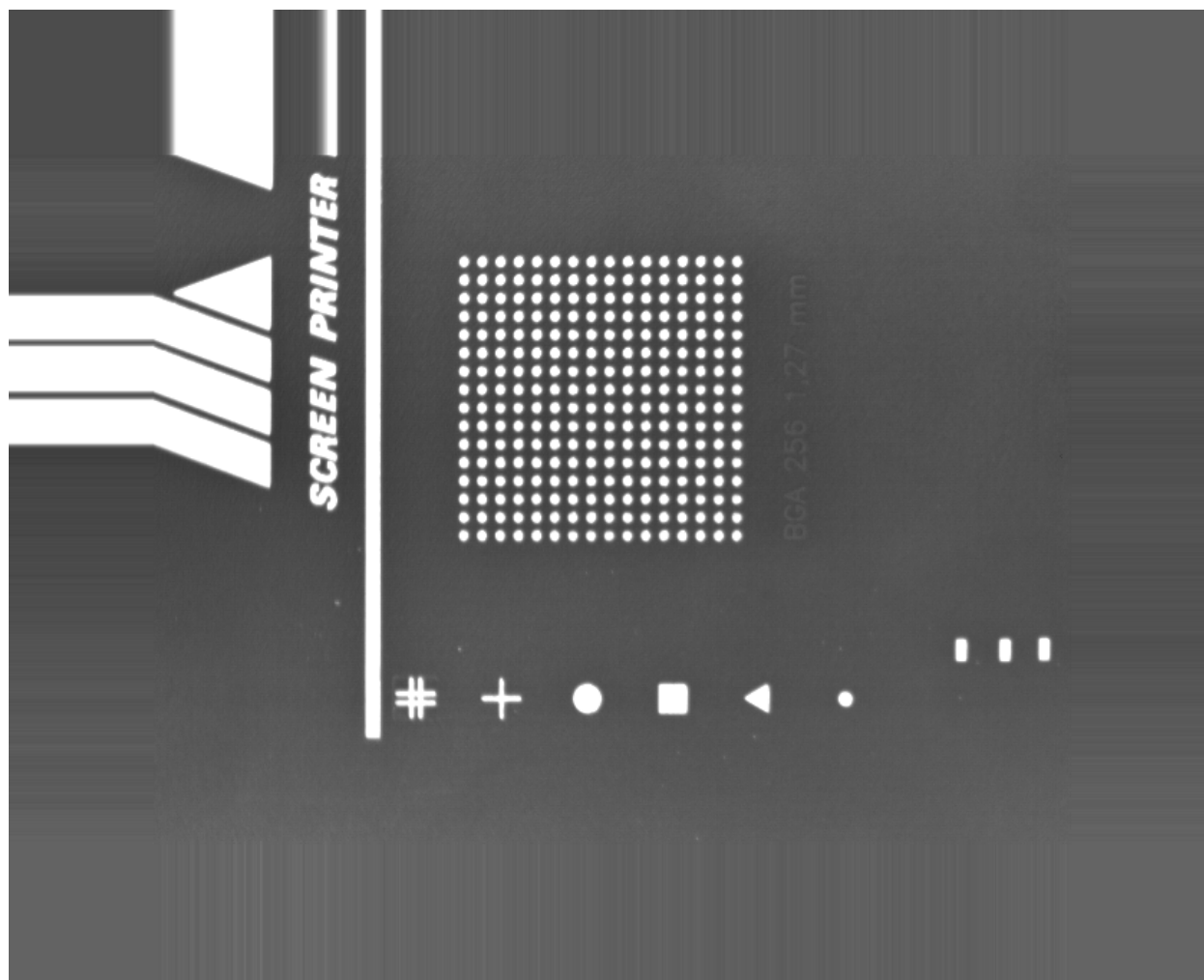
Inputs

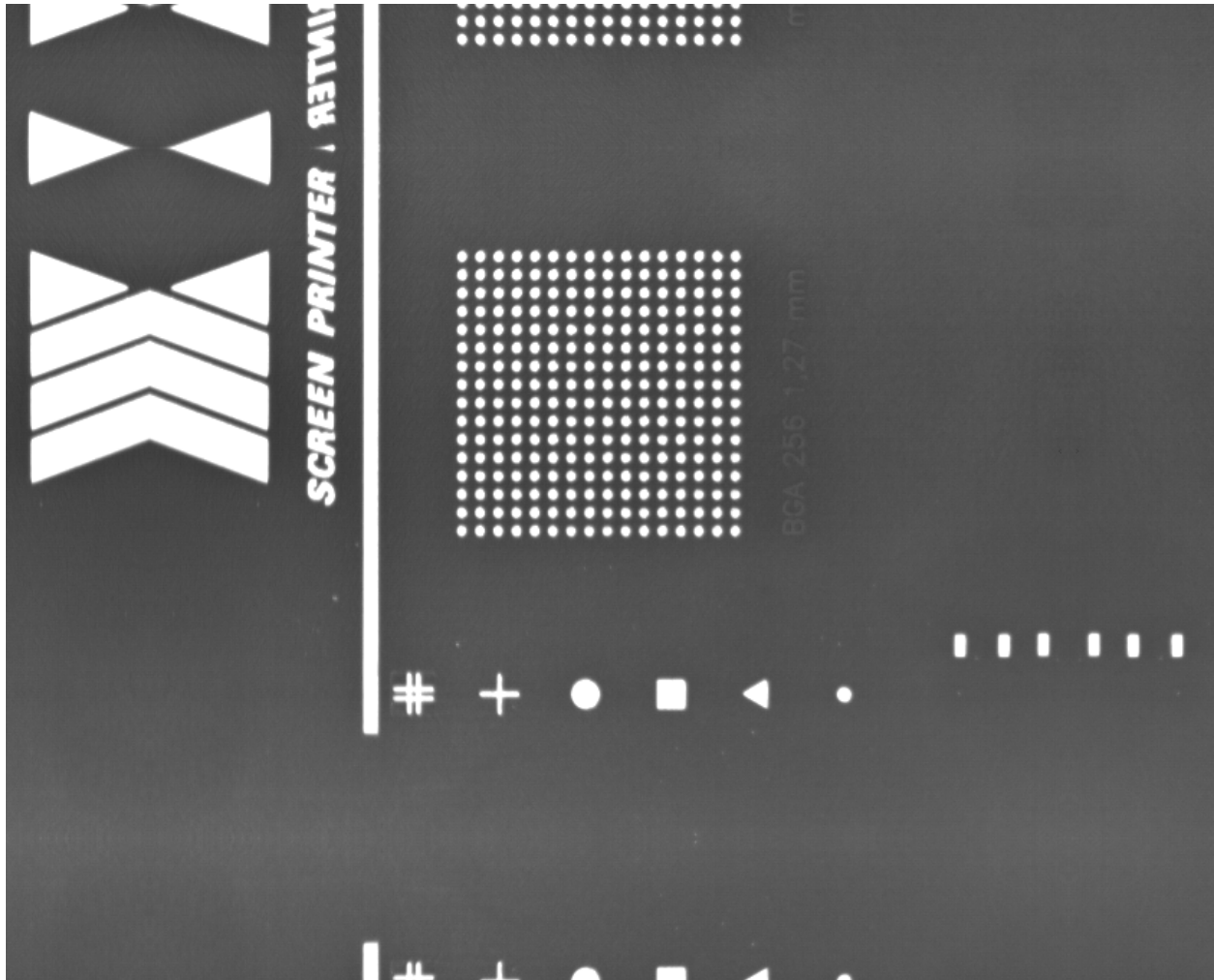
Image (Type: Image)

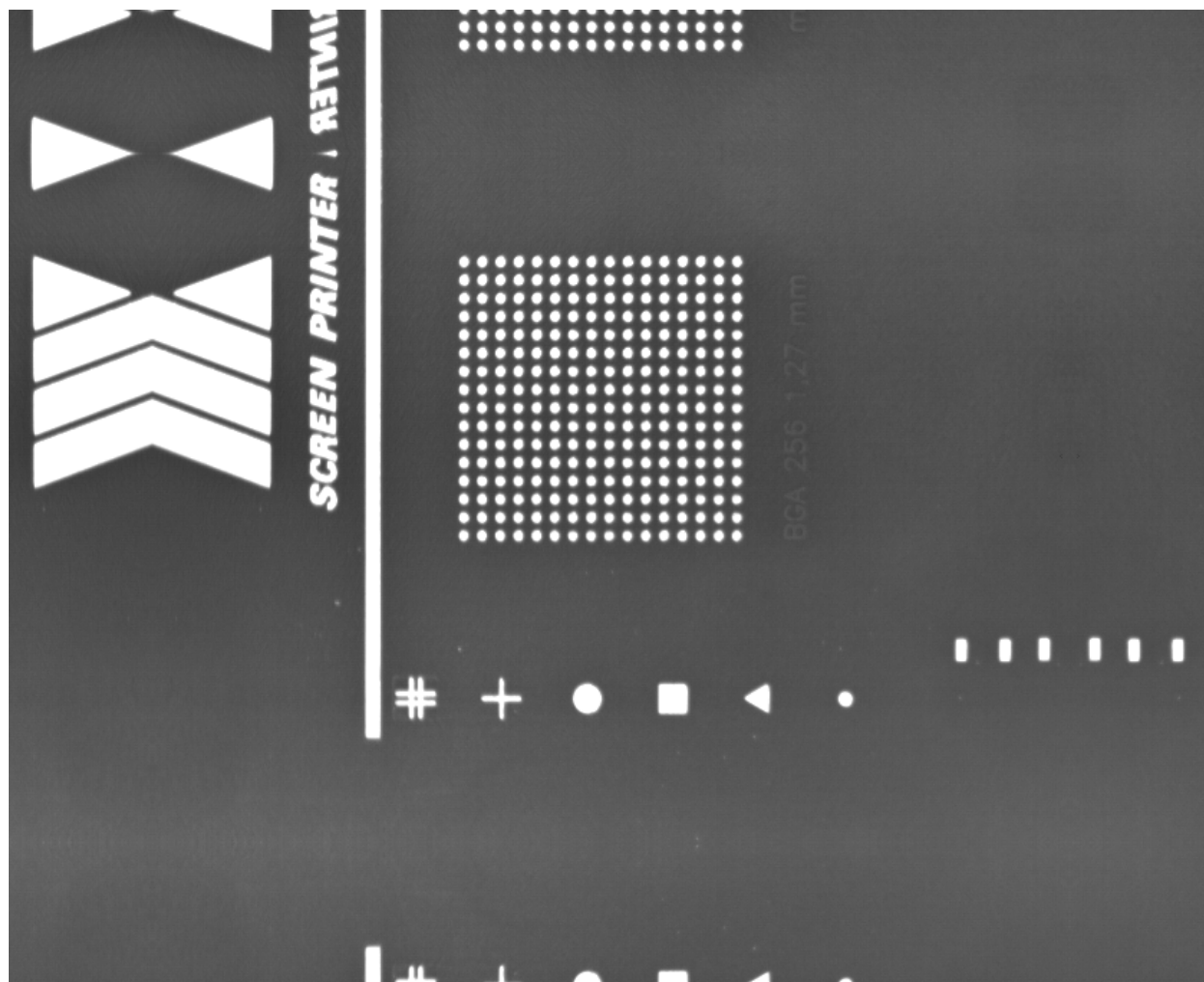
The input image.

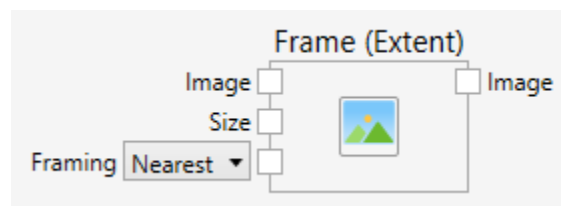
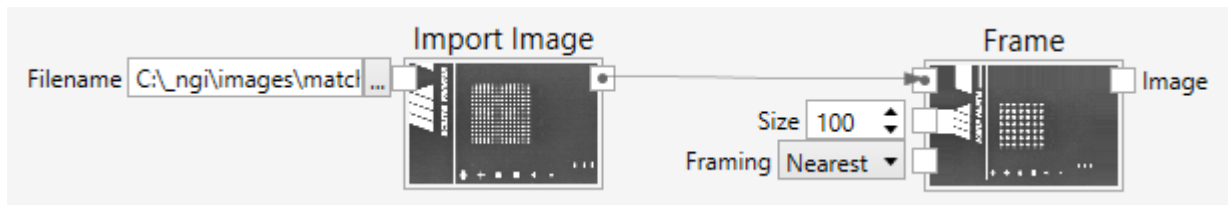
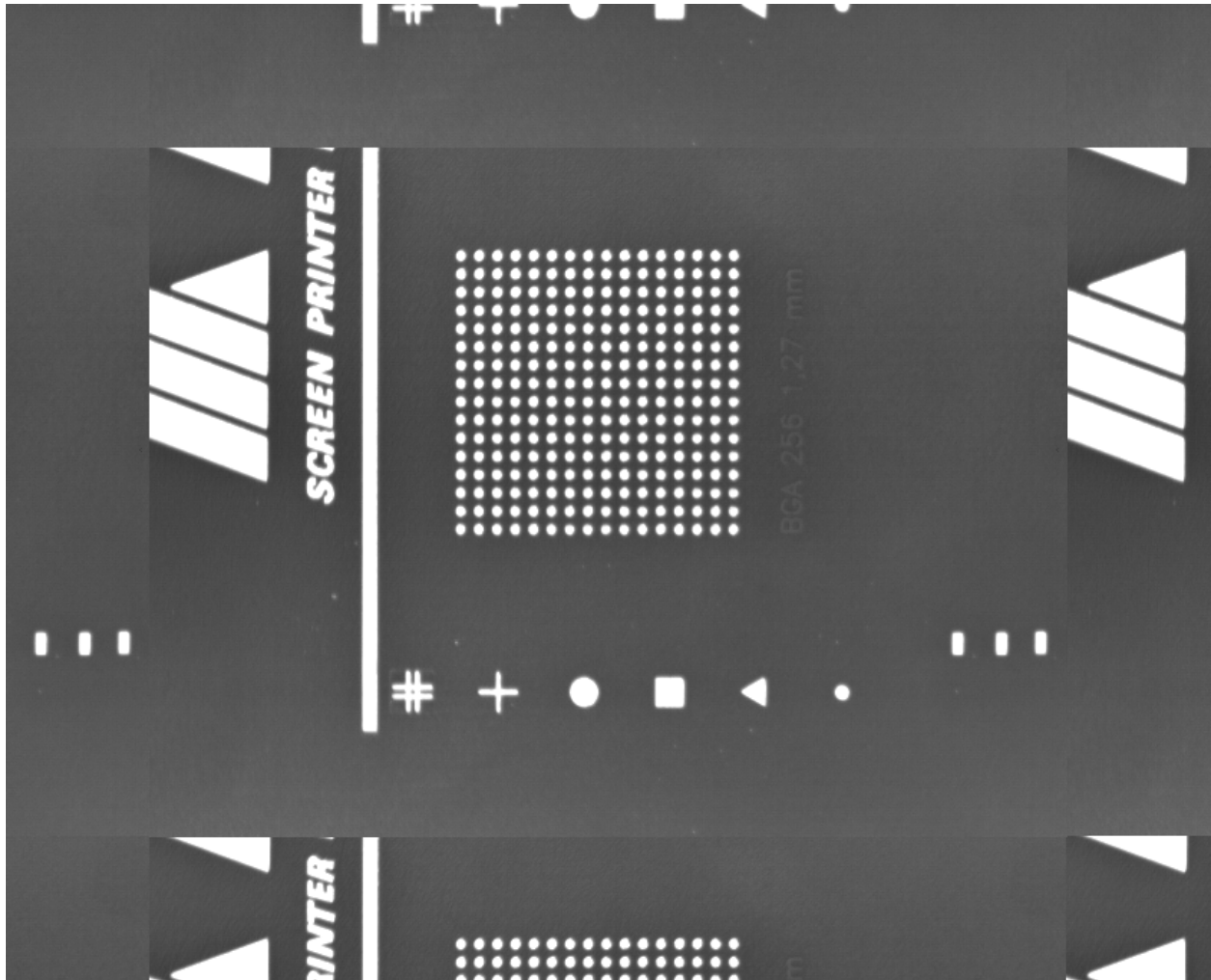
Size (Type: Extent3d)

The frame size.









Framing (Type: String)

The framing method (Nearest, Reflective (Middle), Reflective (Border), 'Periodic').

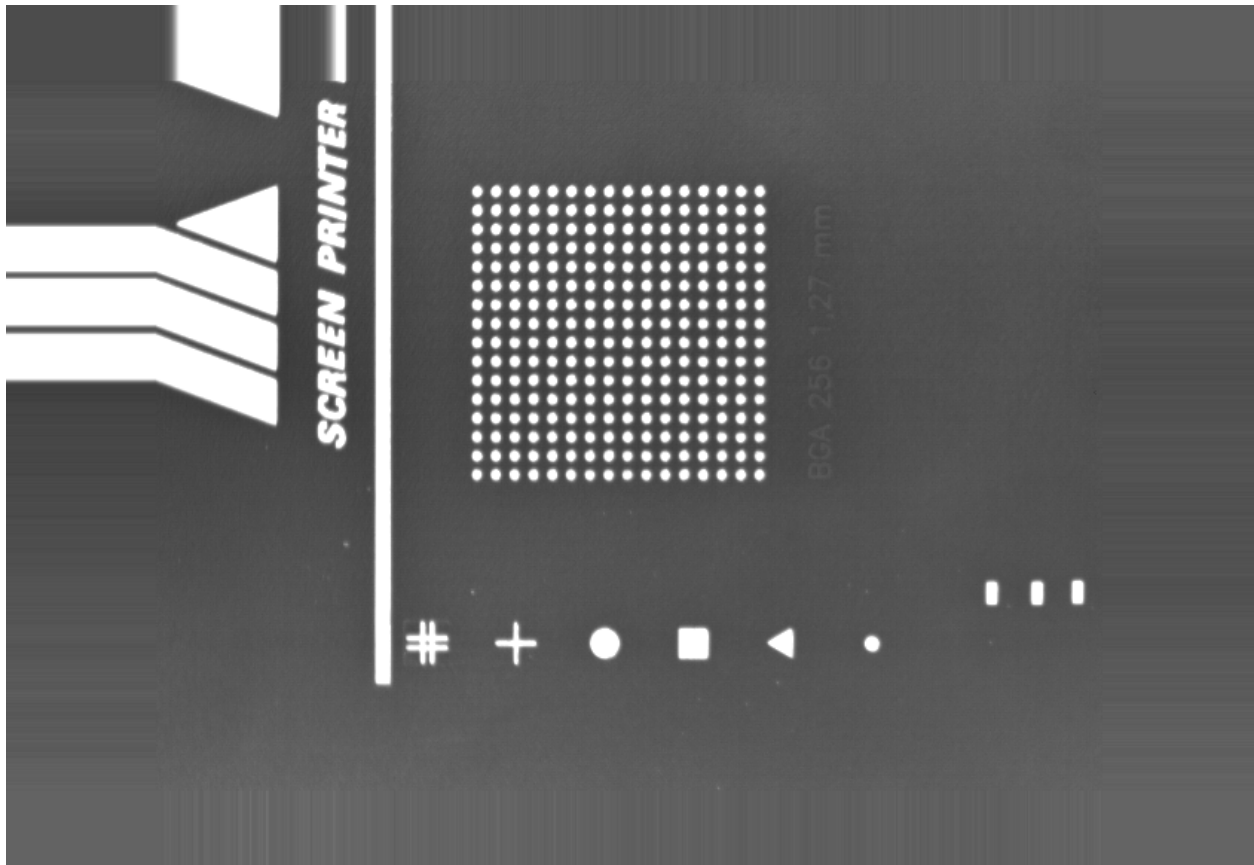
Outputs**Image (Type: Image)**

The output image.

Comments

The **Frame (Extent)** node puts a border around the input image.

The border can be put around in various ways. *Nearest* takes the nearest pixel value to the image border and repeats it.



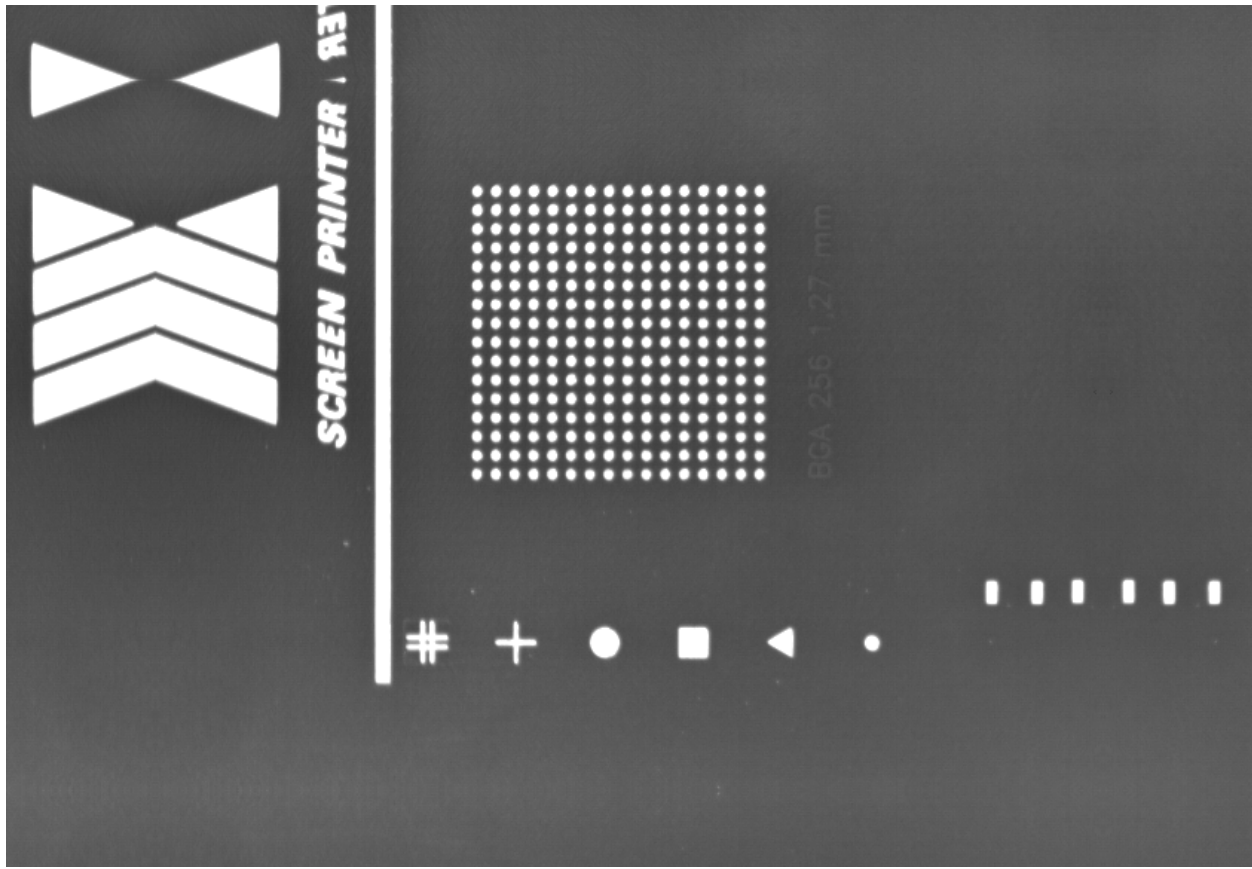
Reflective (Middle) reflects the image at the middle of the border pixel.

Reflective (Border) reflects the image at the border of the border pixel.

Periodic periodically repeats the image.

15.3.112 GammaPalette

Creates a palette with a gamma transfer function.



Inputs

Type (Type: String)

The type of the palette. Choices are `PaletteByte`, `PaletteUInt16`, `PaletteUInt32`, `PaletteDouble`, `PaletteRgbByte`, `PaletteRgbUInt16`, `PaletteRgbUInt32` and `PaletteRgbDouble`.

Width (Type: Int32)

The width of the palette.

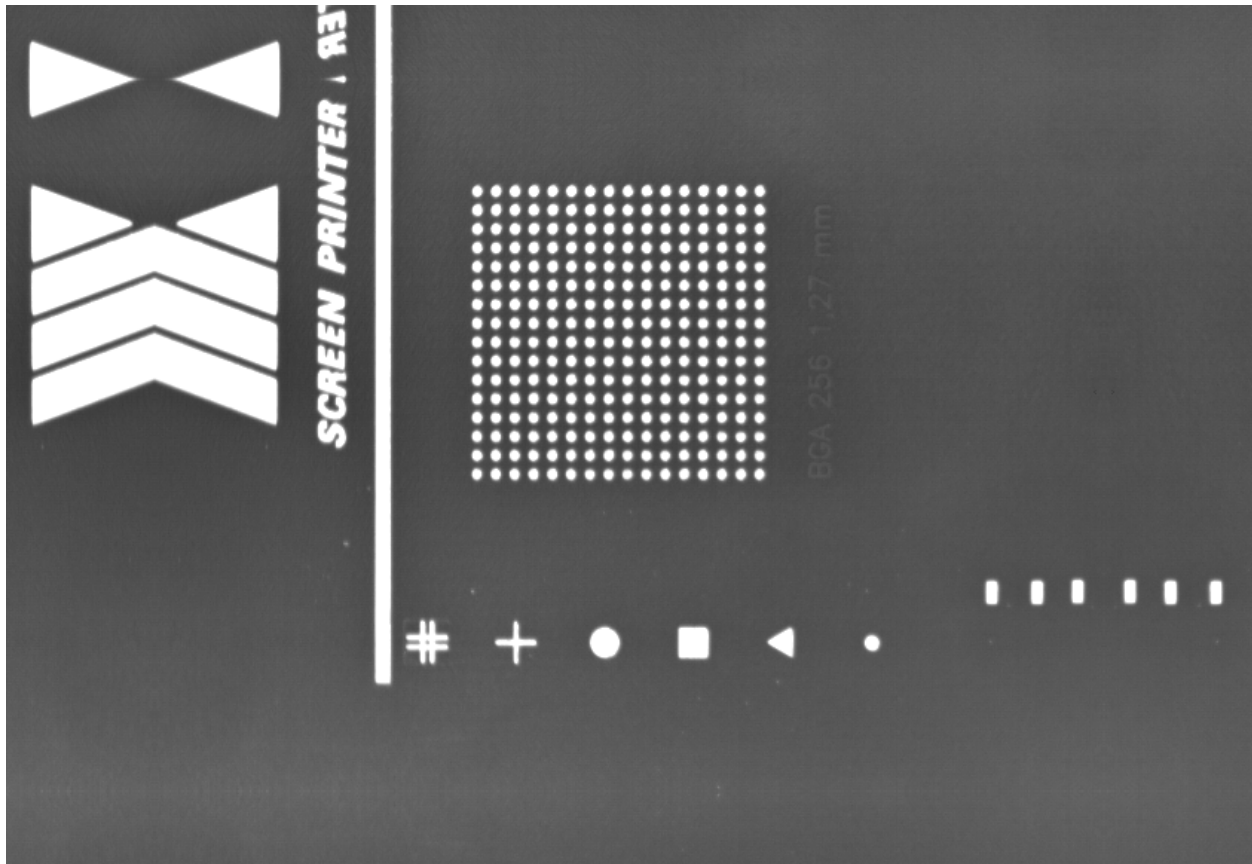
Gamma (Type: Double)

The gamma parameter.

Outputs

Palette (Type: Palette)

The output palette.



Comments

This function creates a palette with a gamma function.

A gamma value below 1.0 compresses darks and expands brights. Here is an example of a gamma palette with a gamma of 0.5:

A gamma value of exactly 1.0 creates a linear transfer function:

A gamma value above 1.0 expands darks and compresses brights. Here is an example of a gamma palette with a gamma of 2.0:

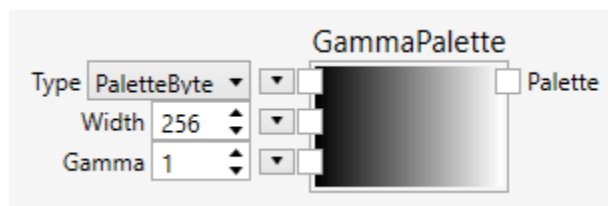
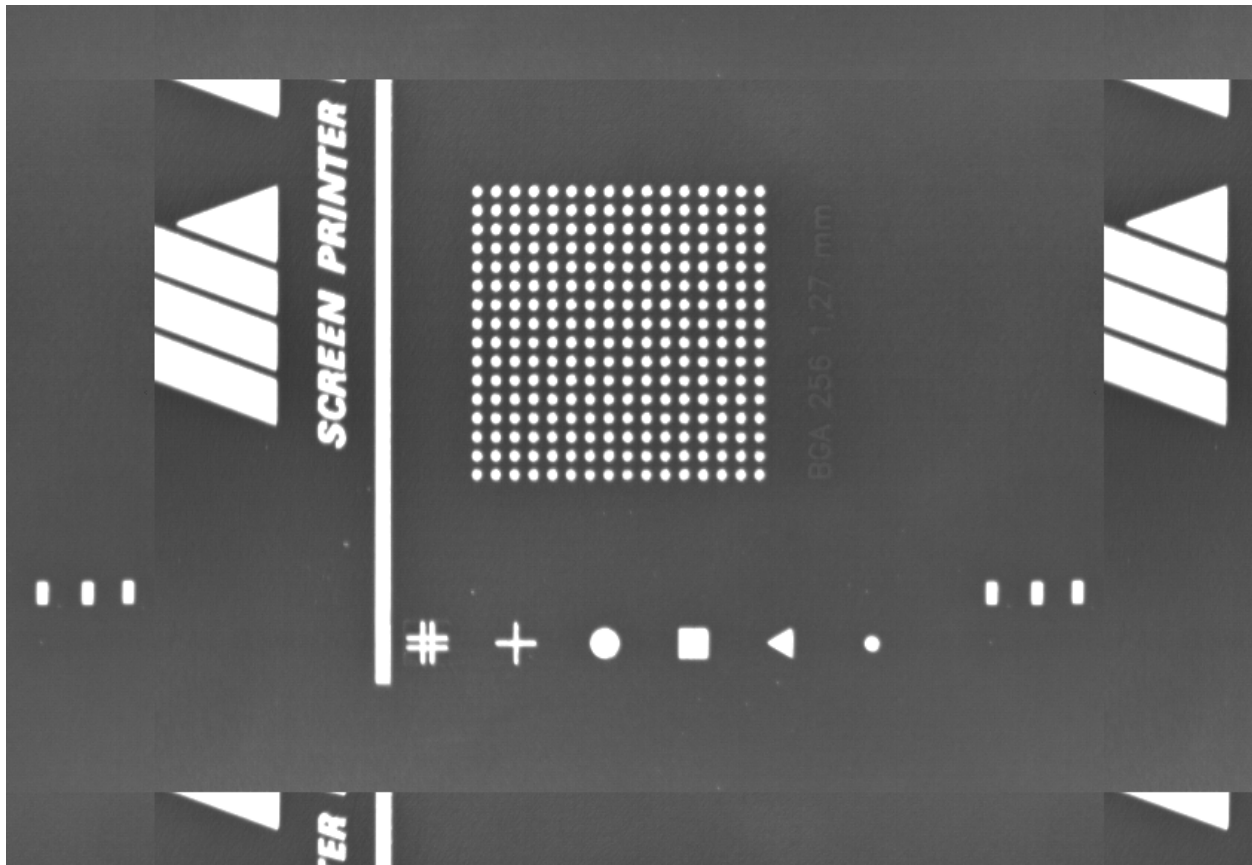
15.3.113 Gauss

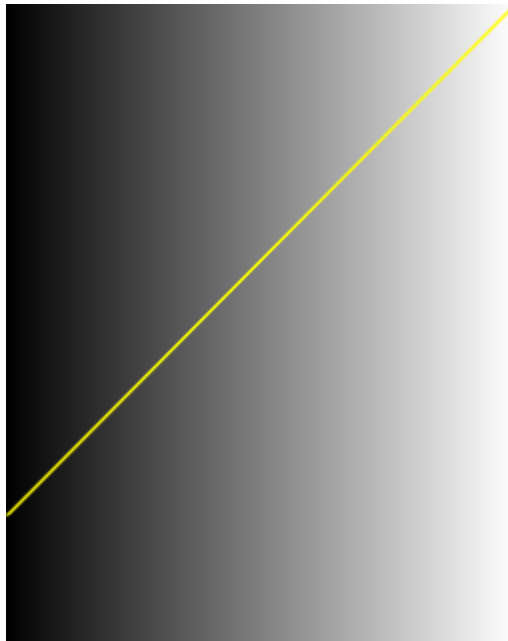
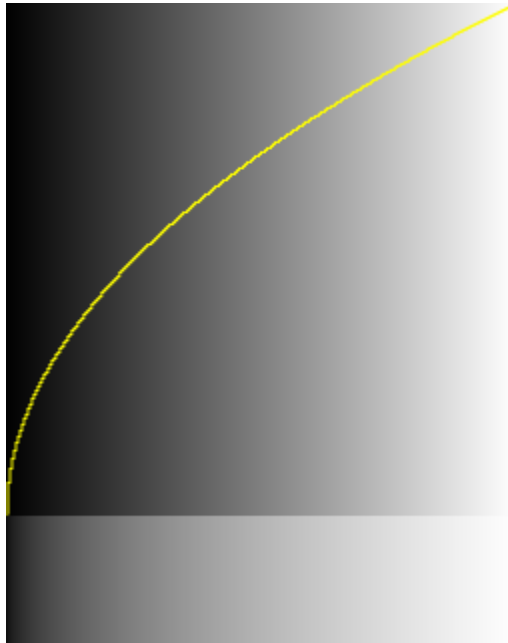
Filters an image using a gaussian kernel of size 3x3 or 5x5. A gaussian filter is a lowpass and blurs an image.

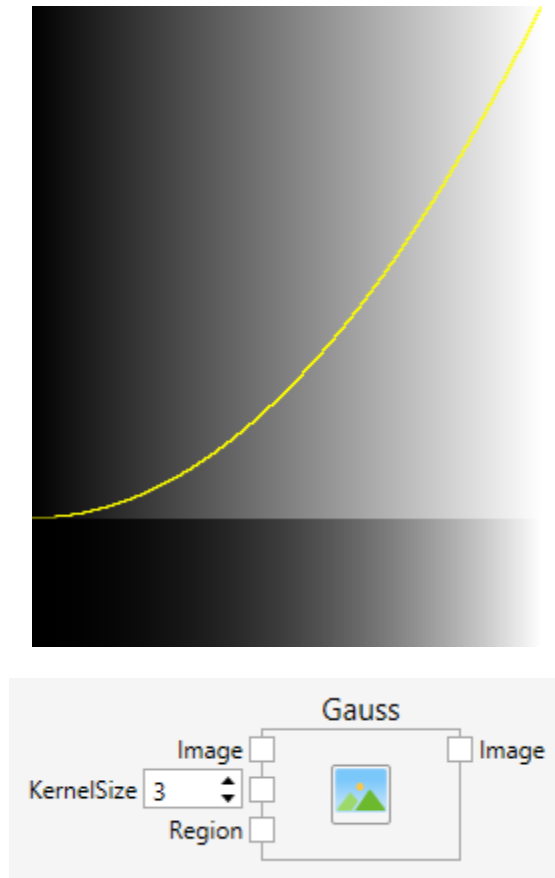
Inputs

Image (Type: Image)

The input image.





**KernelSize (Type: Int32)**

The kernel size (3 -> 3x3, 5 -> 5x5).

Region (Type: Region)

Specifies an optional area of interest.

Outputs**Image (Type: Image)**

The output image.

Comments

The function applies a gauss filter. The corresponding kernel is either a 3x3 matrix with the following values:

```
1 2 1
2 4 2
1 2 1
```

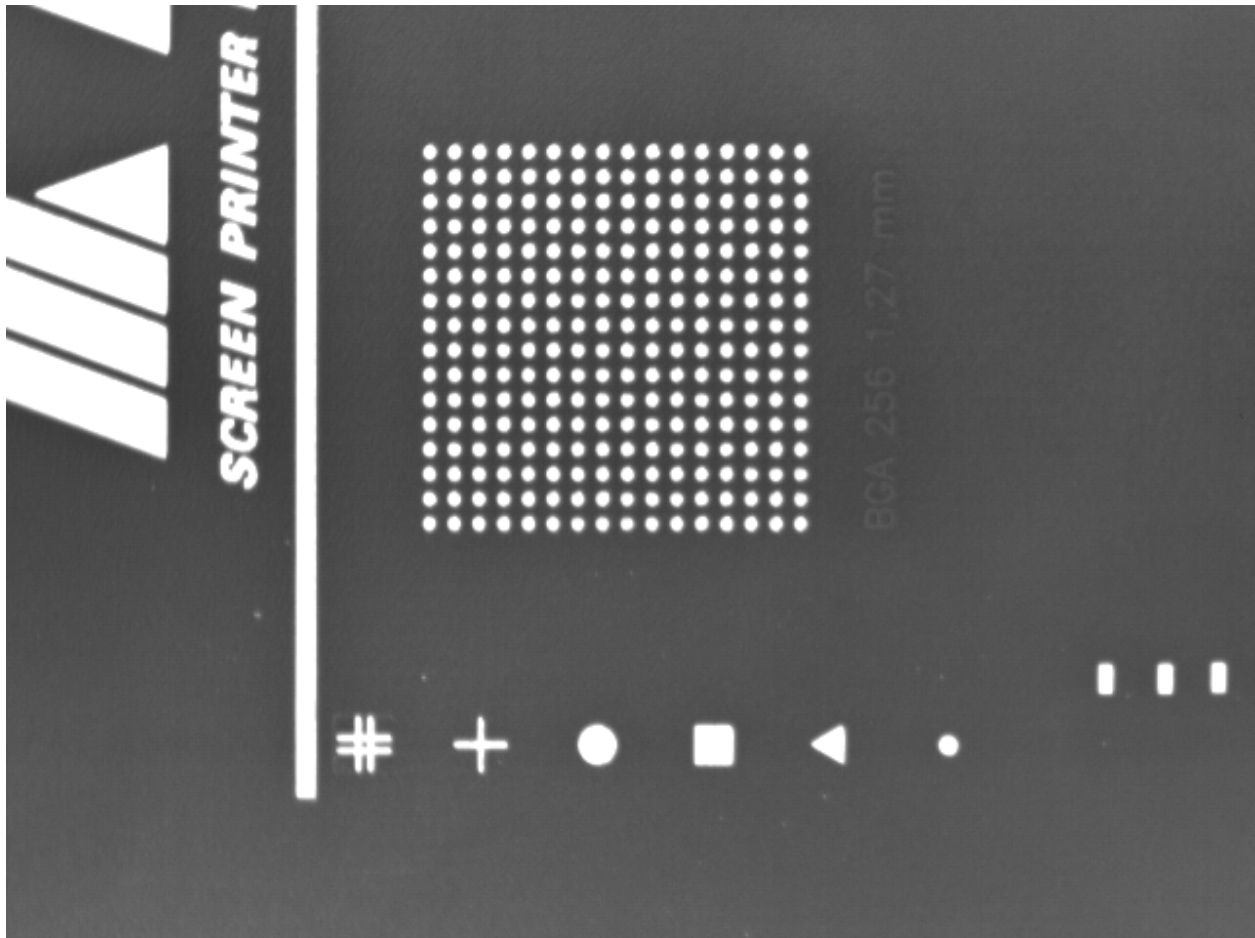
or a 5x5 matrix with the following values:

```
2  7  12  7  2
7 31  52 31  7
12 52 127 52 12
7 31  52 31  7
2  7  12  7  2
```

The effect of a gauss filter is a lowpass. The lowpass filter emphasizes low frequencies and attenuates high frequencies. The strength of the lowpass filter depends on the size of the kernel.

Here are a few results of the gauss filter with increasing kernel sizes:

Original:

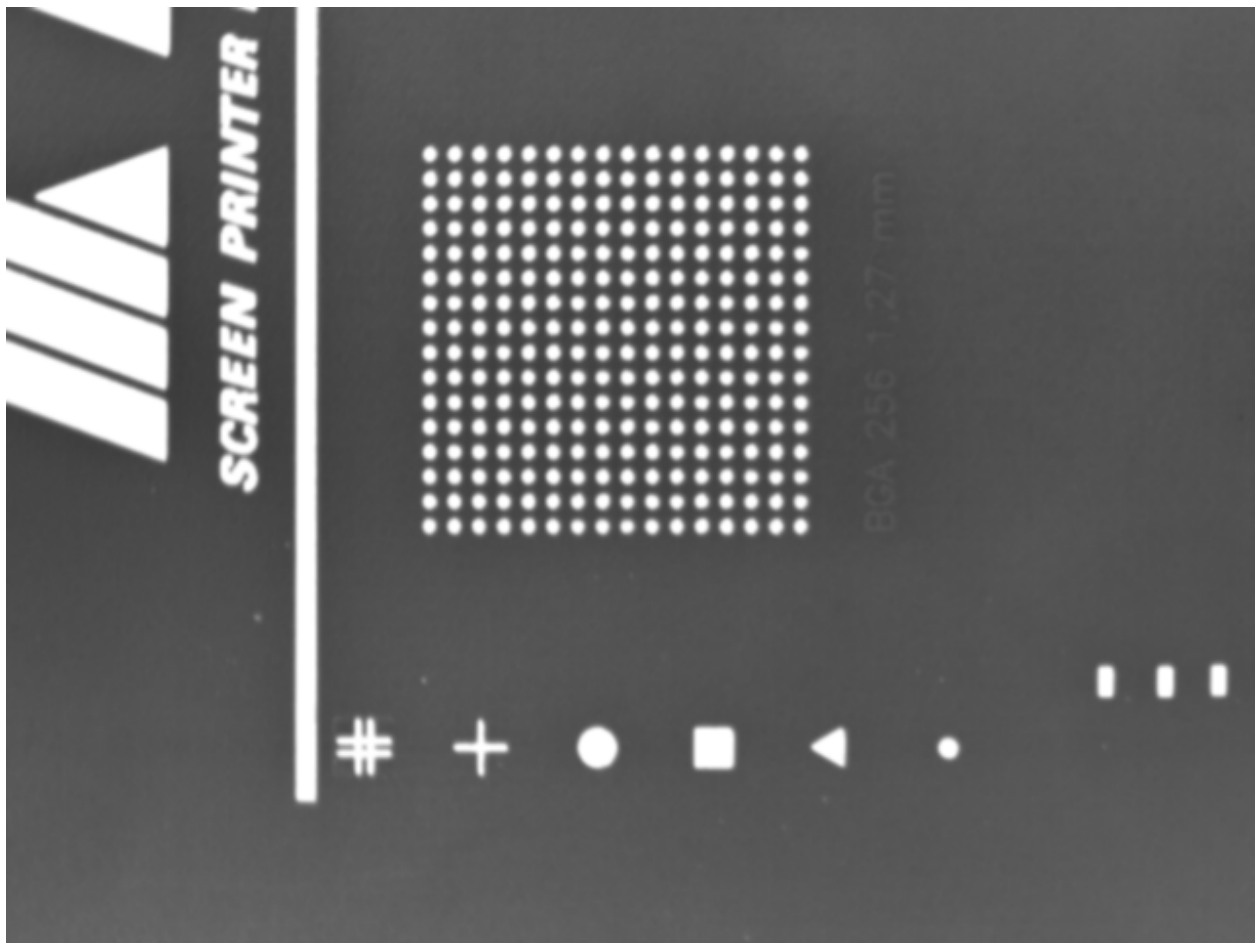


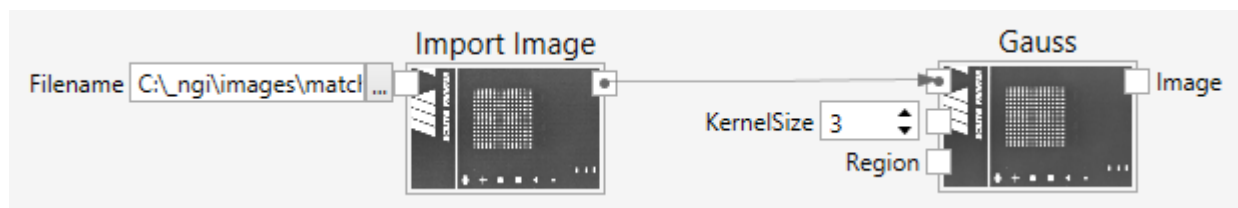
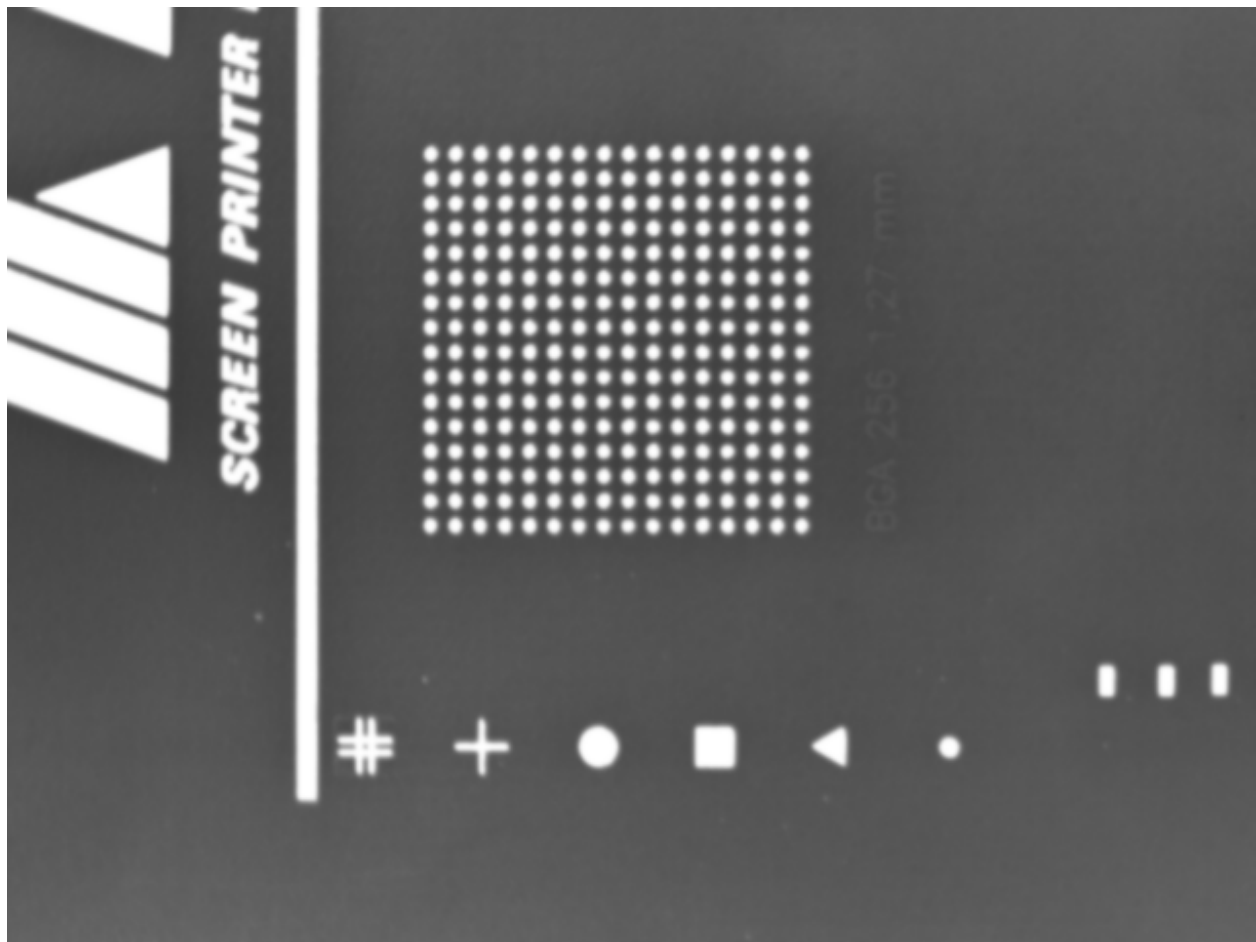
KernelSize = 3:

KernelSize = 5:

Sample

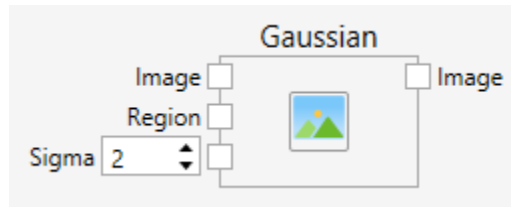
Here is an example that shows how to use the gauss filter.





15.3.114 Gaussian

Filters an image using a gaussian kernel. A gaussian filter is a lowpass and blurs an image. The amount of blurring is determined by the parameter sigma, which in turn determines the size and the parameters of the convolution kernel.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Specifies an optional area of interest.

Sigma (Type: Double)

Determines the size of the kernel.

Outputs

Image (Type: Image)

The output image.

Comments

The gaussian filter is considered an optimal lowpass or blurring filter. It is isotropic if the filter kernel is large enough for a sufficient approximation (at least 5x5, i.e. $\sigma > 5/3$). It behaves well in frequency space and is clearly superior to a box filter. The kernel size of the gaussian is 3 times sigma, rounded up to the next odd integer.

The filter kernel corresponds to a two-dimensional gaussian, where sigma determines to the width of the bell-shaped curve and

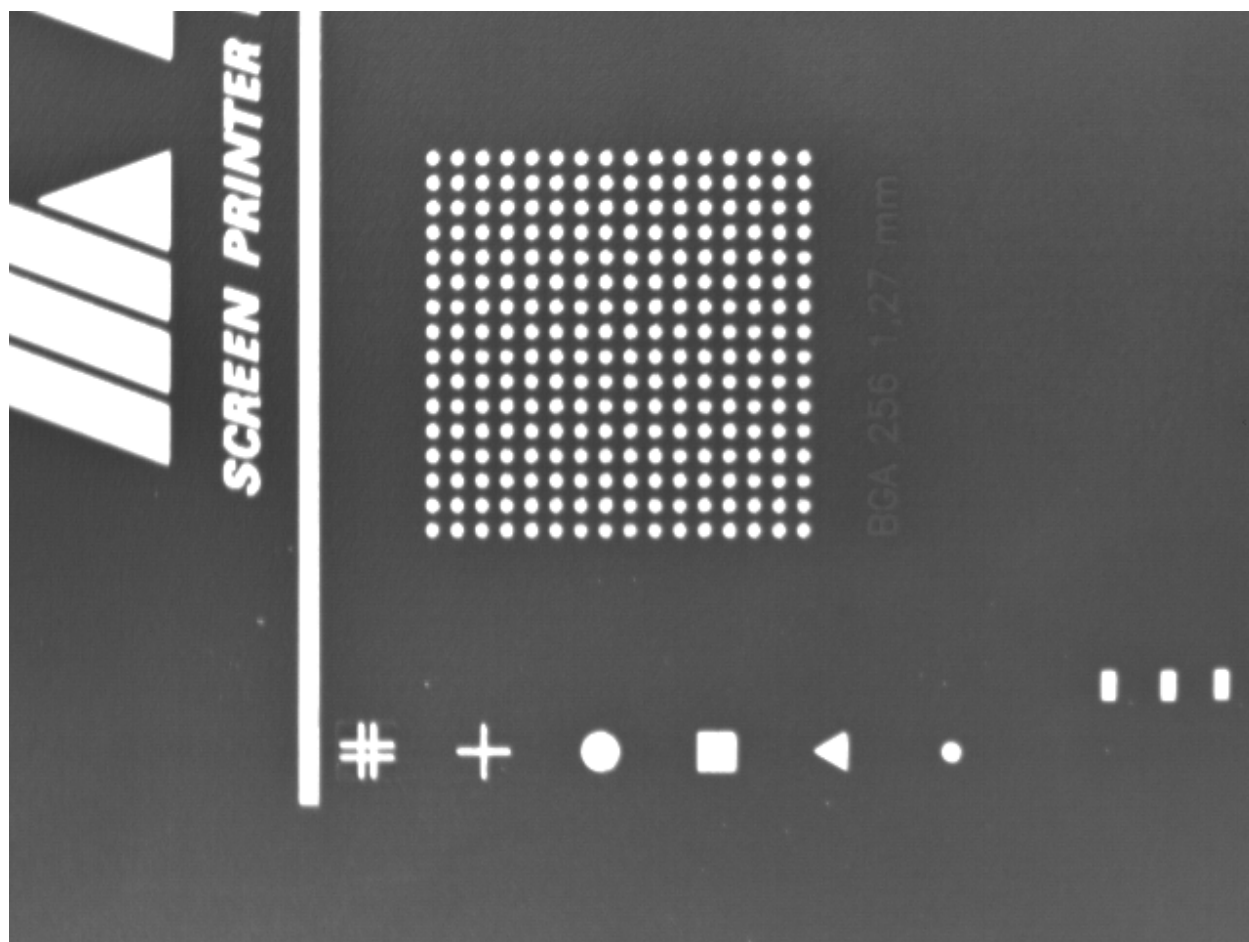
$$r = (x^2 + y^2)$$

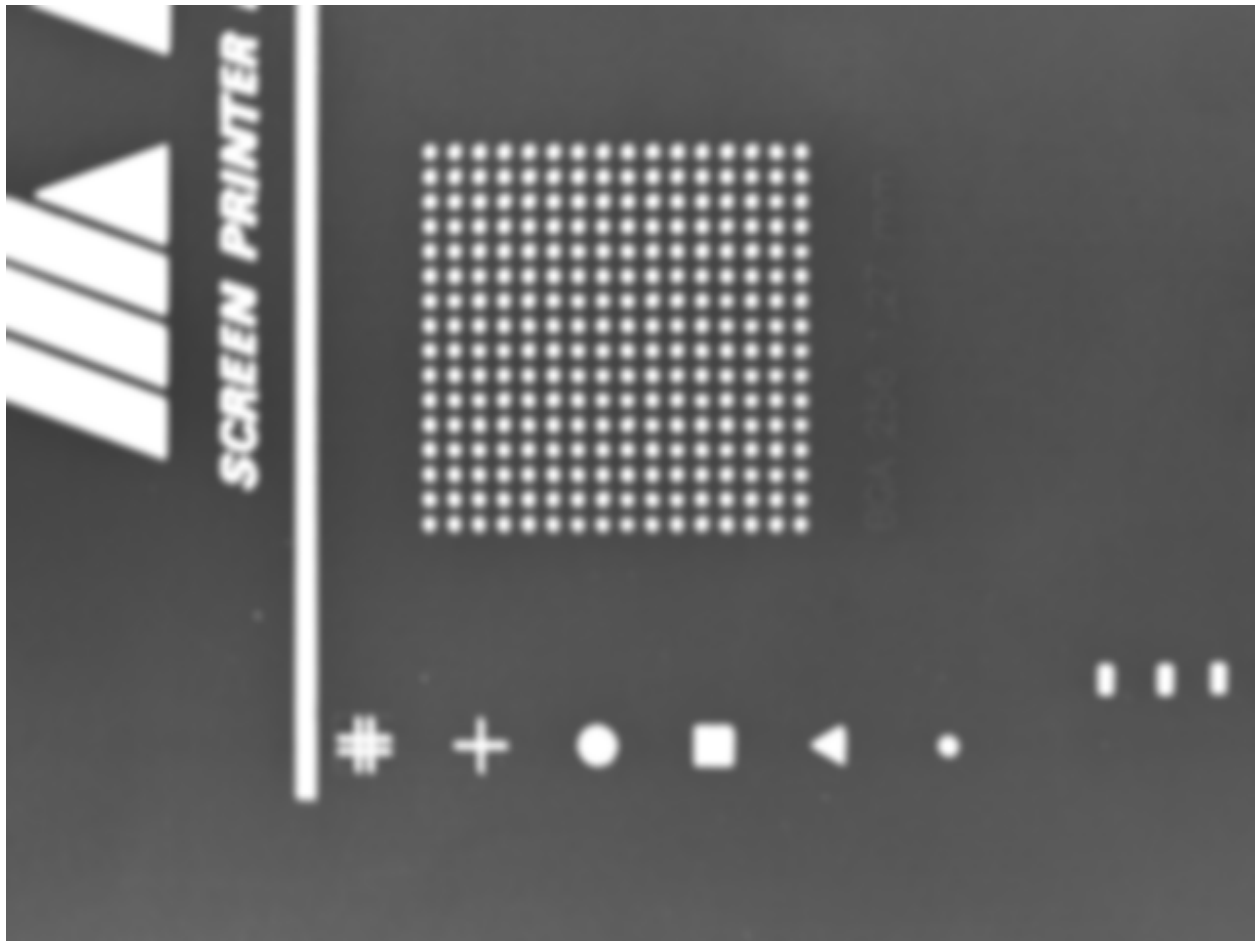
is the distance from the center:

$$G(x, y) = e^{-(x^2 + y^2)/(2 * \sigma^2)}$$

Here are a few results of the gaussian filter with increasing sigma values:

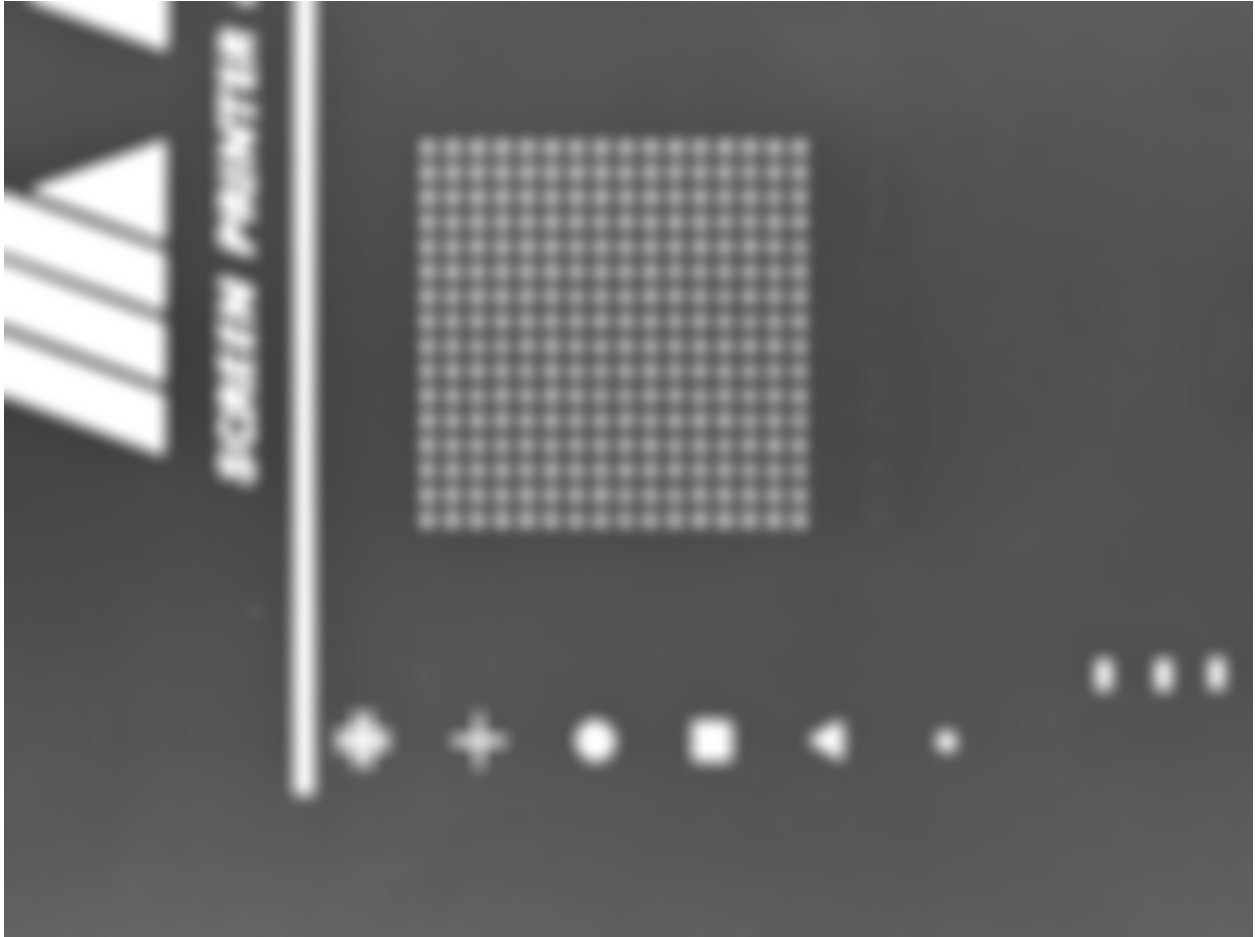
Original:





Sigma = 2:

Sigma = 4:



Sigma = 8:

Sigma = 16:

Sigma = 32:

Sigma = 64:

Sample

Here is an example that shows how to use the gaussian filter.

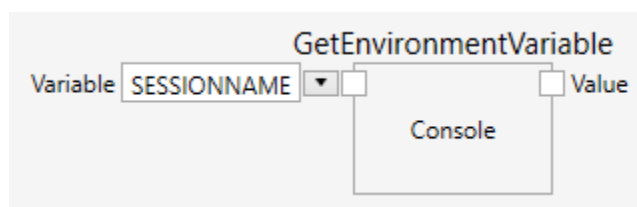
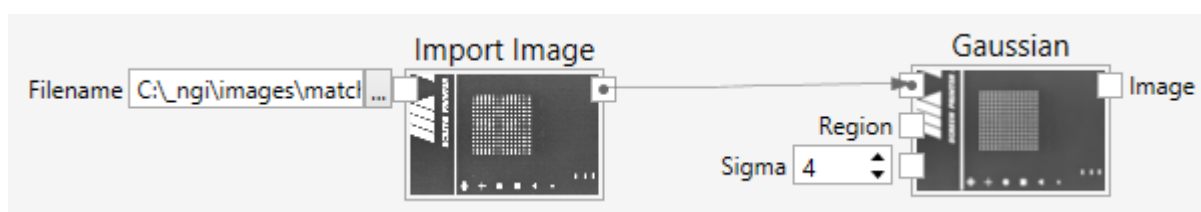
15.3.115 GetEnvironmentVariable

Retrieves the value of an environment variable from the current process.









Inputs

Variable (Type: string)

The name of the environment variable.

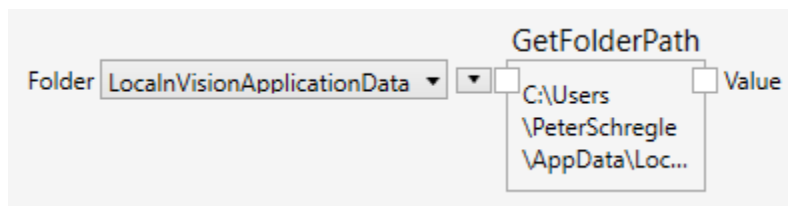
Outputs

Value (Type: string)

The value of the environment variable specified by variable, or null if the environment variable is not found.

15.3.116 GetFolderPath

Gets the path to the system special folder that is identified by the specified enumeration.



Inputs

Folder (Type: string)

A string that identifies a system special folder.

Outputs

Path (Type: string)

The path to the specified system special folder, if that folder physically exists on your computer; otherwise, an empty string ("").

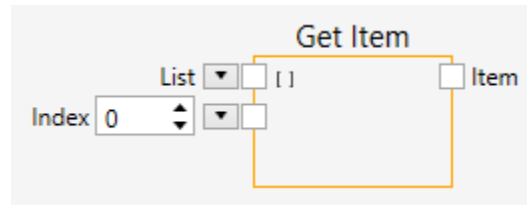
Comments

The following folders can be identified:

folder	description
Applica- tionData	The directory that serves as a common repository for application-specific data for the current roaming user. A roaming user works on more than one computer on a network. A roaming user's profile is kept on a server on the network and is loaded onto a system when the user logs on.
Com- monAp- plication- Data	The directory that serves as a common repository for application-specific data that is used by all users.
Com- mon- Desk- topDirec- tory	The file system directory that contains files and folders that appear on the desktop for all users.
Com- monDoc- uments	The file system directory that contains documents that are common to all users.
Com- monMu- sic	The file system directory that serves as a repository for music files common to all users.
Com- monPic- tures	The file system directory that serves as a repository for image files common to all users.
Com- monPro- gram- Files	The directory for components that are shared across applications. To get the x86 common program files directory on a non-x86 system, use the ProgramFilesX86 member.
Com- monPro- gram- FilesX86	The Program Files folder.
Com- mon- Videos	The file system directory that serves as a repository for video files common to all users.
Cookies	The directory that serves as a common repository for Internet cookies.
Desktop	The logical Desktop rather than the physical file system location.
Desk- topDirec- tory	The directory used to physically store file objects on the desktop.
Favorites	The directory that serves as a common repository for the user's favorite items.
LocalAp- plication- Data	The directory that serves as a common repository for application-specific data that is used by the current, non-roaming user.
Local- nVision- Applica- tionData	The directory where nVision stores local data.
MyDoc- uments	The My Documents folder. This member is equivalent to Personal.
MyMu- sic	The My Music folder.
MyPic- tures	The My Pictures folder.
MyVideos	The file system directory that serves as a repository for videos that belong to a user.
Personal	The directory that serves as a common repository for documents. This member is equivalent to My- Documents.
464 Pipeline	The directory where the pipeline is located.
Program- Files	The program files directory. On a non-x86 system, passing ProgramFiles to the GetFolderPath method returns the path for non-x86 programs. To get the x86 program files directory on a non-x86 system, use the ProgramFilesX86 member.

15.3.117 GetListItem

Takes a specific item out of a list.



Inputs

List (Type: List<T>)

The list.

Index (Type: Int32)

The zero-based index of the item. If the index is out of range, an error will be shown.

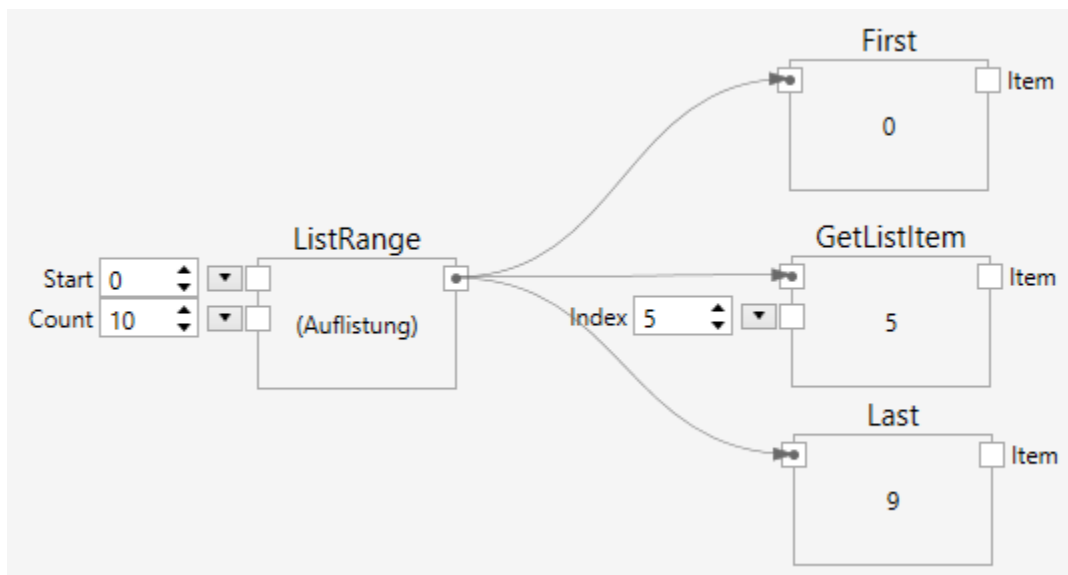
Outputs

Item (Type: T)

The specific item of the list.

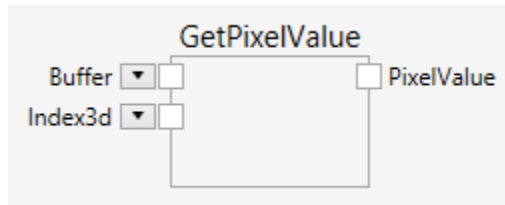
Sample

Here is an example:



15.3.118 GetPixelValue

Reads a pixel value from a buffer.



Inputs

Buffer

The buffer to read from. This can be an image, or a palette, histogram, profile, etc.

Index3d

The three-dimensional index of the pixel.

Outputs

PixelValue

The value of the pixel in the buffer.

Comments

Buffers are three-dimensional: *x* is the horizontal dimension (the positive *x*-axis extents to the right), *y* is the vertical dimension (the positive *y*-axis extends downwards) and *z* is the planar dimension (the positive *z*-axis extends into the paper/screen plane). The origin is at the upper left corner.

Buffers have a variant type: they can be greyscale or color (in different color spaces) with different numeric types (8 bit unsigned, 16 bit unsigned, 32 bit signed, or 64 bit float per channel).

Buffers come in various forms: images are most common, but palettes, histograms or profiles are also buffers.

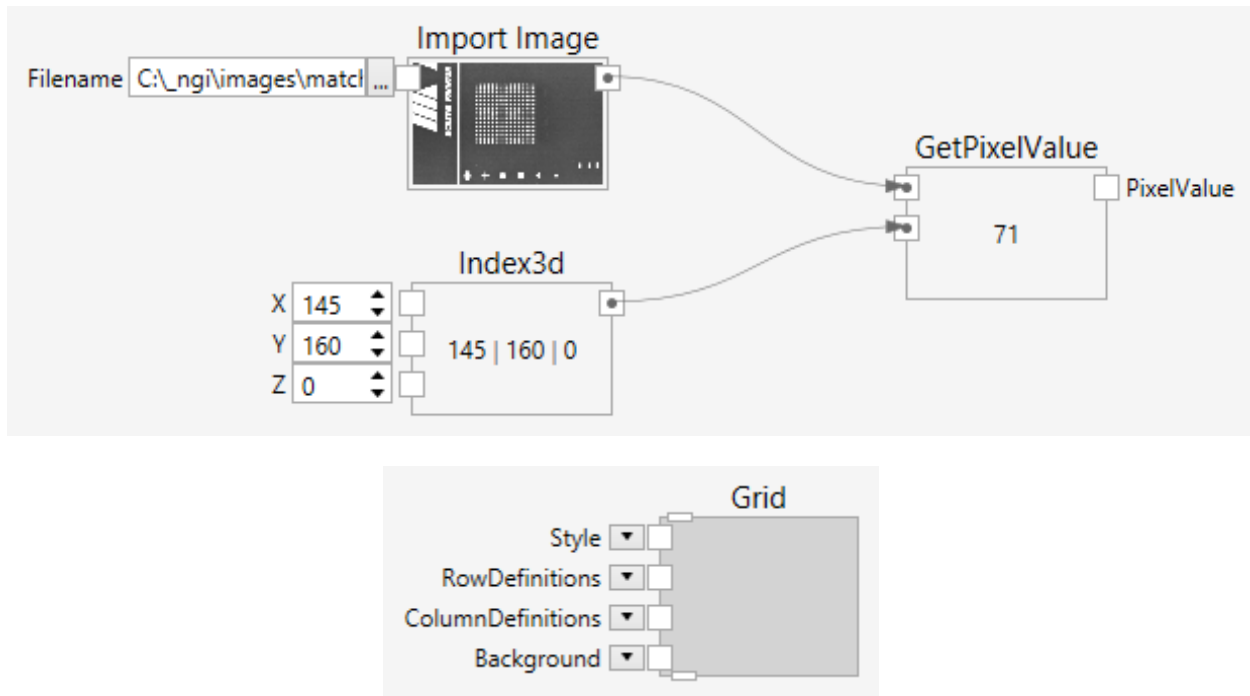
Sample

Here is a sample pipeline that illustrates **GetPixelValue**:

15.3.119 HMI Grid

The **Grid** allows you to split the available area into a rectangular grid with *m* rows and *n* columns. The row and column definitions are build with a list of **GridLength** nodes. The **GridLength** node specifies the size (absolute or relative) of the specific row or column.

At the bottom of the **Grid** node is a pin that allows it to connect several children.



Inputs

Style (Type: `Style`)

Optional styling of the **Grid**.

The **Grid** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Background* styles.

RowDefinitions (Type: `List<GridLength>`)

A list of definitions that determines the number of rows of the grid, as well as their relative or absolute heights. By default the grid has one row.

ColumnsDefinitions (Type: `List<GridLength>`)

A list of definitions that determines the number of columns of the grid, as well as their relative or absolute width. By default the grid has one column.

Background (Type: `SolidColorBrushByte`)

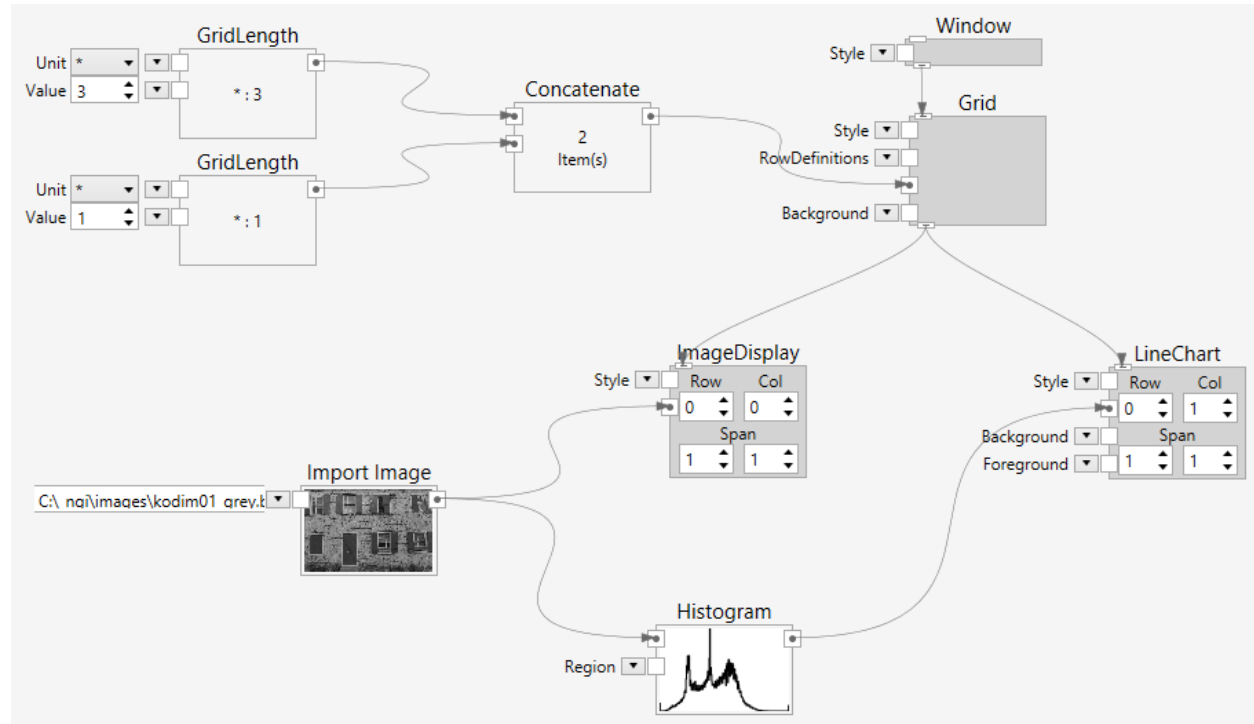
The background of the **Grid**.

Comments

Children of the **Grid** have the attached properties **Row**, **Col**, as well as **RowSpan** and **ColSpan**. These properties define in which grid cell(s) the respective child will be put.

Example

Here is an example that splits the area into two columns and one row, where the left column has 3x size and the right column has 1x size. An image is displayed in the left column and its histogram is displayed in the right column. This definition:



creates the following user interface:

15.3.120 HMI GridLength

The **GridLength** node specifies the size (absolute or relative) of a specific row or column.

Inputs

Unit (Type: string)

The possible values are `Auto`, `*` or `Absolute`.

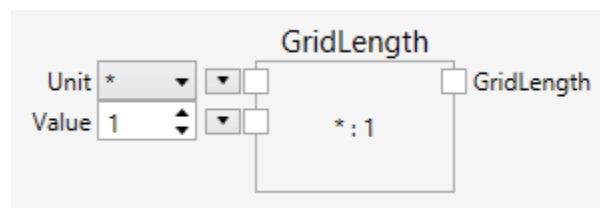
Value (Type: double)

The size of the row or column. The interpretation of the value is determined by the **Unit** as well.

If the **Unit** is `Auto`, the **Value** is ignored and the size is determined by the contents of the controls in the column or row.

If the **Unit** is `Absolute`, the **Value** is interpreted as a size given in pixel.

If the **Unit** is `*`, the **Value** is interpreted as a relative size. The remaining space in the grid, after all `Auto` and `Absolute` columns are taken is split in relation to the values given with `*`.

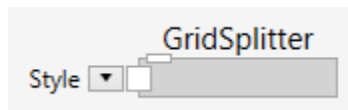


Comments

In order to use the **GridLength** to specify the settings for rows or columns, one or more **GridLength** nodes must be put in a list.

15.3.121 HMI GridSplitter

The **GridSplitter** allows you to enter a splitter inside a grid that enables interactive resize of the neighboring columns or rows.



Inputs

Style (Type: `style`)

Optional styling of the **GridSplitter**.

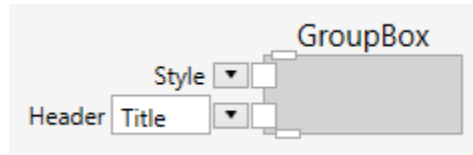
The **GridSplitter** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Background* styles.

Comments

The **GridSplitter** should get it's own grid cell(s), so that it is always visible.

15.3.122 HMI GroupBox

The **GroupBox** puts a group box around its child.



At the bottom of the **GroupBox** node is a pin that allows it to connect one child.

Inputs

Style (Type: `Style`)

Optional styling of the **GroupBox**.

The **GroupBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

Header (Type: `string`)

The header text is displayed at the top left corner of the **GroupBox**.

Example

Here is an example that puts a groupbox around three radio buttons. This definition:
creates the following user interface:

15.3.123 Hipass

Filters an image using a highpass kernel of size 3x3 or 5x5.

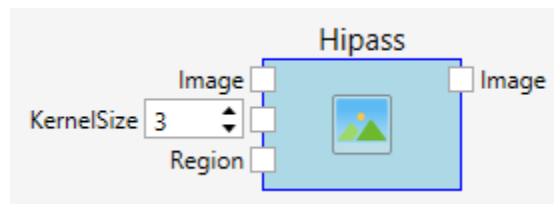
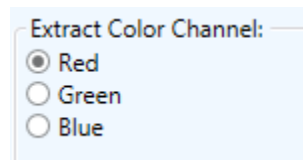
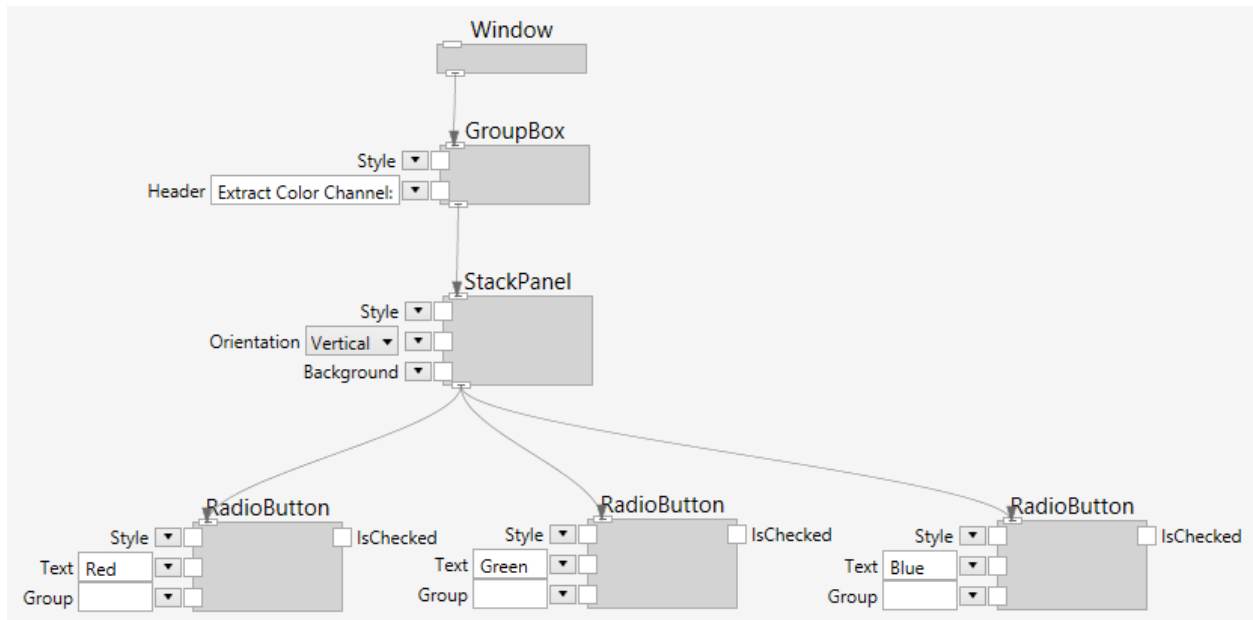
Inputs

Image (Type: `Image`)

The input image.

KernelSize (Type: `Int32`)

The kernel size (3 -> 3x3, 5 -> 5x5).



Region (Type: Region)

Specifies an optional area of interest.

Outputs**Image (Type: Image)**

The output image.

Comments

The function applies a hipass filter. The corresponding kernel is either a 3x3 matrix with the following values:

<code>::</code>

`-1 -1 -1 -1 8 -1 -1 -1 -1`

or a 5x5 matrix with the following values:

<code>-1 -1 -1 -1 -1</code>
<code>-1 -1 -1 -1 -1</code>
<code>-1 -1 24 -1 -1</code>
<code>-1 -1 -1 -1 -1</code>
<code>-1 -1 -1 -1 -1</code>

The effect of a hipass filter is that it amplifies high frequencies and attenuates low frequencies. The strength of the hiwpass filter depends on the size of the kernel.

Here are a few results of the hipass filter with increasing kernel sizes:

Original:

KernelSize = 3:

KernelSize = 5:

Sample

Here is an example that shows how to use the hipass filter.

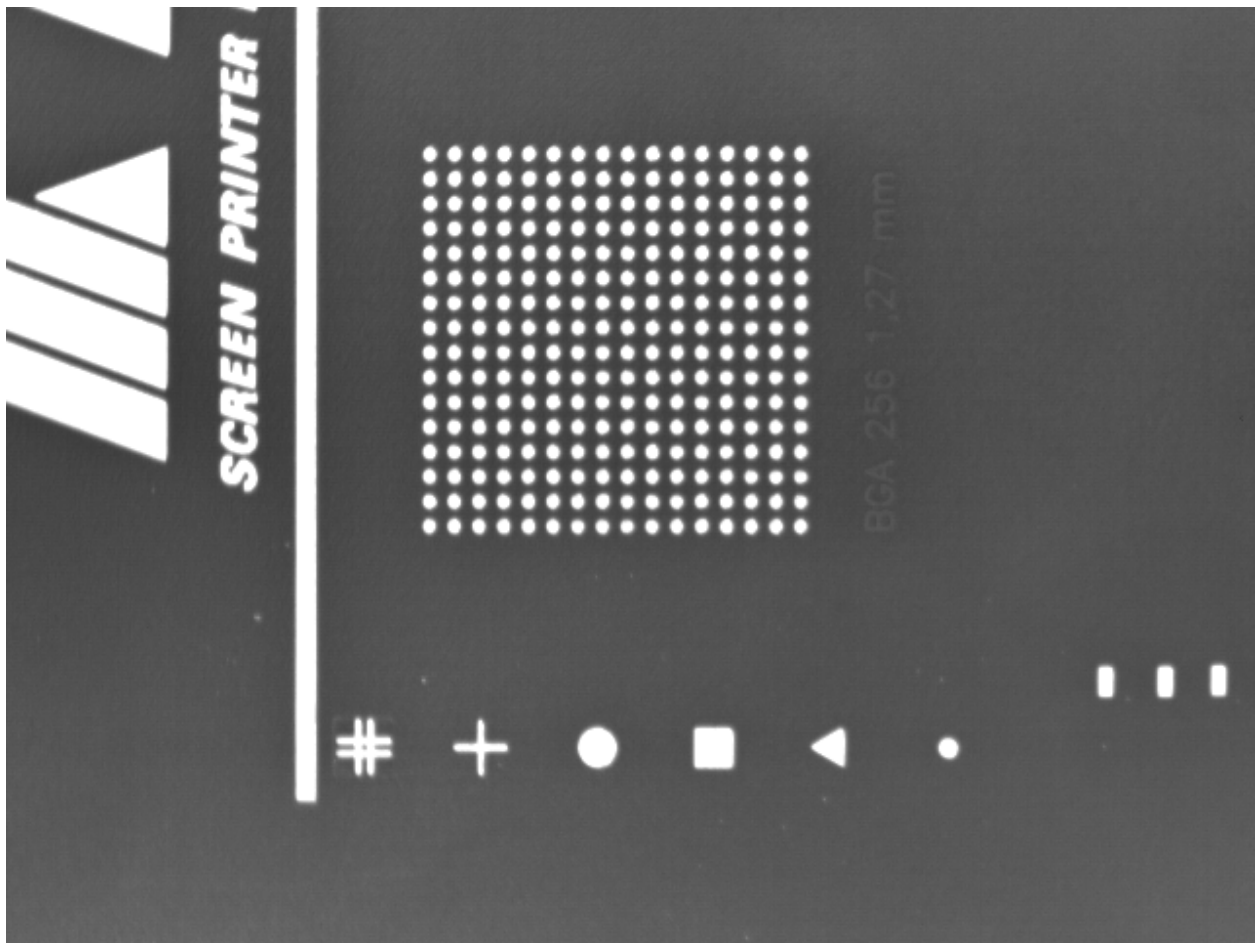
15.3.124 Histogram

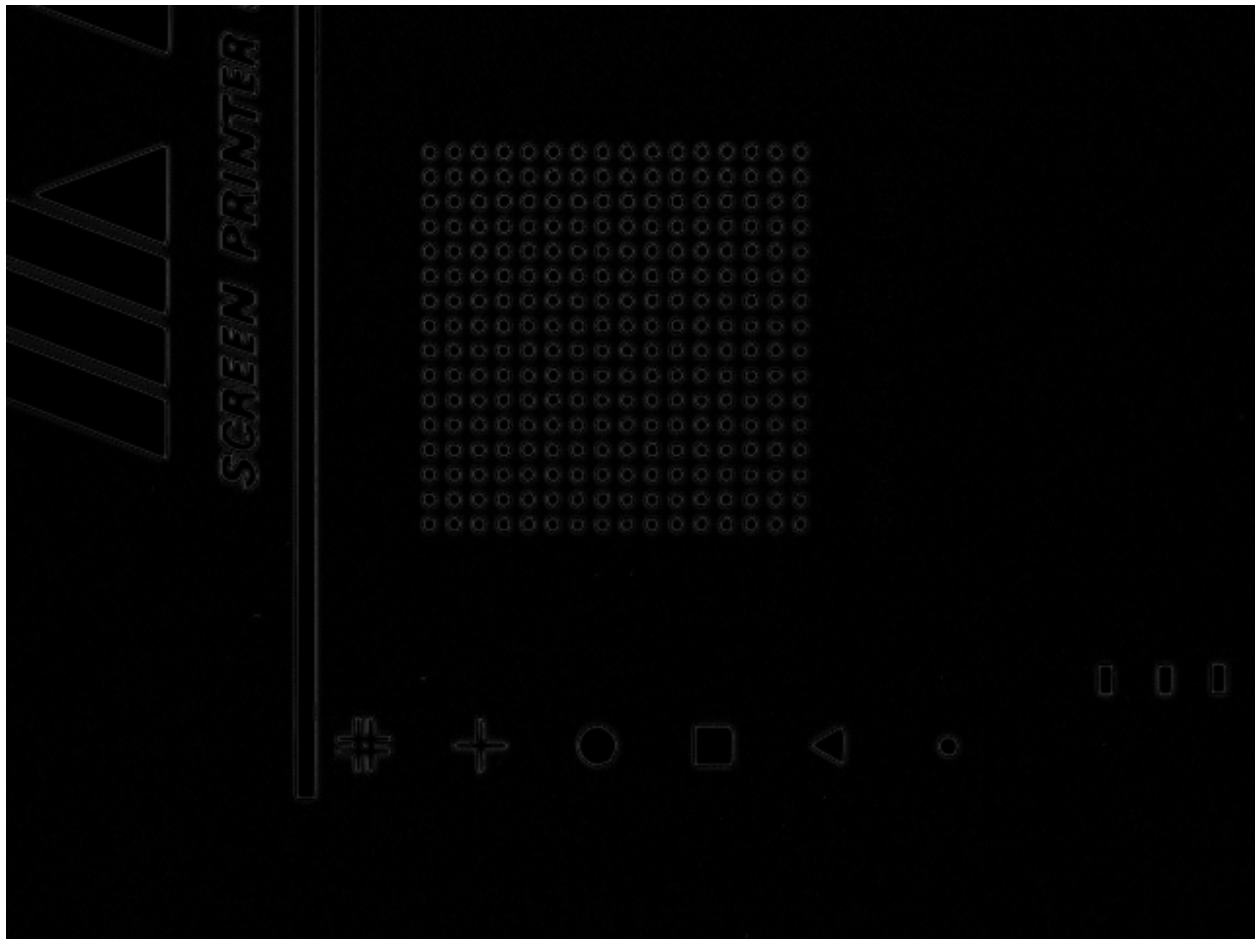
Calculates a histogram of an image to show the greyscale/color distribution.

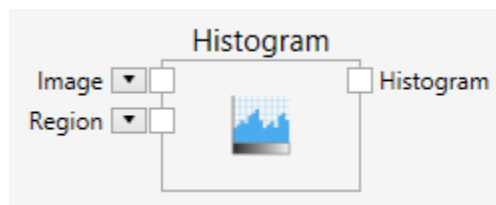
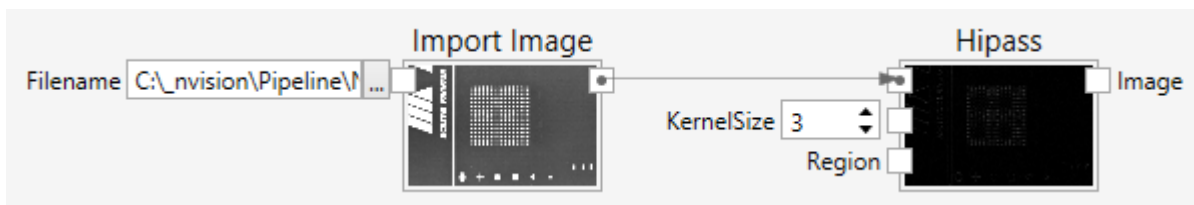
The Histogram node can be followed by several other nodes to yield statistic information about an image or find an optimal threshold (Otsu).

Inputs**Image (Type: Image)**

The input image.







Region (Type: Region)

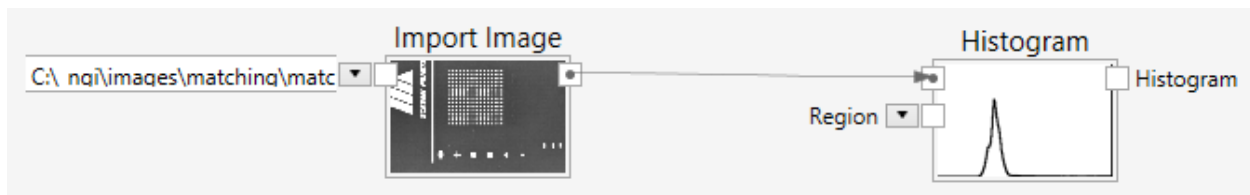
An optional region of interest. If this parameter is connected, the histogram operation is constrained to pixels within the region only.

Outputs**Histogram (Type: Histogram)**

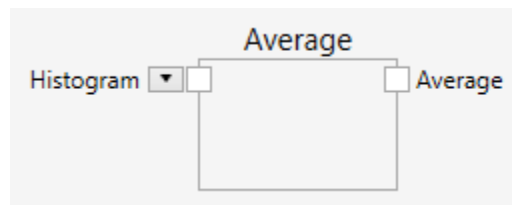
The histogram.

Sample

Here is an example:

**15.3.125 Histogram Average**

Calculates the average value of a buffer, given a histogram of the buffer.

**Inputs****Histogram (Type: Histogram)**

The input histogram.

Outputs**Average (Type: Object)**

The average value.

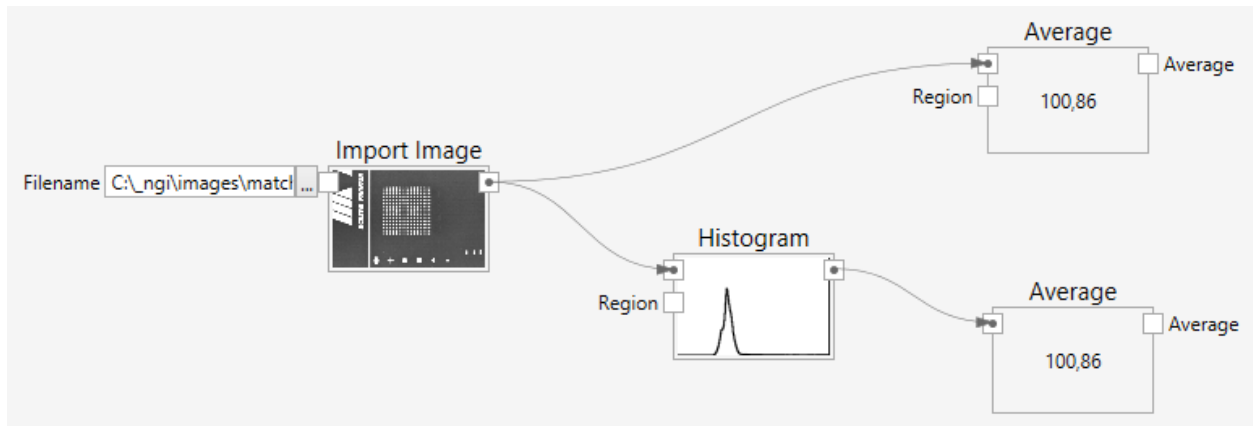
Comments

The **Histogram Average** node calculates the average value of a buffer, given a histogram to it.

If a color buffer is used, the average is calculated channel-wise.

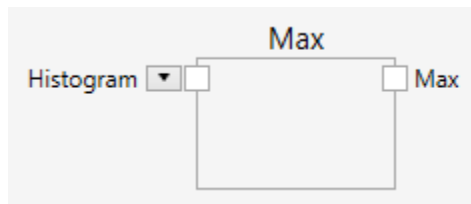
Sample

Here is an example that calculates the average of histogram.



15.3.126 Histogram Maximum

Calculates the maximum value of a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Maximum (Type: Object)

The maximum value.

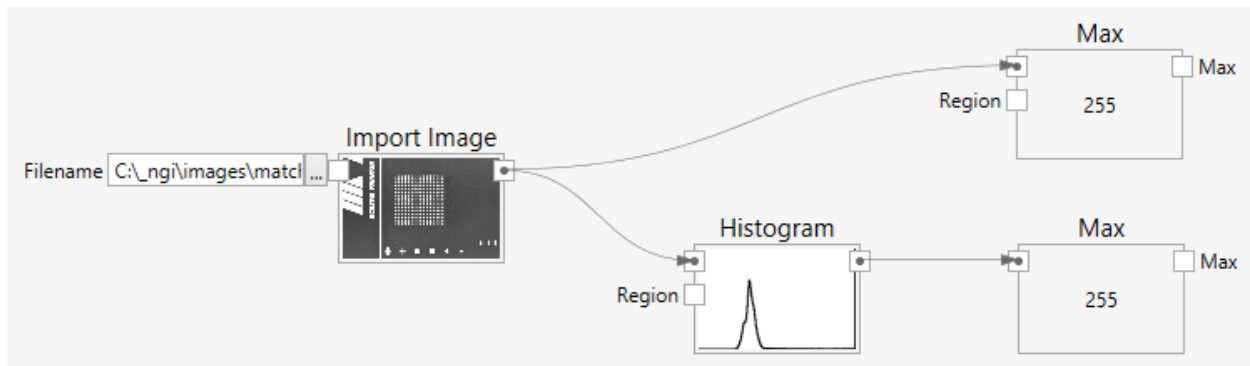
Comments

The **Histogram Maximum** node calculates the maximum value of a buffer, given a histogram to it.

If a color buffer is used, the maximum is calculated channel-wise.

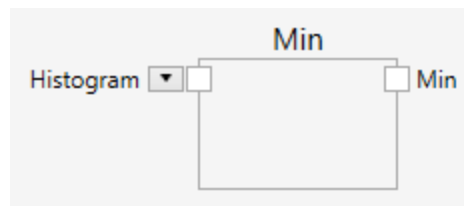
Sample

Here is an example that calculates the maximum of a histogram.



15.3.127 Histogram Minimum

Calculates the minimum value of a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Minimum (Type: Object)

The minimum value.

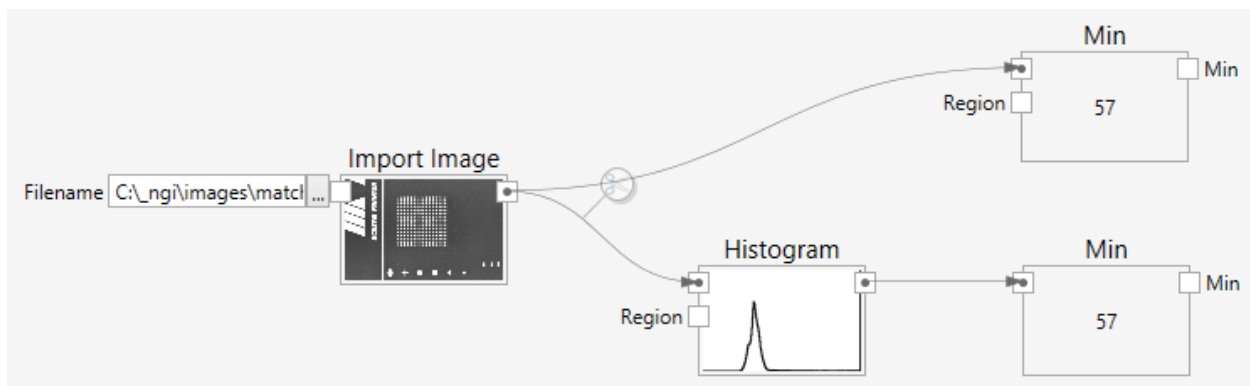
Comments

The **Histogram Minimum** node calculates the minimum value of a buffer, given a histogram to it.

If a color buffer is used, the minimum is calculated channel-wise.

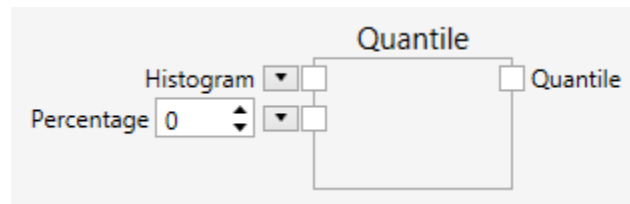
Sample

Here is an example that calculates the minimum of a histogram.



15.3.128 Histogram Quantile

Calculates the quantile of a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Percentage (Type: Double)

Specifies the quantile.

Outputs

Quantile (Type: Object)

The quantile value.

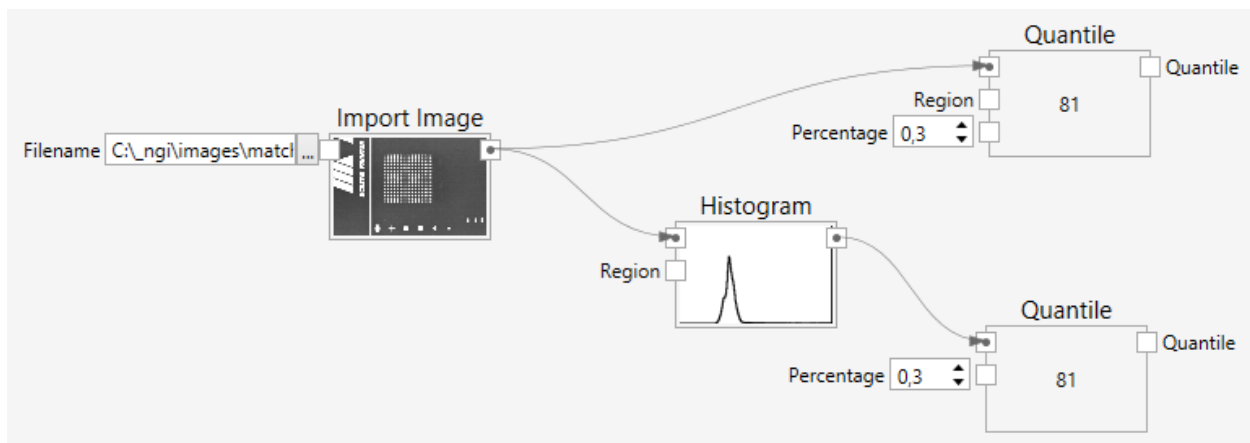
Comments

The **Histogram Quantile** node calculates a quantile of a buffer, given a histogram to it.

If a color buffer is used, the quantile is calculated channel-wise.

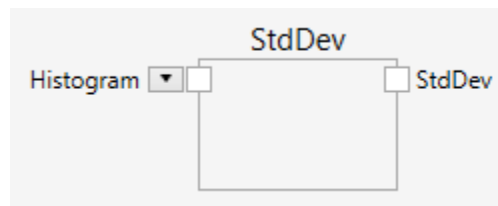
Sample

Here is an example that calculates the 50% quantile of a histogram.



15.3.129 Histogram StdDev

Calculates the standard deviation of all pixels in a buffer, given a histogram



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

StdDev (Type: Object)

The standard deviation.

Comments

The **Histogram StdDev** node calculates the standard deviation of all pixel values of a buffer, given a histogram to it.

If a color buffer is used, the standard deviation is calculated channel-wise.

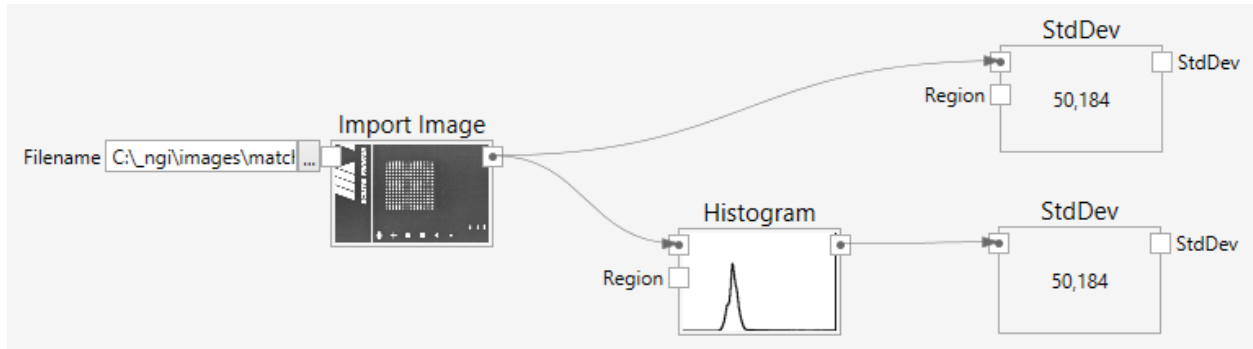
The standard deviation is a measure of statistical dispersion, averaging the squared distance of values from the mean value. The standard deviation is a measure of how much a statistical distribution is spread out. The standard deviation is the square root of the variance.

The standard deviation is calculated according to this formula:

$$\sigma = \sqrt{\frac{1}{n} \sum x^2 - \mu^2}$$

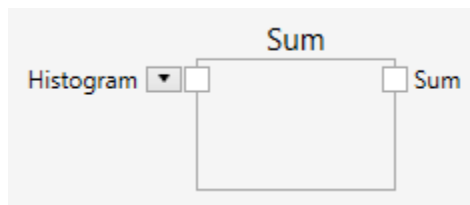
Sample

Here is an example that calculates the standard deviation of a histogram.



15.3.130 Histogram Sum

Calculates the sum of all pixels in a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Sum (Type: Object)

The sum.

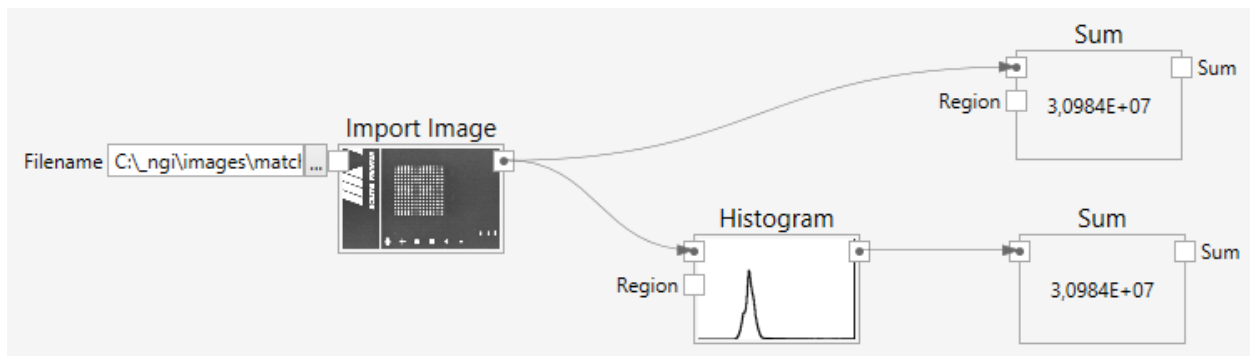
Comments

The **Histogram Sum** node calculates the sum of all pixel values of a buffer, given a histogram to it.

If a color buffer is used, the sum is calculated channel-wise.

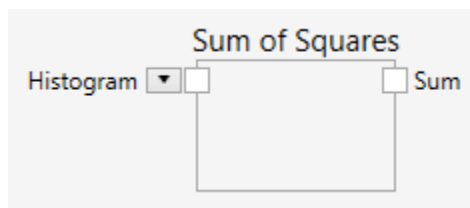
Sample

Here is an example that calculates the sum of a histogram.



15.3.131 Histogram Sum of Squares

Calculates the sum of squares of all pixels in a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Sum (Type: Object)

The sum of squares.

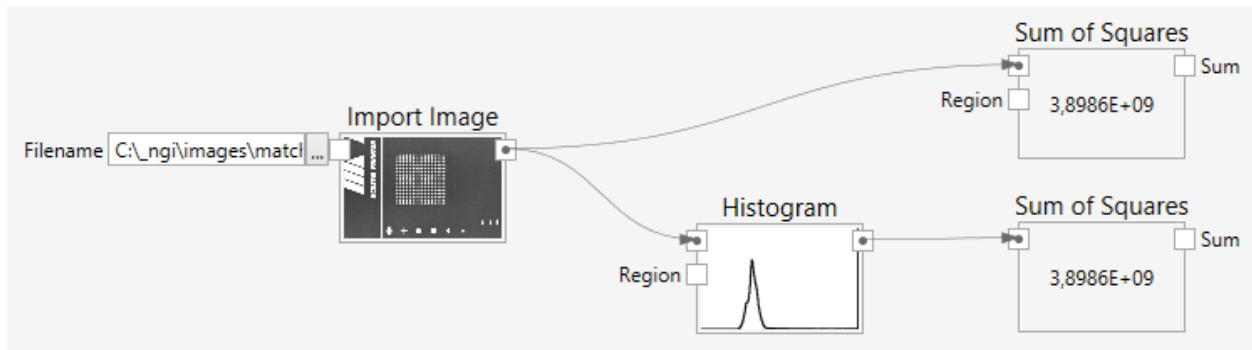
Comments

The **Histogram Sum of Squares** node calculates the sum of squares of all pixel values of a buffer, given a histogram to it.

If a color buffer is used, the sum of squares is calculated channel-wise.

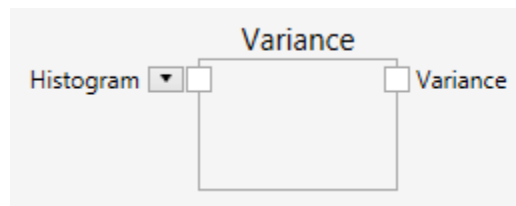
Sample

Here is an example that calculates the sum of squares of a histogram.



15.3.132 Histogram Variance

Calculates the variance of all pixels in a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Variance (Type: Object)

The variance.

Comments

The **Histogram Variance** node calculates the variance of all pixel values of a buffer, given a histogram to it.

If a color buffer is used, the variance is calculated channel-wise.

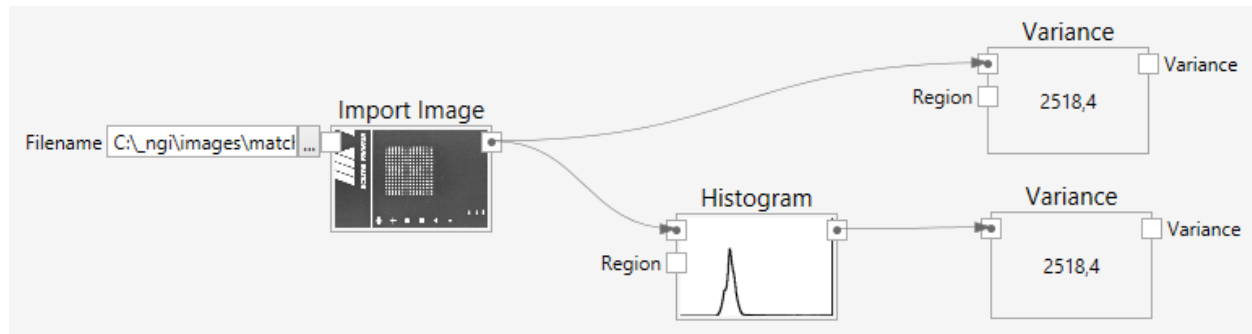
The variance is a measure of statistical dispersion, averaging the squared distance of values from the mean value. The variance is a measure of how much a statistical distribution is spread out.

The variance is calculated according to this formula:

$$\sigma^2 = \frac{1}{n} \sum x^2 - \mu^2$$

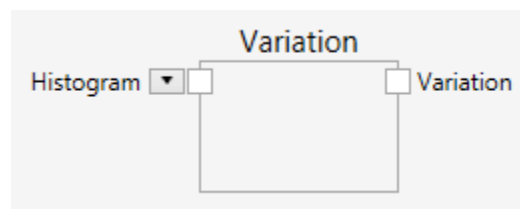
Sample

Here is an example that calculates the variance of a histogram.



15.3.133 HistogramVariation

Calculates the coefficient of variation of all pixels in a buffer, given a histogram.



Inputs

Histogram (Type: Histogram)

The input histogram.

Outputs

Variation (Type: Object)

The coefficient of variation.

Comments

The **HistogramVariation** node calculates the coefficient of variation of all pixel values of a buffer, given a histogram

If a color buffer is used, the coefficient of variation is calculated channel-wise.

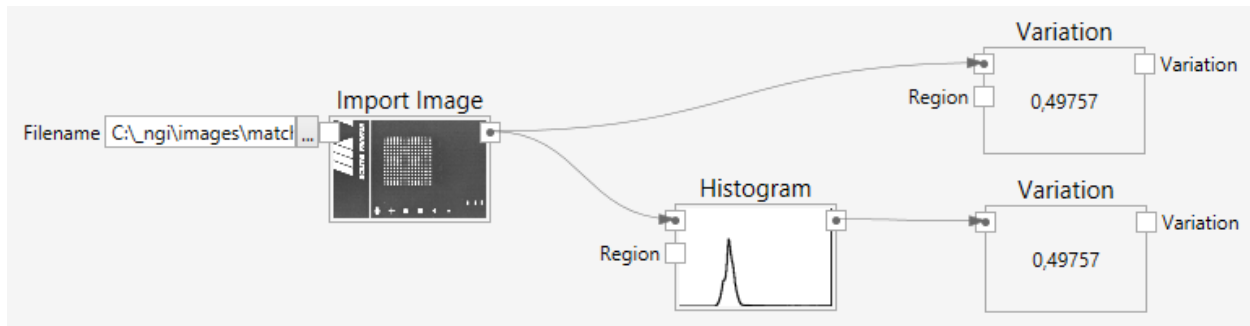
The coefficient of variation is the standard deviation divided by the average.

The coefficient of variation is calculated according to this formula:

$$c_v = \frac{\sigma}{\mu}$$

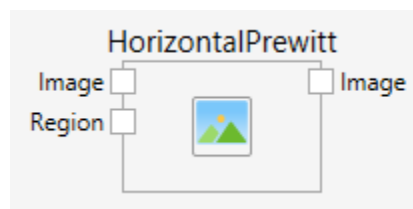
Sample

Here is an example that calculates the coefficient of variation of a histogram:



15.3.134 Horizontal Prewitt

Filters an image using a horizontal prewitt kernel.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a horizontal prewitt filter. The corresponding kernel is a 3x3 matrix with the following values:

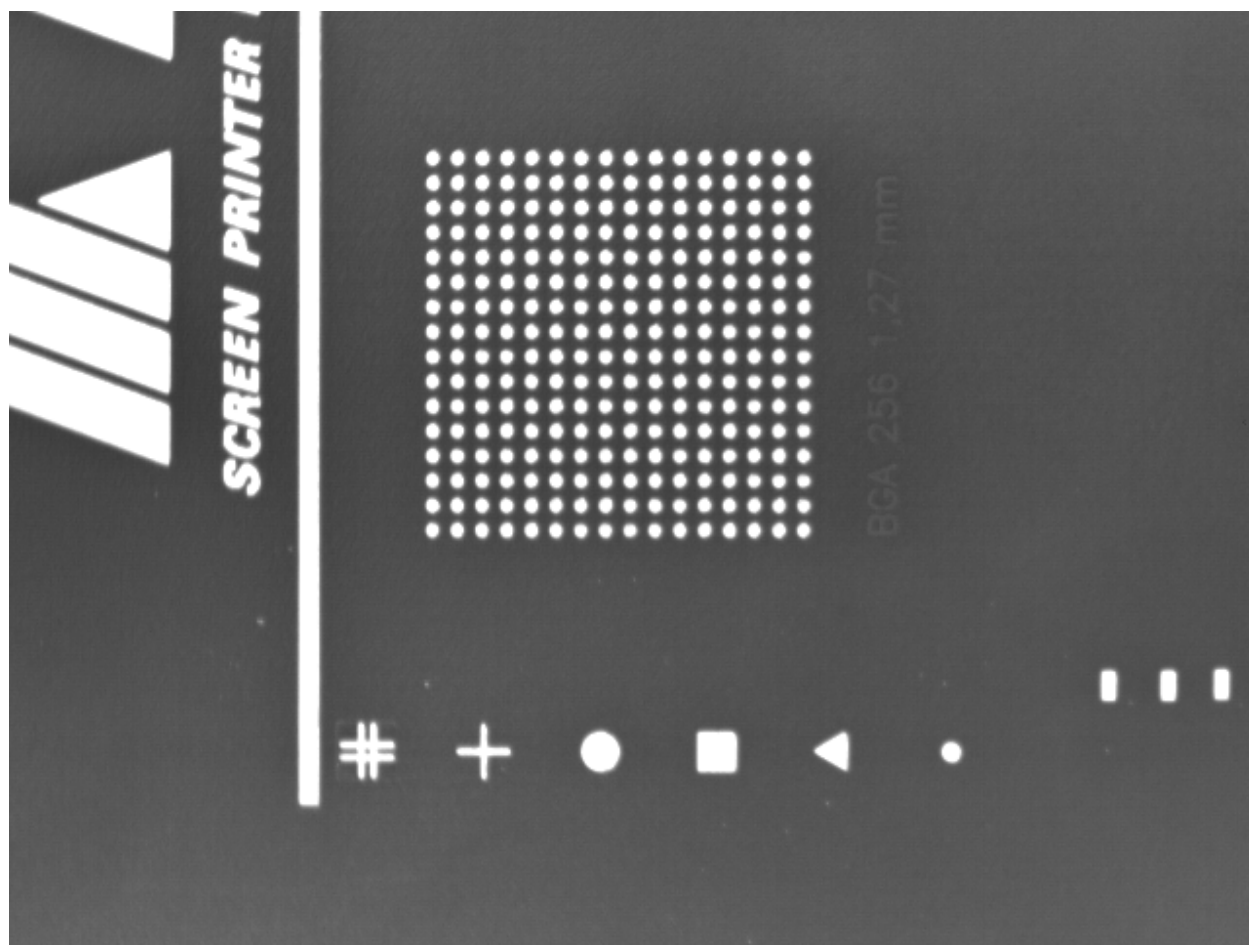
```

1  1  1
0  0  0
-1 -1 -1

```

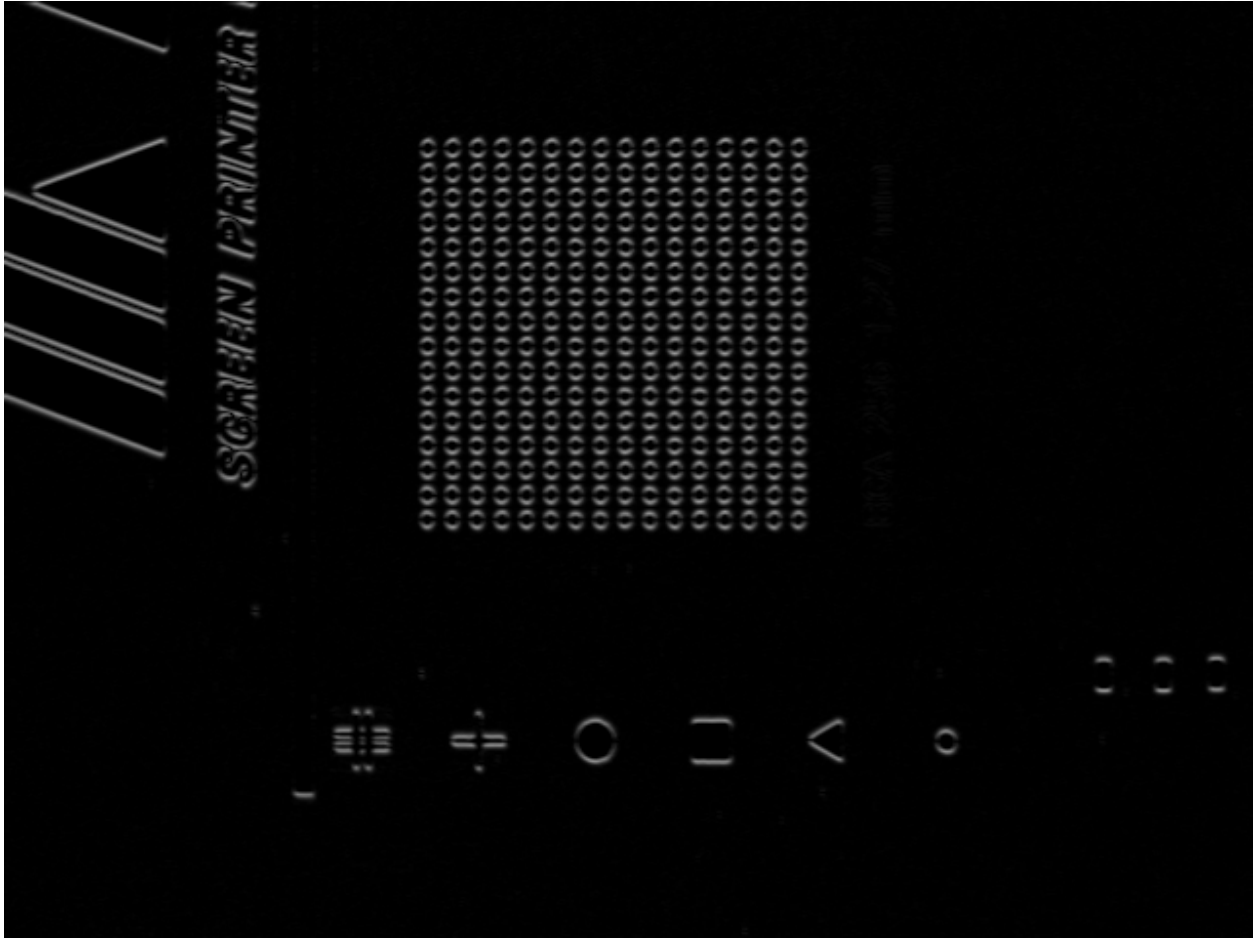
The filter attenuates horizontal edges while at the same time smoothing in the horizontal direction.

Here are a few results of the horizontal prewitt filter with increasing kernel sizes:



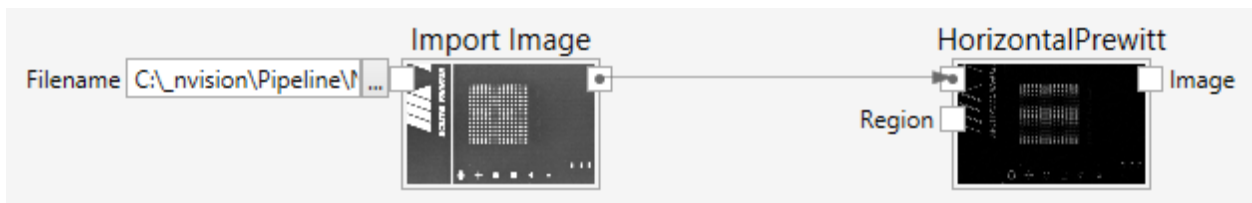
Original:

Result:



Sample

Here is an example that shows how to use the horizontal prewitt filter.

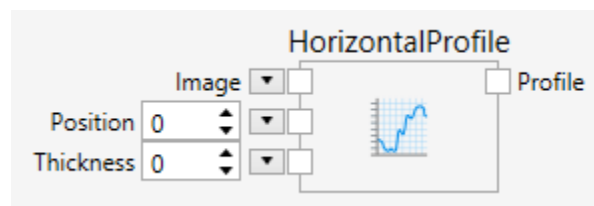
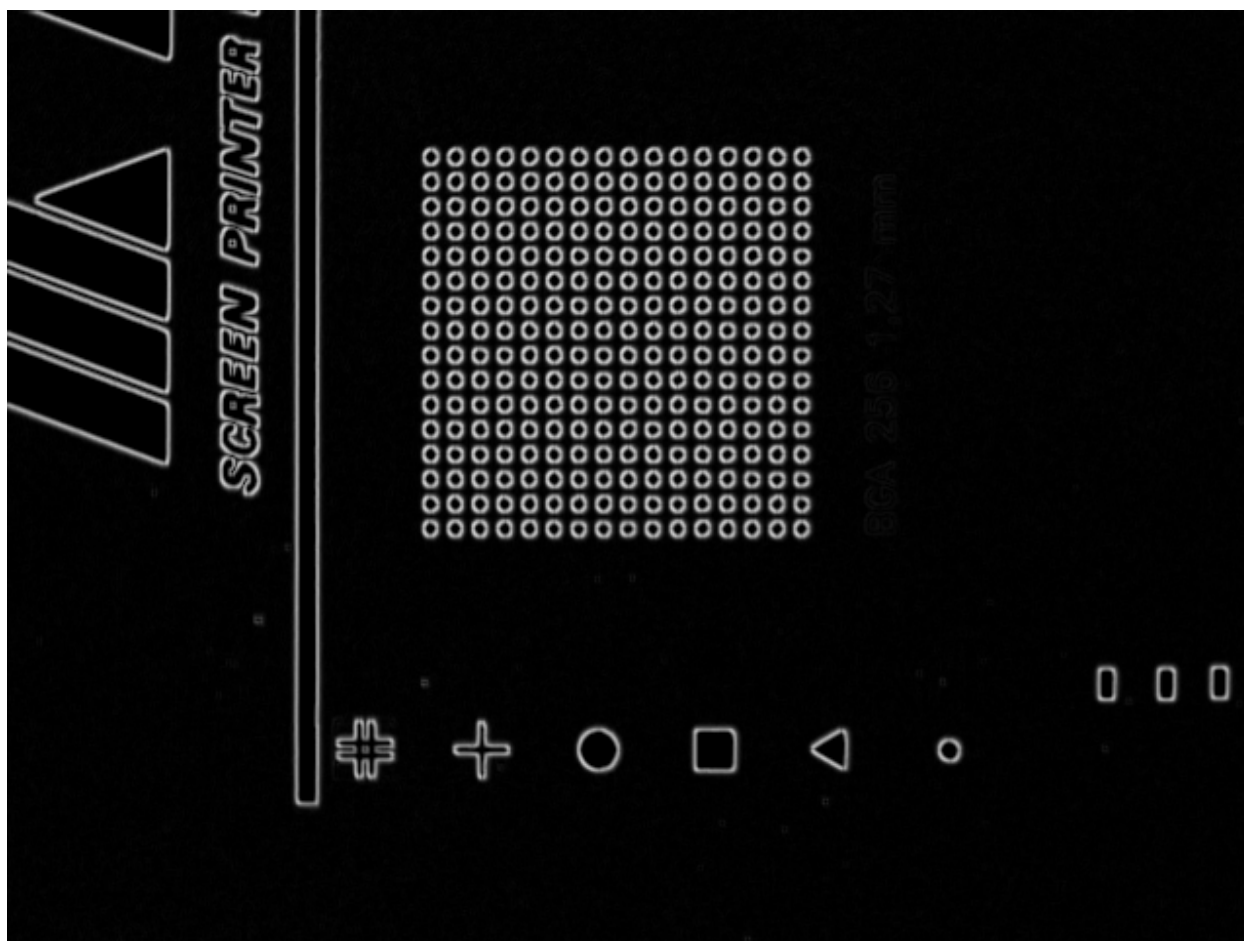
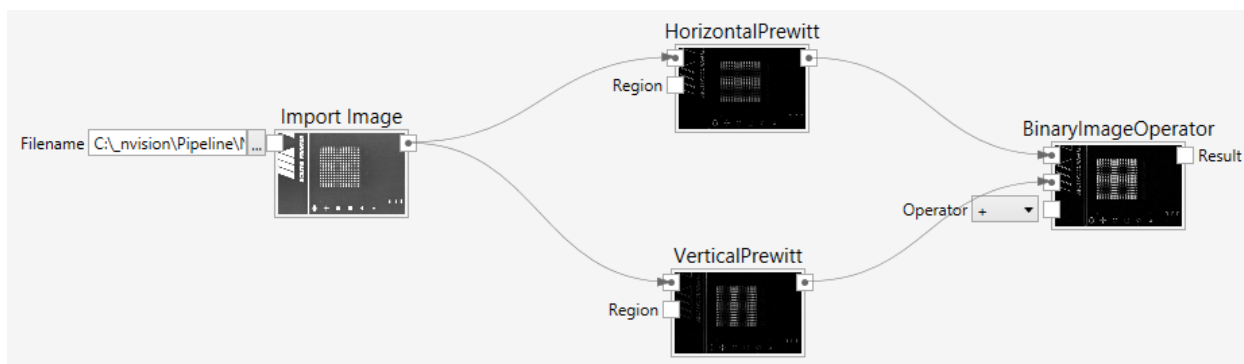


Often, you want to combine the horizontal and the vertical prewitt filter to create an edge map. A simple way to do this is shown in the following sample:

Here is an example image of such an edge map:

15.3.135 Horizontal Profile

Calculates a horizontal profile of an image to show the greyscale/color distribution.



The Horizontal Profile node gives you information about the greyscale/color distribution along a horizontal line. The line can have a certain thickness.

Inputs

Image (Type: Image)

The input image.

Position (Type: Int32)

The vertical coordinate of the horizontal line.

Thickness (Type: Int32)

The thickness in pixel of the horizontal line.

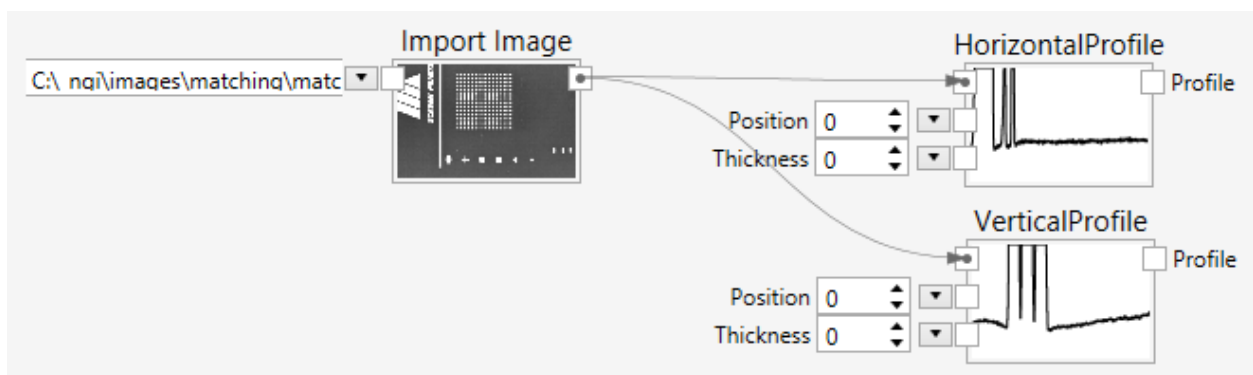
Outputs

Profile (Type: Profile)

The profile.

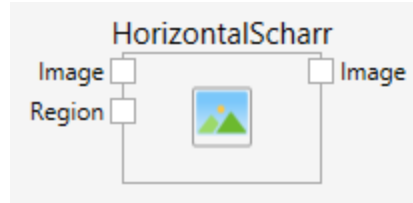
Sample

Here is an example:



15.3.136 Horizontal Scharr

Filters an image using a horizontal scharr kernel.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a horizontal prewitt filter. The corresponding kernel is a 3x3 matrix with the following values:

```
3  10  3
0   0  0
-3 -10 -3
```

The filter attenuates horizontal edges while at the same time smoothing in the horizontal direction.

Here are a few results of the horizontal scharr filter with increasing kernel sizes:

Original:

Result:

Sample

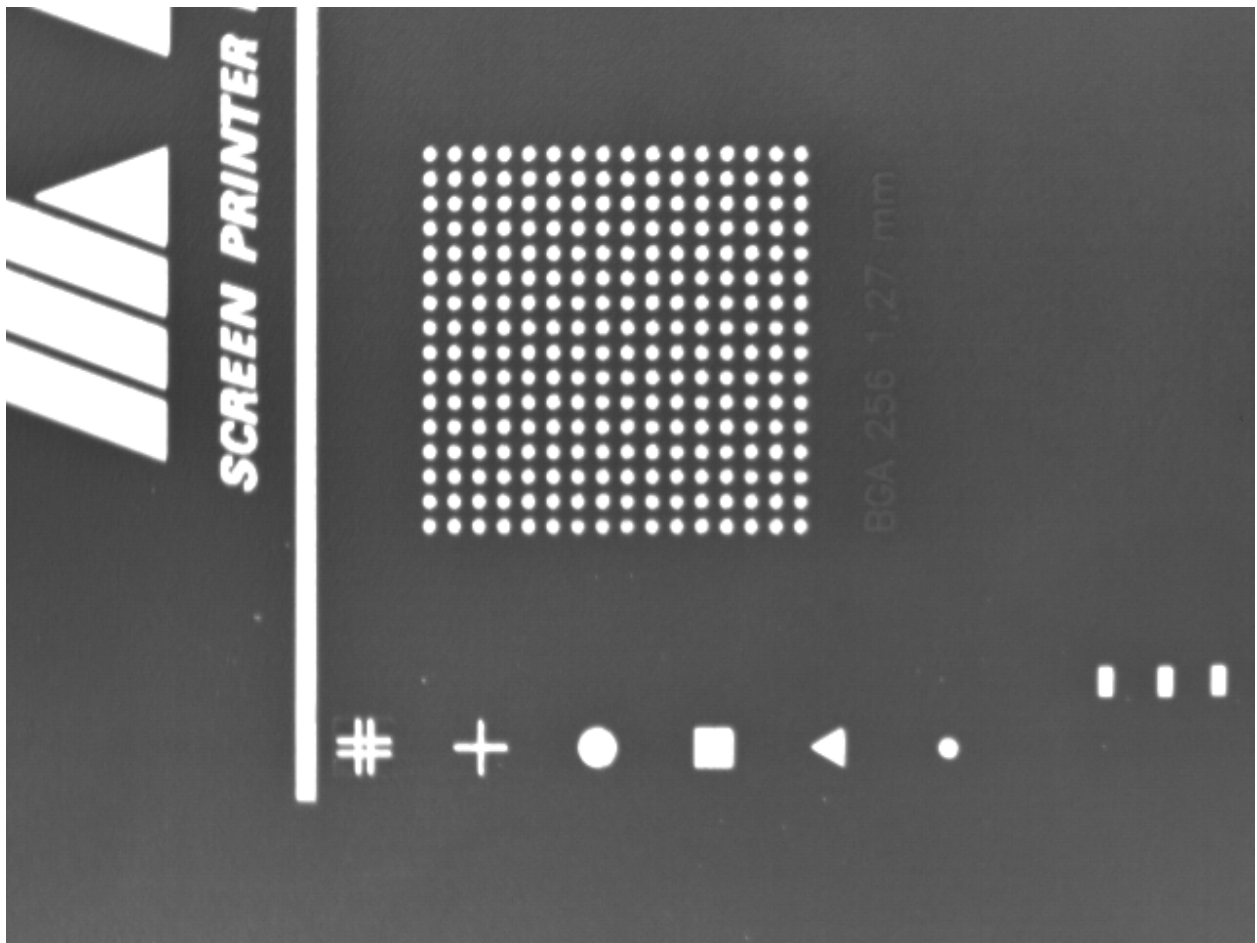
Here is an example that shows how to use the horizontal scharr filter.

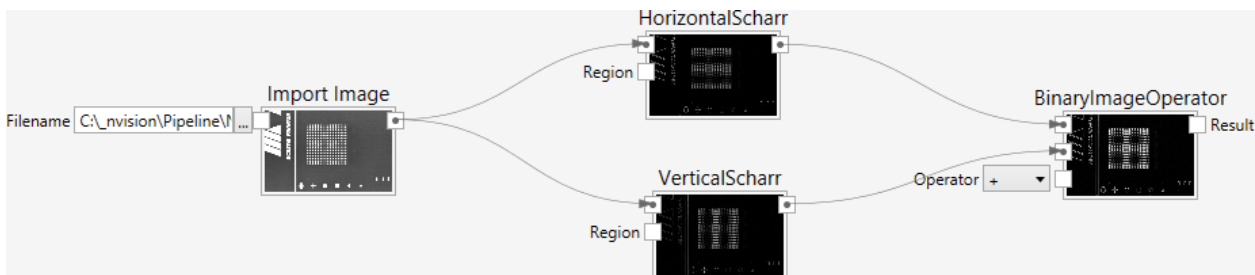
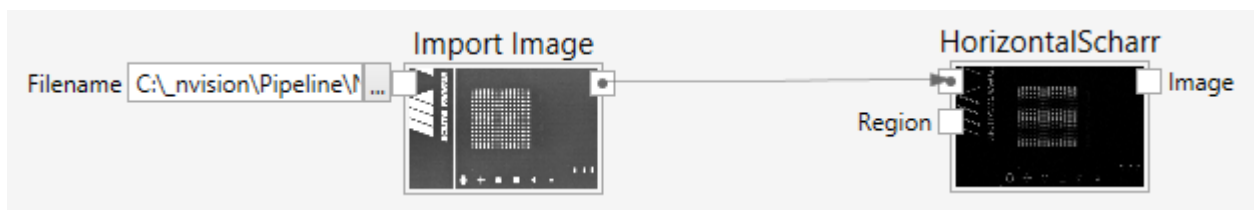
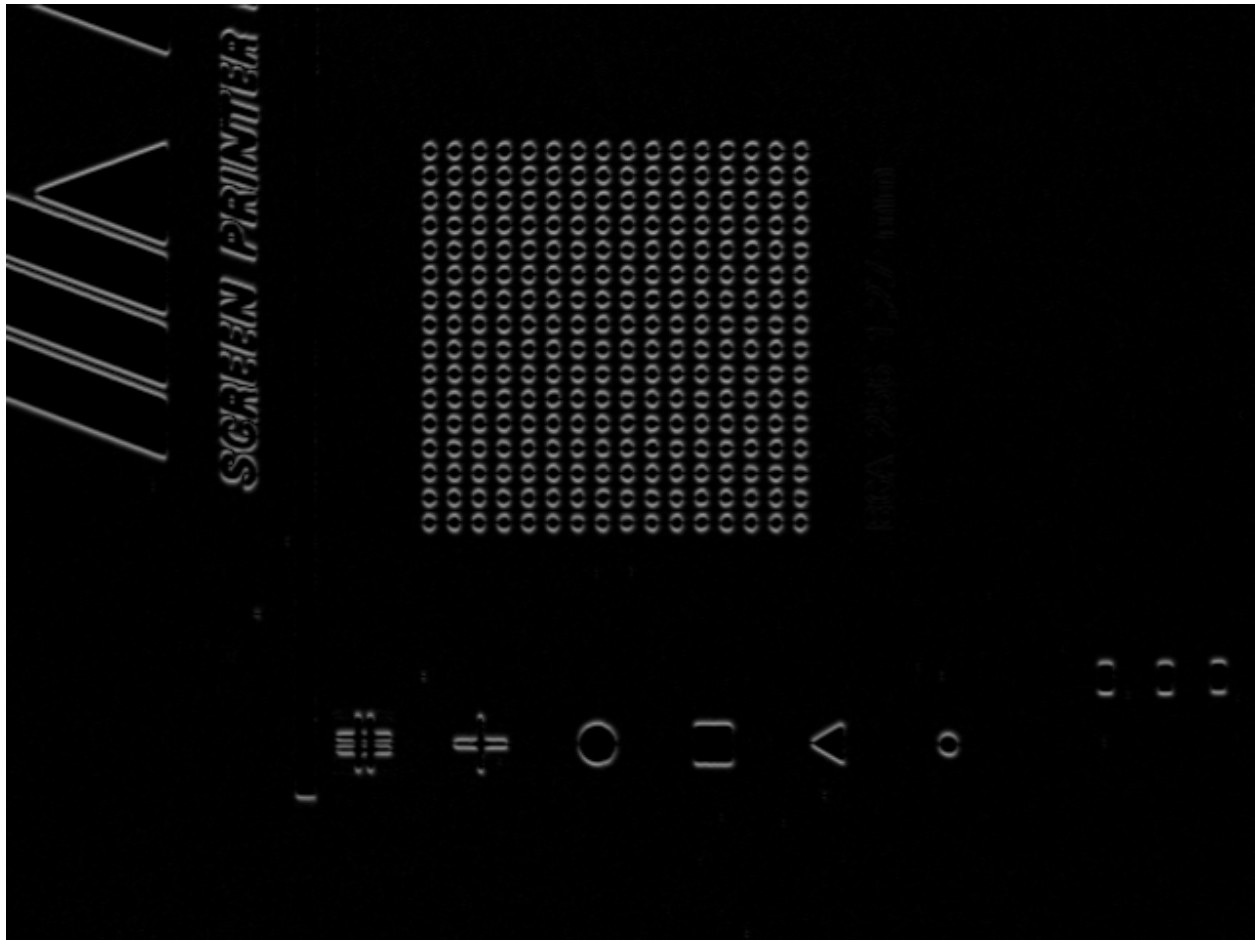
Often, you want to combine the horizontal and the vertical scharr filter to create an edge map. A simple way to do this is shown in the following sample:

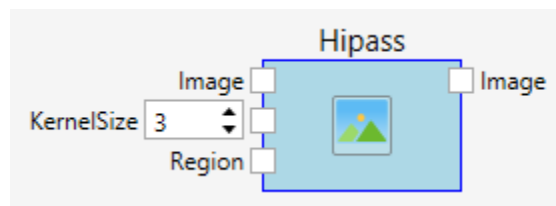
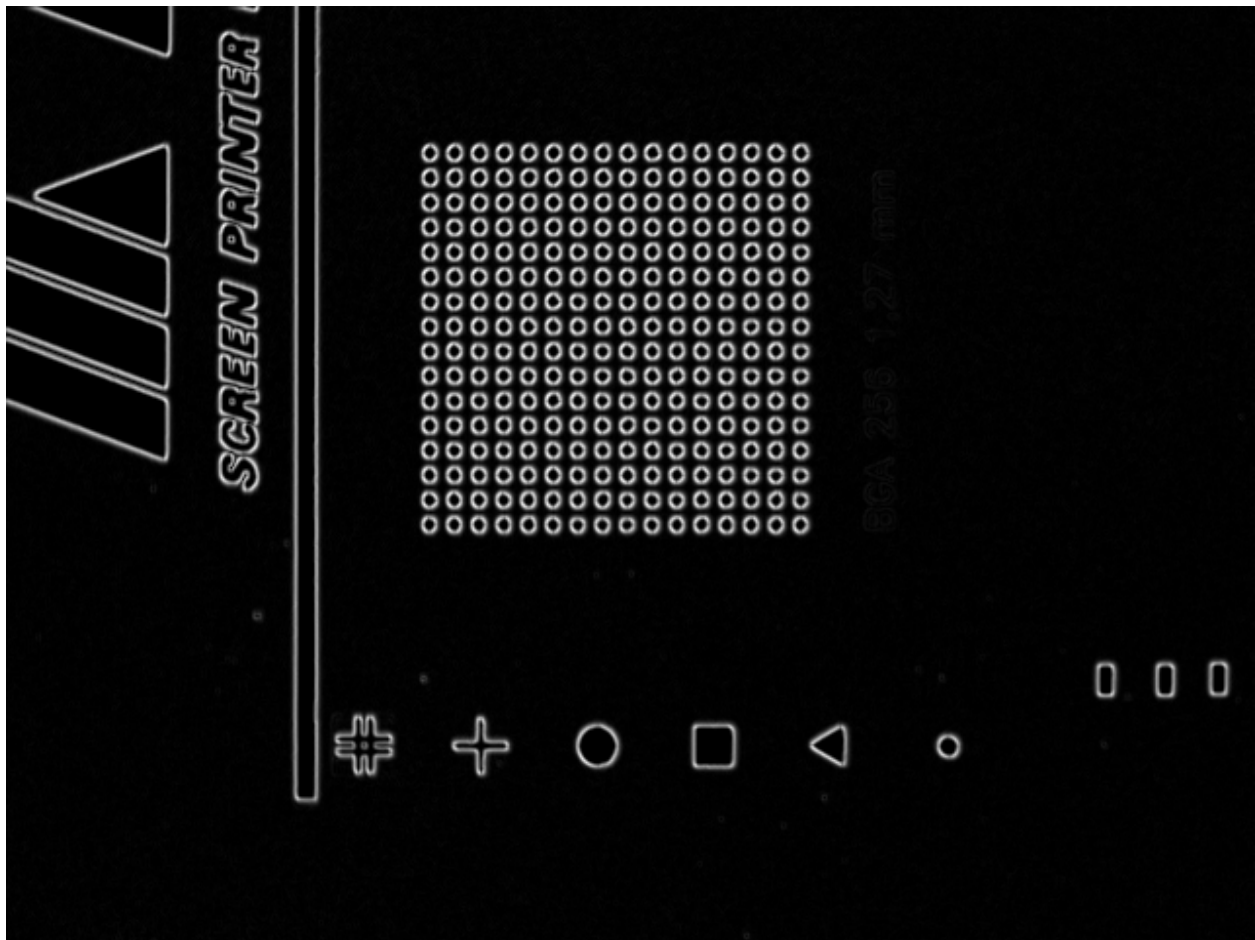
Here is an example image of such an edge map:

15.3.137 Horizontal Sobel

Filters an image using a horizontal sobel kernel of size 3x3 or 5x5.







Inputs

Image (Type: Image)

The input image.

KernelSize (Type: Int32)

The kernel size (3 -> 3x3, 5 -> 5x5).

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a horizontal_sobel filter. The corresponding kernel is either a 3x3 matrix with the following values:

```
1  2  1
0  0  0
-1 -2 -1
```

or a 5x5 kernel with the following values:

```
1  4  6  4  1
2  8 12  8  2
0  0  0  0  0
-2 -8 -12 -8 -2
-1 -4  -6 -4 -1
```

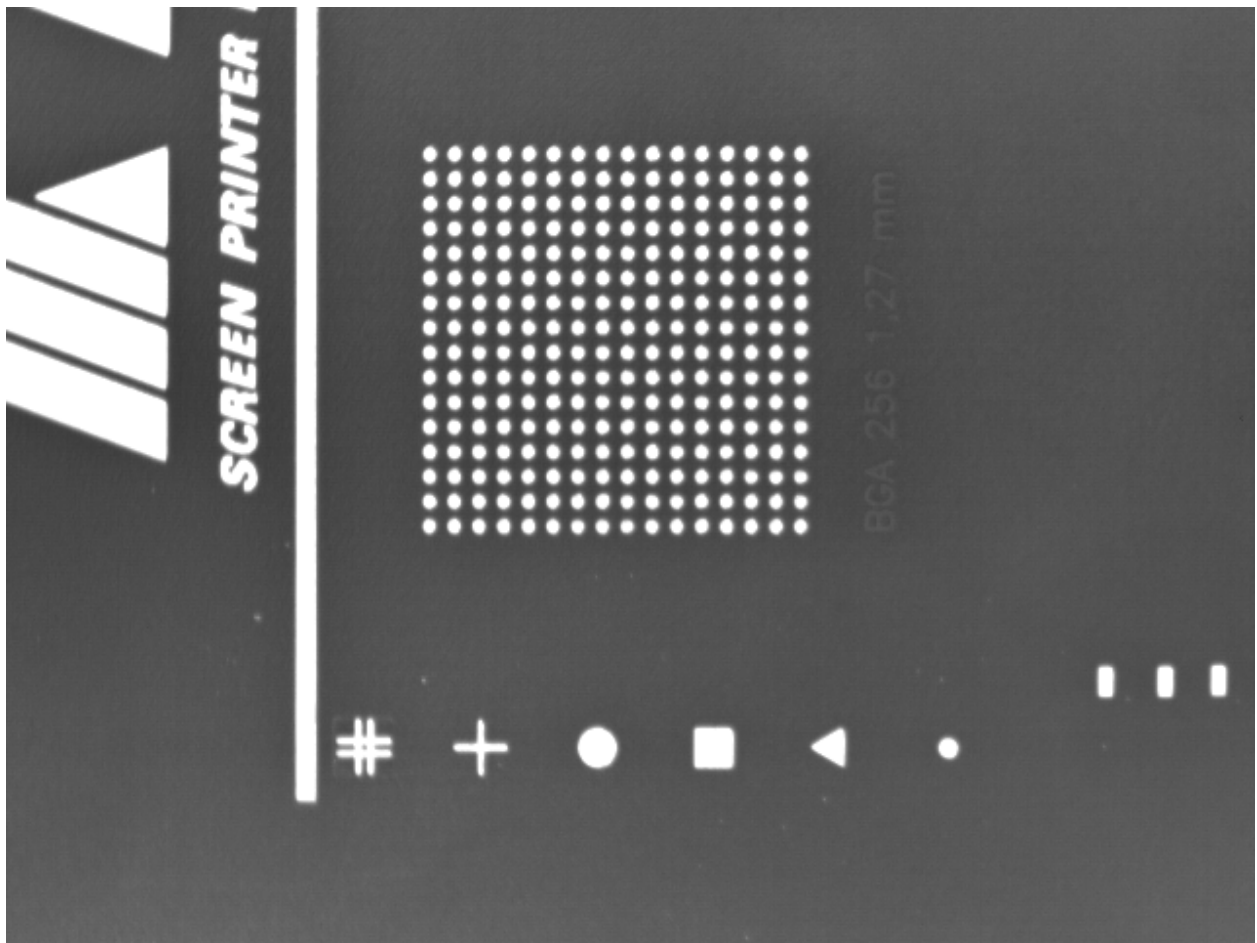
The effect of a horizontal sobel filter is that it amplifies horizontal edges and attenuates everything else. The strength of the sobel filter depends on the size of the kernel.

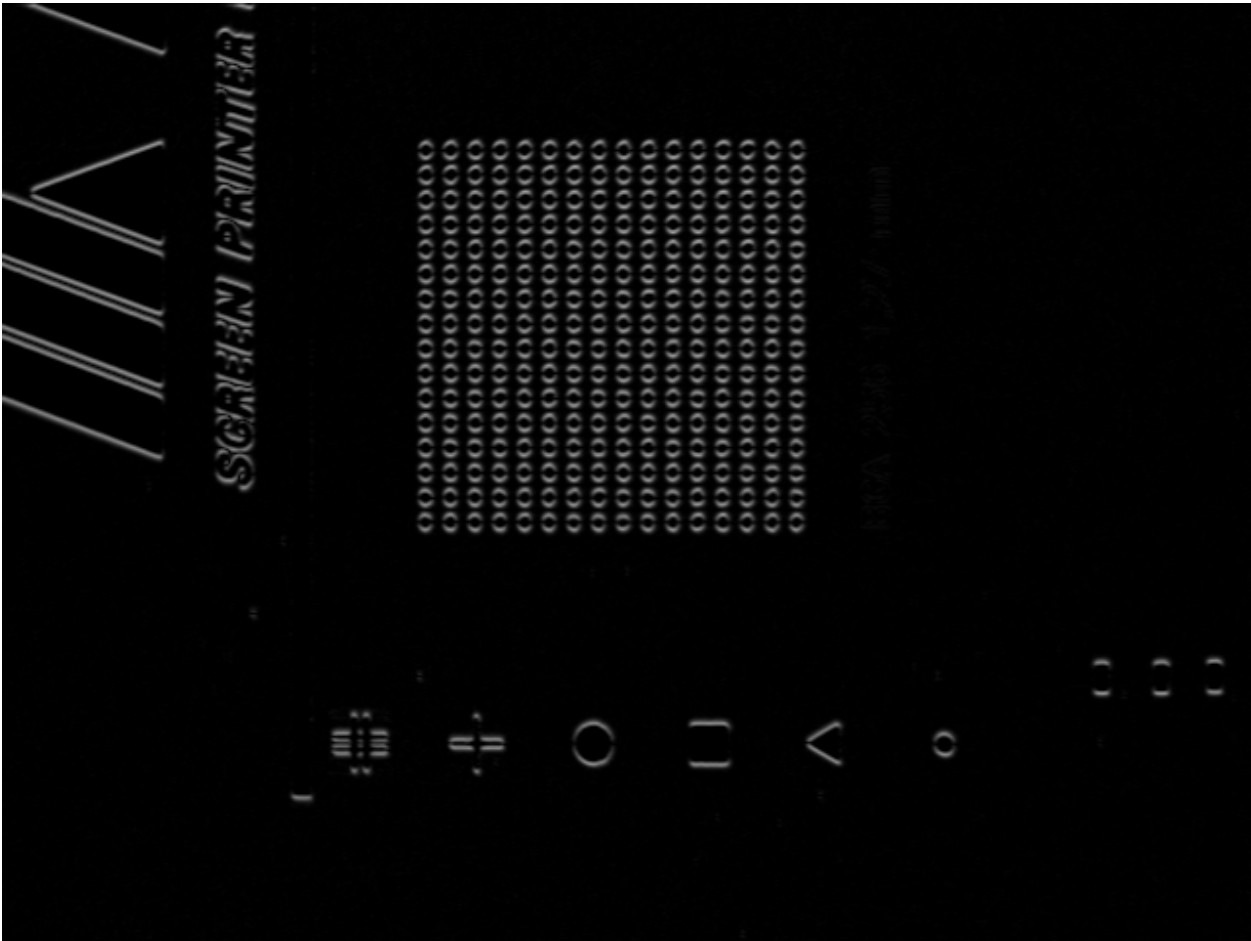
Here are a few results of the horizontal sobel filter with increasing kernel sizes:

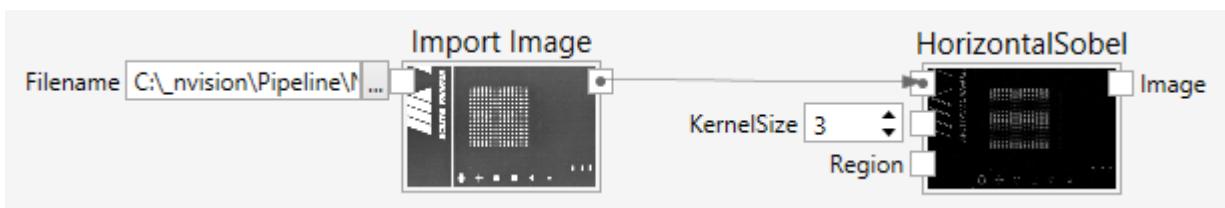
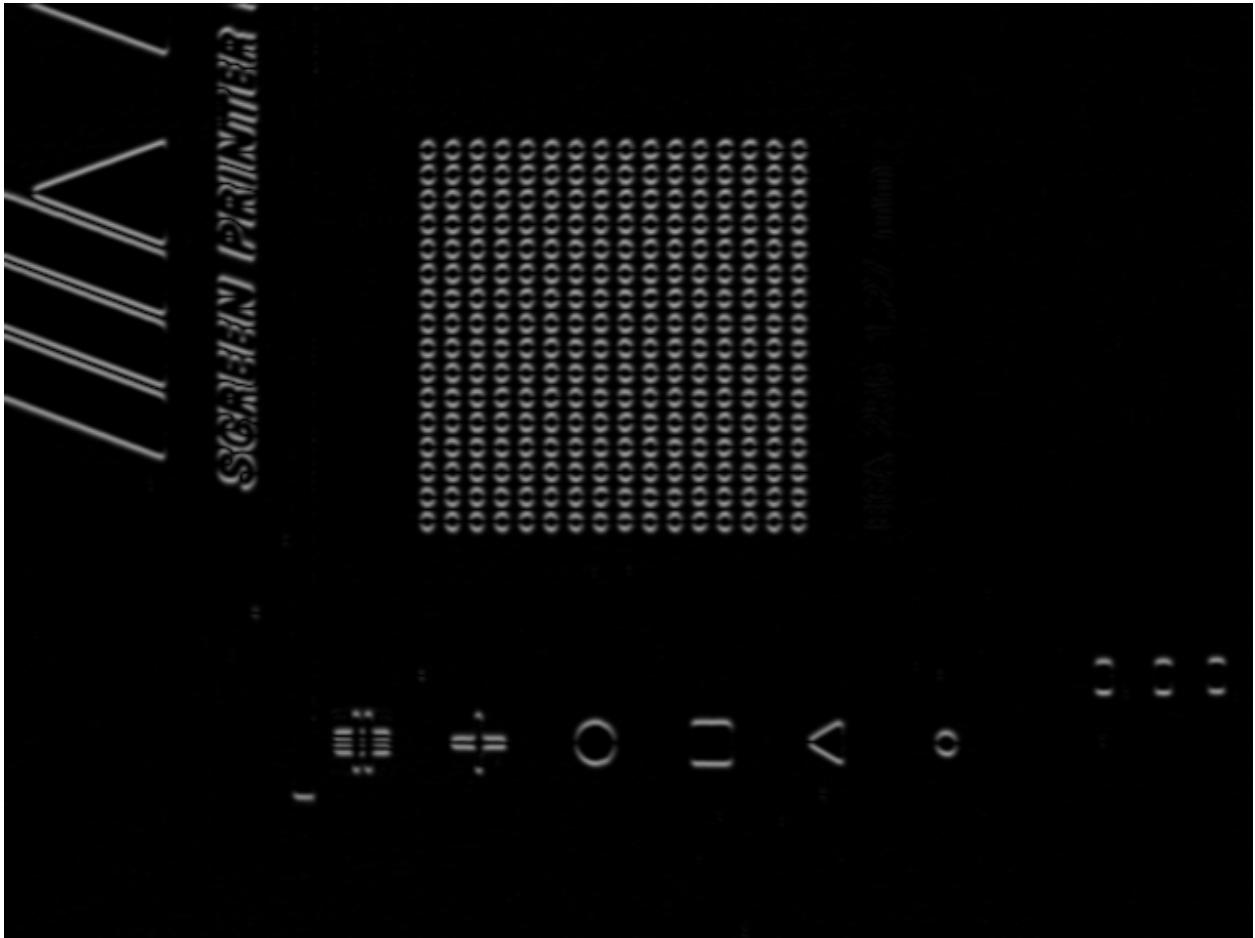
Original:

KernelSize = 3:

KernelSize = 5:



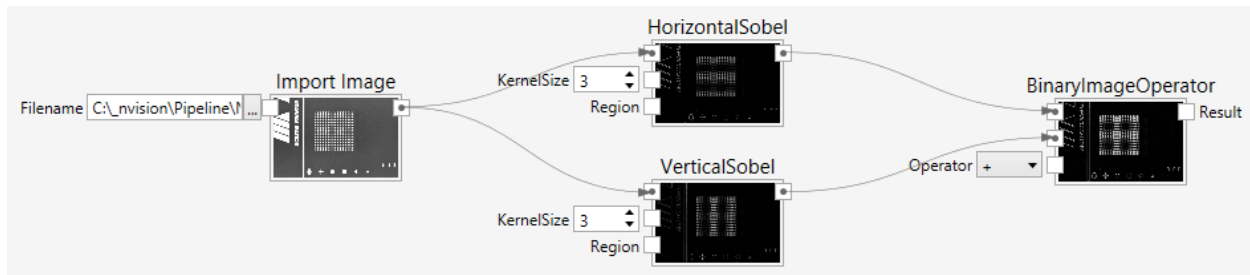




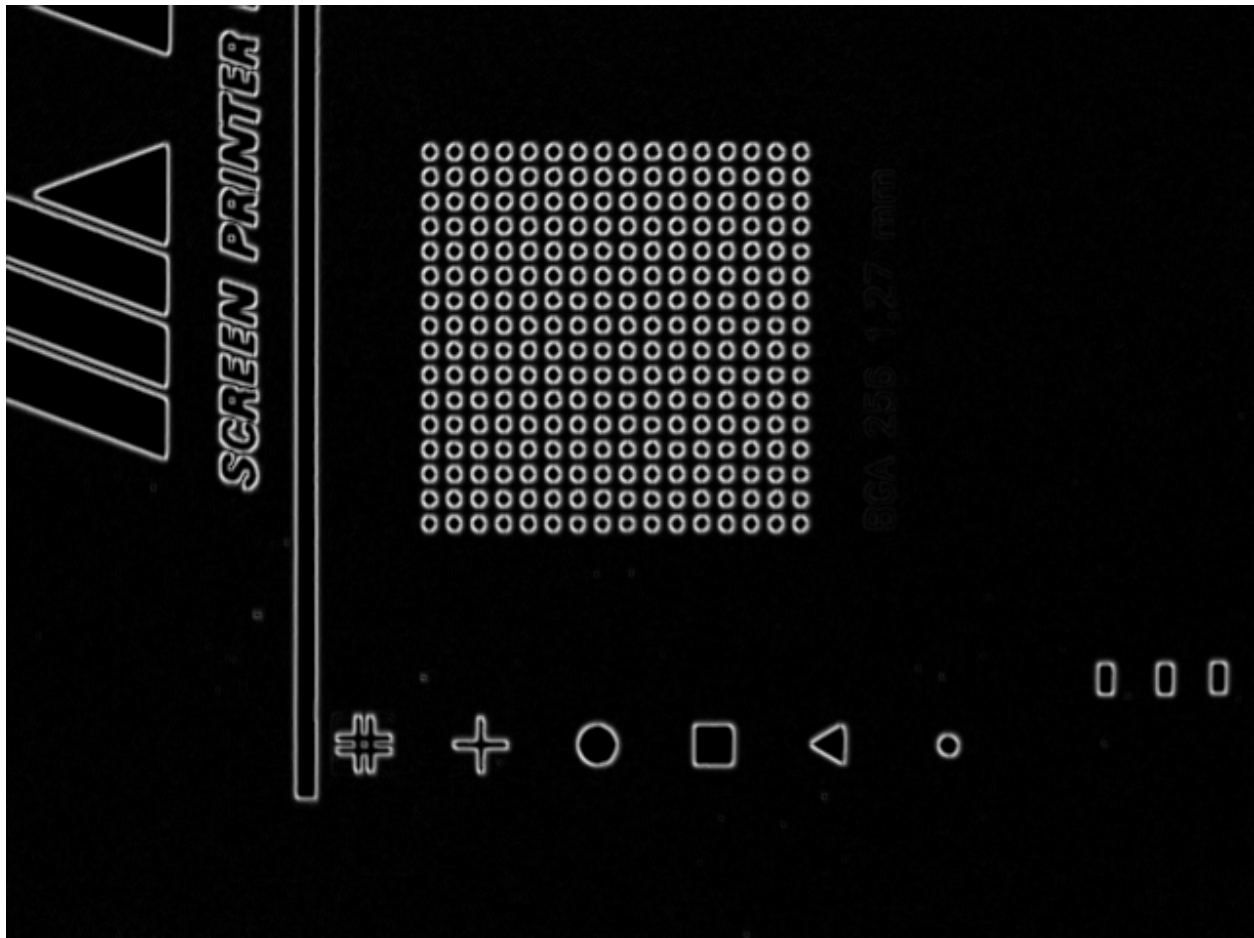
Sample

Here is an example that shows how to use the horizontal sobel filter.

Often, you want to combine the horizontal and the vertical sobel filter to create an edge map. A simple way to do this is shown in the following sample:

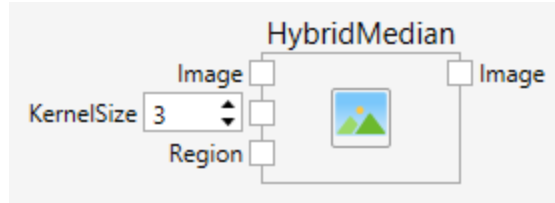


Here is an example image of such an edge map:



15.3.138 HybridMedian

Perform a hybrid median filter.



Inputs

Image (Type: Image)

The input image.

KernelSize (Type: Int32)

The kernel size.

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The hybrid median is a noise-reduction filter, which often performs better than simple averaging, especially for salt-and-pepper noise.

The hybrid median calculates the median of a cross-shaped kernel, the median of an x-shaped kernel, and finally the median of the center pixel and these two intermediate results.

The hybrid median is more edge-preserving than the standard median filter.

Here are a few results of the median filter with increasing kernel sizes:

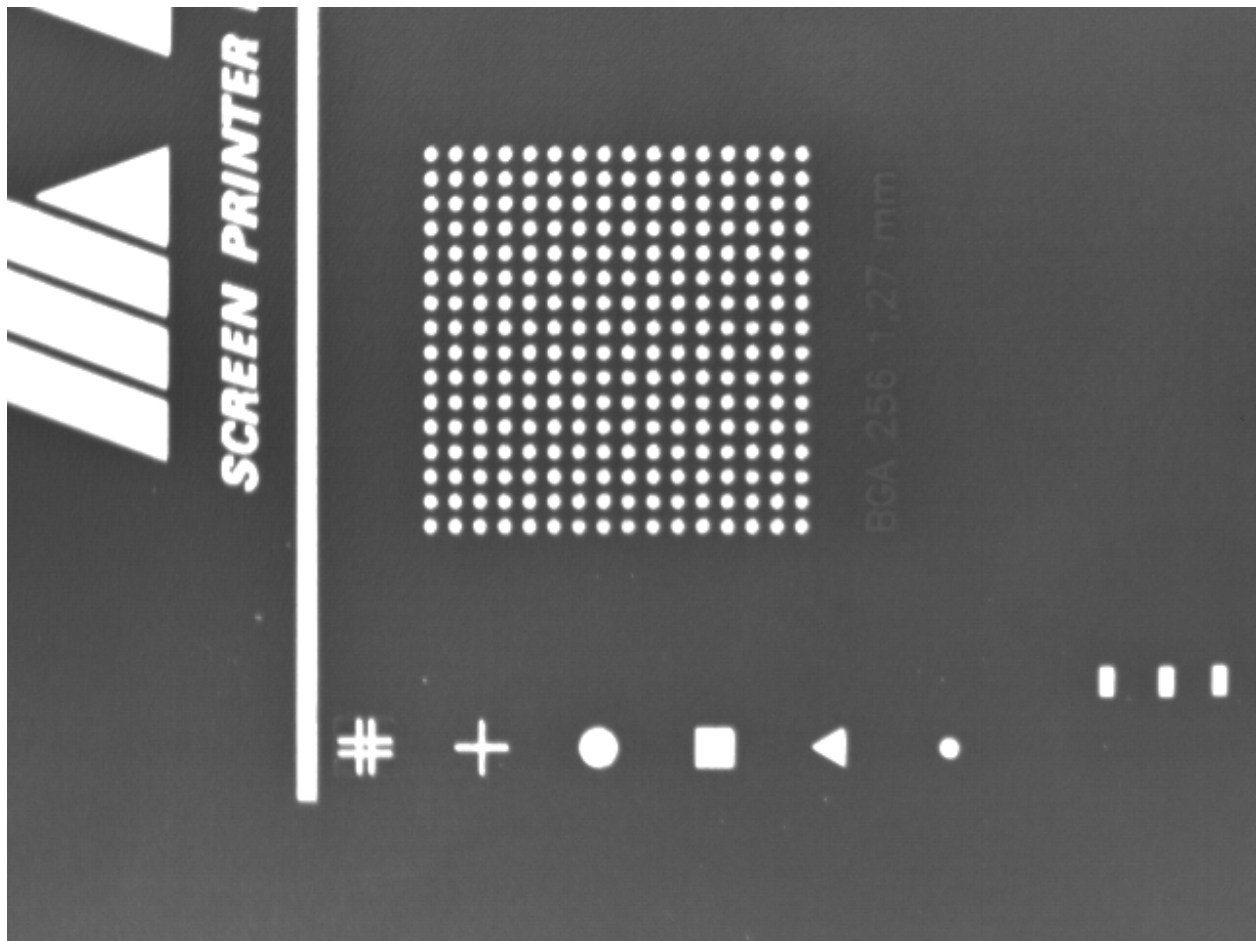
Original:

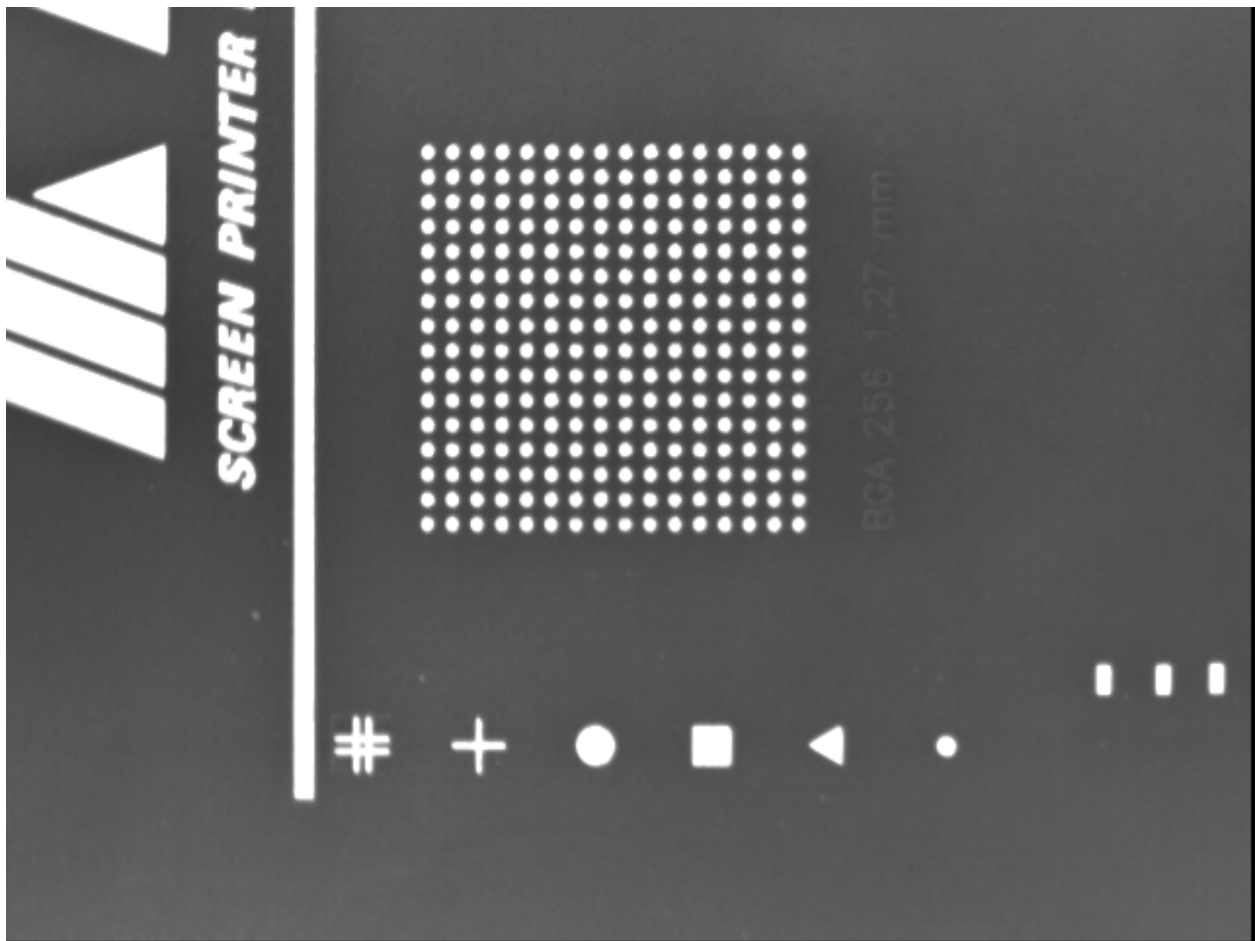
KernelSize = 3:

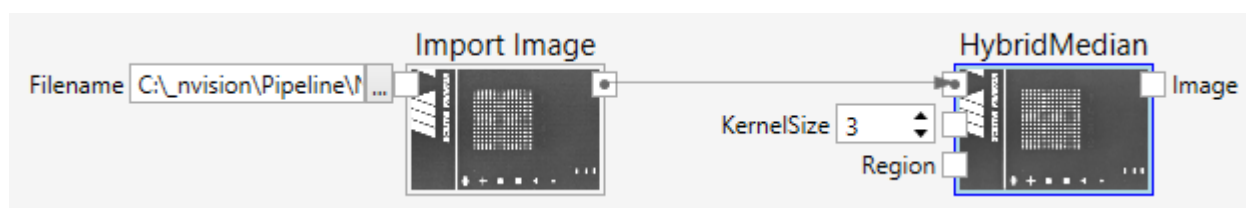
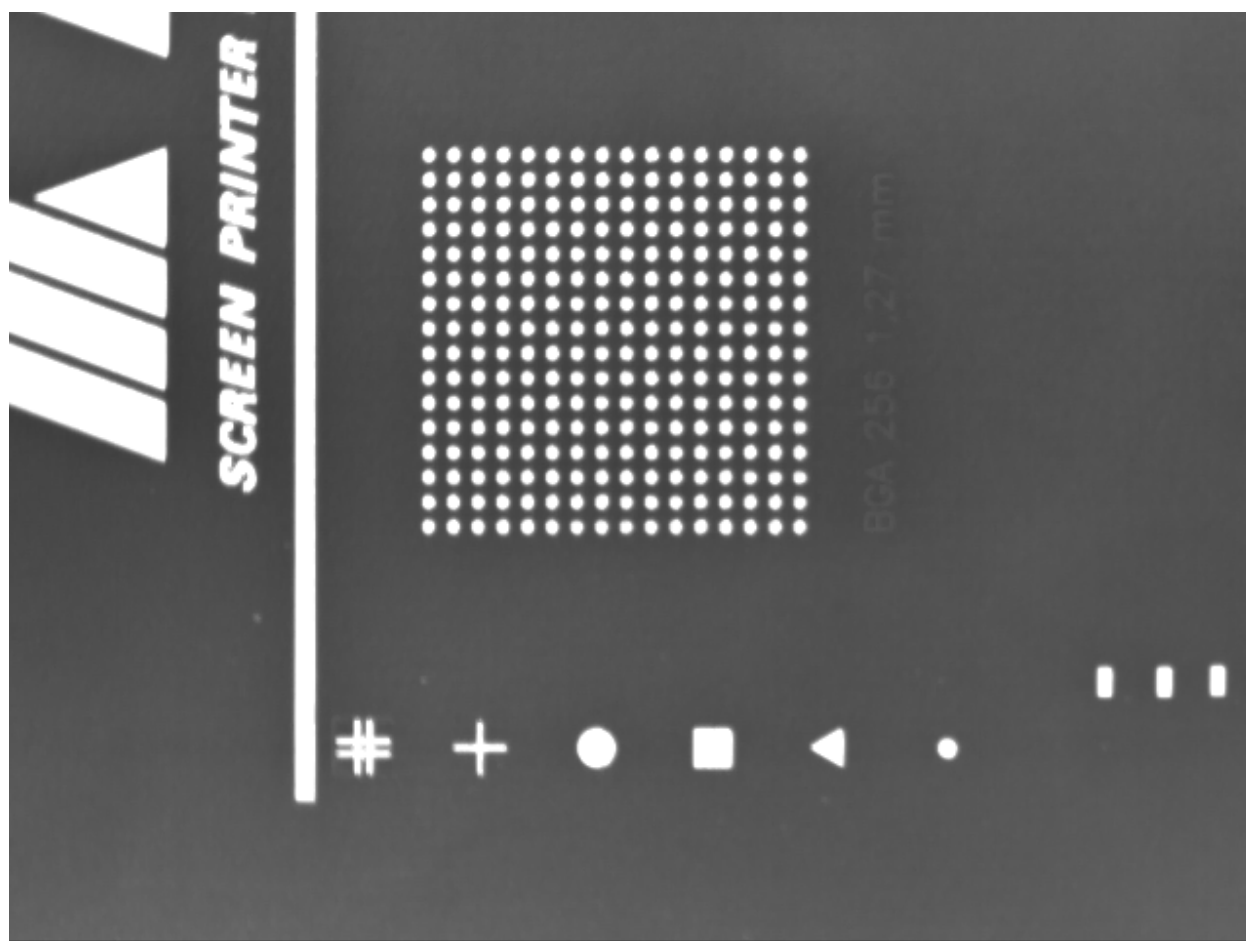
KernelSize = 5:

Sample

Here is an example that shows how to use the hybrid median filter.

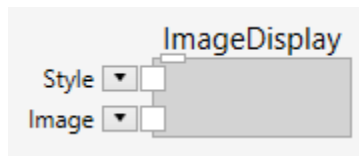






15.3.139 HMI Image

An **Image** can be used to display an image.



Inputs

Style (Type: Style)

Optional styling of the **Image**.

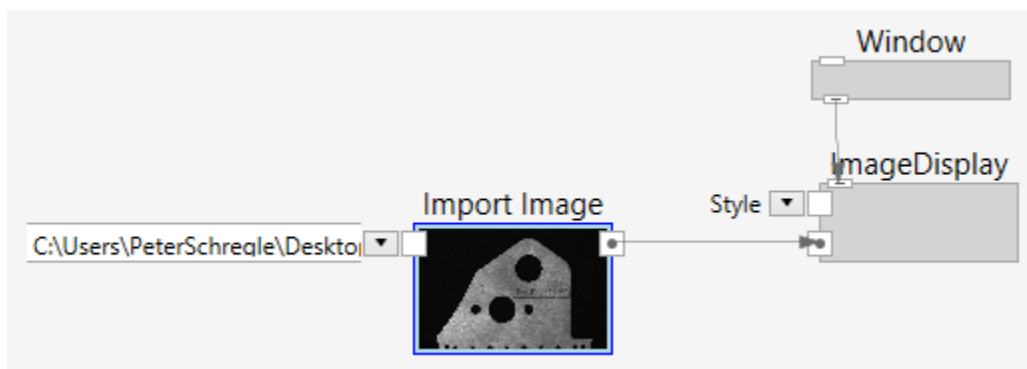
The **Image** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Image (Type: Image)

The iamge.

Example

Here is an example that shows an **Image**. This definition:



creates the following user interface:

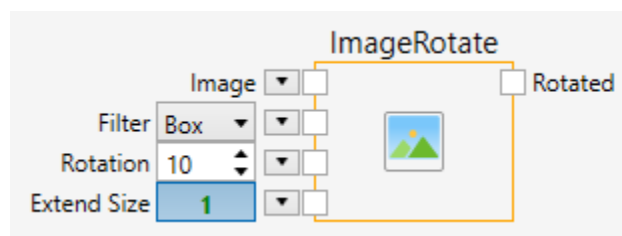
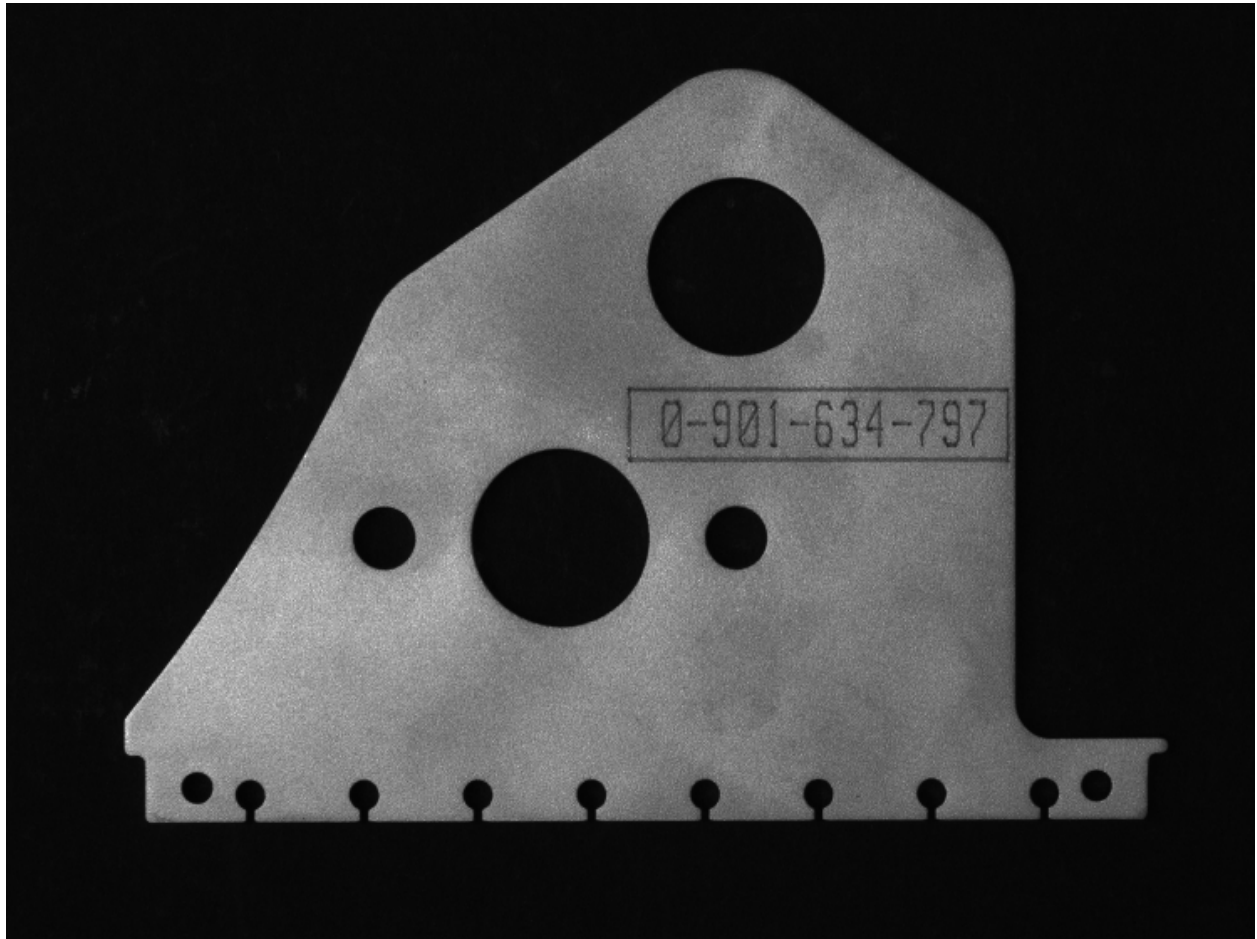
15.3.140 ImageRotate

Rotates an image by an angle.

Inputs

Image (Type: Image)

The input image.



Rotation (Type: Double)

The rotation angle in degrees. A positive value will rotate clockwise.

Filter (Type: String)

The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

Extend Size (Type: bool)

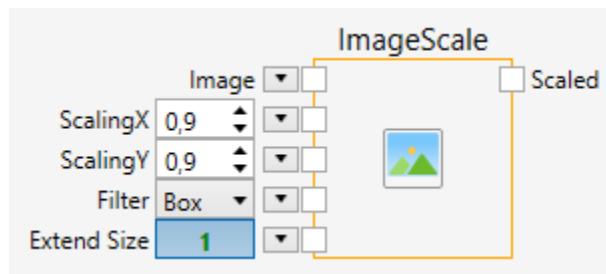
The geometric transformation may produce pixels that are outside of the input image bounds. If the parameter is set, the size of the output image is adapted to make room for these pixels. If the parameter is cleared, the size of the output image is kept the same as the input image.

Outputs**Rotated (Type: Image)**

The output image.

15.3.141 ImageScale

Scales an image by two scaling factors vector.

**Inputs****Image (Type: Image)**

The input image.

ScalingX (Type: Double)

The horizontal scaling factor.

ScalingY (Type: Double)

The vertical scaling factor.

Filter (Type: String)

The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

Extend Size (Type: bool)

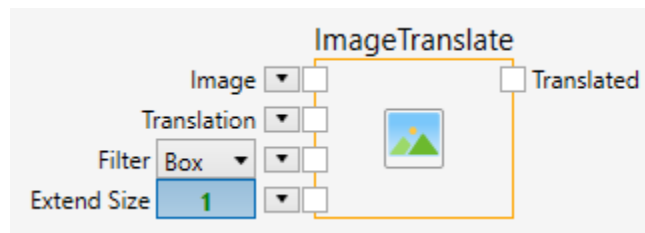
The geometric transformation may produce pixels that are outside of the input image bounds. If the parameter is set, the size of the output image is adapted to make room for these pixels. If the parameter is cleared, the size of the output image is kept the same as the input image.

Outputs**Scaled (Type: Image)**

The output image.

15.3.142 ImageTranslate

Translates an image by a translation vector.

**Inputs****Image (Type: Image)**

The input image.

Translation (Type: Ngui.VectorDouble)

The translation vector.

Filter (Type: String)

The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

Extend Size (Type: bool)

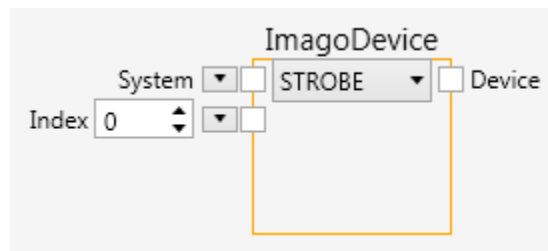
The geometric transformation may produce pixels that are outside of the input image bounds. If the parameter is set, the size of the output image is adapted to make room for these pixels. If the parameter is cleared, the size of the output image is kept the same as the input image.

Outputs**Translated (Type: Image)**

The output image.

15.3.143 Imago Device

A device implementation.

**Inputs****System (Type: VIB_NET.VIBSystem)**

The factory instance.

Index (Type: Int32)

The index of the device. The device itself is selected with the combobox inside the node.

Outputs**Device**

The device. The specify type depends on the device type selected within the node.

Comments

The first step in using any Imago device functionality is to create the system. With a system, you can then create an Imago device, and with a device you can call any method or property on it to carry out the respective operation.

For a more detailed description see the Imago .NET documentation (VIBSystem::OpenDevice method).

15.3.144 Imago Device Operation

An operation on an Image device.

Inputs

Device

The device. The specify type depends on the device type selected within the node.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs

Sync (Type: object)

Synchronizing object. It is used to establish an order of execution.

Comments

The first step in using any Imago device functionality is to create the system. With a system, you can then create an Imago device, and with a device you can call any method or property on it to carry out the respective operation.

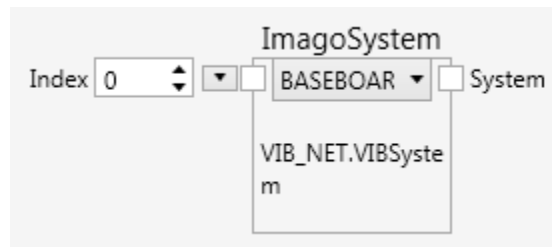
An imago system may have the following devices:

device	purpose
CameraLink	Represents the CameraLink-Input module which allows capturing images over CameraLink.
CameraTrigger	Represents the camera trigger output.
DigitalInput	Represents the digital inputs.
DigitalOutput	Represents the digital outputs.
IOScheduler	Controls the IOScheduler which allows to strobe out bit-values in hard real-time.
Led	Represents the Leds.
Multiplexer	Represents the multiplexer between trigger sources and sinks.
PowerOverEthernet	Represents the PowerOverEthernet (PoE) module, which allows gathering information about the current state of attached PoE devices.
RS232	Represents the RS232 which allows sending and receiving data over a RS232 port.
RS422	Represents the RS422 differential IO lines.
Service	Contains everything that is no physical interface.
Strobe	Represent the strobe (current) sources.
TriggerGenerator	This module contains functions which control the FPGA Trigger Unit.
TriggerOverEthernet	Represents the TriggerOverEthernet (ToE) module which allows the generation of GigE Action Commands.

For a more detailed description see the Imago .NET documentation (VIB_NET.iDevice interface and derived classes).

15.3.145 Imago System

Factory to acquire/release every device implementation.



Inputs

Index (Type: Int32)

The index of the desired system. The system itself is selected with the combobox inside the node.

Outputs

System (Type: VIB_NET.VIBSystem)

The factory instance.

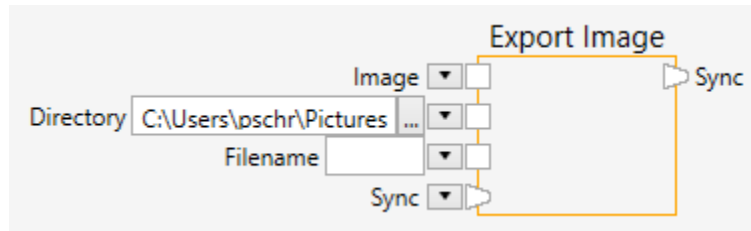
Comments

The first step in using any Imago device functionality is to create the system. With a system, you can then create an Imago device, and with a device you can call any method or property on it to carry out the respective operation.

For a more detailed description see the Imago .NET documentation (VIBSystem::VIBSystem method).

15.3.146 Export Image (Immediate)

Exports an image to a file.



Inputs

Image (Type: Image)

An image.


Directory (Type: String)

A directory in the filesystem. This is the directory, where the saved files will be placed. If the directory does not exist, an error message will be shown.

Filename (Type: String)

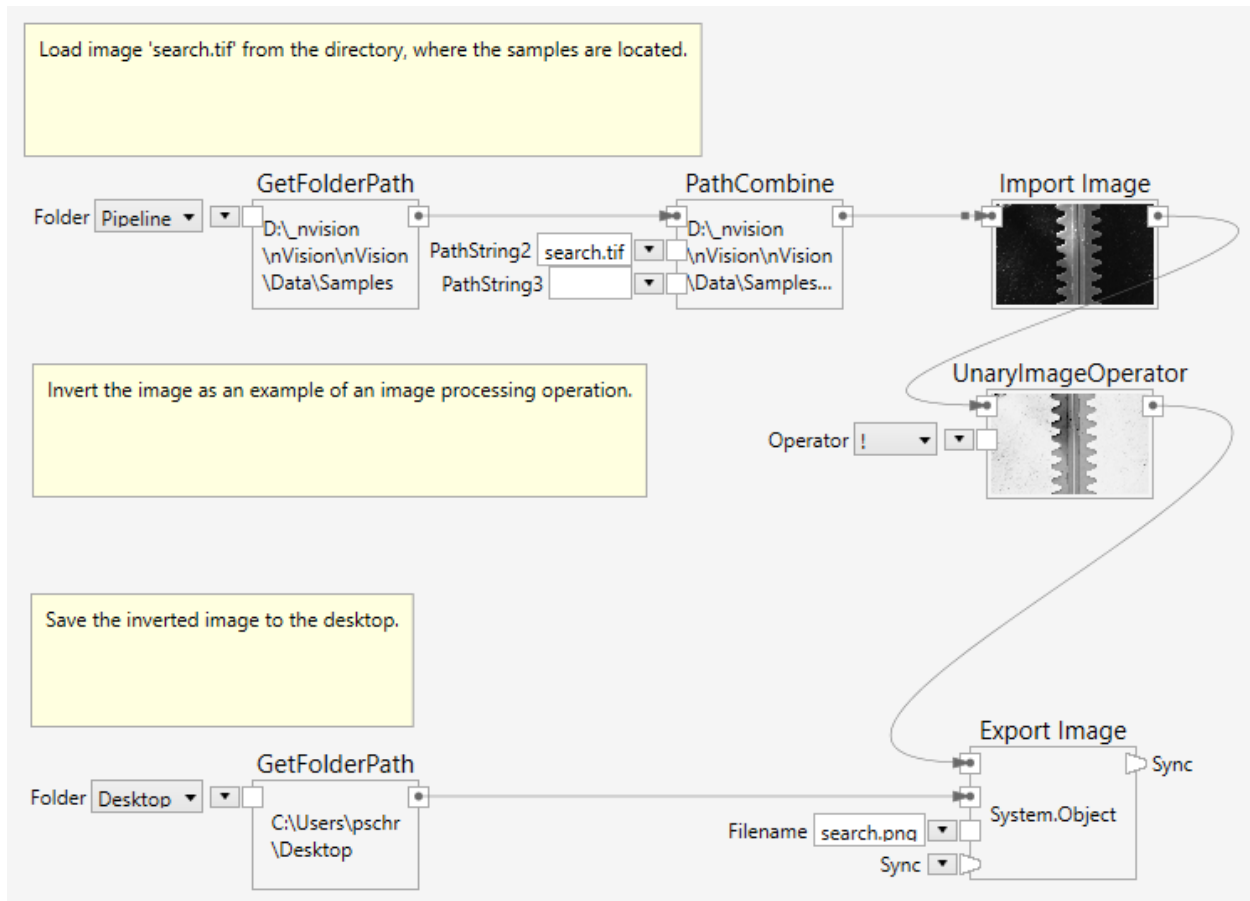
A filename. You can either enter a filename (with extension) or leave this empty. If you leave it empty, the system will create a name in the form of “image_00000.tif”, where the number will be incremented automatically. If you enter a filename, the file will be overwritten in every execution of the pipeline node.

Comments

The **Export Image** node saves an image to the filesystem. The directory where the file will be saved can either be typed, or selected with a Search Folder dialog - by clicking on the  button. The node supports a variety of image file formats, such as TIF, PNG, JPG and BMP to name a few. If a file could not be saved for some reason, an appropriate error message is shown.

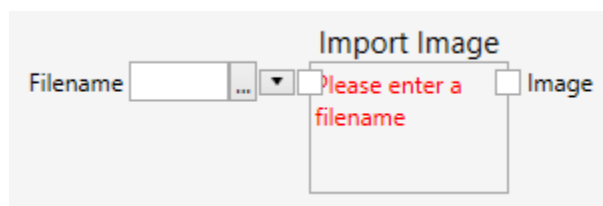
Sample

Here is an example that shows how to use the **Export Image** node (click on the image to load the example):



15.3.147 Import Image

Imports an image from a file.



Inputs

Filename (Type: String)


A filename in the filesystem.

Outputs

Image (Type: Image)

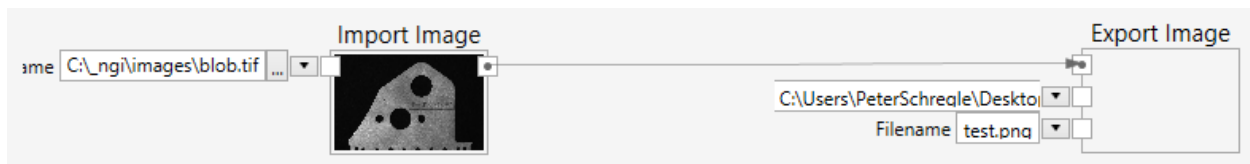
The loaded image.

Comments

The **ImportImage** node loads an image from the filesystem. The name of the file can either be typed, or selected with a File Open dialog - by clicking on the  button. The node supports a variety of image file formats, such as TIF, PNG, JPG and BMP to name a few. If a file could not be loaded for some reason, an appropriate error message is shown.

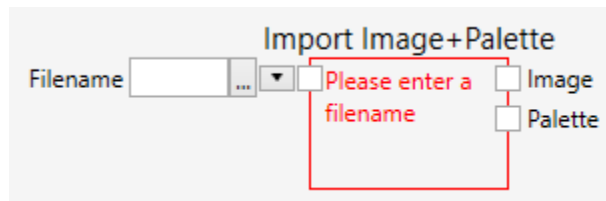
Sample

Here is an example that shows how to use the **ImportImage** node.



15.3.148 Import Image + Palette

Imports an image and its associated palette from a file.



Inputs

Filename (Type: String)

A filename in the filesystem.

Outputs


Image (Type: Image)

The loaded image.

Palette (Type: Palette)

The loaded palette.

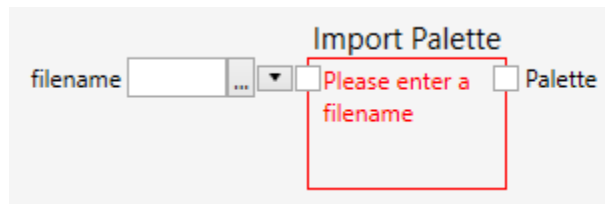
Comments

The **ImportImage** node loads an image and its associated palette from the filesystem. The name of the file can either be typed, or selected with a File Open dialog - by clicking on the  button. The node supports a variety of image file formats, such as TIF, PNG, JPG and BMP to name a few. If a file could not be loaded for some reason, an appropriate error message is shown.

The loaded image is not mapped over the palette, and you may need to do this if you want to make sure that the image is displayed in a visual correct manner.

15.3.149 ImportPalette

Imports a palette from a file.


**Inputs****Filename (Type: String)**

A filename in the filesystem.

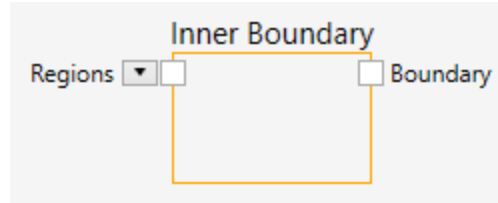
Outputs**Palette (Type: Palette)**

The loaded palette.

Comments

The **ImportPalette** node loads a palette from the filesystem. The name of the file can either be typed, or selected with a File Open dialog - by clicking on the  button.

The supported format is a binary file consisting of either 256 bytes (for a monochrome palette) or 768 bytes (for a color palette).



15.3.150 Inner Boundary

Extracts the inner boundary from a list of regions.

The inner boundary of a region consists of the pixels of the region that touch the background. The inner boundary lies on the object.

Inputs

Regions (Type: RegionList)

The list of regions.

Outputs

Boundary (Type: RegionList)

The objects reduced to their inner boundary.

Comments

Here is an example of an inner boundary:

15.3.151 Jaehne

Presets an image with a pattern designed by Prof. Bernd Jaehne.

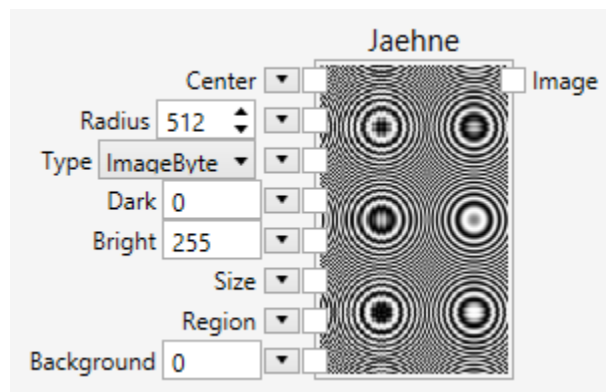
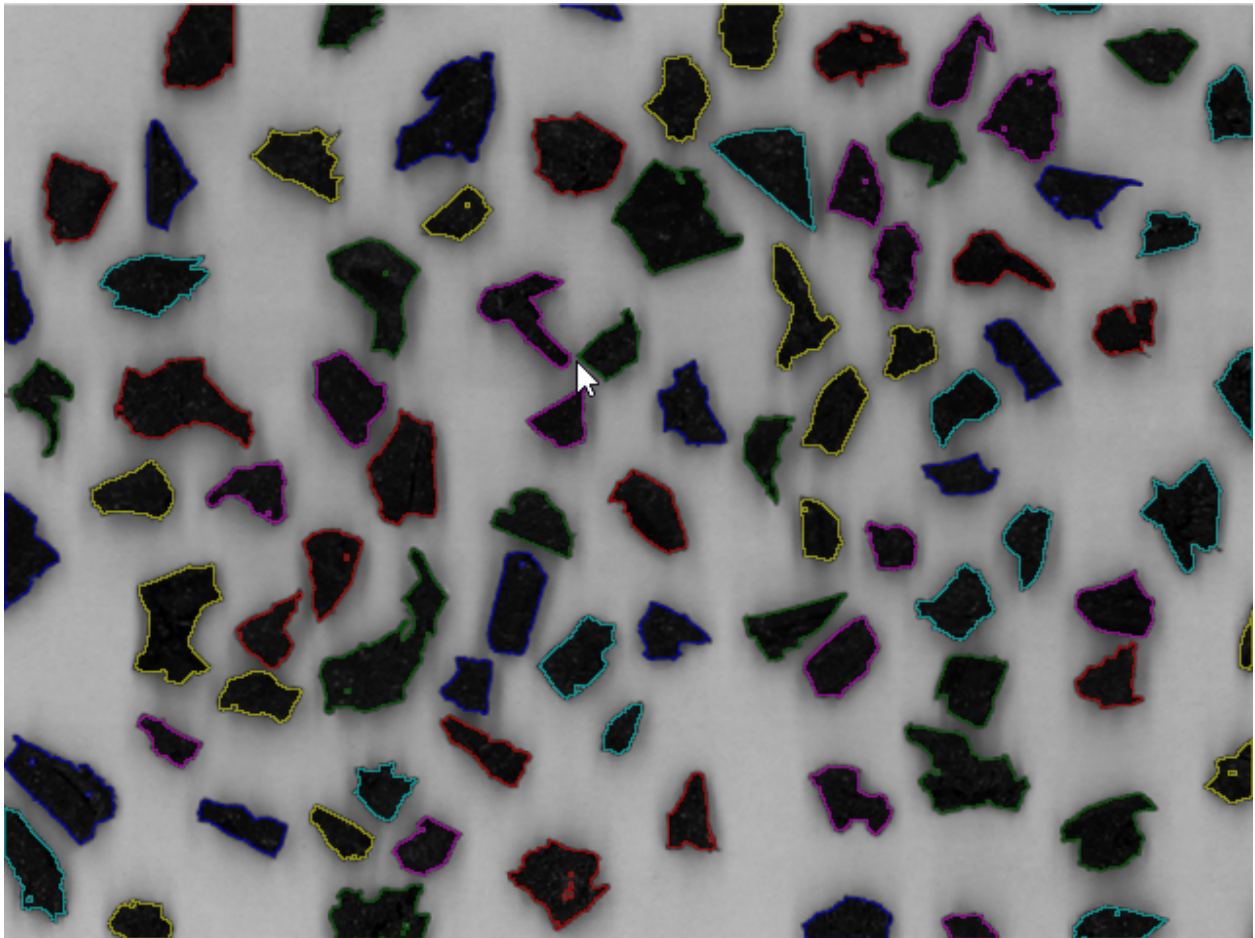
Inputs

Center (Type: PointDouble)

The center of the Jaehne pattern. The default is 512, 512.

Center (Radius: Double)

The radius of the Jaehne pattern. The default is 512.



Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble

Dark (Type: String)

The value of a dark checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: String)

The value of a bright checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

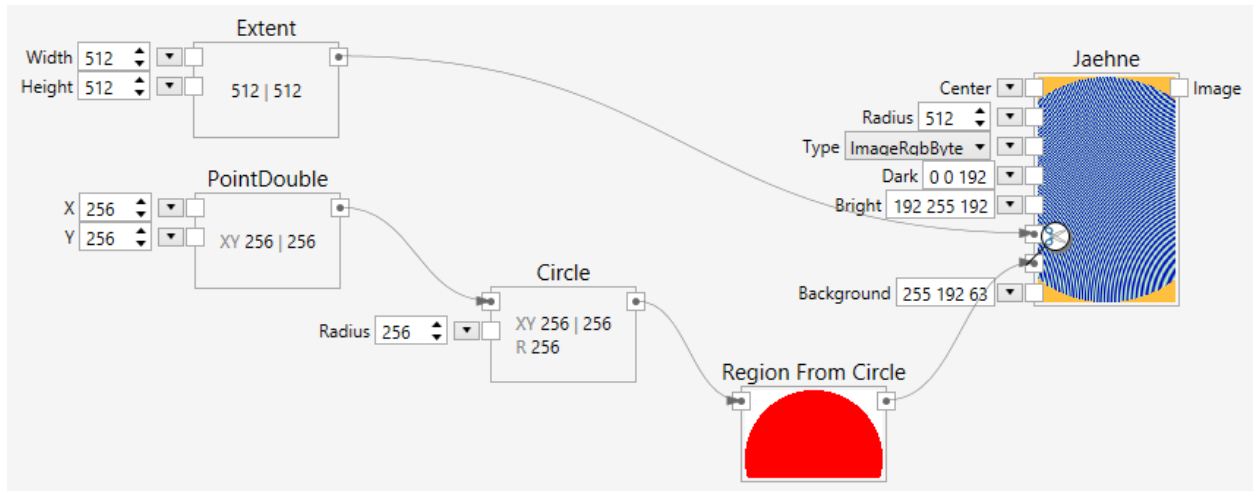
Outputs

Image (Type: Image)

The image preset with a Jaehne pattern.

Comments

The **Jaehne** node presets an image with a sinusoidal pattern designed by Prof. Bernd Jaehne. The size of the created image, the radius and center of the pattern, the colors for dark and bright, the region of interest and the background color can all be specified.



Sample

Here is an example that shows how to use the **Jaehne** node.

Here is the image created with the sample:

15.3.152 Laplace

Filters an image using a laplace kernel of size 3x3 or 5x5.

Inputs

Image (Type: Image)

The input image.

KernelSize (Type: Int32)

The kernel size (3 -> 3x3, 5 -> 5x5).

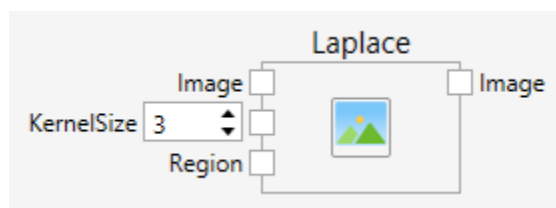
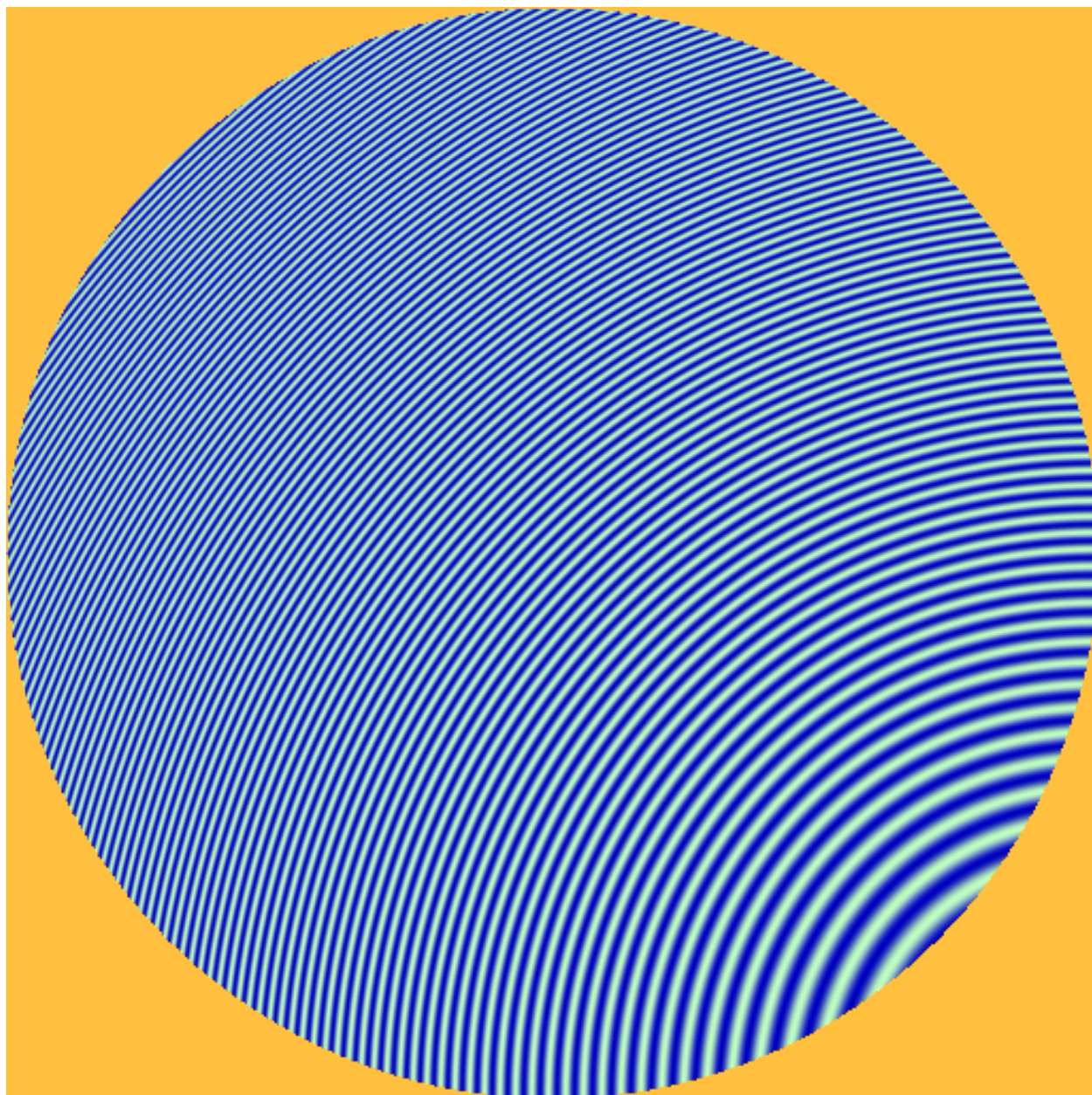
Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.



Comments

The function applies a laplace filter. The corresponding kernel is either a 3x3 matrix with the following values:

```
-1 -1 -1
-1  8 -1
-1 -1 -1
```

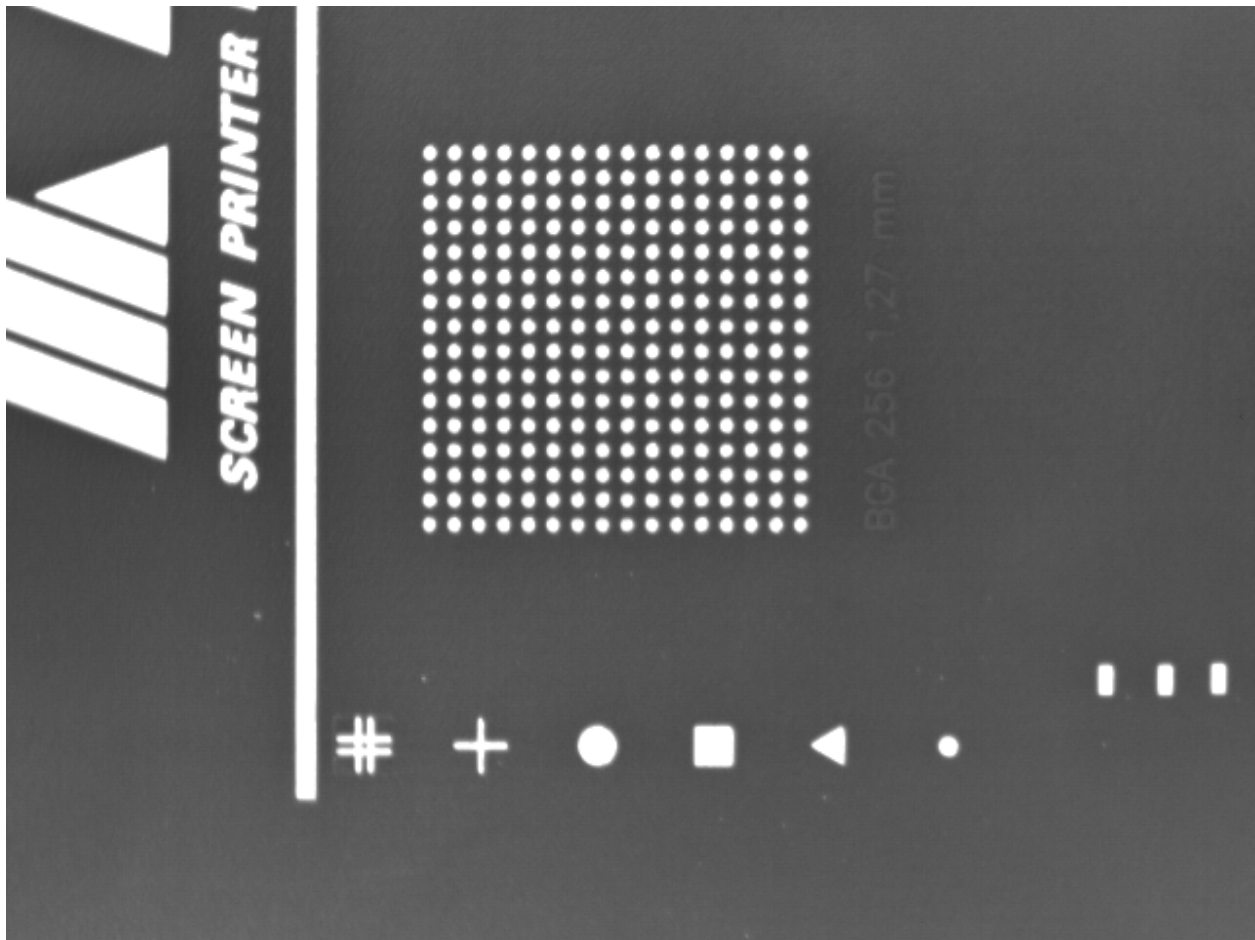
or a 5x5 kernel with the following values:

```
-1 -3 -4 -3 -1
-3  0  6  0 -3
-4  6 20  6 -4
-3  0  6  0 -3
-1 -3 -4 -3 -1
```

The effect of a laplace filter is that it amplifies high frequencies and attenuates low frequencies. The strength of the laplace filter depends on the size of the kernel.

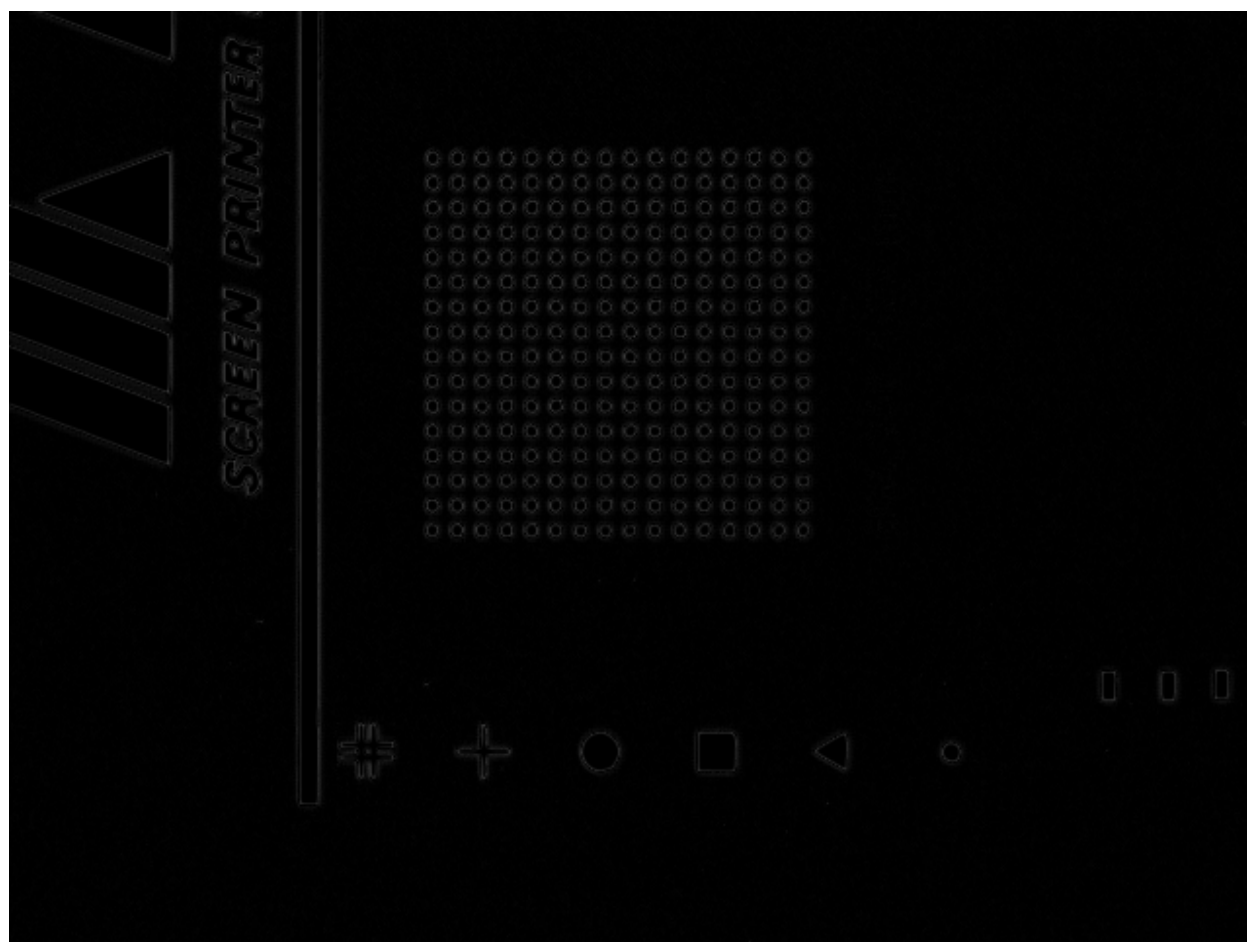
Here are a few results of the laplace filter with increasing kernel sizes:

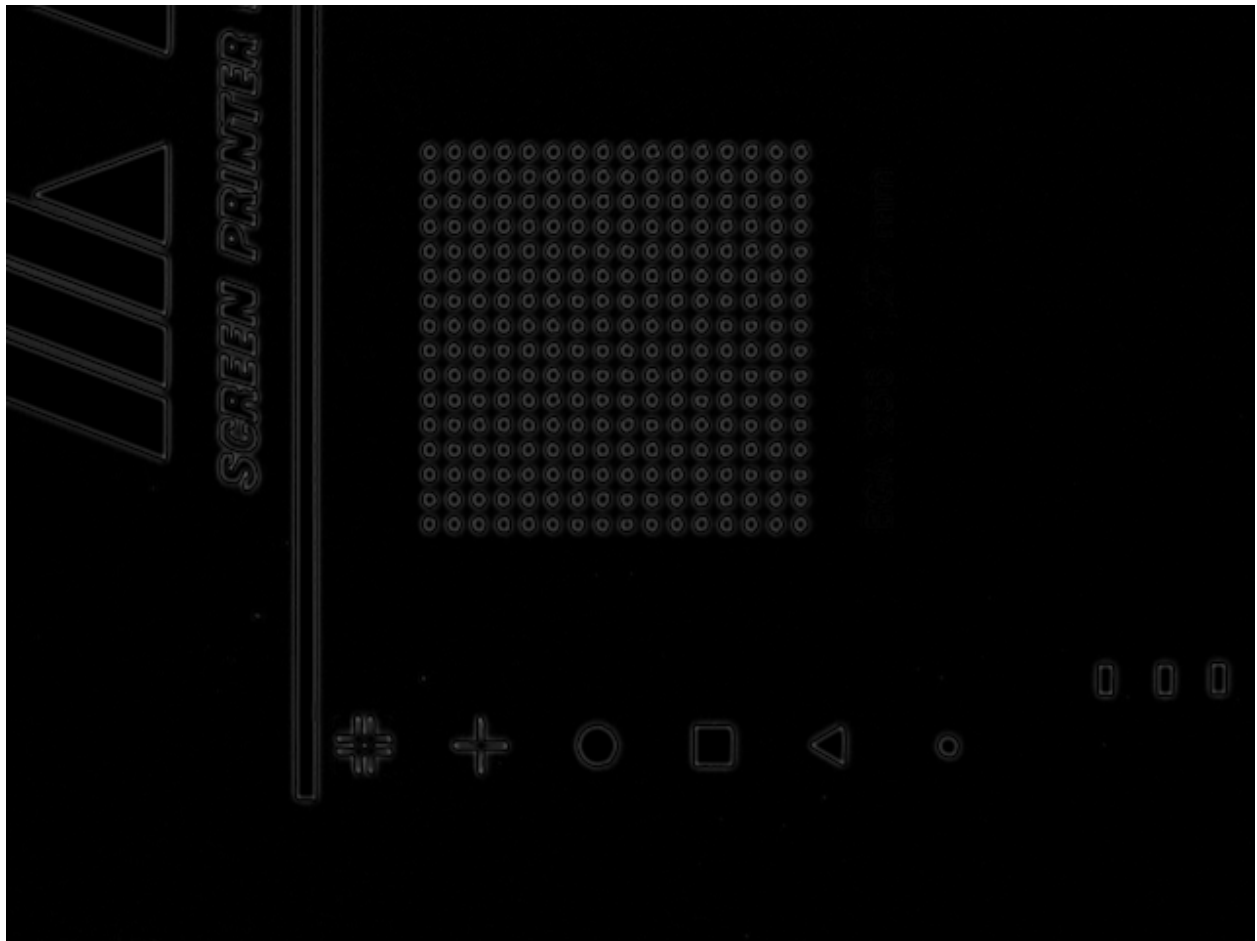
Original:



KernelSize = 3:

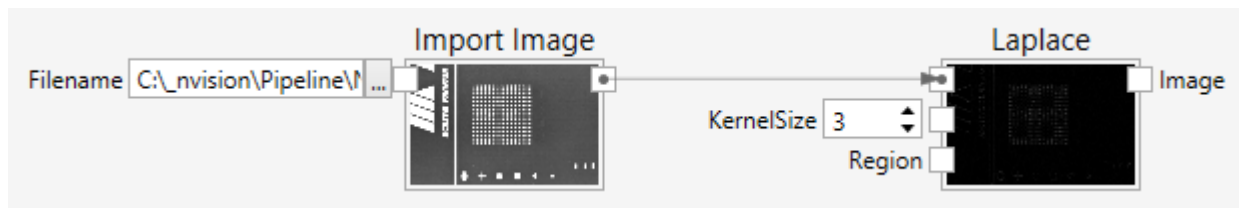
KernelSize = 5:





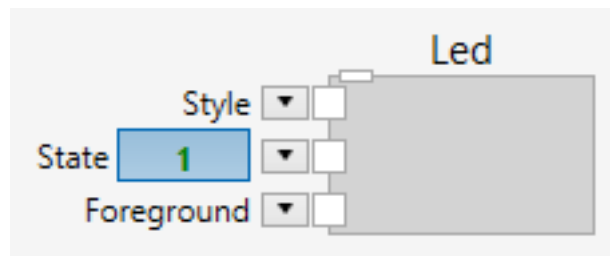
Sample

Here is an example that shows how to use the laplace filter.



15.3.153 HMI Led

An **Led** can be used to display a state.



Inputs

Style (Type: `style`)

Optional styling of the **Led**.

The **WebBrowser** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment* and *Margin* styles.

State (Type: `boolean`)

The **Led** is one if this is `True`, the **Led** is off otherwise.

Foreground (Type: `SolidColorBrush`)

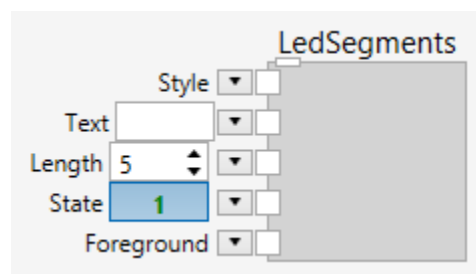
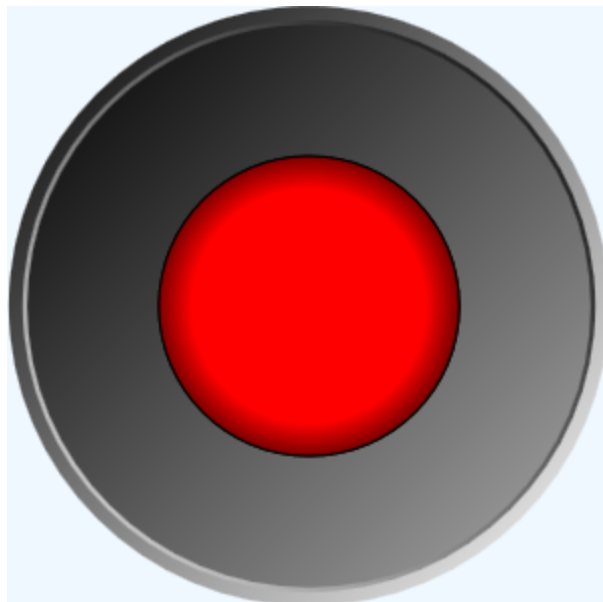
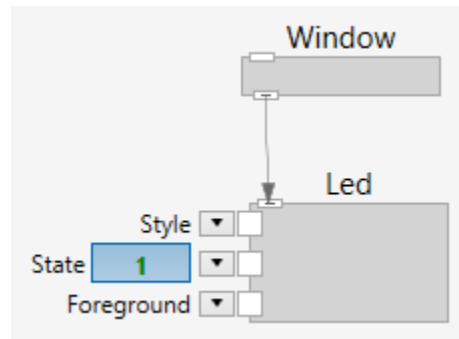
The color of the **Led**.

Example

Here is an example that shows an **Led**. This definition:
creates the following user interface:

15.3.154 HMI LedSegments

An **LedSegments** can be used to display text in a segmented led-style display.



Inputs

Style (Type: `Style`)

Optional styling of the **Led**.

The **WebBrowser** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment* and *Margin* styles.

Text (Type: `string`)

The text.

Length (Type: `Int32`)

The number of characters in the display.

State (Type: `boolean`)

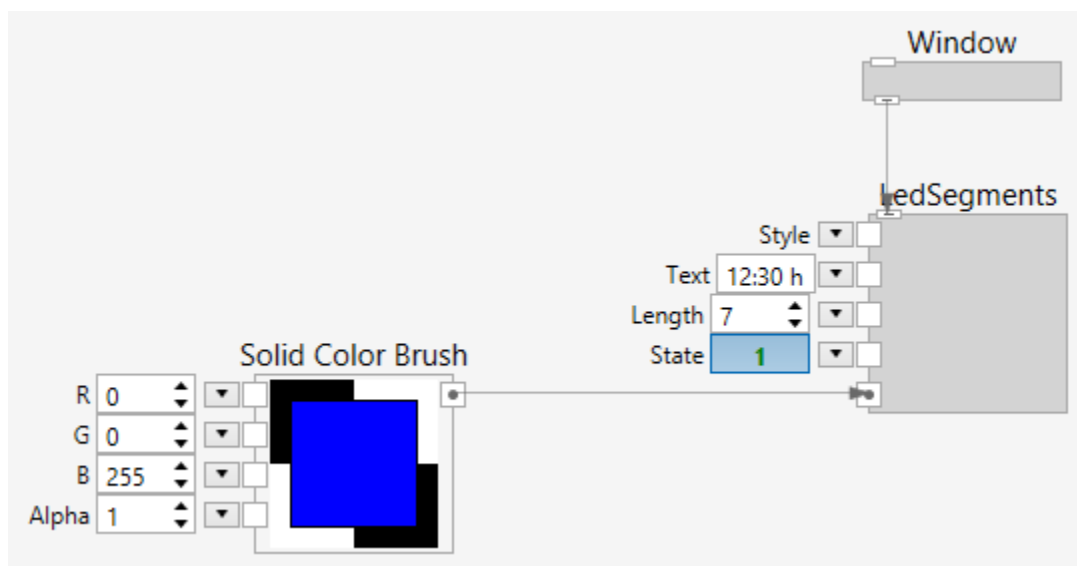
The **LedSegments** is one if this is `True`, the **LedSegments** is off otherwise.

Foreground (Type: `SolidColorBrush`)

The color of the **LedSegments**.

Example

Here is an example that shows an **LedSegments**. This definition:

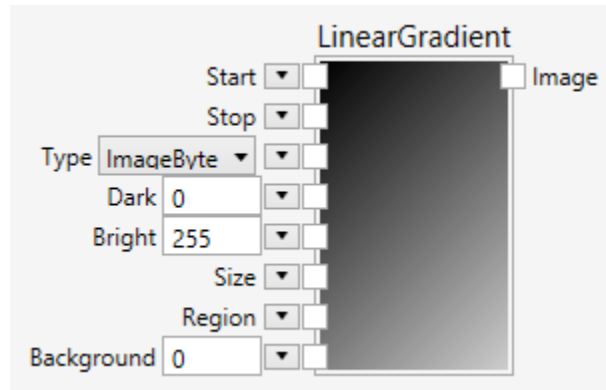


creates the following user interface:



15.3.155 LinearGradient

Presets an image with a linear gradient pattern.



Inputs

Start (Type: `PointDouble`)

The start point of the gradient. At this point the dark color is used. Between the start and stop points the colors are interpolated linearly between dark and bright.

Stop (Type: `PointDouble`)

The stop point of the gradient. At this point the bright color is used. Between the start and stop points the colors are interpolated linearly between dark and bright.

Type (Type: `String`)

The image type. The following types can be chosen: `ImageByte`, `ImageUInt16`, `ImageUInt32`, `ImageDouble`, `ImageRgbByte`, `ImageRgbUInt16`, `ImageRgbUInt32`, `ImageRgbDouble`

Dark (Type: `String`)

The value of a dark checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: `String`)

The value of a bright checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three

numbers separated by a space.

Size (Type: **Extent3d**)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: **Region**)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: **String**)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs

Image (Type: **Image**)

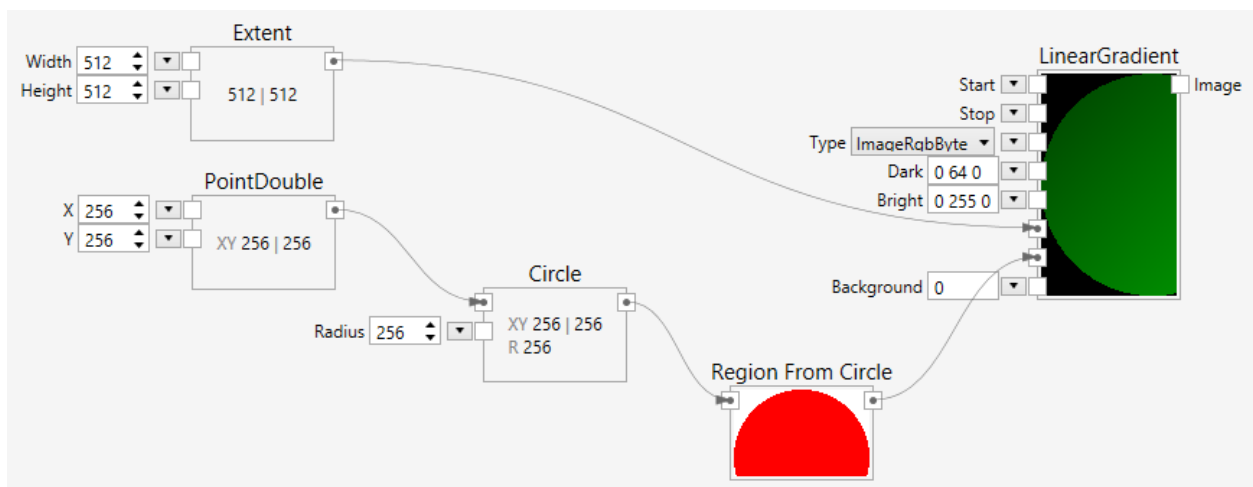
The image preset with a linear gradient pattern.

Comments

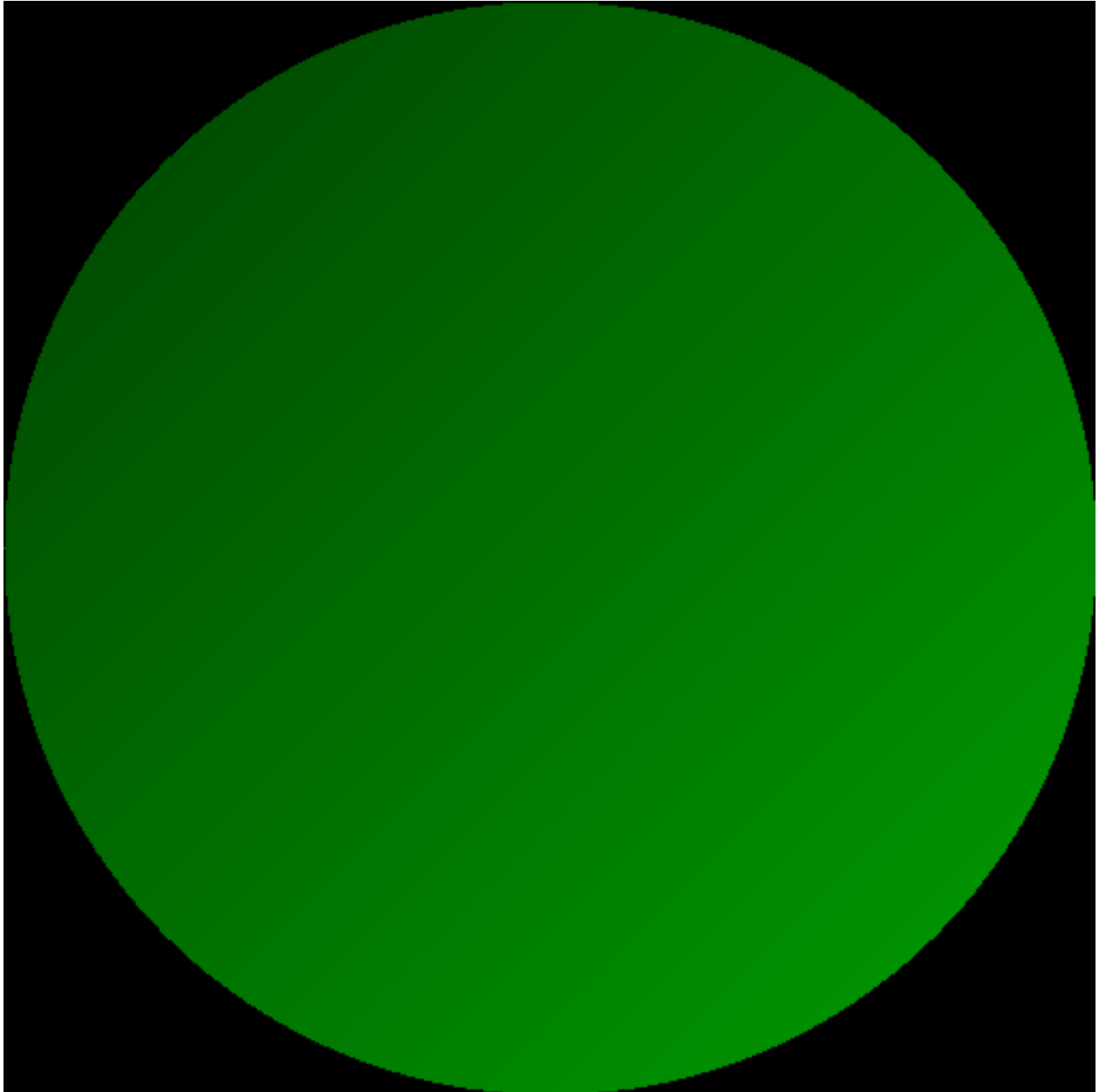
The **LinearGradient** node presets an image with a linear gradient pattern. The size of the created image, the points and colors for dark and bright, the region of interest and the background color can all be specified.

Sample

Here is an example that shows how to used the **LinearGradient** node.

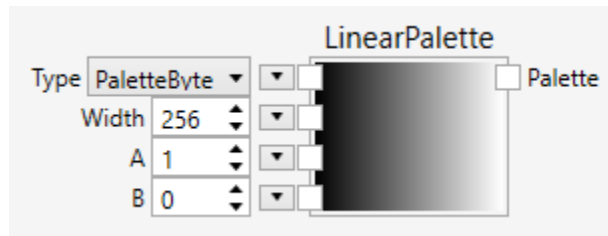


Here is the image created with the sample:



15.3.156 LinearPalette

Creates a palette with a linear transfer function.



Inputs

Type (Type: String)

The type of the palette. Choices are `PaletteByte`, `PaletteUInt16`, `PaletteUInt32`, `PaletteDouble`, `PaletteRgbByte`, `PaletteRgbUInt16`, `PaletteRgbUInt32` and `PaletteRgbDouble`.

Width (Type: Int32)

The width of the palette.

A (Type: Double)

The slope of the linear curve.

B (Type: Double)

The intercept of the linear curve.

Outputs

Palette (Type: Palette)

The output palette.

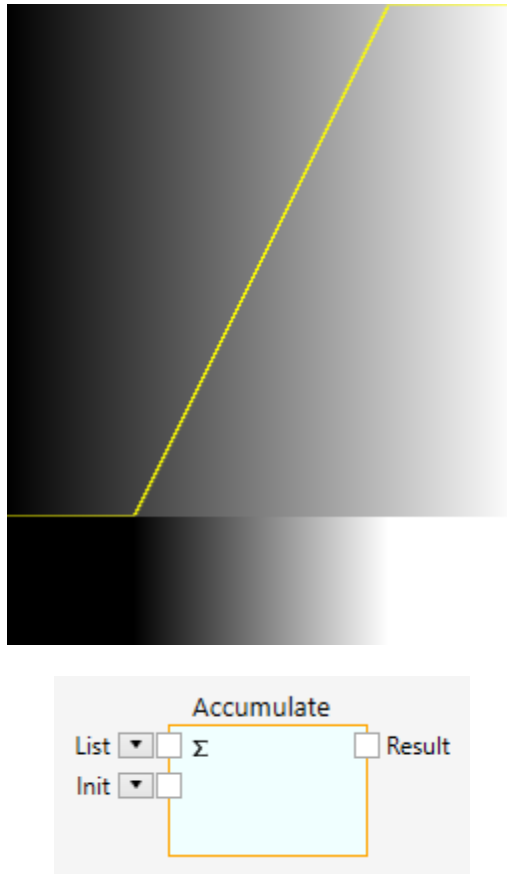
Comments

This function creates a palette with a linear function.

Here is an example of a linear curve with a slope of 2 and an intercept of -127:

15.3.157 Accumulate

Accumulate takes all items from a list and combines them into a single result. Calculating the sum, or the minimum or the maximum are examples for accumulation. The accumulation is specified as a graphical pipeline, for a single item, which combines a single item with the accumulation so far.



Outside View

At the outside, the **Accumulate** node appears like a single node, with inputs and outputs.

Inputs

List (Type: List<T>)

The input list.

Init (Type: U)

The initial accumulator value.

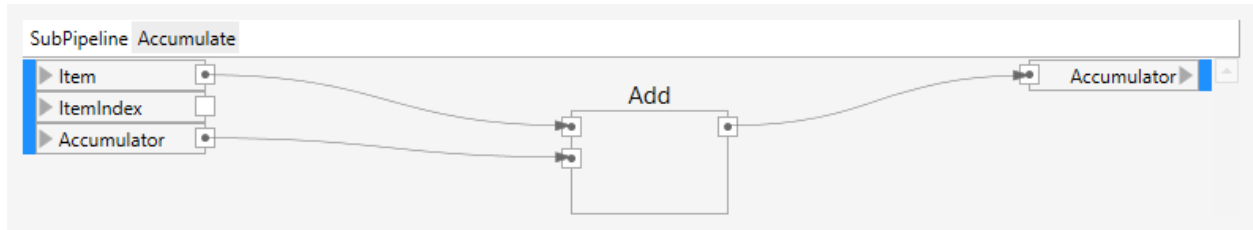
Outputs

Result (Type: U)

The resulting accumulator value.

Inside View

Inside, the **Accumulate** node looks like a normal branched pipeline. This pipeline is used to specify the accumulation, which is evaluated for each item.



Inputs

Item (Type: \mathbb{T})

One item of the input list.

ItemIndex (Type: `Int32`)

The index of the item of the input list.

Accumulator (Type: \mathbb{U})

The input accumulator value.

Outputs

Accumulator (Type: \mathbb{U})

The resulting accumulator value.

Comments

Accumulate nodes can be nested in similar ways as subpipelines.

The nesting is shown with a breadcrumbs control at the top.

First Level Second Level Third Level

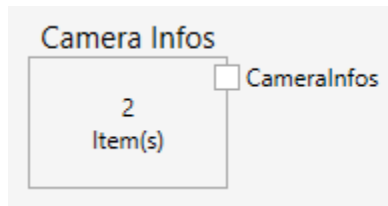
Double-click an **Accumulate** node to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

By default, a **Accumulate** node has three inputs and one output.

You can add additional inputs with the **Add Input (Ctrl-I)** command. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

15.3.158 Camera Infos

Lists information about the available cameras.



The preview shows the number of available cameras.

Outputs

CameraInfos (Type: List<CameraInfo>)

A list of **CameraInfo** objects. You can use **GetItem** to take a specific item from the list and **GetProperty** to read a specific piece of information, such as the *Model* or the *Vendor*.

Example

Here is an example that lists some information about cameras:

15.3.159 Concatenate

Concatenates two lists of the same type.

Inputs

ListA (Type: List<T>)

The first input list.

ListB (Type: List<T>)

The second input list.

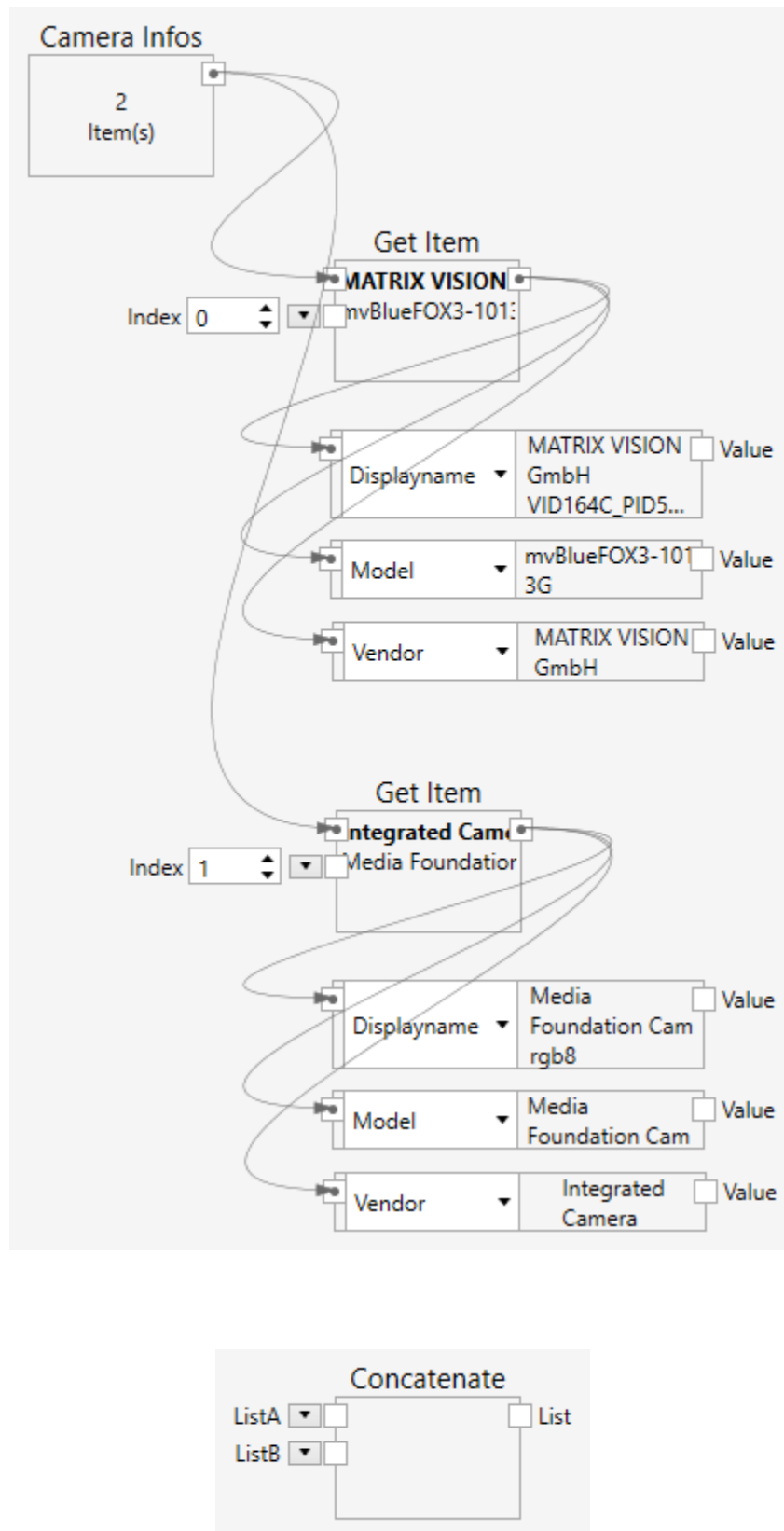
Outputs

List (Type: List<T>)

The output list, which consists of the items of the first input list followed by the items of the second input list.

15.3.160 List Files

Lists the available files in a directory.





Inputs

Directory (Type: String)

The directory (provided as a string, or selected via the ... button. This can be a relative or absolute path to the directory to search. This string is not case-sensitive.

Filter (Type: String)

The filter string to match against the names of the files in the directory. This parameter can contain a combination of valid literal path and wildcard characters (* to match zero or more characters and ? to match zero or one characters).

Subdirectories (Type: Boolean)

If this parameter is set to false, the files in the specified directory only are returned. If this parameter is set to true, the files in the specified directory as well as in all its subdirectories are returned.

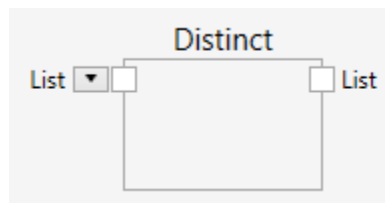
Outputs

FileNames (Type: List<String>)

The list of files.

15.3.161 Distinct

Removes duplicates from a list, so that only distinct items remain in the list.



Inputs

List (Type: List<T>)

The input list.

Outputs

List (Type: List<T>)

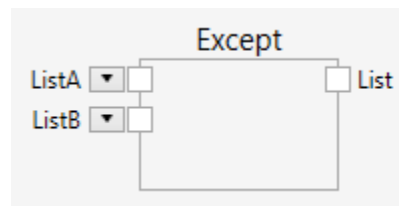
The output list, which consists of the distinct items of the input list only.

Comments

The output list is a set in the mathematical sense, which can have distinct objects only.

15.3.162 Except

Calculates the difference of two sets.



Inputs

ListA (Type: List<T>)

The first input list, whose distinct elements form the first set for the set difference operation.

ListB (Type: List<T>)

The second input list, whose distinct elements form the second set for the set difference operation.

Outputs

List (Type: List<T>)

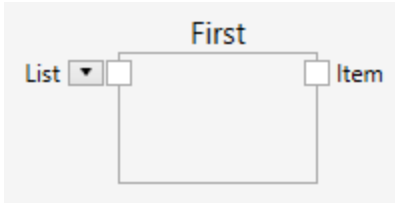
The output list, which consists of the set difference of the two lists, i.e. the output list consists of the items of the first input list that are not also in the second input list.

Comments

The lists are treated as mathematical sets, which can have distinct objects only.

15.3.163 First

Takes the first item out of a list.



Inputs

List (Type: List<T>)

The list.

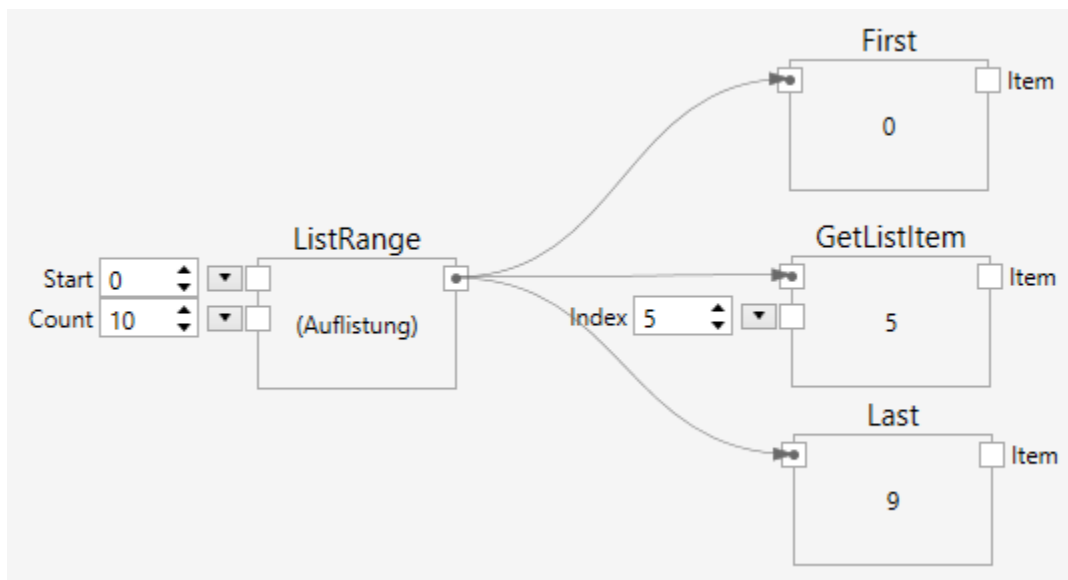
Outputs

Item (Type: T)

The first item of the list.

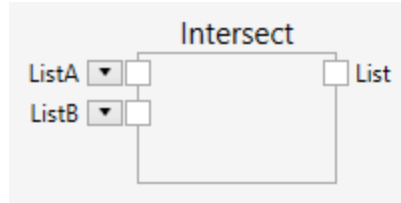
Sample

Here is an example:



15.3.164 Intersect

Calculates the intersection of two sets.



Inputs

ListA (Type: List<T>)

The first input list, whose distinct elements form the first set for the intersection.

ListB (Type: List<T>)

The second input list, whose distinct elements form the second set for the union.

Outputs

List (Type: List<T>)

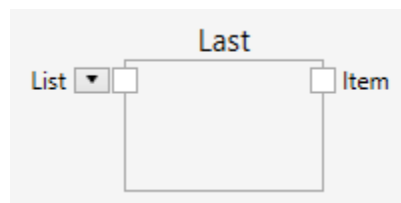
The output list, which consists of the set intersection of the two lists.

Comments

The lists are treated as mathematical sets, which can have distinct objects only.

15.3.165 Last

Takes the last item out of a list.



Inputs

List (Type: List<T>)

The list.

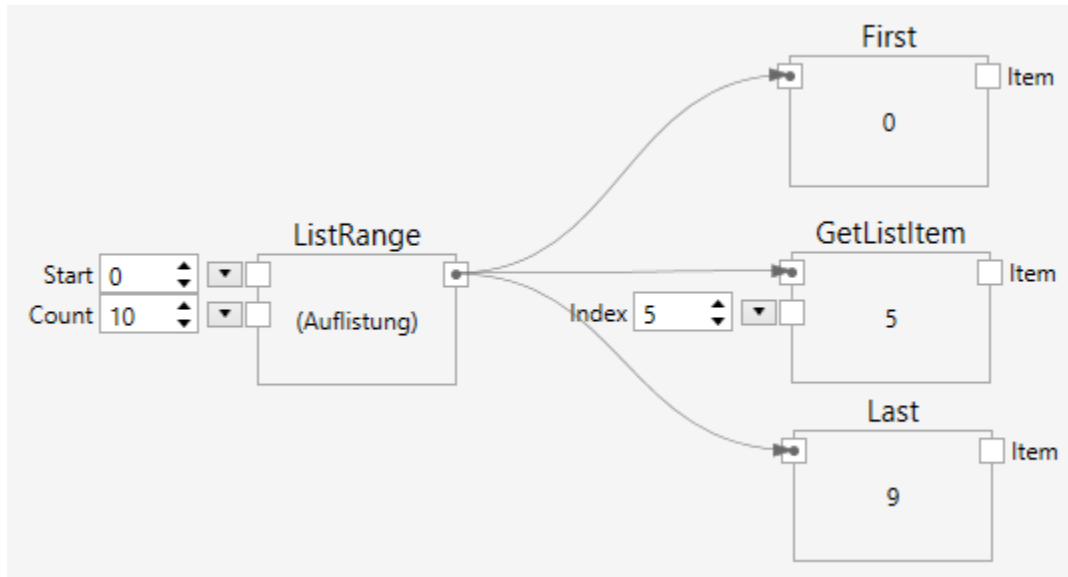
Outputs

Item (Type: T)

The last item of the list.

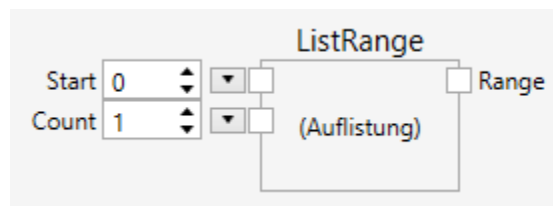
Sample

Here is an example:



15.3.166 ListRange

Creates a list of increasing integer numbers.



By default the list starts at 0 and has 1 entry.

The ListRange node is often used to provide the equivalent of a loop in **nVision's** dataflow pipelines.

Inputs

Start (Type: Int32)

The start of the list of numbers.

Count (Type: Int32)

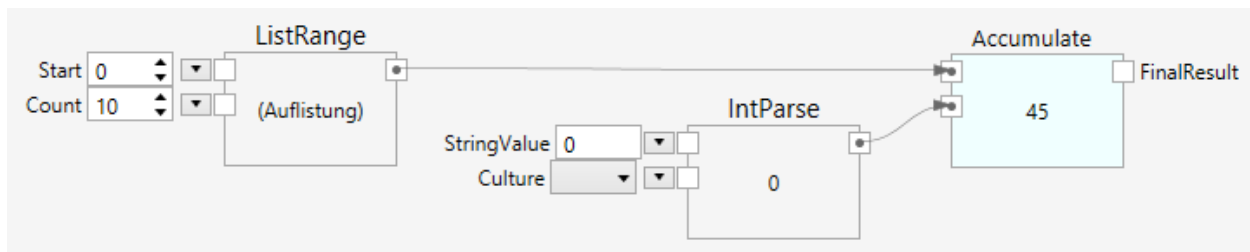
The count of increasing numbers.

Outputs**Range (Type: List<Int32>)**

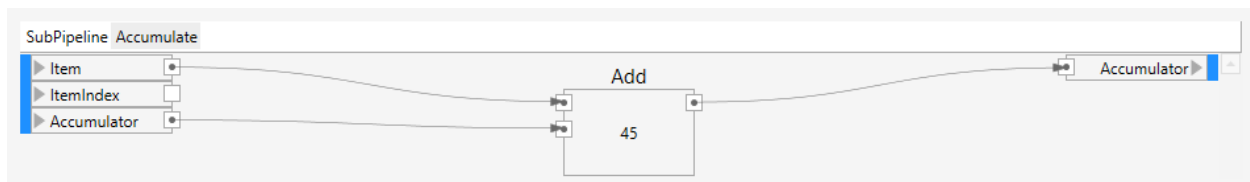
The list of numbers.

Sample

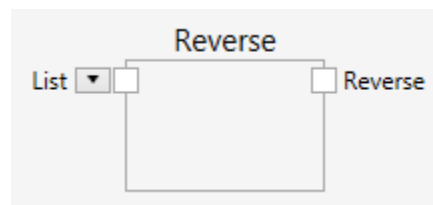
Here is an example that calculates the sum of all numbers from 0 to 9:



The summation is carried out with the following **Accumulate** node:

**15.3.167 Reverse**

Reverses a list.

**Inputs****List (Type: List<T>)**

The input list.

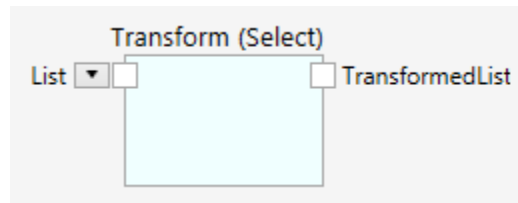
Outputs

Reverse (Type: List<T>)

The output list, with the items in reverse order.

15.3.168 Transform (Select)

Transform (Select) takes items from a list and transforms them to a list of different items. The transformation is specified as a graphical pipeline, for a single item. All transformed items are added to the output list.



Outside View

At the outside, the **Transform (Select)** node appears like a single node, with inputs and outputs.

Inputs

List (Type: List<T>)

The input list.

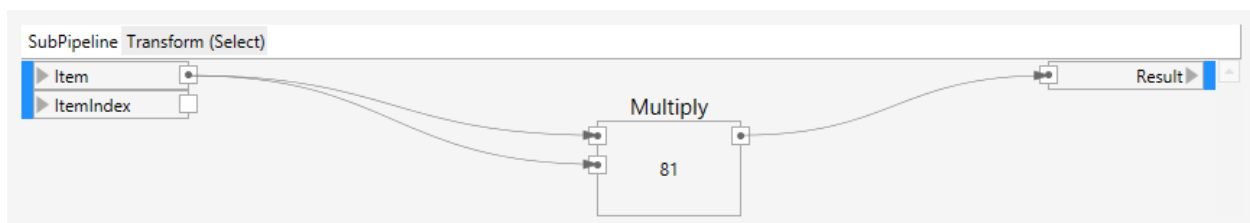
Outputs

TransformedList (Type: List<U>)

The output list, consisting of a list of transformed itmes from the input list.

Inside View

Inside, the **Transform (Select)** node looks like a normal branched pipeline. This pipeline is used to specify the transformation, which is evaluated for each item.



Inputs

Item (Type: τ)

One item of the input list.

ItemIndex (Type: Int_{32})

The index of the item of the input list.

Outputs

Result (Type: υ)

The result item. The result item can be of the same type as the input item or of a different type.

Comments

Transform (Select) nodes can be nested in similar ways as subpipelines.

The nesting is shown with a breadcrumbs control at the top.

First Level Second Level Third Level

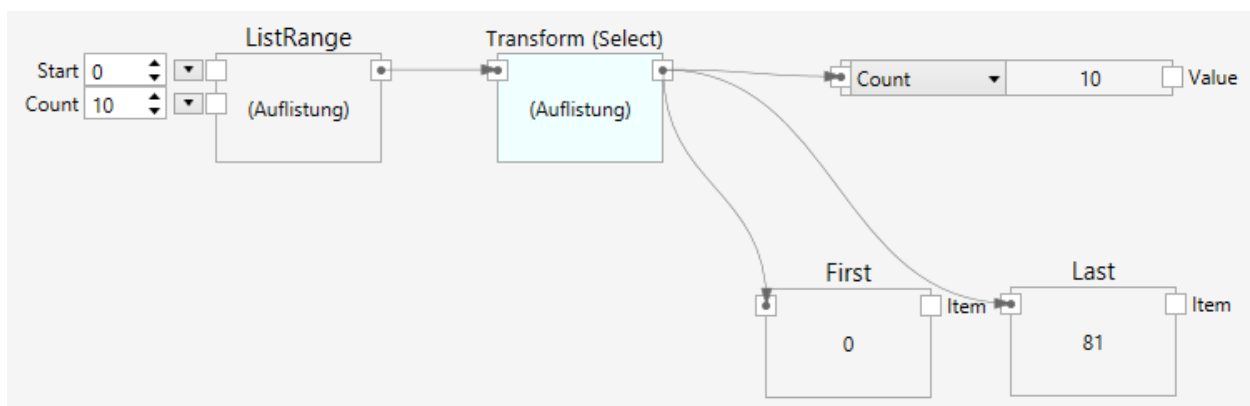
Double-click a **Transform (Select)** node to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

By default, a **Transform (Select)** node has one input and one output.

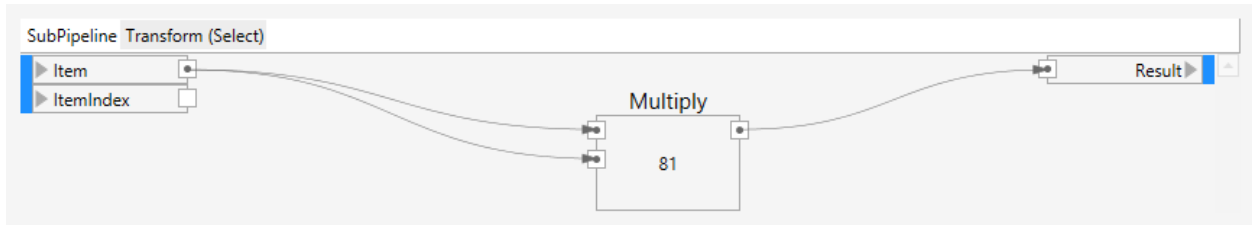
You can add additional inputs with the **Add Input (Ctrl-I)** command. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

Sample

Here is an example:

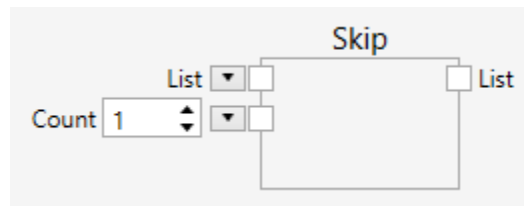


It takes items from the first list (0, 1, ..., 9) and transforms it to a list with each item squared (0, 1, ..., 81).



15.3.169 Skip

Skips a number of items at the beginning of a list. The remaining items are put in the output list.



Inputs

List (Type: List<T>)

The input list.

Count (Type: Int32)

The number of items to skip.

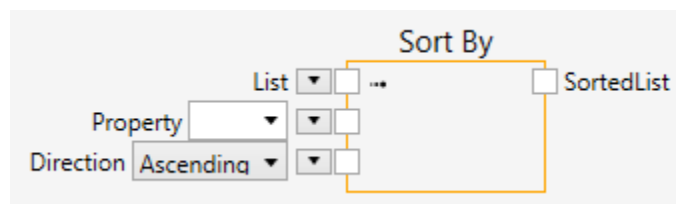
Outputs

List (Type: List<T>)

The output list, consisting of the part of the input list that remains after skipping.

15.3.170 Sort By

Sorts a list of items by some property.



Sort By can be applied on objects which have a property that can be used for sorting. The sort order can be specified.

Inputs

List (Type: `List<T>`)

The list of items.

Property (Type: `string`)

A property or a property path, such as `X` (`Point` has a property named `X`) or `Size.X` (`Image` has a property named `Size`, which in turn has a property named `X`).

Direction (Type: `string`)

The sort direction, use `Ascending` or `Descending` to sort in the respective direction.

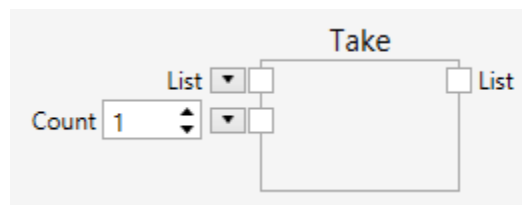
Outputs

SortedList (Type: `List<T>`)

The sorted list.

15.3.171 Take

Takes a number of items at the beginning of a list. The remaining items are not put in the output list.



Inputs

List (Type: `List<T>`)

The input list.

Count (Type: `Int32`)

The number of items to take.

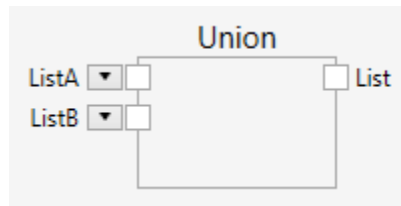
Outputs

List (Type: `List<T>`)

The output list, consisting of the beginning part of the input list that has been taken.

15.3.172 Union

Calculates the union of two sets.



Inputs

ListA (Type: List<T>)

The first input list, whose distinct elements form the first set for the union.

ListB (Type: List<T>)

The second input list, whose distinct elements form the second set for the union.

Outputs

List (Type: List<T>)

The output list, which consists of the set union of the two lists.

Comments

The lists are treated as mathematical sets, which can have distinct objects only.

15.3.173 Filter (Where)

Filter (Where) takes those items from a list where a predicate is true. The predicate is specified itself as a graphical pipeline. The predicate is evaluated for each item of the list, beginning with the first item. Items where the predicate evaluates to true are added to the output list.



Outside View

At the outside, the **Filter (Where)** node appears like a single node, with inputs and outputs.

Inputs

List (Type: List<T>)

The input list.

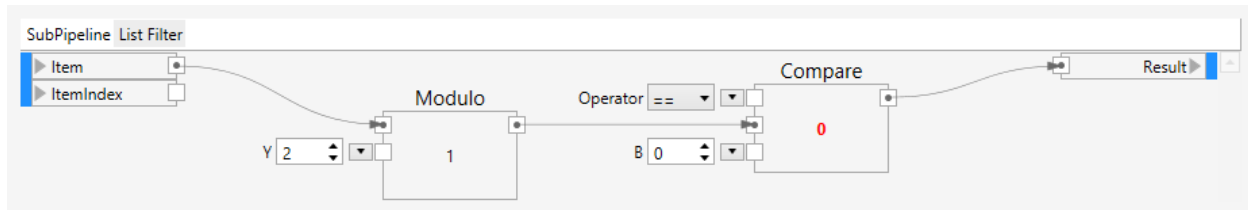
Outputs

FilteredList (Type: List<T>)

The output list, consisting of a part of the input list, as determined by consecutive execution of the predicate.

Inside View

Inside, the **Filter (Where)** node looks like a normal branched pipeline. This pipeline is used to specify the predicate, which is evaluated for each item.



Inputs

Item (Type: T)

One item of the input list.

ItemIndex (Type: Int32)

The index of the item of the input list.

Outputs

Result (Type: Boolean)

The boolean predicate.

Comments

Filter (Where) nodes can be nested in similar ways as subpipelines.

The nesting is shown with a breadcrumbs control at the top.

Double-click a **Filter (Where)** node to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

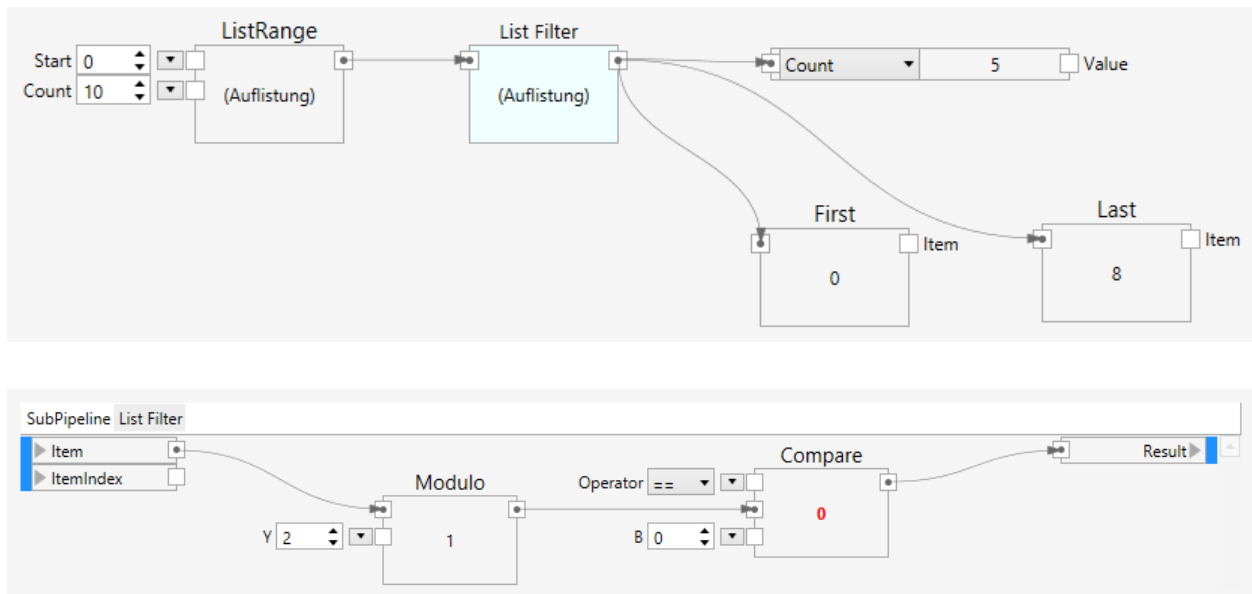
First Level Second Level Third Level

By default, a **Filter (Where)** node has one input and one output.

You can add additional inputs with the **Add Input (Ctrl-I)** command. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

Sample

Here is an example:



It takes items from the first list (0, 1, ..., 9) where the predicate (is even) is true and returns the resulting list (0, 2, ..., 8).

15.3.174 Load File

Loads an entire file.

Inputs

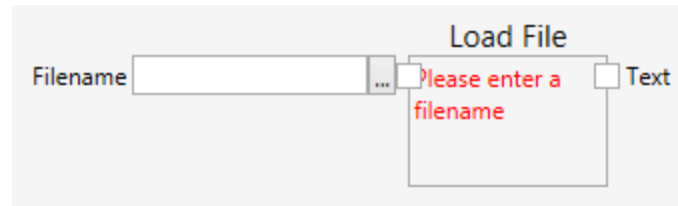
Filename

The path to the file.

Outputs

Text

The contents of the file.

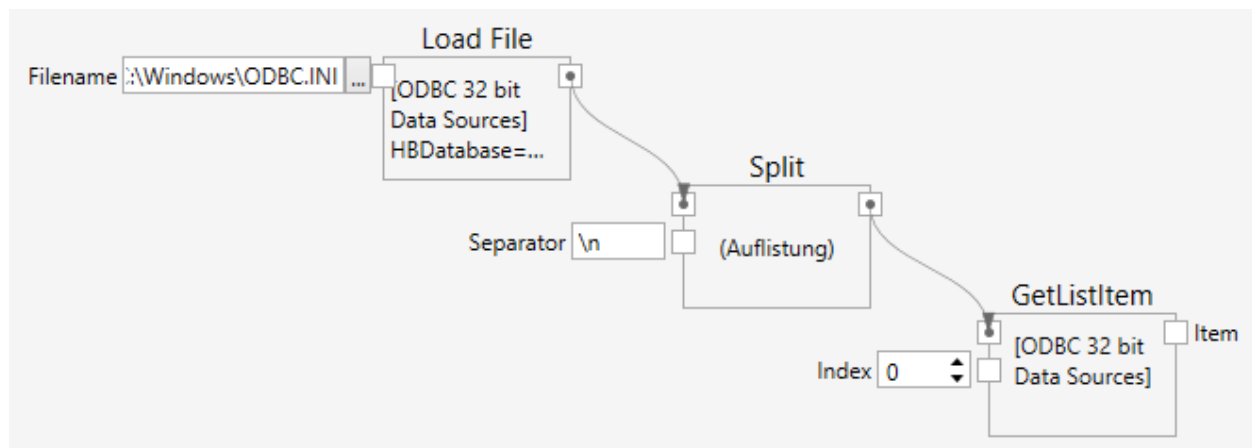


Comments

The **Load File** node checks the time the file was last written to, and compares it with the time it last read from the file. If the file has been written to after it has been last read, the node reads the file again.

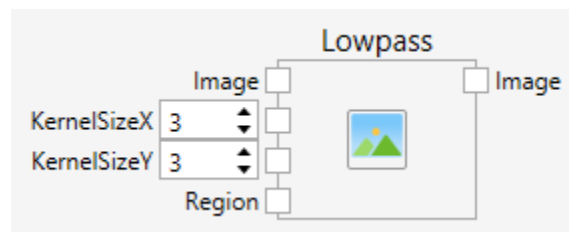
Sample

You can use standard text operations to work with the file contents. Here is an example that splits a file into lines.



15.3.175 Lowpass

Perform a lowpass filter.



Inputs

Image (Type: Image)

The input image.

KernelSizeX (Type: Int32)

The kernel width.

KernelSizeY (Type: Int32)

The kernel height.

Region (Type: Region)

Specifies an optional area of interest.

Outputs**Image (Type: Image)**

The output image.

Comments

The function applies a lowpass filter. The kernel is rectangular with all $n \times m$ values set to $1 / (n*m)$.

The effect of a lowpass filter is a lowpass. The lowpass filter emphasizes low frequencies and attenuates high frequencies. The strength of the lowpass filter depends on the size of the kernel.

Here are a few results of the lowpass filter with increasing kernel sizes:

Original:

KernelSizeX = KernelSizeY = 3:

KernelSizeX = KernelSizeY = 5:

KernelSizeX = KernelSizeY = 7:

KernelSizeX = KernelSizeY = 9:

Sample

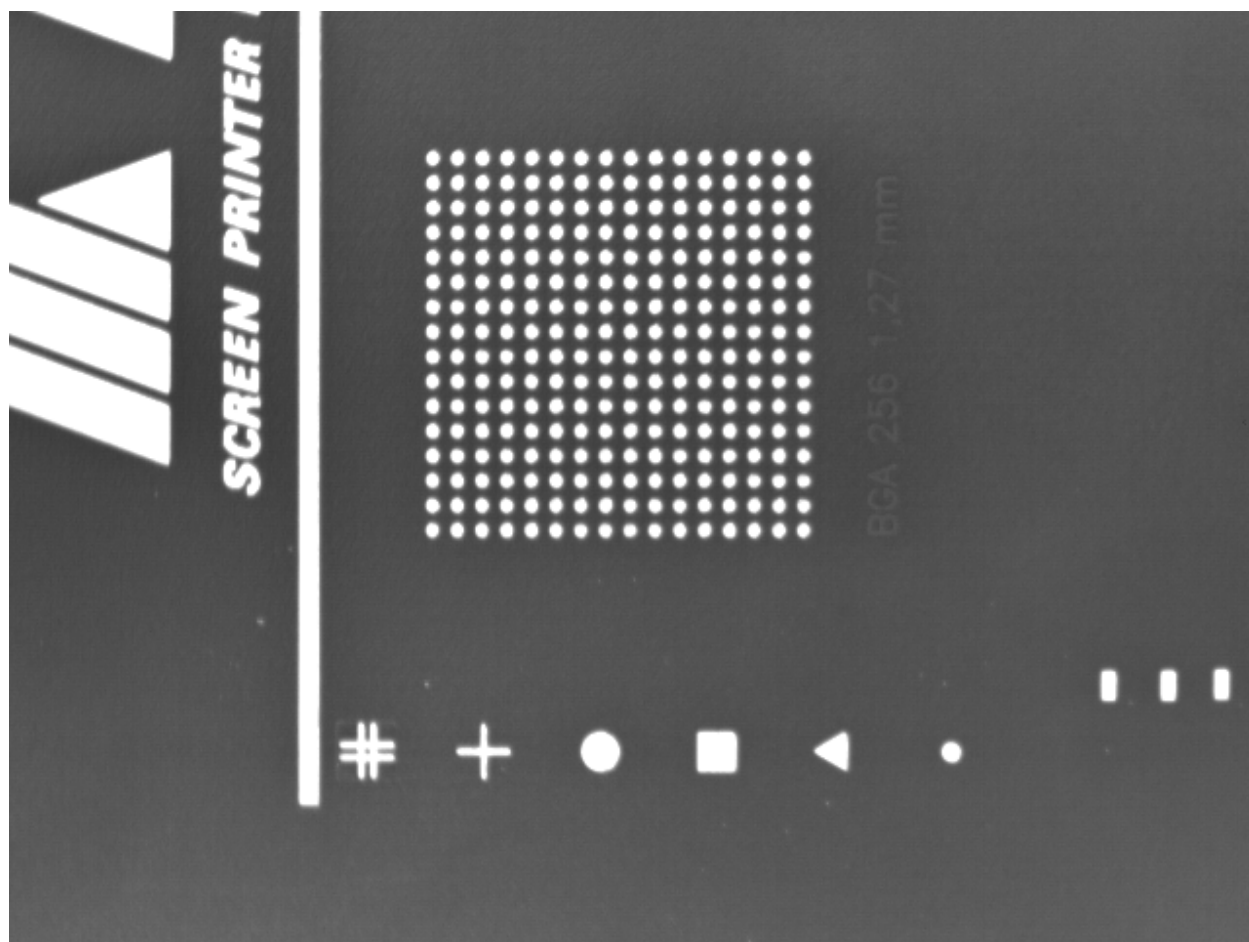
Here is an example that shows how to use the median filter.

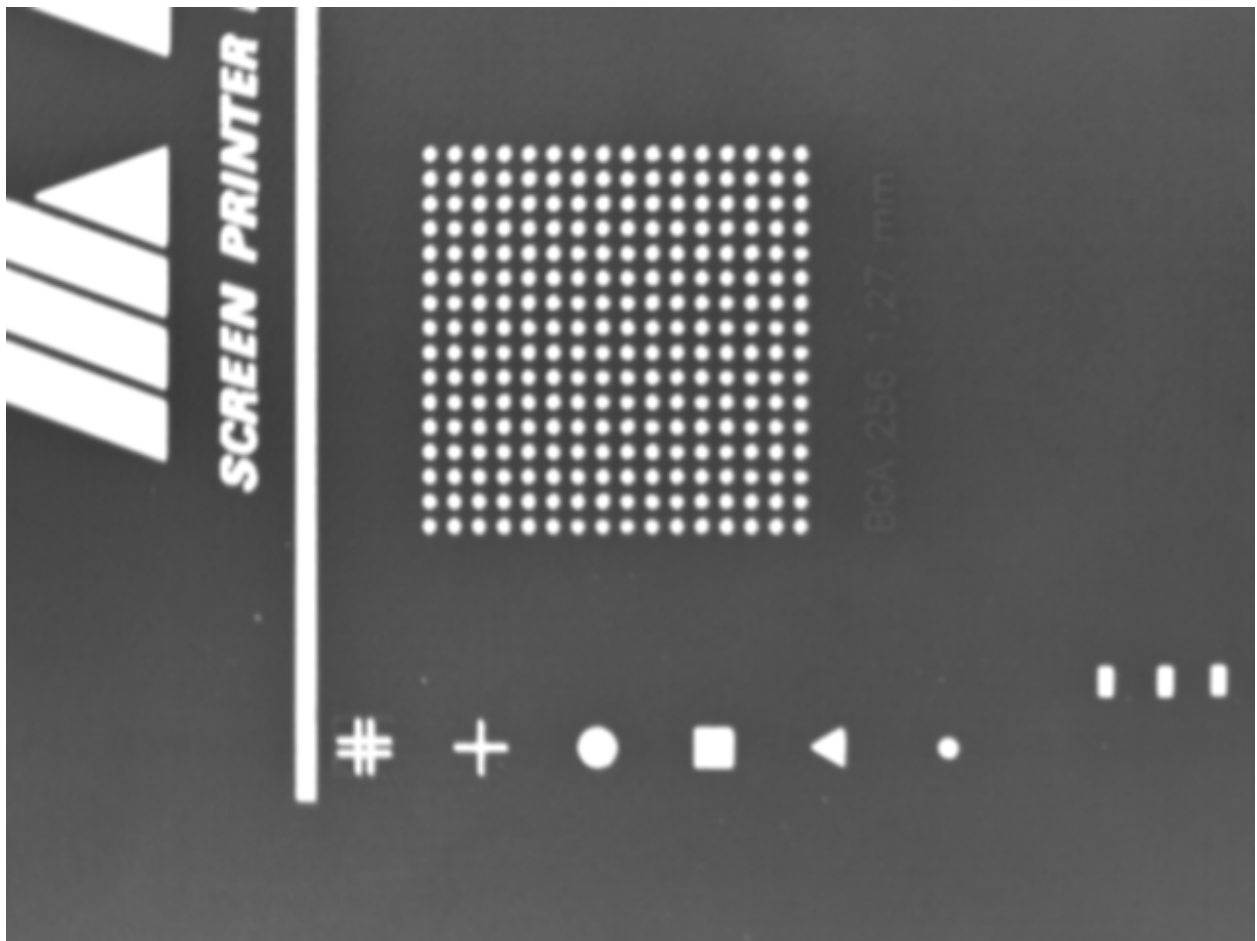
15.3.176 Lowpass

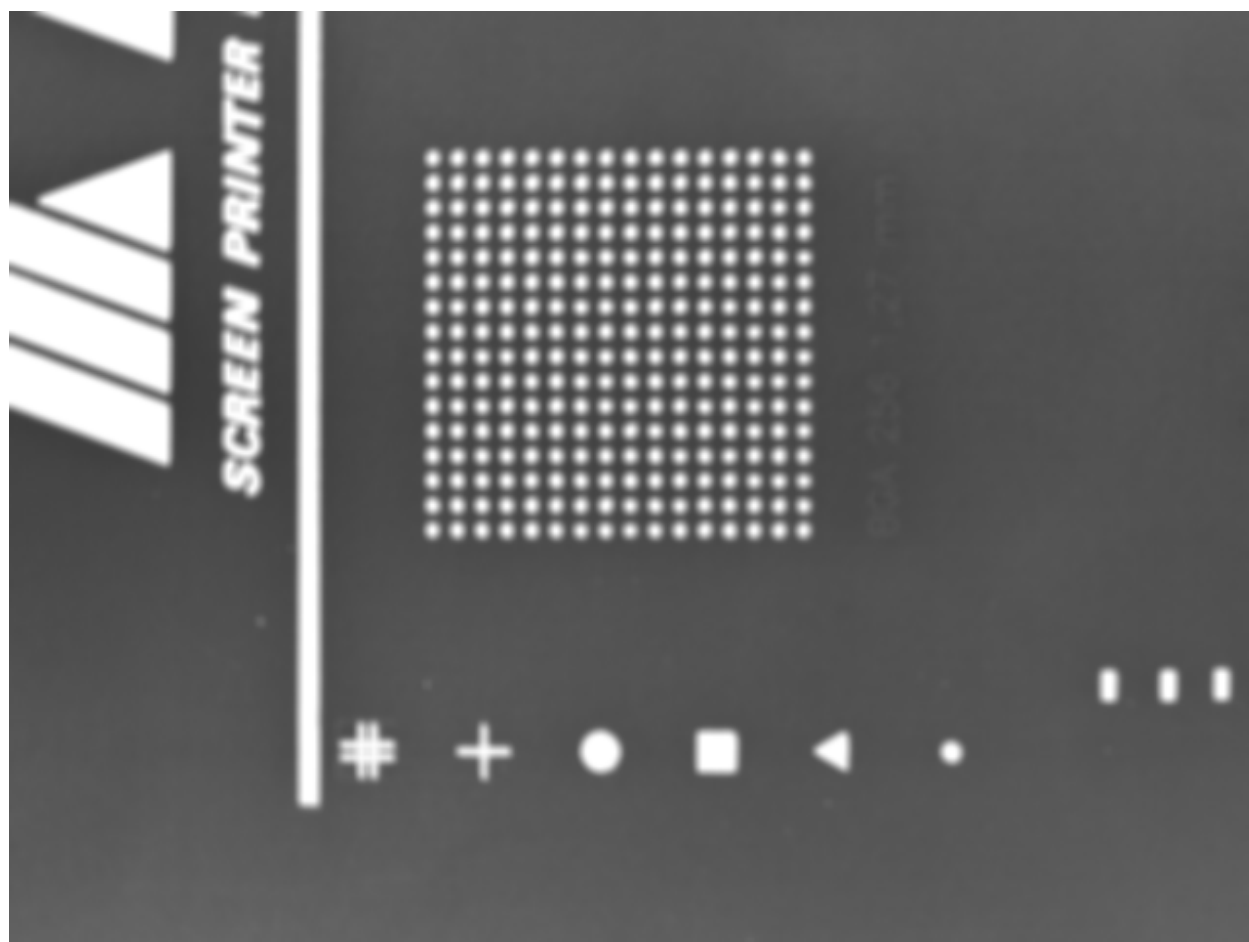
Filters an image using a lowpass kernel of rectangular size. A lowpass filter blurs an image.

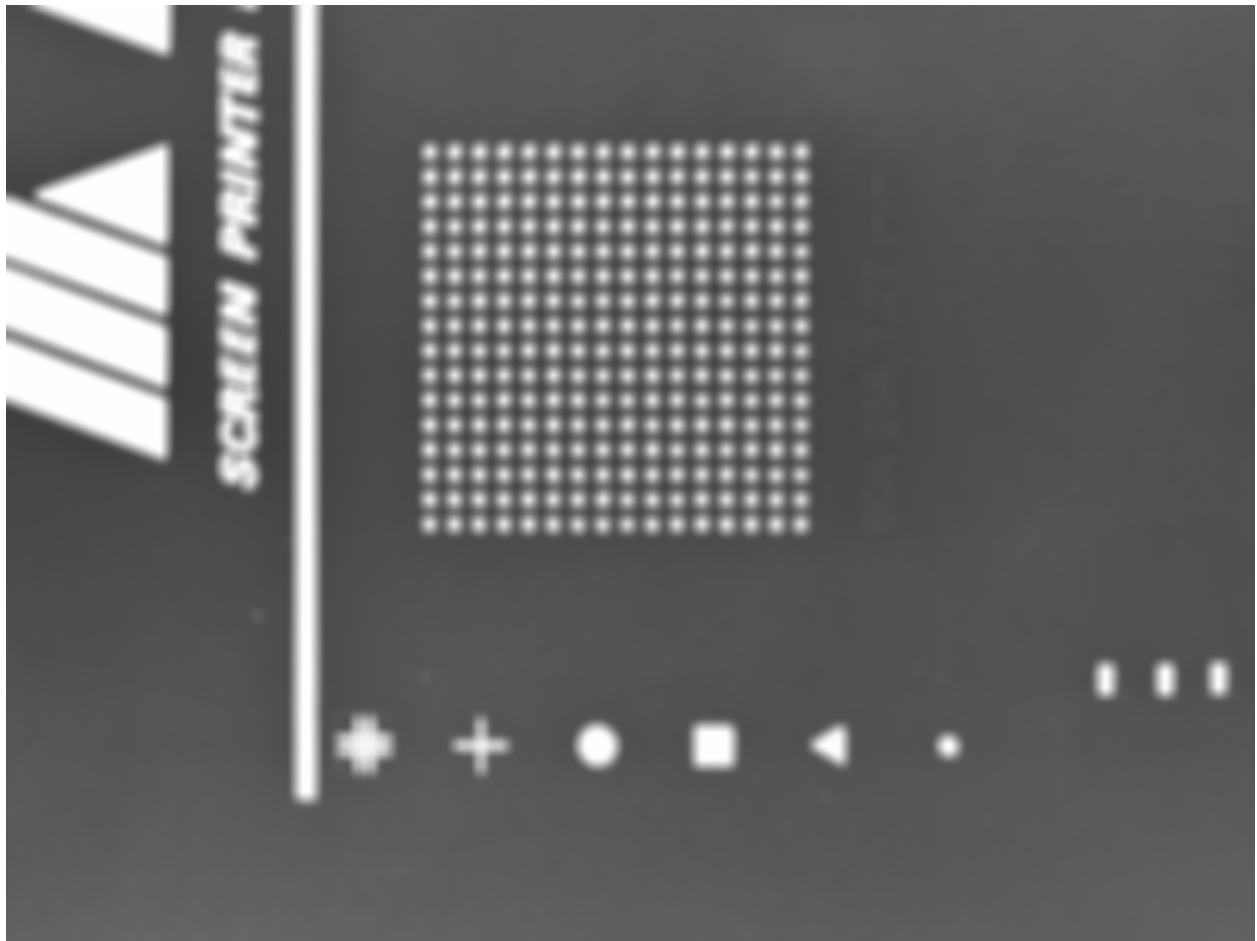
The lowpass filter emphasizes low frequencies and attenuates high frequencies. The strength of the lowpass filter depends on the size of the kernel.

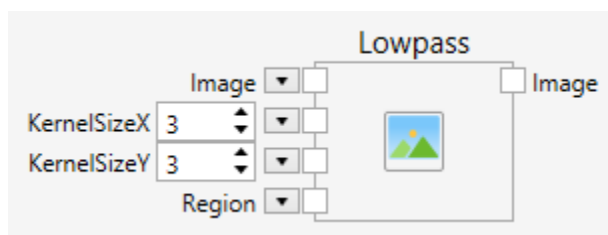
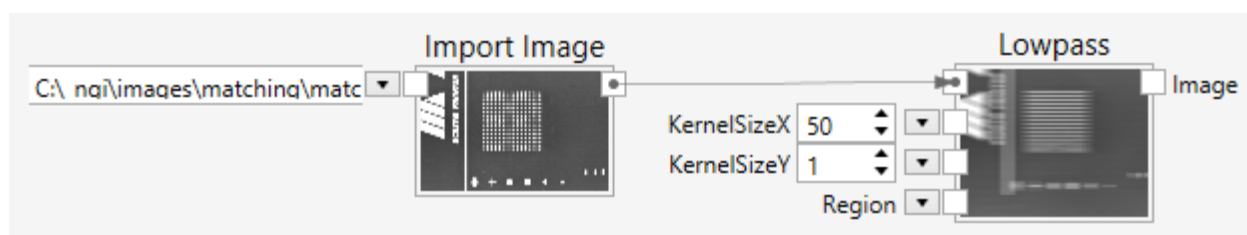
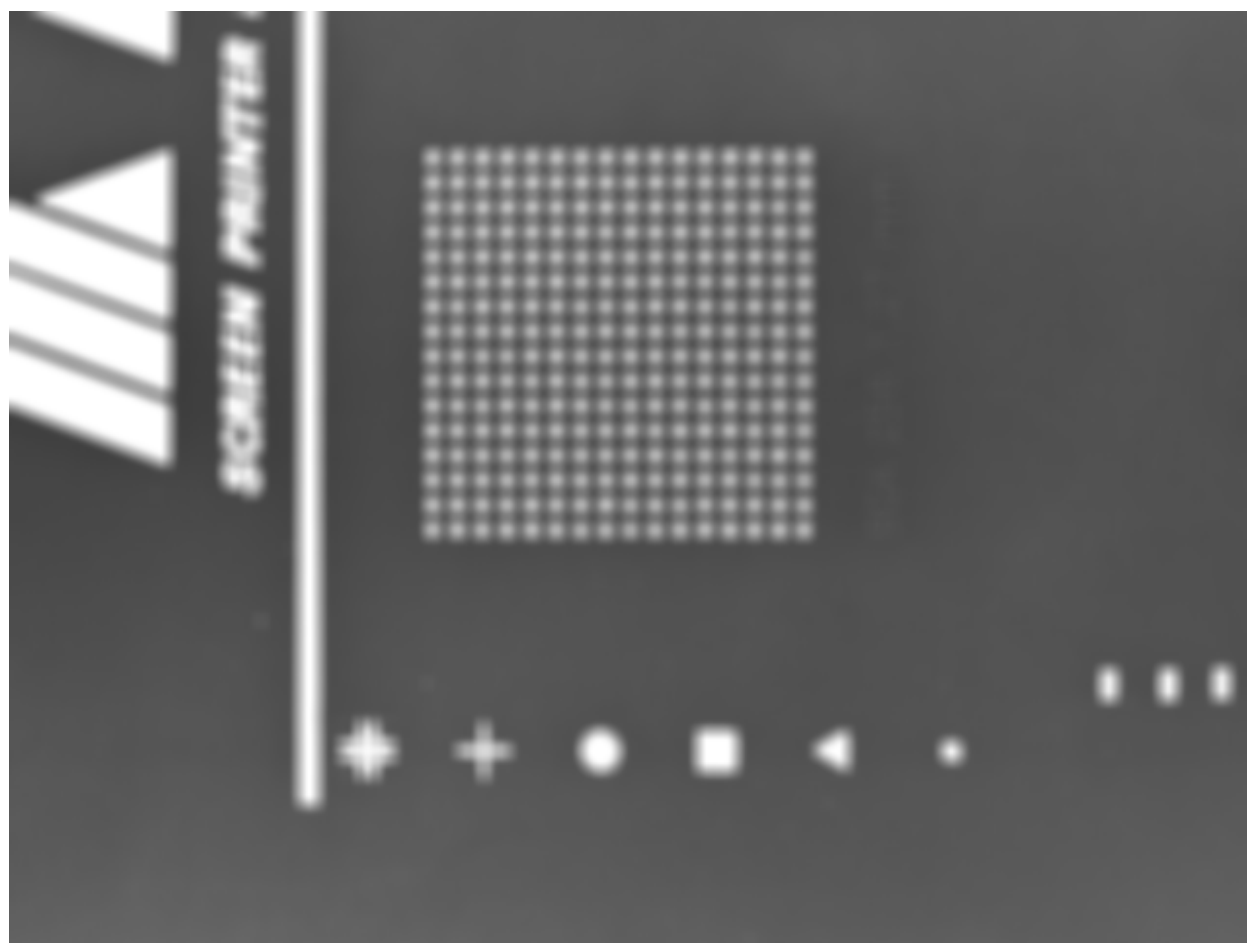
The kernel size is anisotropic and can be different in horizontal and vertical directions.











Inputs

Image (Type: Image)

The input image.

KernelSizeX (Type: Int32)

The horizontal kernel size (≥ 1).

KernelSizeY (Type: Int32)

The vertical kernel size (≥ 1).

Region (Type: Region)

Specifies an optional area of interest.

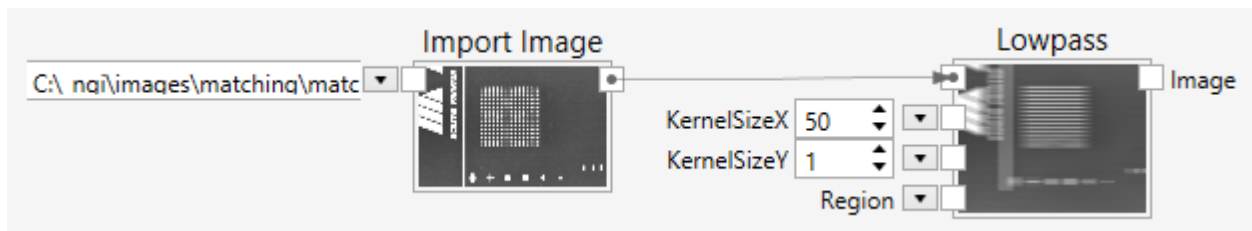
Outputs

Image (Type: Image)

The output image.

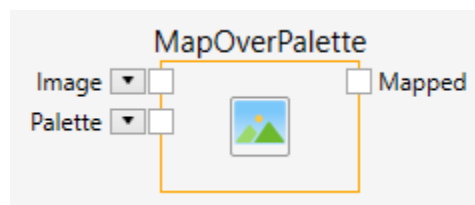
Sample

Here is an example:



15.3.177 MapOverPalette

Maps an image over a palette.



Inputs

Image (Type: Image)

The input image.

Palette (Type: Palette)

The mapping palette.

Outputs

Mapped (Type: Image)

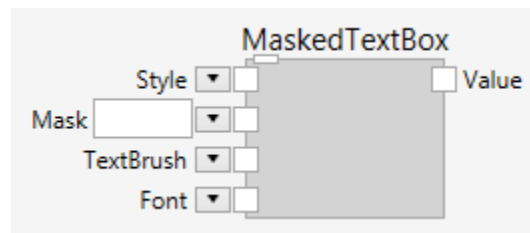
The output image.

Comments

For every pixel in the input image, the corresponding color entry in the palette is read and written to the output image.

15.3.178 MaskedTextBox

The **MaskedTextBox** is a HMI control that can be used to restrict user input based on a specified regular expression mask.



Inputs

Style (Type: Style)

Optional styling of the **MaskedTextBox**.

The **MaskedTextBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

Mask (Type: string)

The desired input pattern.

TextBrush (Type: SolidColorBrush)

The text color.

Font (Type: Font)

The font.

Outputs**Value (Type: string)**

The text entered by the user.

Description

The *Mask* input is used to define the desired pattern that must be input. Any characters that violate the mask will be discarded. This ensures that only data in the proper format is entered.

Regular Expression Masks

The mask is entered in the form of a regular expression, as described below.

Literals

Literal characters can be included in the regular expression, but special characters must be escaped using the backslash character (\).

Character Classes

Character classes can be used to define one or more characters, using ranges or by explicitly defining individual characters. Character classes are enclosed in square brackets. Ranges can be defined using a start character, followed by a dash (–), followed by an end character. For example, the character class "[0–9]", includes all digits. Additional ranges can be appended to specify discontinuous ranges. For example, the character class "[0–37–9]", includes all digits except 4, 5, and 6.

The caret (^) can be used to negate the character class. For example, the character class "[^0–9]" includes all characters except the digits.

Groups

Any number of regular expression elements can be grouped together by enclosing them in parenthesis. Alternations, which are described below, are a special type of group which are also enclosed in parenthesis. Groups can be repeated using one of the support quantifiers after the closing parenthesis.

For example, the regular expression "ABC*" requires A and B, then allows for zero or more of C. The regular expression "(ABC)*", which groups ABC into a single element, allows for zero or more of ABC together.

In terms of literal completion, only the A character would ever be entered by the end-user for the "(ABC) *" regular expression. When the user types an A, then the BC are required and are therefore automatically inserted by the control. If an additional A is typed, then again the BC would be auto completed, resulting in "ABCABC" as the text.

Alternation

Alternation allows for two or more alternate branches to be taken and is defined by using the vertical bar in parenthesis. For example, the "(AB|CD)" regular expression can match the string AB or the string CD. The alternate branches can be any other supported regular expression element, including other groups or alternations. This allows for very complex scenarios to be defined and used as a mask.

When showing prompt indicators for uncompleted sections of the masks, the control will choose the shortest branch of an alternation that would result in completion of the mask.

Quantifier

Quantifiers can be used to specify the number of times a regular expression element should be repeated. There are several methods for defining quantifiers, which are:

Quantifier	Description
?	Indicates that the regular expression element is optional. For example, the regular expression "AB?C" will match the string ABC or AC.
*	Indicates that the regular expression element can be repeated zero or more times, with no upper limit. For example, the regular expression "AB*C" will match the string ABC or AC, as well as ABBBBBBBC.
+	Indicates that the regular expression element must occur once, but can be repeated any number of times. For example, the regular expression "AB+C" will match the string ABC or ABBBBBBBC, but will not match AC.
{N}	Indicates that the regular expression element must occur exactly N times, where N is a positive integer value. For example, the regular expression "AB{2}C" will only match the string ABBC.
{N,}	Indicates that the regular expression element must occur exactly N times, where N is a positive integer value, but can be repeated any number of times. For example, the regular expression "AB{2,}C" will match the string ABBC or ABBBBBBBC.
{N,M}	Indicates that the regular expression element must occur exactly N times, where N is a positive integer value, but can be repeated M times, where M is a positive integer value greater than N. For example, the regular expression "AB{2,3}C" will match the string ABBC or ABBBC, but not ABBBBBC.

Prompt Indicators

The control will render one or more prompt indicators (a circle) when additional input is required. Literals will also be rendered ahead of the user input, when possible.

Caret

The caret is used to indicate where the next typed character (or pasted text) will be inserted.

The caret is only shown when the control currently has the keyboard focus.

Example

Here is an example that shows the various edit controls. This definition:
creates the following user interface:

15.3.179 Max

Finds the maximum of two or more numbers.

Inputs

A (Type: Double)

The first operand.

B (Type: Double)

The second operand.

C...Z (Type: Double)

Additional operands, added on demand.

Outputs

Max (Type: Double)

The maximum of all operands.

Comments

This node calculates the maximum of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

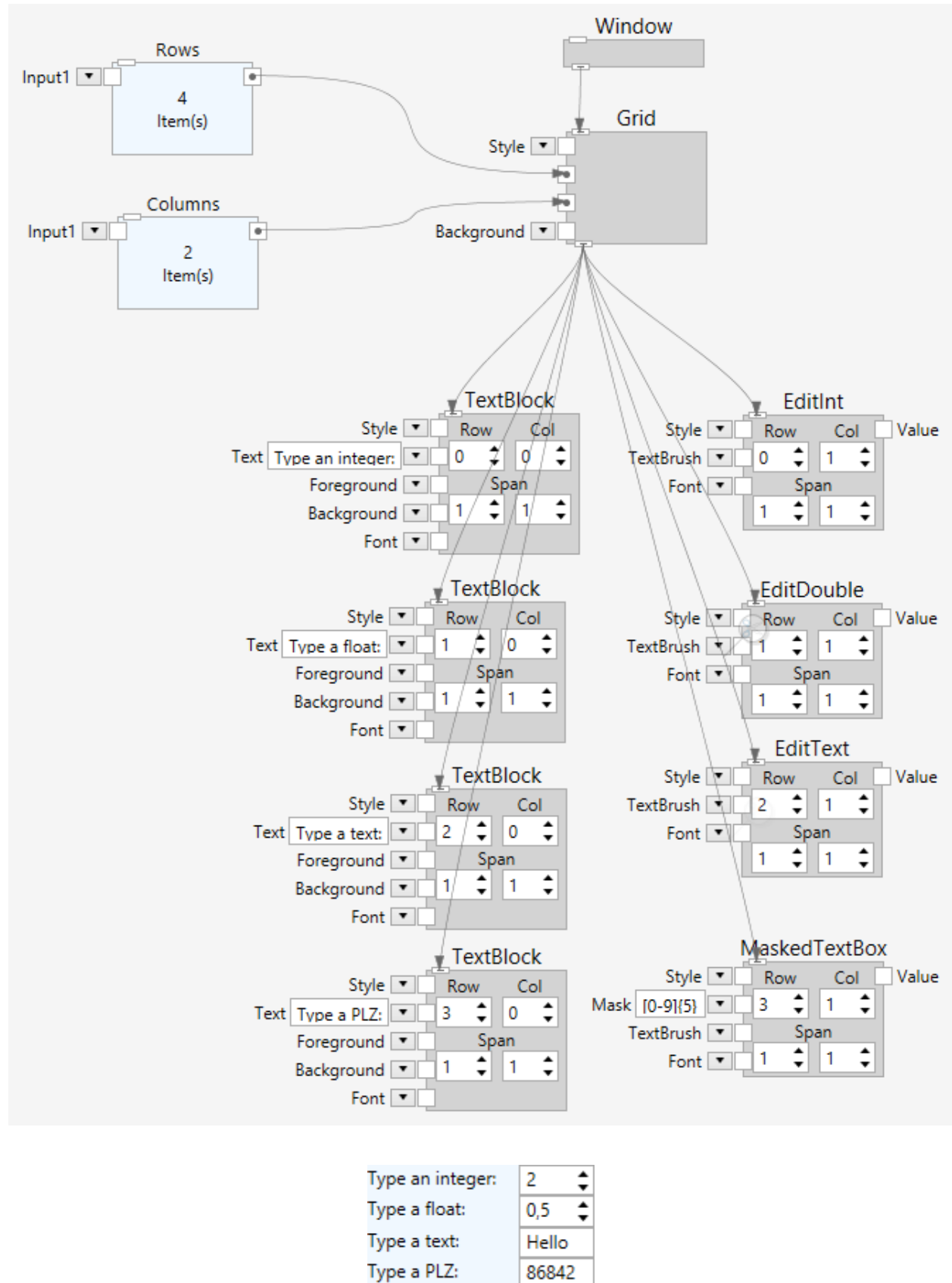
15.3.180 Median

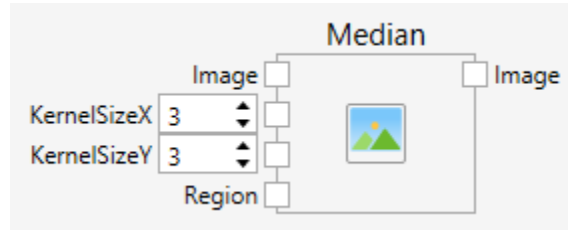
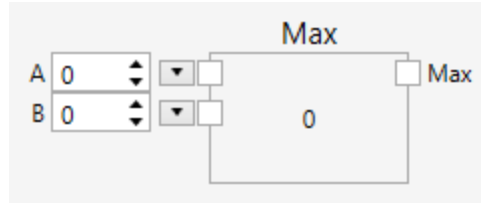
Perform a median filter.

Inputs

Image (Type: Image)

The input image.



**KernelSizeX (Type: Int32)**

The kernel width.

KernelSizeY (Type: Int32)

The kernel height.

Region (Type: Region)

Specifies an optional area of interest.

Outputs**Image (Type: Image)**

The output image.

Comments

The median is a noise reduction filter, which often performs better than simple averaging, especially for salt-and-pepper noise.

For every pixel, the median filter selects the median value of all values in the neighborhood of the pixel.

Here are a few results of the median filter with increasing kernel sizes:

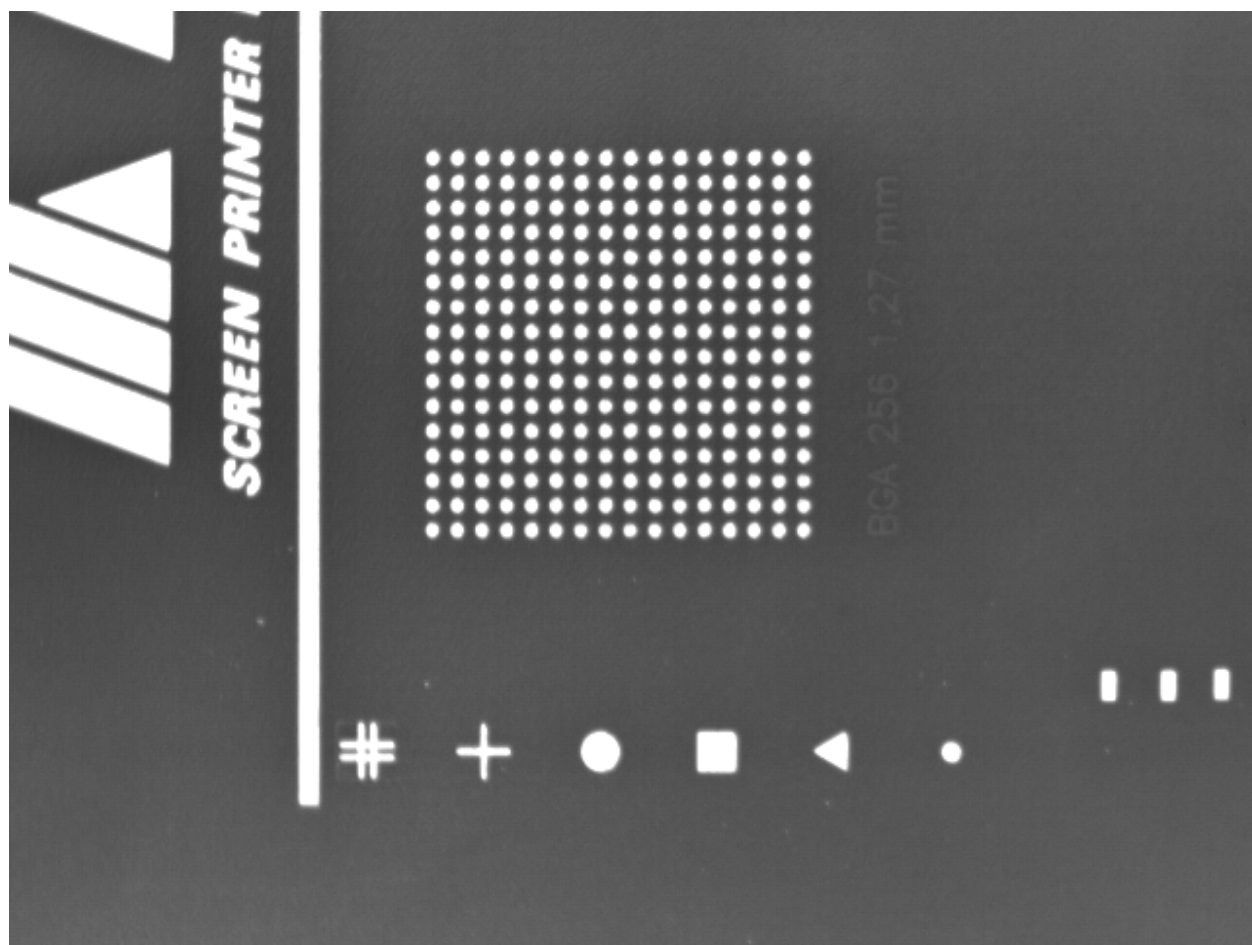
Original:

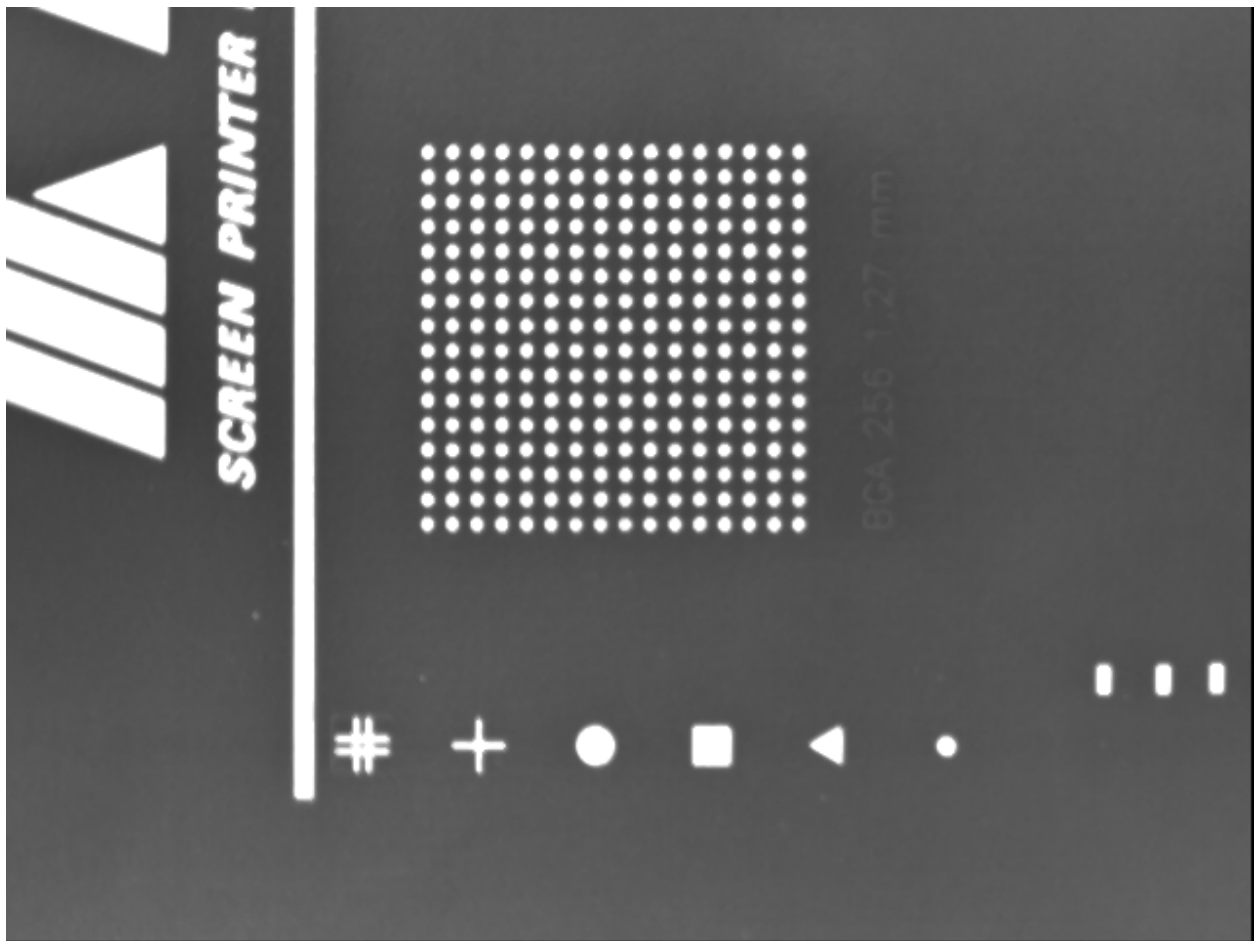
KernelSizeX = KernelSizeY = 3:

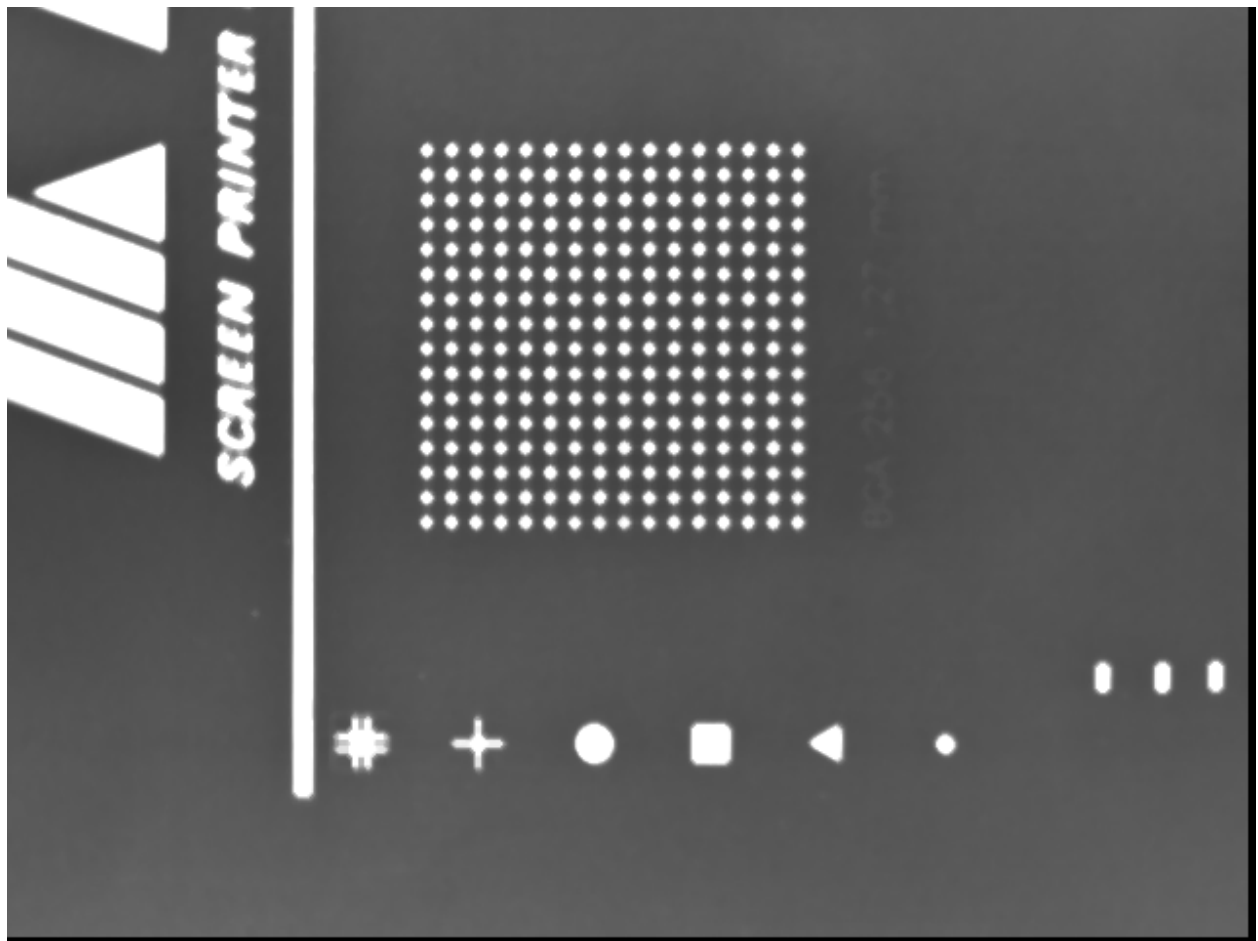
KernelSizeX = KernelSizeY = 5:

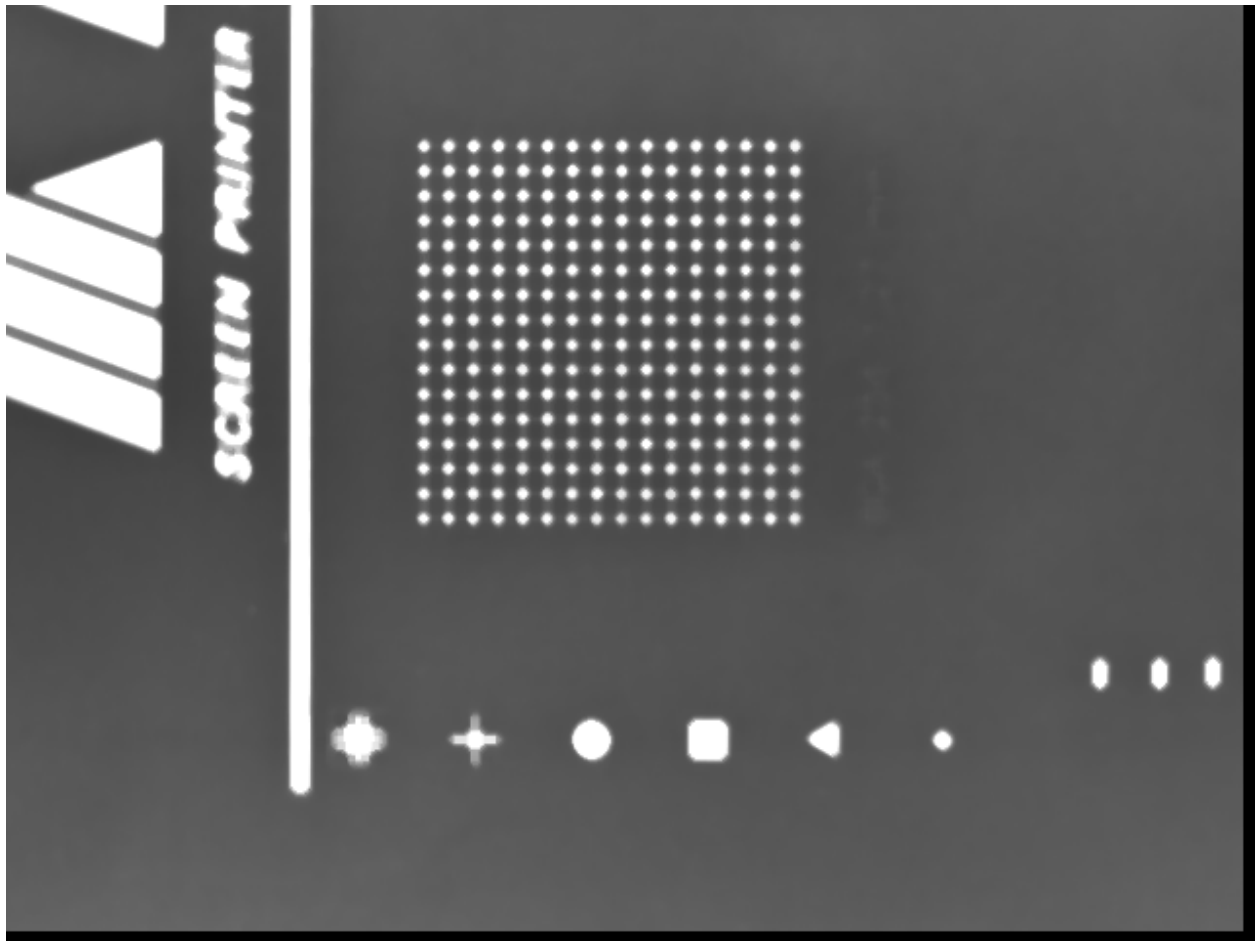
KernelSizeX = KernelSizeY = 7:

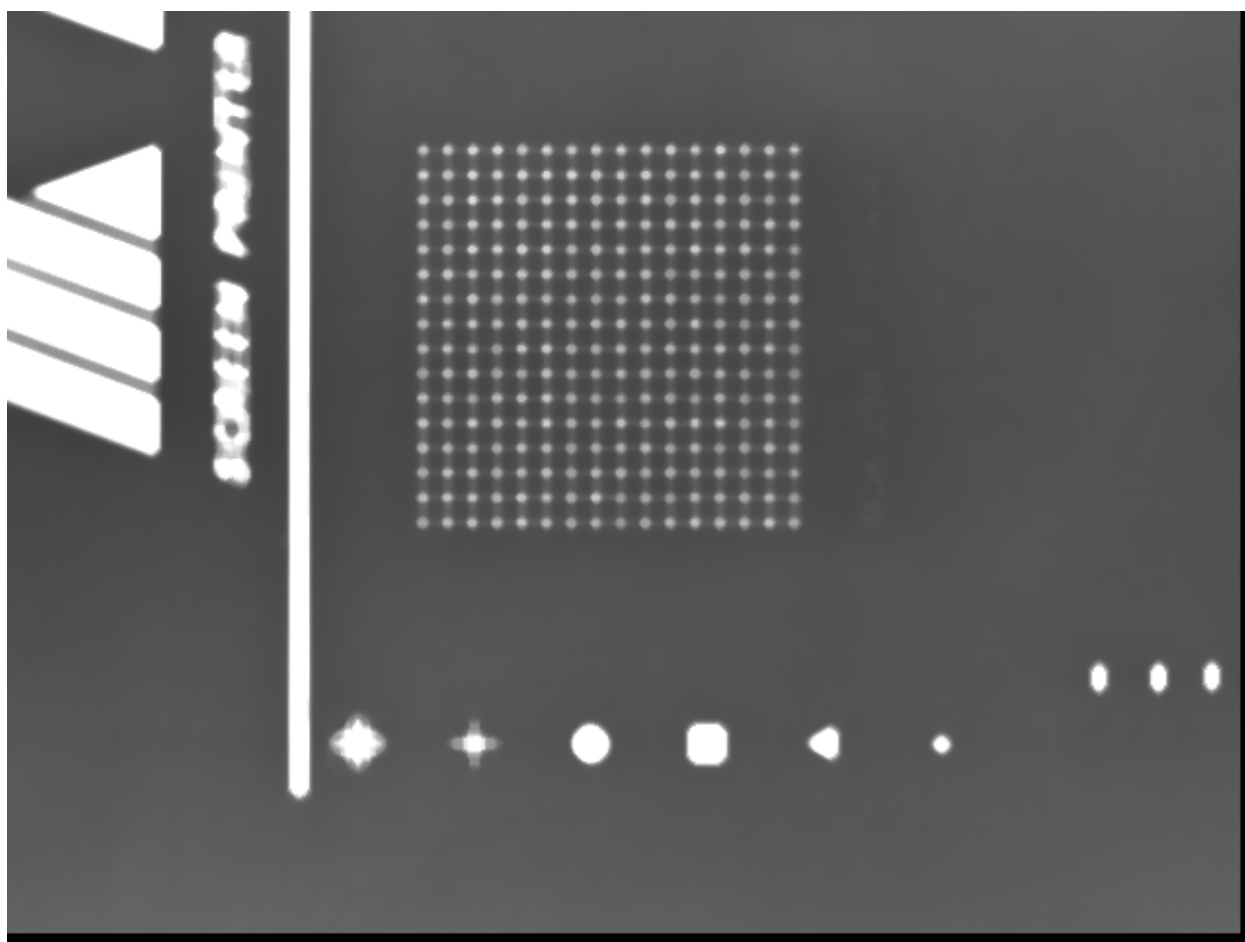
KernelSizeX = KernelSizeY = 9:





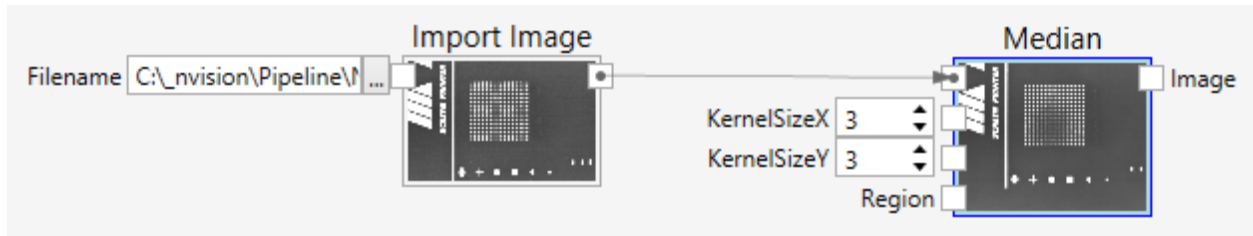






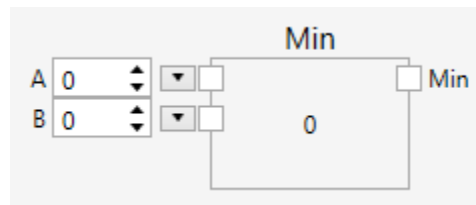
Sample

Here is an example that shows how to use the median filter.



15.3.181 Min

Finds the minimum of two or more numbers.



Inputs

A (Type: Double)

The first operand.

B (Type: Double)

The second operand.

C...Z (Type: Double)

Additional operands, added on demand.

Outputs

Min (Type: Double)

The minimum of all operands.

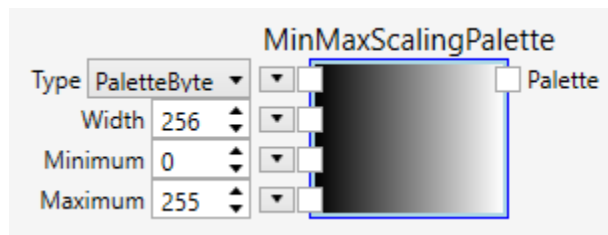
Comments

This node calculates the minimum of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.182 MinMaxScalingPalette

Creates a palette with a linear transfer function, given minimum and maximum values of an image.



Inputs

Type (Type: String)

The type of the palette. Choices are `PaletteByte`, `PaletteUInt16`, `PaletteUInt32`, `PaletteDouble`, `PaletteRgbByte`, `PaletteRgbUInt16`, `PaletteRgbUInt32` and `PaletteRgbDouble`.

Width (Type: Int32)

The width of the palette.

Minimum (Type: Double)

The minimal value.

Maximum (Type: Double)

The maximal value.

Outputs

Palette (Type: Palette)

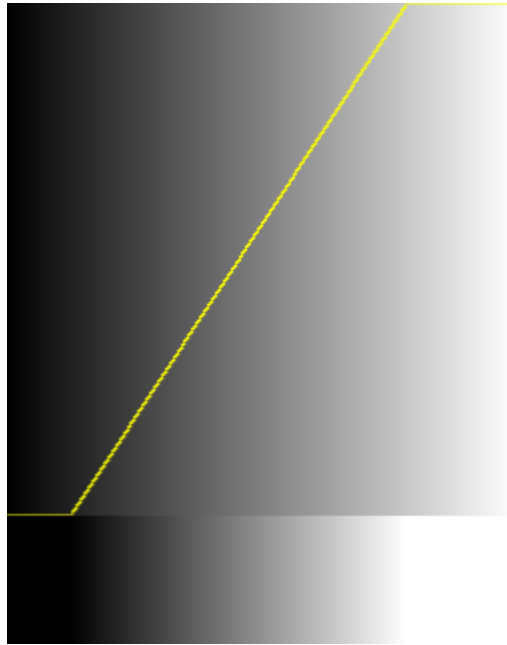
The output palette.

Comments

This function creates a palette with a linear function.

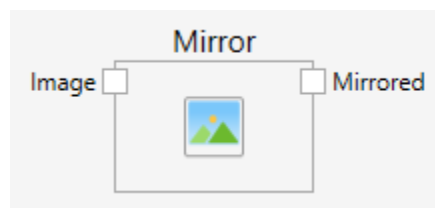
The minimum and maximum values are usually determined by inspecting an image. When the image is then mapped over a palette created with this function, it will be remapped to gray values between 0 and 255.

Here is an example of a linear curve with a minimum of 32 and a maximum of 200:



15.3.183 Mirror

Mirrors an image.



Inputs

Image (Type: Image)

The input image.

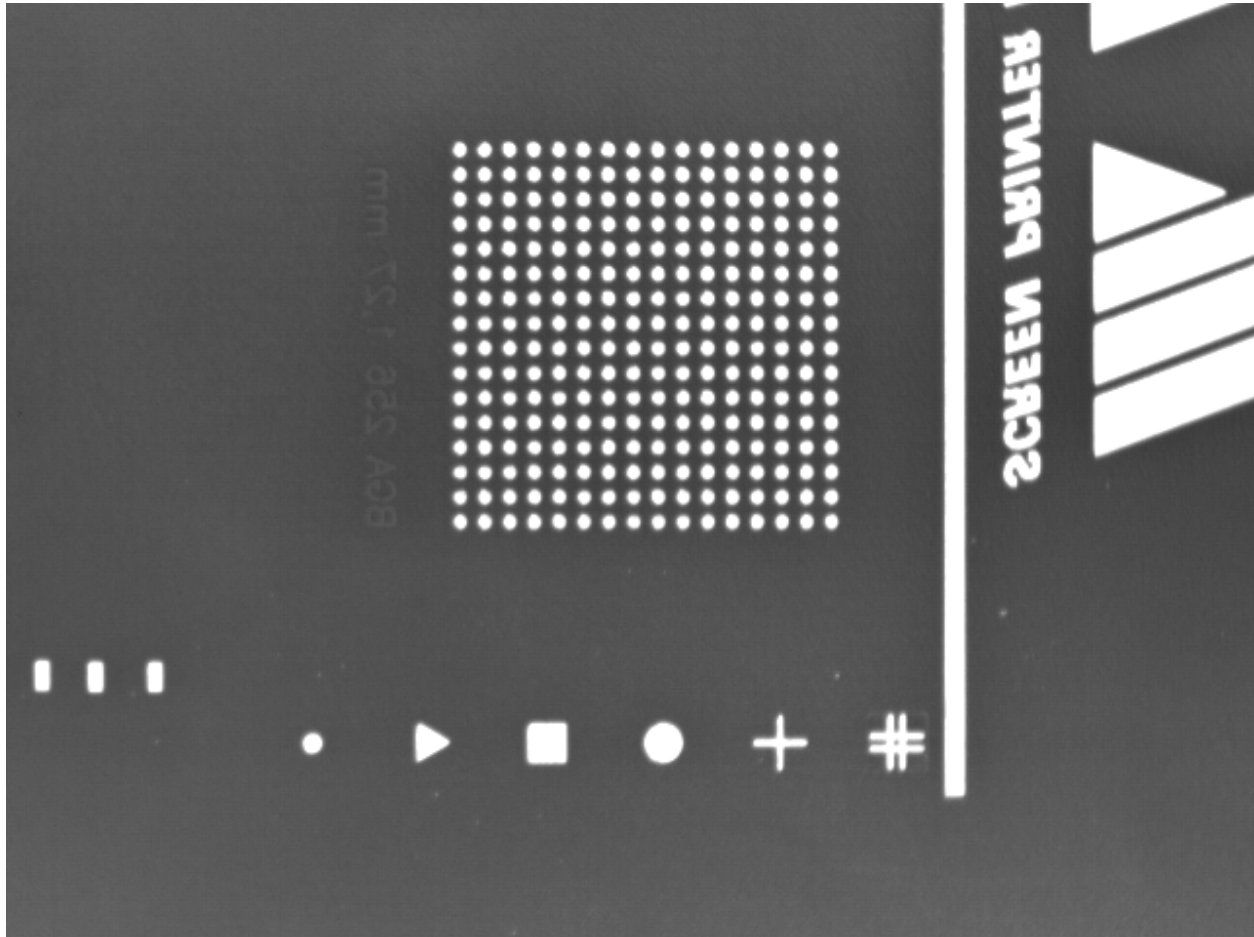
Outputs

Mirrored (Type: Image)

The output image.

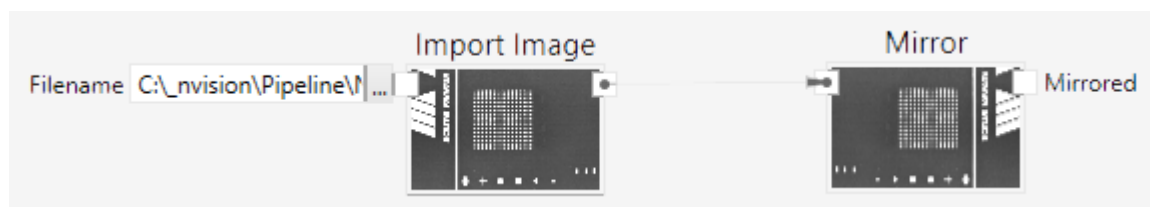
Comments

This function reverses an image in the horizontal direction (exchanges left and right).



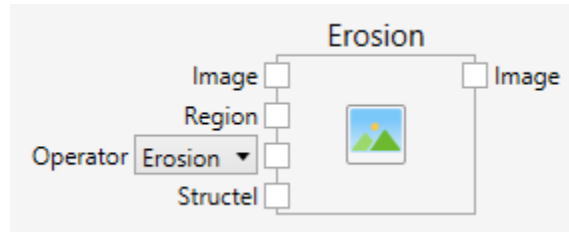
Sample

Here is an example that mirrors an image.



15.3.184 Morphological Image Operation

Performs a morphological image operation. Possible operations are erosion, dilation, opening, closing and morphological gradient.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Constrains the function to a region of interest.

Operator (Type: String)

Possible operators are *Erosion*, *Dilation*, *Closing*, *Opening* and *MorphologicalGradient*.

Structel (Type: Region)

A structuring element.

Outputs

Flipped (Type: Image)

The output image.

Comments

This function performs several morphological operations.

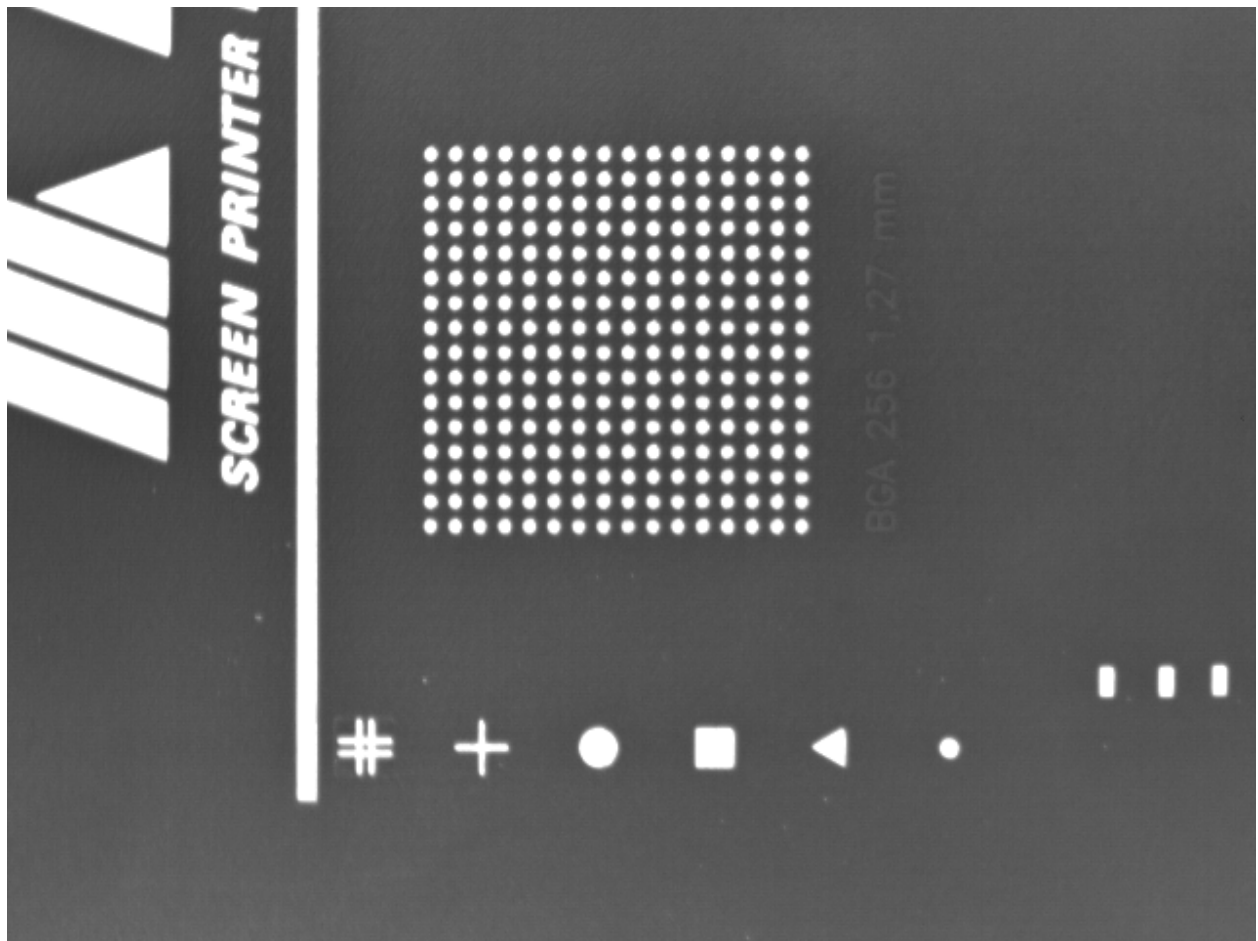
Given the following original image, we show the results of different morphological operations:

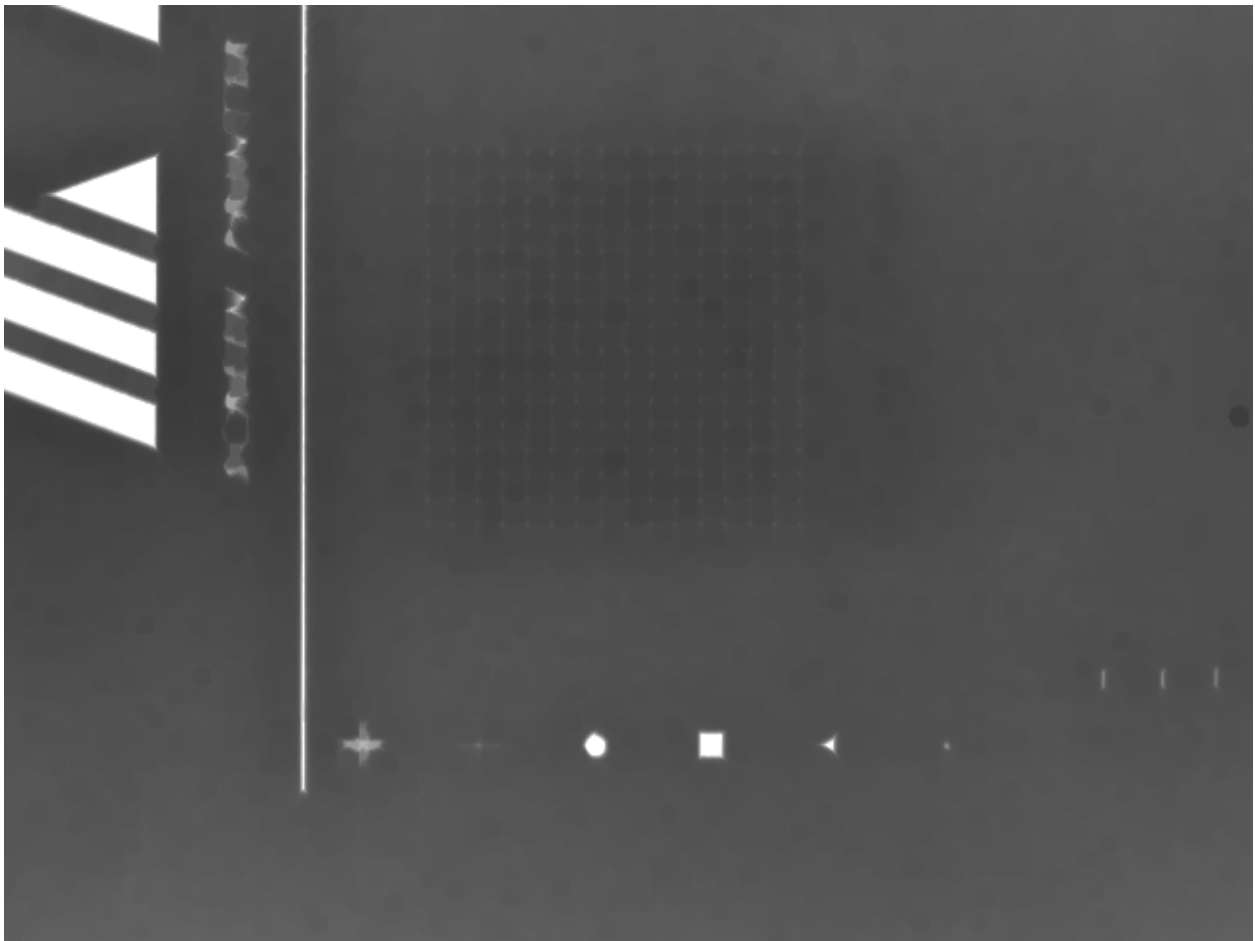
Here is an example of an erosion with a circular structuring element of diameter 10:

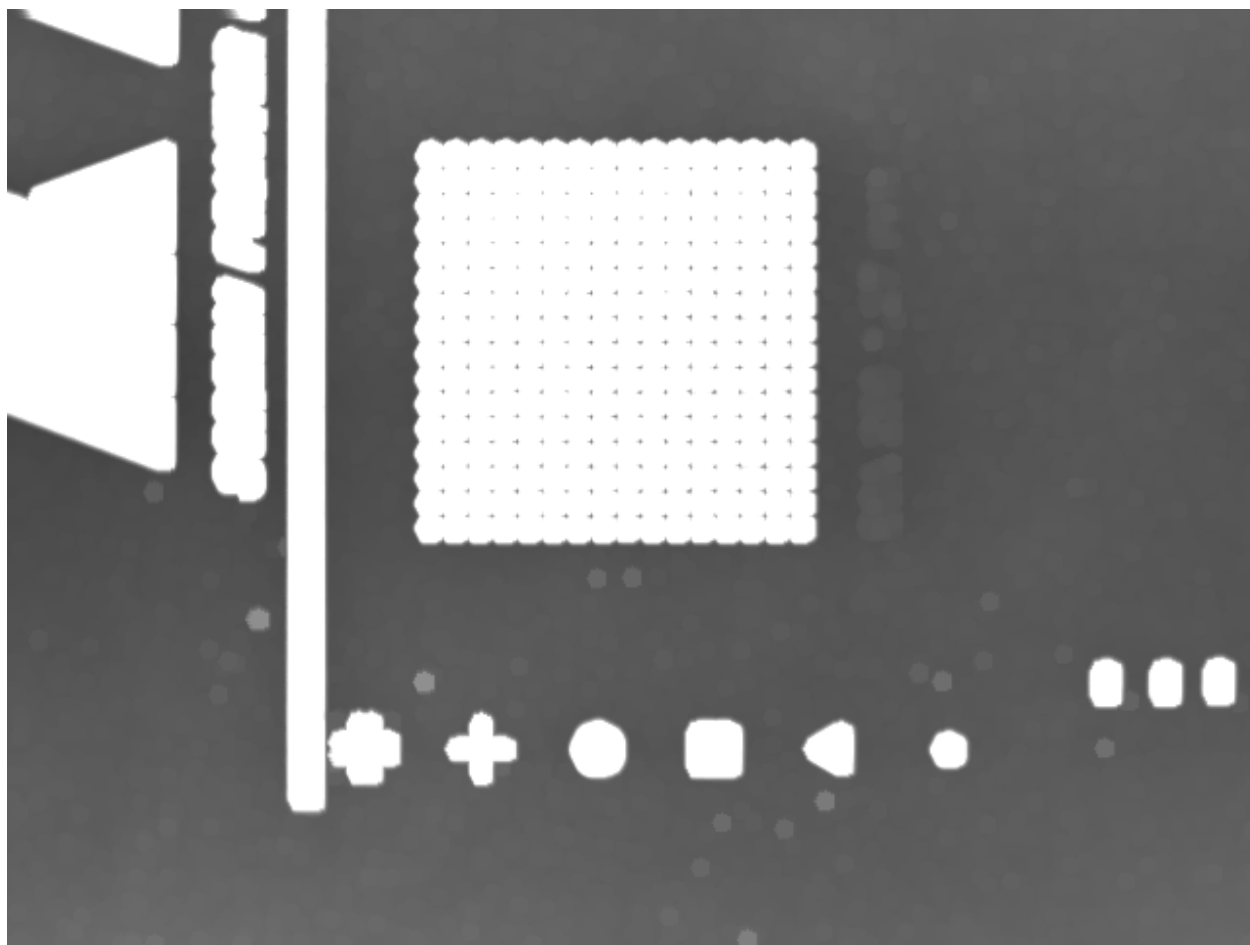
Here is an example of a dilation with a circular structuring element of diameter 10:

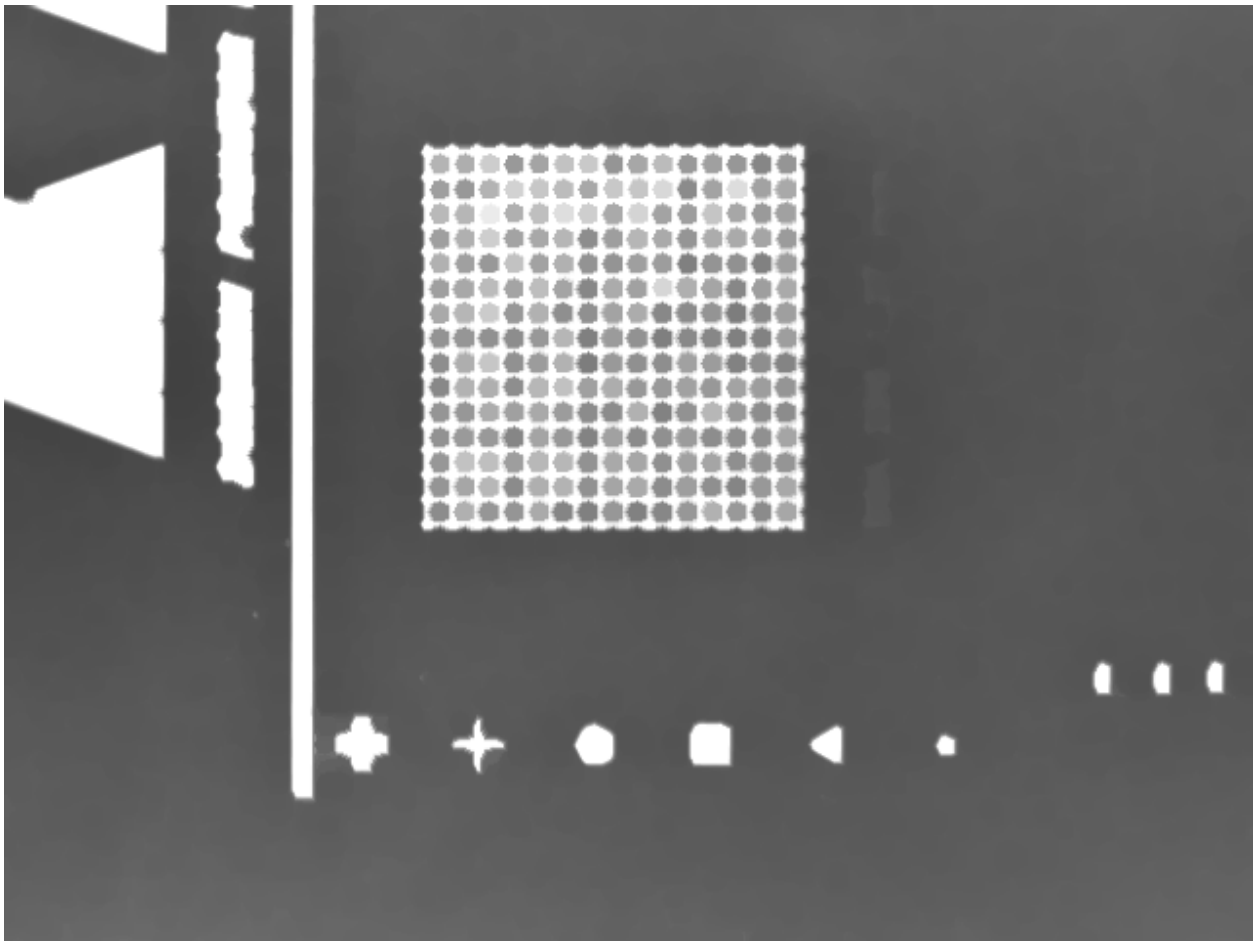
Here is an example of closing with a circular structuring element of diameter 10:

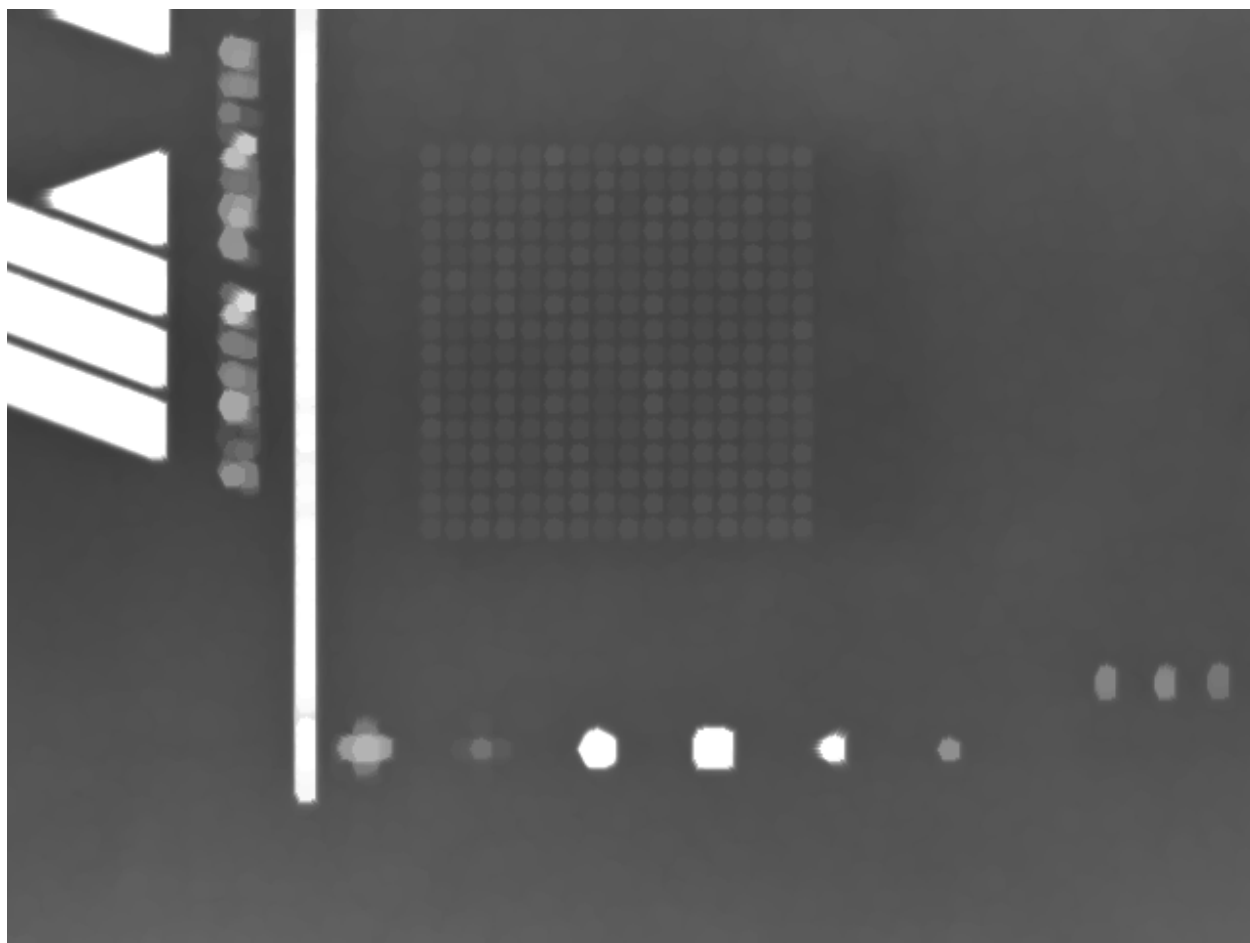
Here is an example of opening with a circular structuring element of diameter 10:



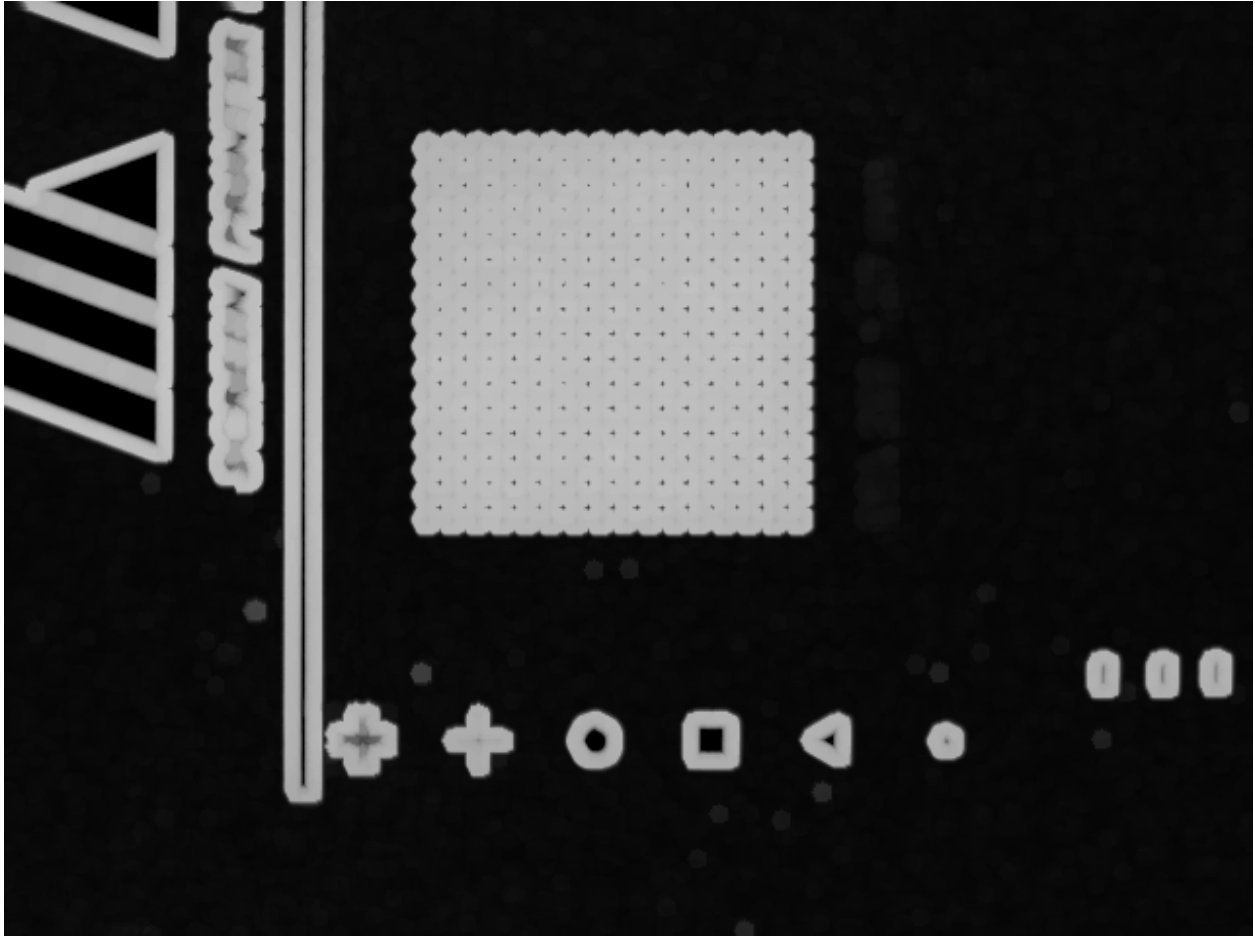








Here is an example of the morphological gradient with a circular structuring element of diameter 3:



Sample

Here is an example that erodes an image.

15.3.185 MorphologicalRegionListOperation

Applies a morphological operation to all regions in a list.

The following operators are available: erosion, dilation, closing, opening and morphological gradient.

Erosion

Calculates the erosion of regions in a list (regions are all the holes in the part).

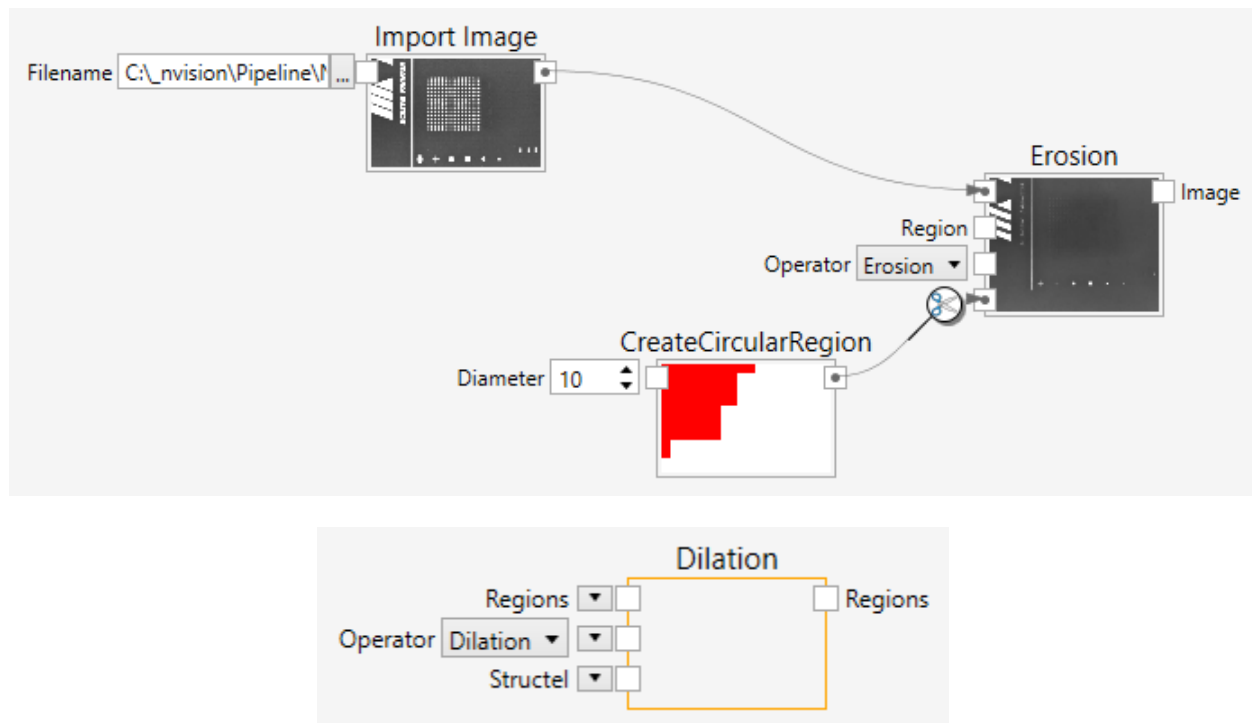
(Erosion with a circular structuring element of radius 10).

Dilation

Calculates the erosion of regions in a list (regions are all the holes in the part).

(Dilation with a circular structuring element of radius 10).

Closing



Calculates the closing of regions in a list (regions are all the holes in the part).

(Closing with a circular structuring element of radius 10).

Opening

Calculates the opening of regions in a list (regions are all the holes in the part).

(Opening with a circular structuring element of radius 10).

Morphological Gradient

Calculates the morphological gradient of regions in a list (regions are all the holes in the part).

(Morphological gradient with a circular structuring element of radius 5).

Inputs

Regions (Type: RegionList)

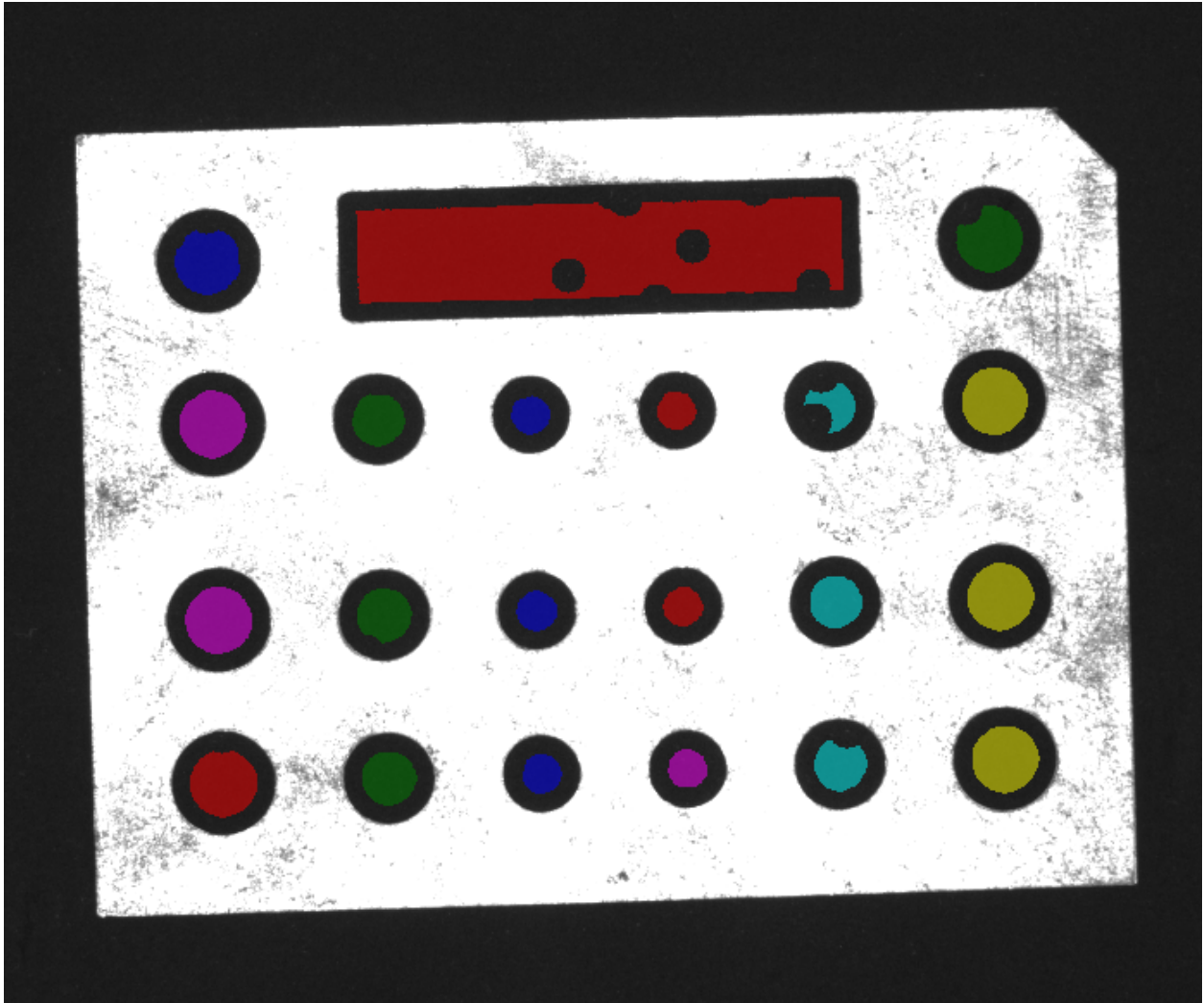
The list of regions.

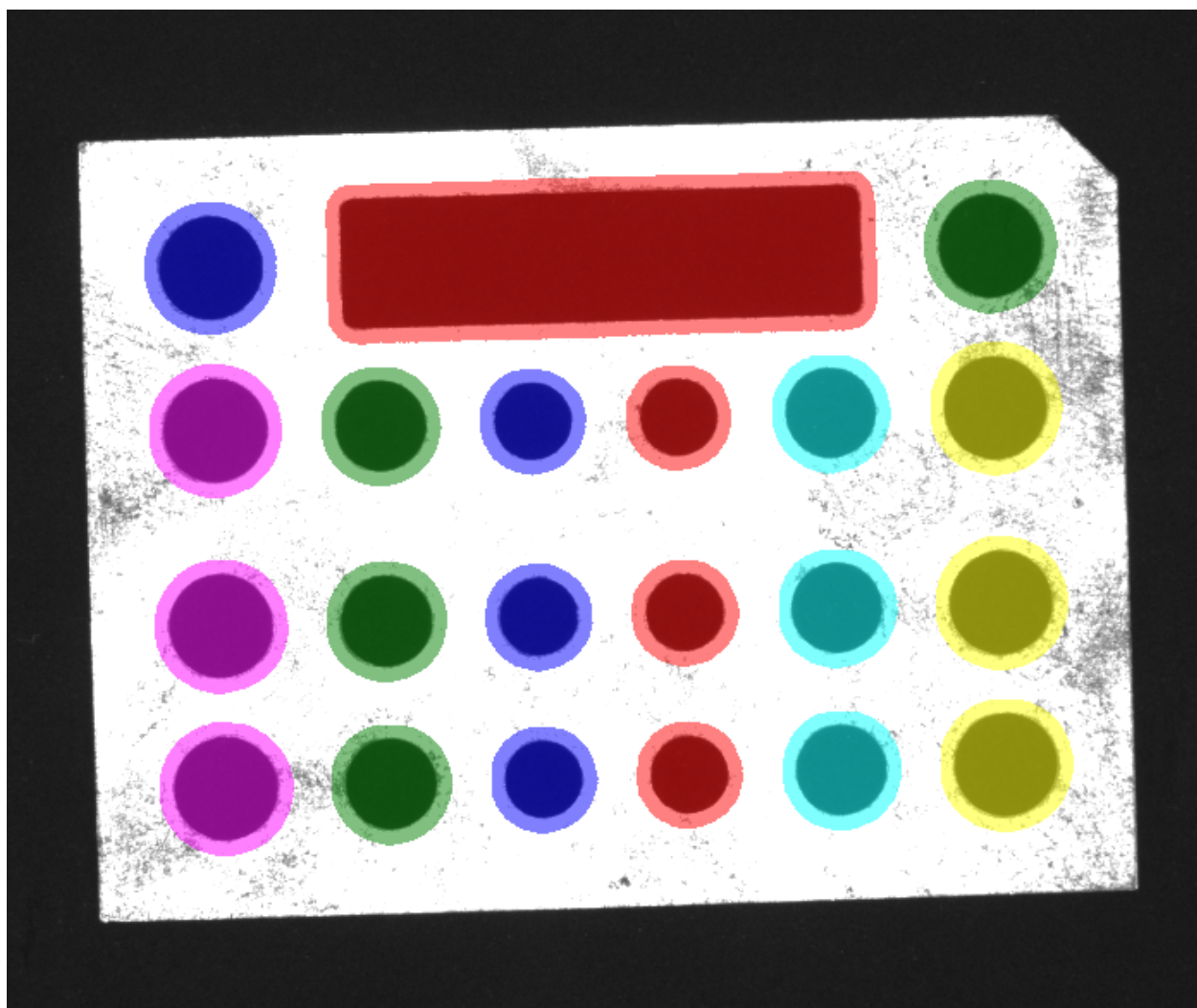
Operator (Type: string)

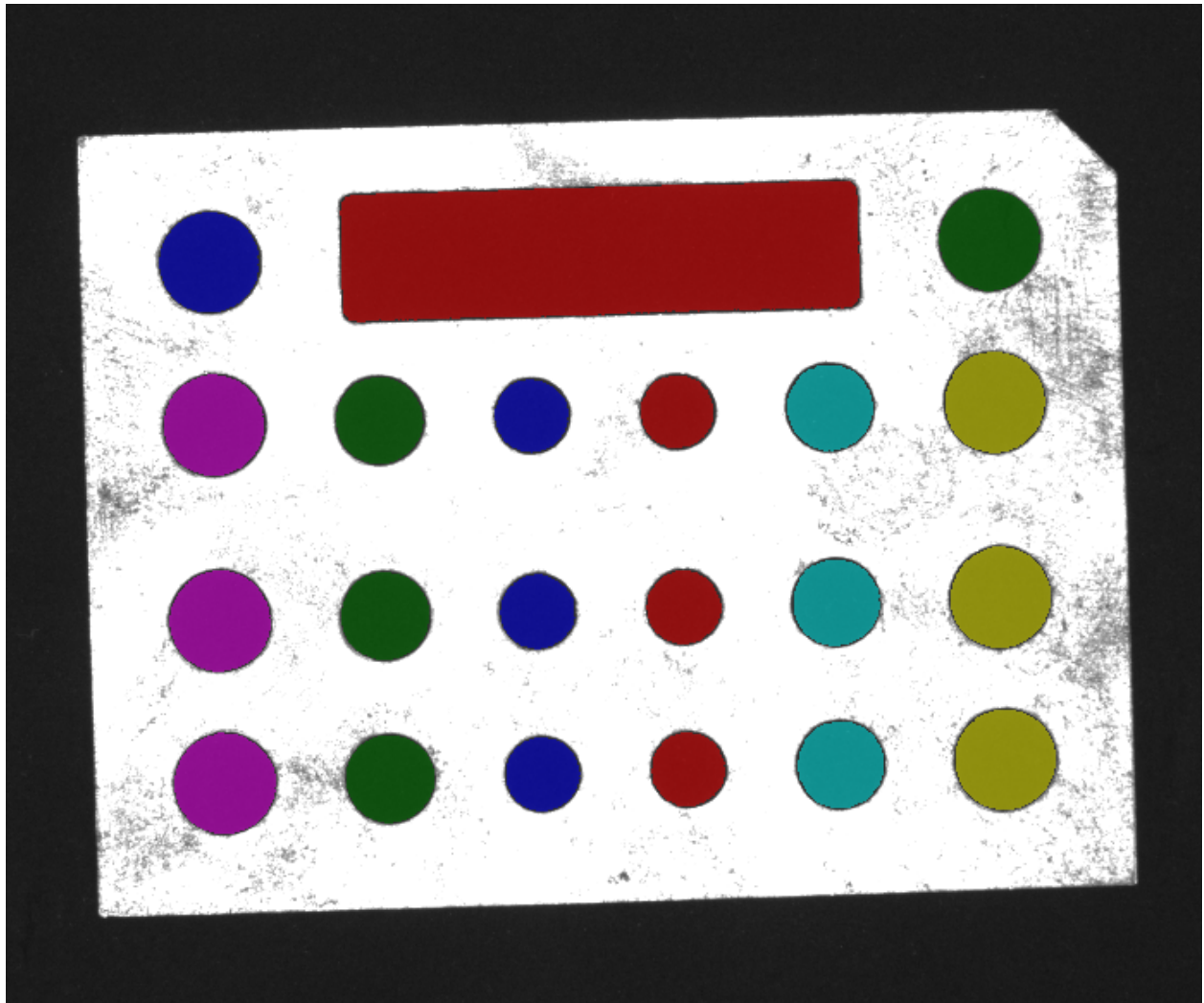
Specifies the operator (Erosion, Dilation, Closing, Opening or MorphologicalGradient).

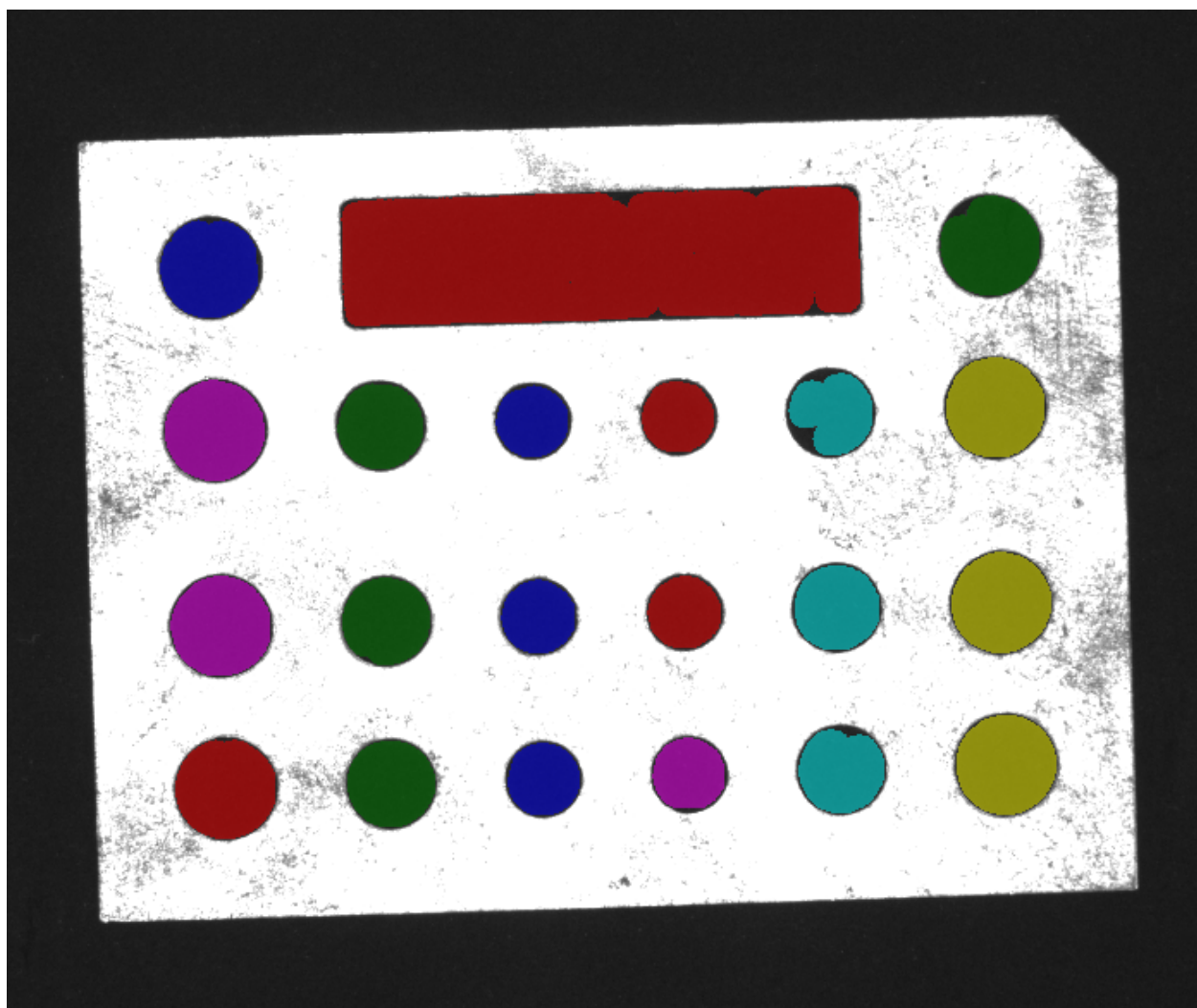
Structel (Type: Region)

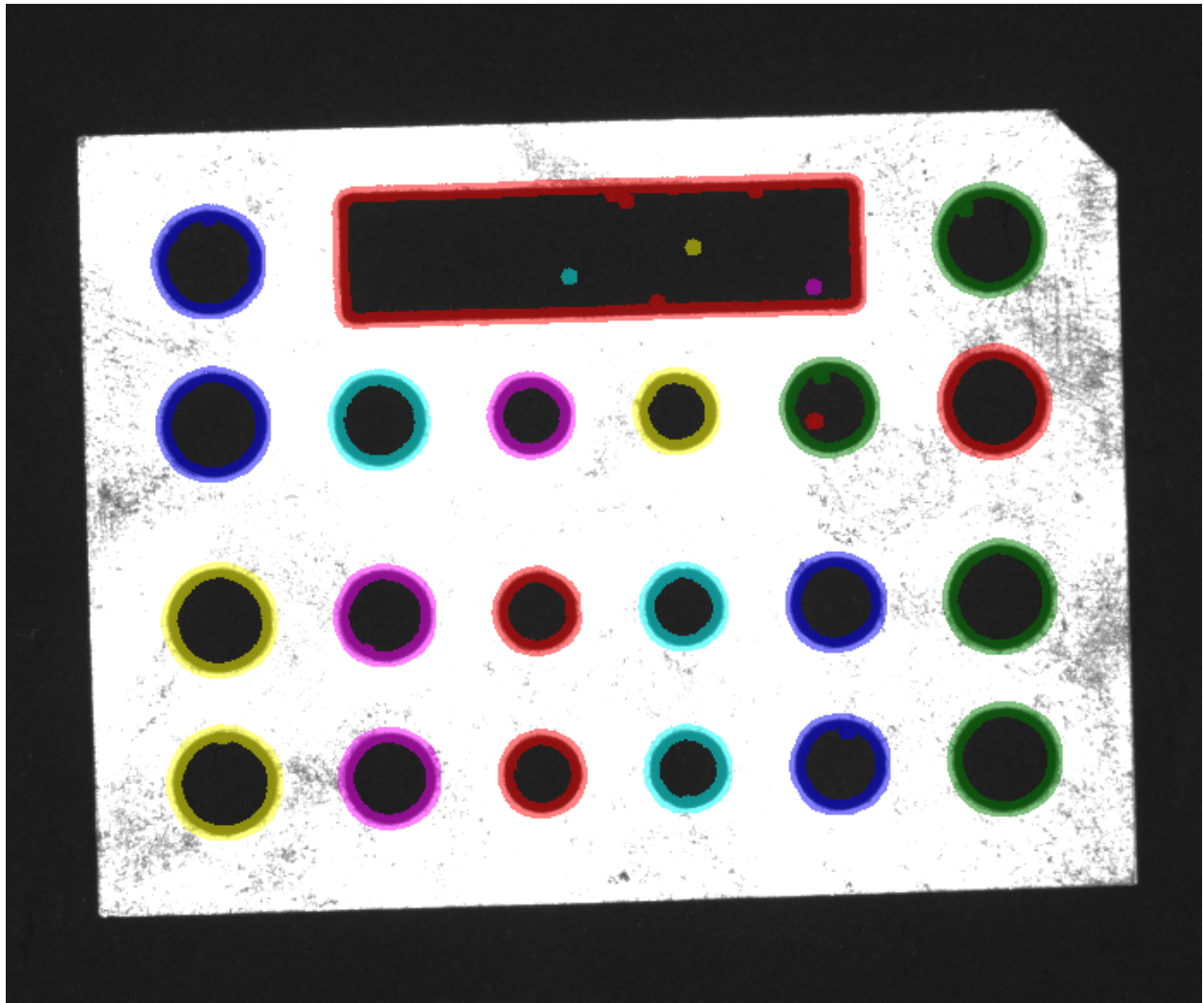
The structuring element (in the form of a region).











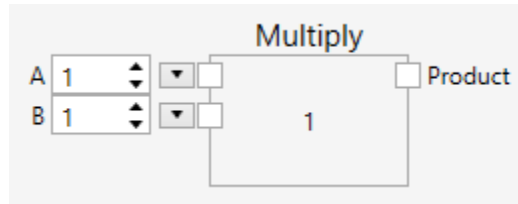
Outputs

Regions (Type: RegionList)

The resulting list of regions.

15.3.186 Multiply

Multiplies two or more numbers.



Inputs

A (Type: Double)

The first operand.

B (Type: Double)

The second operand.

C...Z (Type: Double)

Additional operands, added on demand.

Outputs

Product (Type: Double)

The product of all operands.

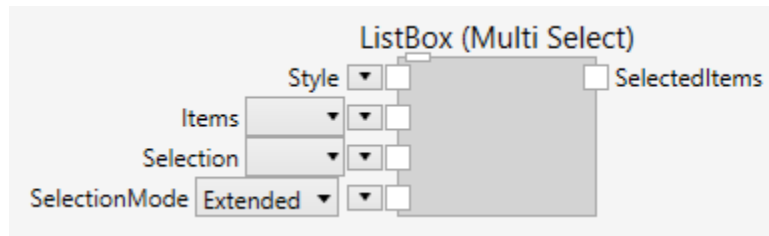
Comments

This node calculates the product of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.187 HMI ListBox (Multi Select)

A multi select **ListBox** can be used to select one from a number of entries.



Inputs

Style (Type: `Style`)

Optional styling of the **ListBox**.

The **ListBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Items (Type: `List<string>`)

A list of text items that is displayed in the **ListBox**.

Selection (Type: `List<string>`)

The initial selection of the **ListBox**. This value is used only when the *Items* pin is connected.

SelectionMode (Type: `string`)

Orientation of the **ListBox**, `Extended` or `Multiple`.

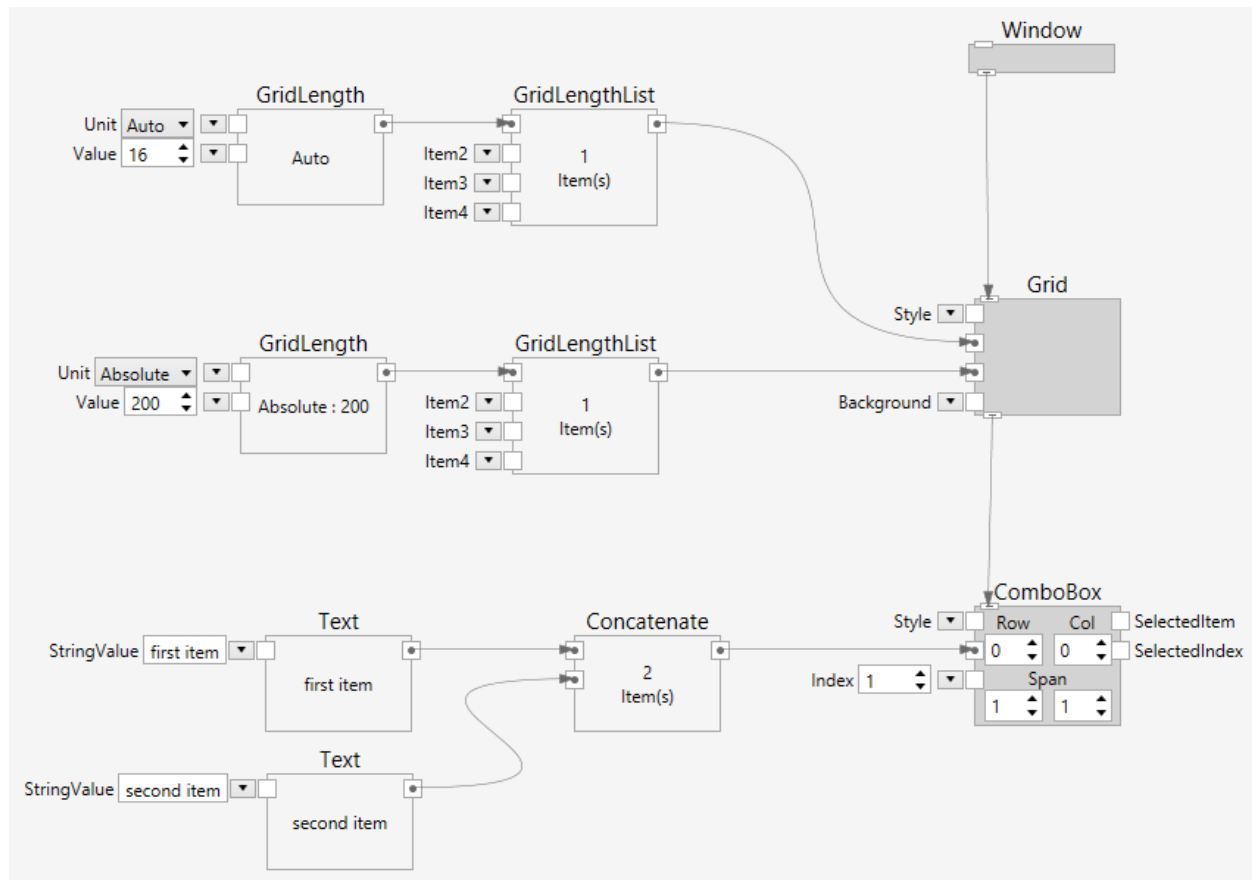
Outputs

SelectedItems (Type: `List<string>`)

The selected items.

Example

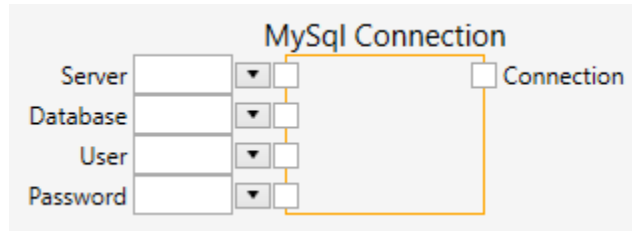
Here is an example that shows a **ListBox**. This definition:
creates the following user interface:



first item
second item

15.3.188 MySQL Connection

Establishes a connection to a MySQL database.



Inputs

Server (Type: String)

The name or network address of the instance of MySQL to which to connect. Examples are `localhost`, `127.0.0.1`, etc.

Database (Type: String)

The case-sensitive name of the database to use.

User (Type: String)

The MySQL login account being used.

Password (Type: String)

The password for the MySQL account being used.

Outputs

Connection (Type: `MySQL.Data.MySqlClient.MySqlConnection`)

The database connection.

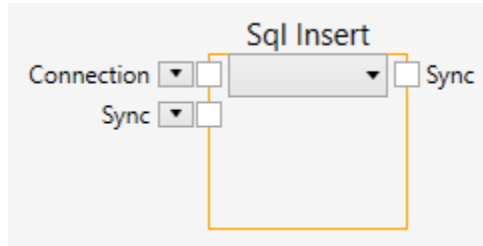
Comments

Establishing a database connection is the first step in communicating with a database. All other queries need the connection. Therefore, establishing the database connection has a somewhat global nature, and in many cases it can be put into the system globals pipeline or in the globals pipeline. If the connection is created in such a way it will be available from everywhere in your pipeline.

See the [MySQL documentation](#) for more information about how to establish connections to MySQL databases.

15.3.189 Sql Insert

Inserts data into a database table.



Inputs

Connection (Type: `MySQL.Data.MySqlClient.MySqlConnection`)

The database connection.

Sync (Type: `Object`)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs

Sync (Type: `Object`)

This output can be connected to any other object. It is used to establish an order of execution.

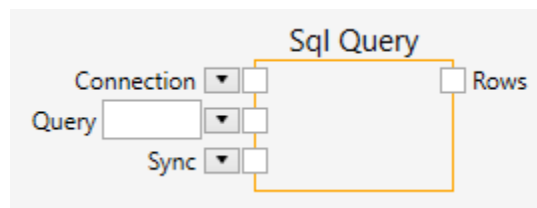
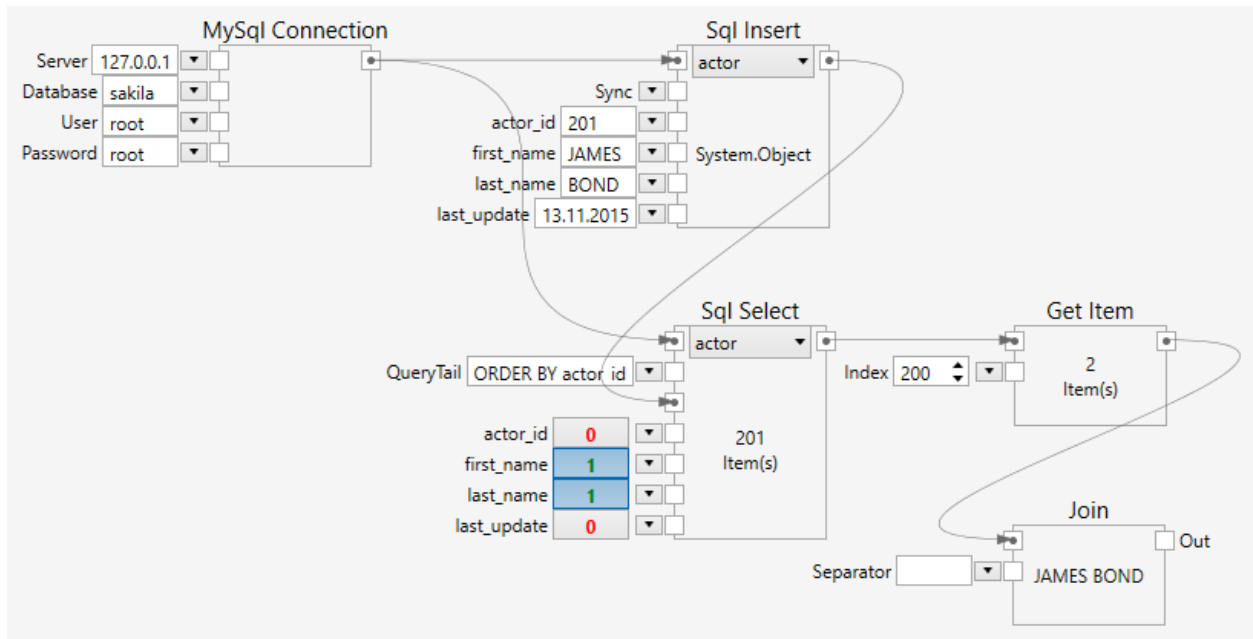
Comments

Once the database connection has been established, the node will fill its selection combo box with the tables that are available in the database. Once you have selected the desired table, the node will be populated dynamically with more inputs of type `String`, which correspond to the column names of the selected table. You can either type values or connect to outputs from elsewhere in the pipeline. If connections are made, you cannot change the table inside the node, because this could break existing connections.

The following example assumes that you have MySQL installed and that you can connect to the `sakila` sample database.

15.3.190 Sql Query

Executes a query on a database connection.



Inputs

Connection (Type: `MySQL.Data.MySqlClient.MySqlConnection`)

The database connection.

Query (Type: `String`)

The query string.

Sync (Type: `Object`)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs

Rows (Type: `List<List<String>>`)

The result of the query.

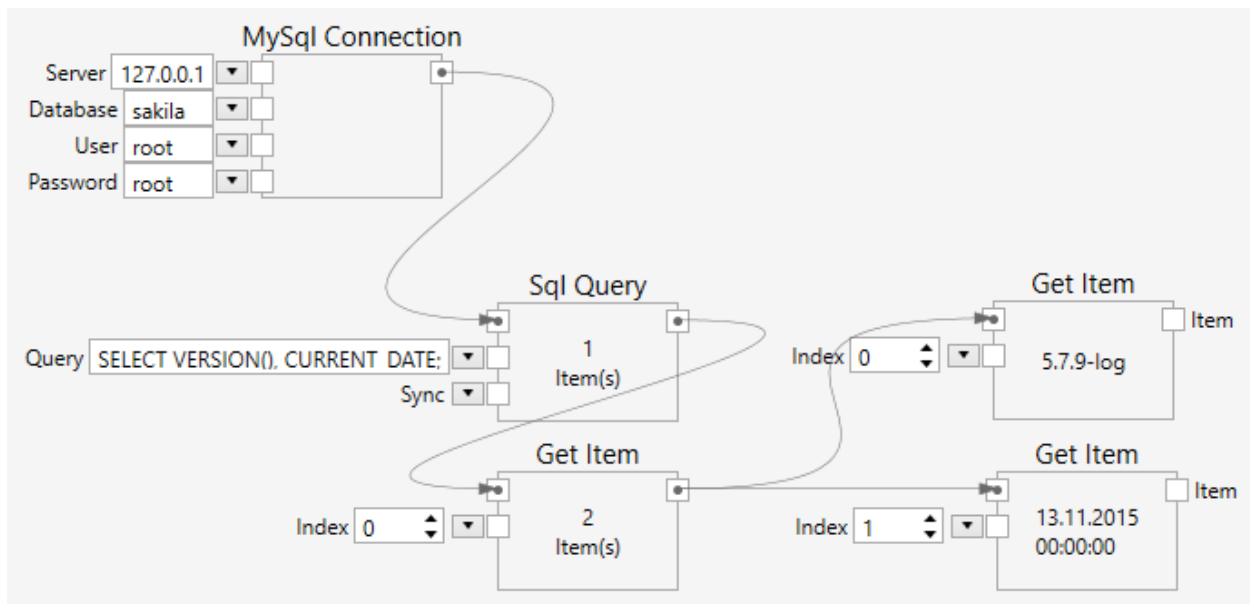
Comments

This command allows you to execute any query on a database.

See the [MySQL documentation](#) for more information about how to construct SQL queries.

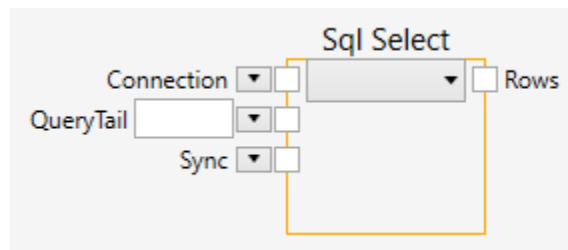
The [MySQL tutorial](#) may help you in constructing query strings.

Here is an example taken from this tutorial that shows how to build a query ('SELECT VERSION(), CURRENT_DATE;') and how to read the results. The example assumes that you have MySQL installed and that you can connect to the *sakila* sample database.



15.3.191 Sql Select

Selects data from a database table.



Inputs

Connection (Type: `MySql.Data.MySqlClient.MySqlConnection`)

The database connection.

QueryTail (Type: String)

The node constructs a query consisting of a Sql Select statement, such as `SELECT column FROM table`. The contents of the query tail is appended to this string and allows you to add clauses like `WHERE ...`, `ORDER BY ...`, etc. You do not need to add a semicolon, this will be appended automatically.

Sync (Type: Object)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs**Sync (Type: Object)**

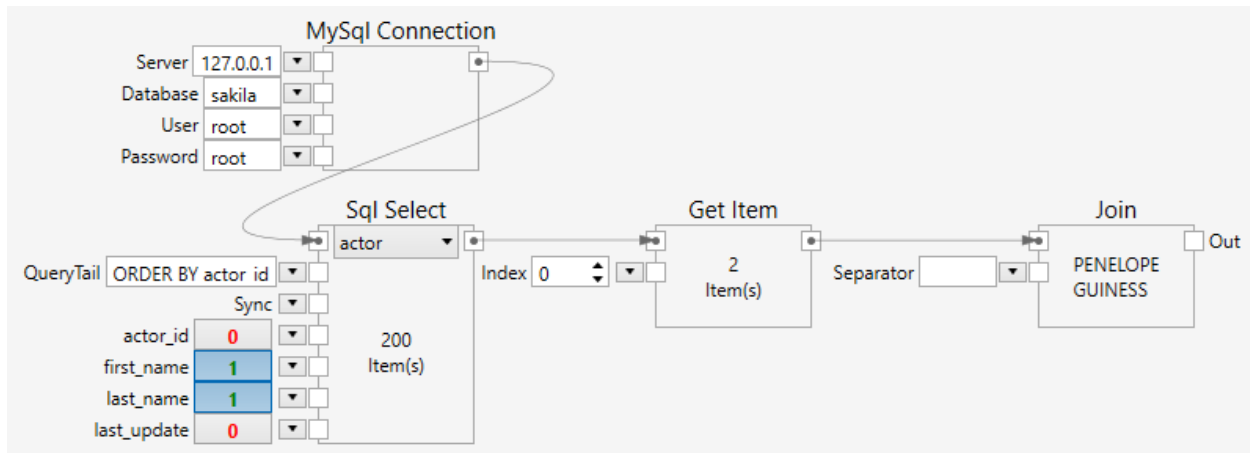
This output can be connected to any other object. It is used to establish an order of execution.

Comments

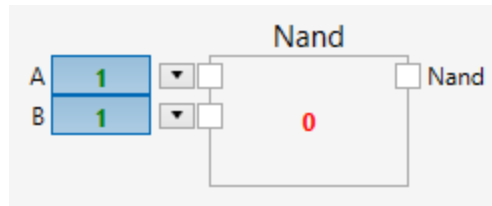
Once the database connection has been established, the node will fill its selection combo box with the tables that are available in the database. Once you have selected the desired table, the node will be populated dynamically with more inputs of type `Boolean`, which correspond to the column names of the selected table. You can either select 1 (true) or 0 (false) or connect to outputs from elsewhere in the pipeline. If connections are made, you cannot change the table inside the node, because this could break existing connections.

See the [MySQL documentation](#) for more information about how to construct SQL select statements.

The following example assumes that you have MySQL installed and that you can connect to the `sakila` sample database.

**15.3.192 Nand**

Logical Nand of two or more boolean values.



Inputs

A (Type: Boolean)

The first operand.

B (Type: Boolean)

The second operand.

C...Z (Type: Boolean)

Additional operands, added on demand.

Outputs

Nand (Type: Boolean)

The logical Nand of all operands.

Comments

This node calculates the logical Nand of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

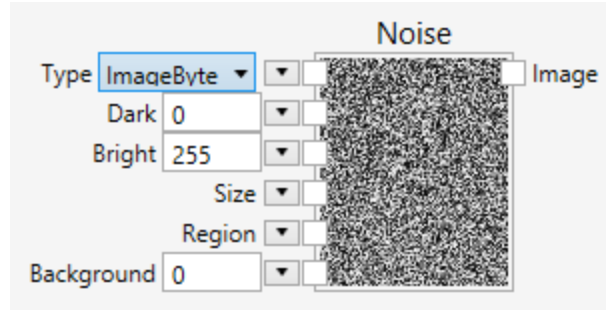
15.3.193 Noise

Presets an image with a noise pattern.

Inputs

Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble



Dark (Type: String)

The dark value of the noise fill. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: String)

The bright value of the noise fill. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs

Image (Type: Image)

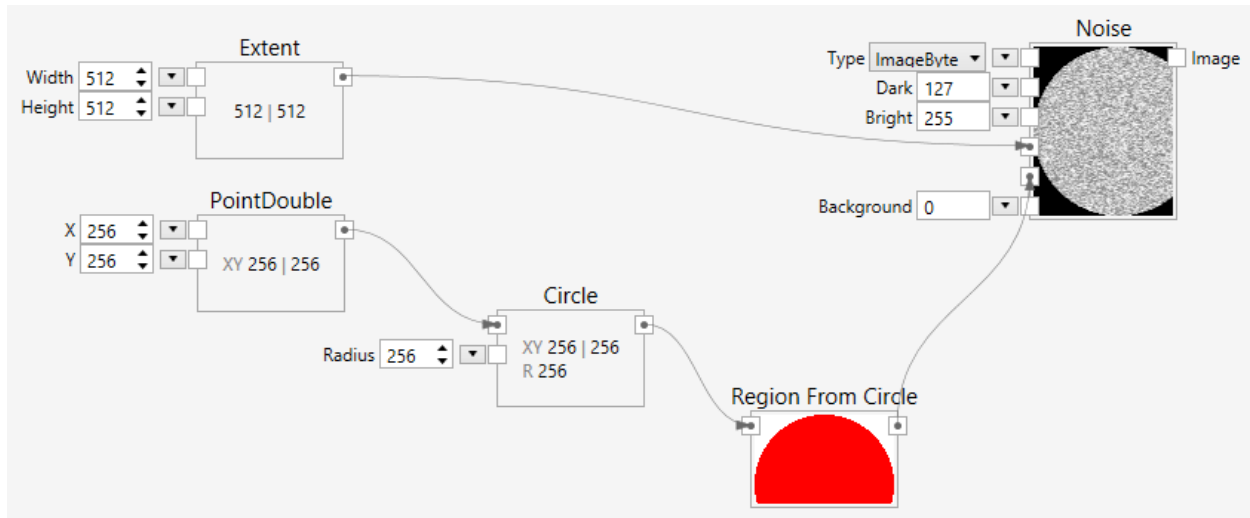
The image preset with a noise pattern.

Comments

The **Noise** node presets an image with a noise pattern. The size of the created image, the range of the colors, the region of interest and the background color can all be specified.

Sample

Here is an example that shows how to use the **Noise** node.



Here is the image created with the sample:

15.3.194 Nor

Logical Nor of two or more boolean values.

Inputs

A (Type: Boolean)

The first operand.

B (Type: Boolean)

The second operand.

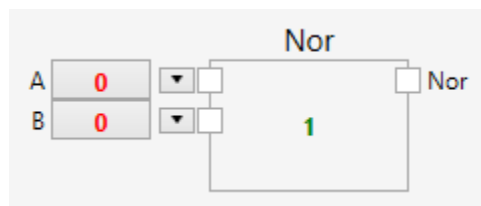
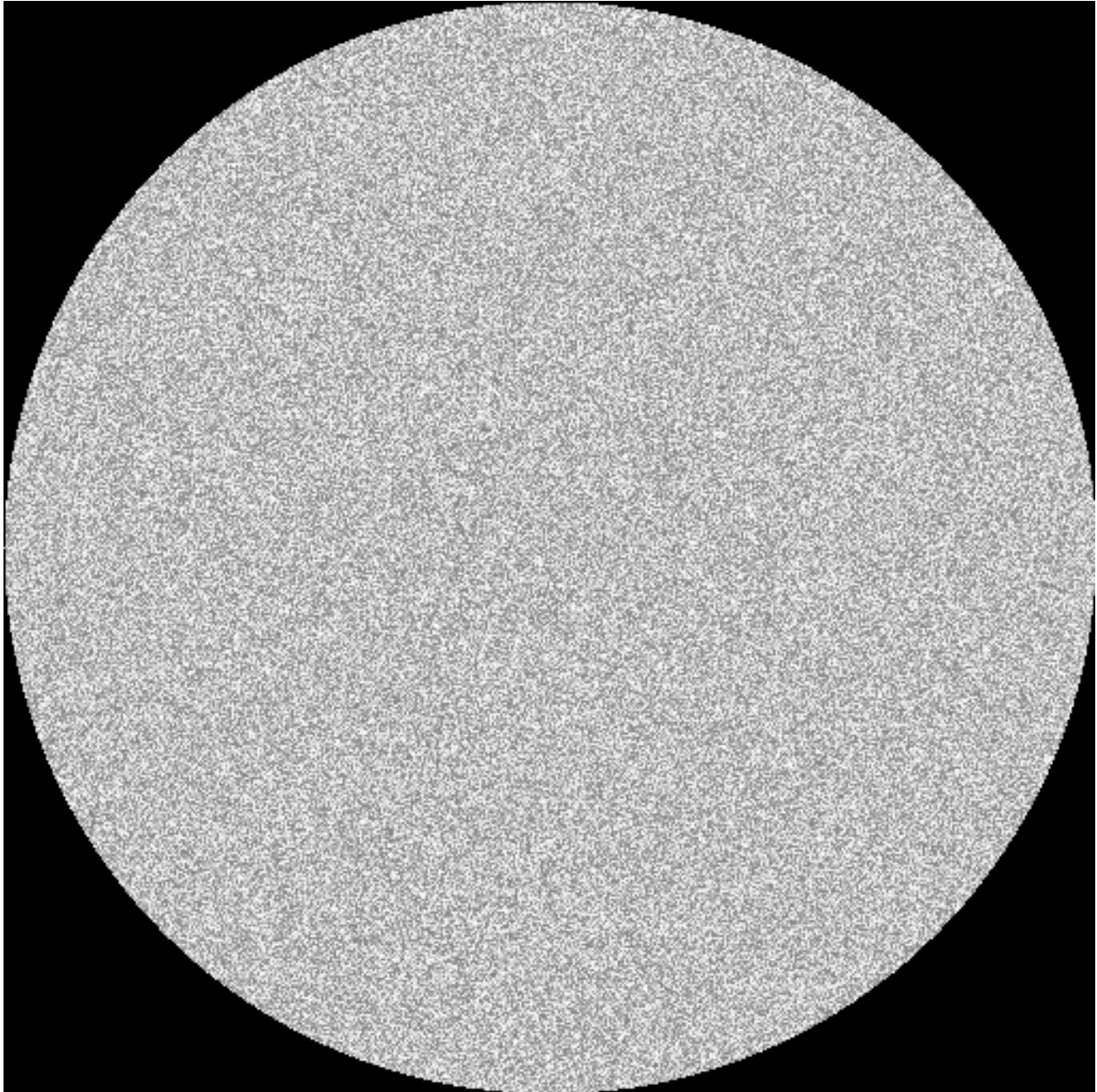
C...Z (Type: Boolean)

Additional operands, added on demand.

Outputs

Nor (Type: Boolean)

The logical Nor of all operands.



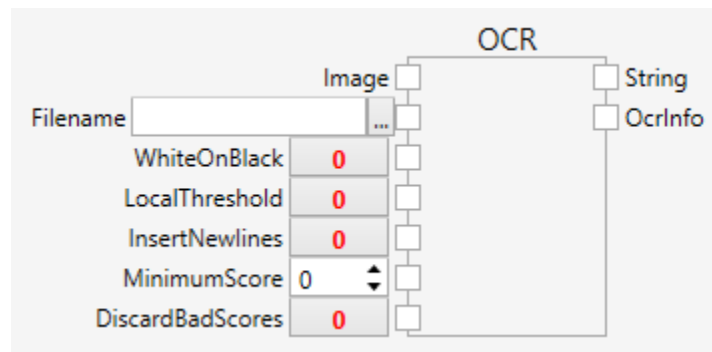
Comments

This node calculates the logical Nor of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.195 OCR

Performs optical character recognition (OCR) or optical character verification (OCV).



Inputs

Image (Type: Image)

The input image.

Filename (Type: String)

The OCR font file.

WhiteOnBlack (Type: Boolean)

Detect characters printed black on white (default, 0) or white on black (1).

LocalThreshold (Type: Boolean)

Use a global threshold (default, 0) or a local threshold (1) for character segmentation.

InsertNewlines (Type: Boolean)

Do not insert newline characters (default, 0) or insert newline characters (1) in decoded result.

MinimumScore (Type: Double)

The minimum score for character acceptance. The recognition score is between 0 and 1. The higher the score, the better the character.

DiscardBadScores (Type: Boolean)

Do not discard characters with bad scores (default, 0) or discard them (1).

Outputs**String (Type: String)**

The decoded text string.

OcrInfo (Type: OcrCharacterInfo)

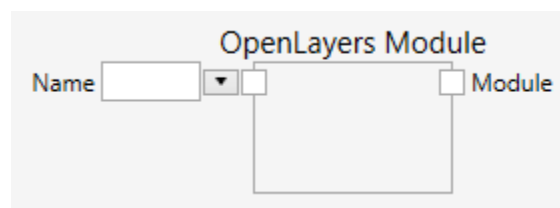
Additional information, such as bounding boxes and scores of segmented characters and the average score.

Comments

The **OCR** node tries to segment characters in the input image. After segmentation, the segmented patterns are compared to taught characters from the font file. Finally, the relative position of the characters is analyzed and a text result is created. Additional information, such as the position of the segmented characters and their match scores is calculated.

15.3.196 OpenLayers Module

Establish a connection to an OpenLayers module from Data Translation, which allows you to read and write digital signals.



This node initializes the module. If you connect its output to a **GetProperty** node, you can gather information about the device. Usually you would connect the output to an **OpenLayers DigIn** node to read electrical input, or an **OpenLayers DigOut** node to write electrical output.

The OpenLayers modules are connected via USB or PCI to a PC and support a variety of input/output lines. The modules may have additional features, such as counters, etc., but these are not yet supported.



module	bus	features
DT9817	USB	28 digital inputs/outputs
DT9817-H	USB	28 digital inputs/outputs
DT9817-R	USB	8 digital inputs, 8 digital outputs
DT9835	USB	64 digital inputs/outputs
DT335	PCI	32 digital inputs/outputs
DT251	PCI	8 digital inputs, 8 digital outputs
DT340	PCI	32 digital inputs, 8 digital outputs

Inputs

Name (Type: `string`)

The name of the OpenLayers module.

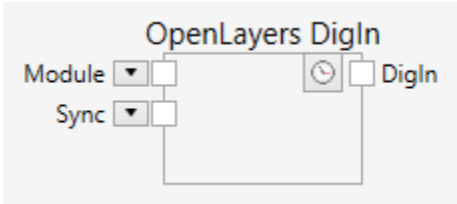
Outputs

Module (Type: `OpenLayersModule`)


The OpenLayers module instance.

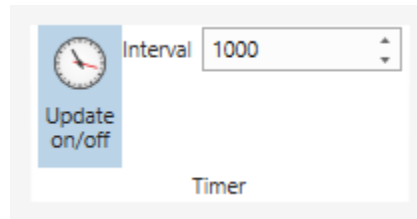
15.3.197 OpenLayers DigIn


Reads digital electrical inputs from an OpenLayers Module from Data Translation.



This node reads digital signals from an OpenLayers module.

If the clock button inside the node is pressed , the node polls the module regularly. The frequency of the polling is controlled with the timer interval setting on the Pipeline tab on the ribbon.



If the clock button inside the node is released , the module is not polled, and the node executes during regular pipeline updates only.

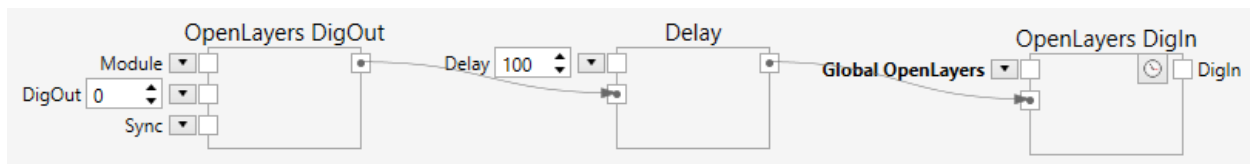
Inputs

Module (Type: OpenLayersModule)

An OpenLayers module instance.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.



Here, the digital input is read **after** the digital output has been written.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

Outputs

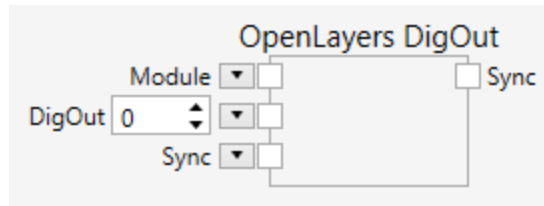
DigIn (Type: Int32)

The digital input in the form of a 32 bit integer number.

15.3.198 OpenLayers DigOut

Writes digital electrical outputs to an OpenLayers module from Data Translation.

This node writes digital signals to an OpenLayers module.



Inputs

Module (Type: OpenLayersModule)

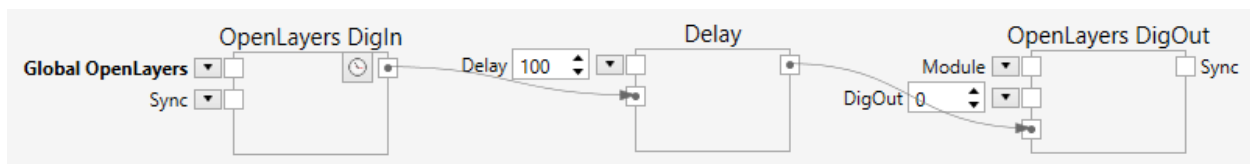
An OpenLayers module instance.

DigOut (Type: Int32)

The numeric value that is written to the electrical outputs.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.



Here, the digital output is written **after** the digital input has been read.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

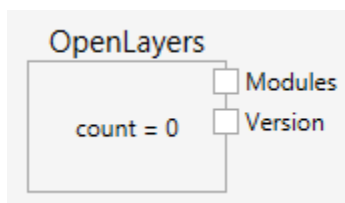
Outputs

Sync (Type: object)

Synchronizing object. It is used to establish an order of execution.

15.3.199 OpenLayers

Get information about the OpenLayers system of Data Translation.



This node initializes the OpenLayers system and delivers a list of connected module names.



Outputs

Modules (Type: List<string>)

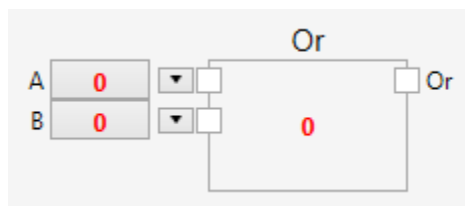
A list of module names.

Version (Type: string)

The version of the installed OpenLayers software.

15.3.200 Or

Logical Or of two or more boolean values.



Inputs

A (Type: Boolean)

The first operand.

B (Type: Boolean)

The second operand.

C...Z (Type: Boolean)

Additional operands, added on demand.

Outputs**Or (Type: Boolean)**

The logical Or of all operands.

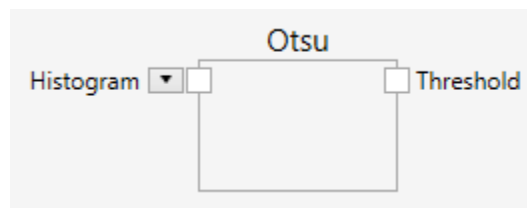
Comments

This node calculates the logical Or of multiple operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.201 Otsu Threshold

Calculates an optimal threshold with the Otsu method, given a histogram.

**Inputs****Histogram (Type: Histogram)**

The input histogram.

Outputs**Threshold (Type: Double)**

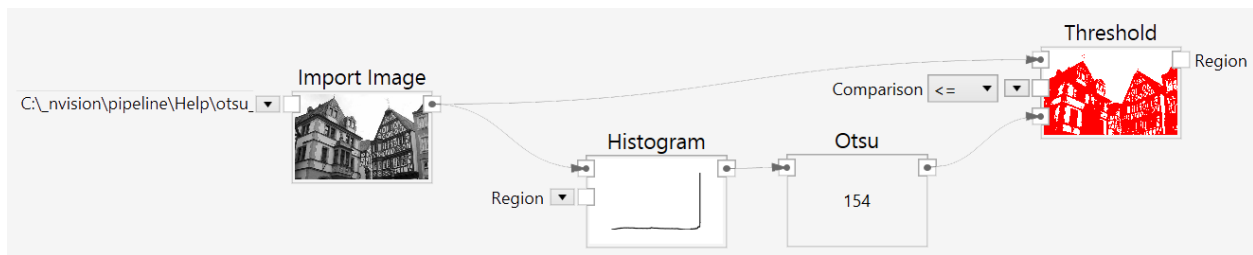
The threshold value.

Comments

The **Otsu Threshold** node calculates a threshold value that can be used to segment an image into foreground and background. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal.

Sample

Here is an example that calculates a threshold with the Otsu method.



Here is the original image:

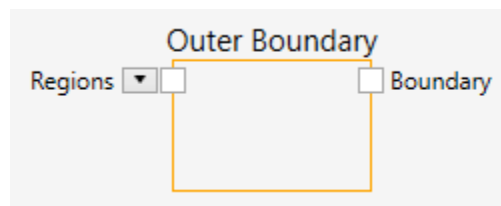


And here is the binary image thresholded with the Otsu threshold:

15.3.202 Outer Boundary

Extracts the outer boundary from a list of regions.

The outer boundary of a region consists of the pixels of the background that touch the object. The outer boundary lies on the background.



Inputs

Regions (Type: RegionList)

The list of regions.

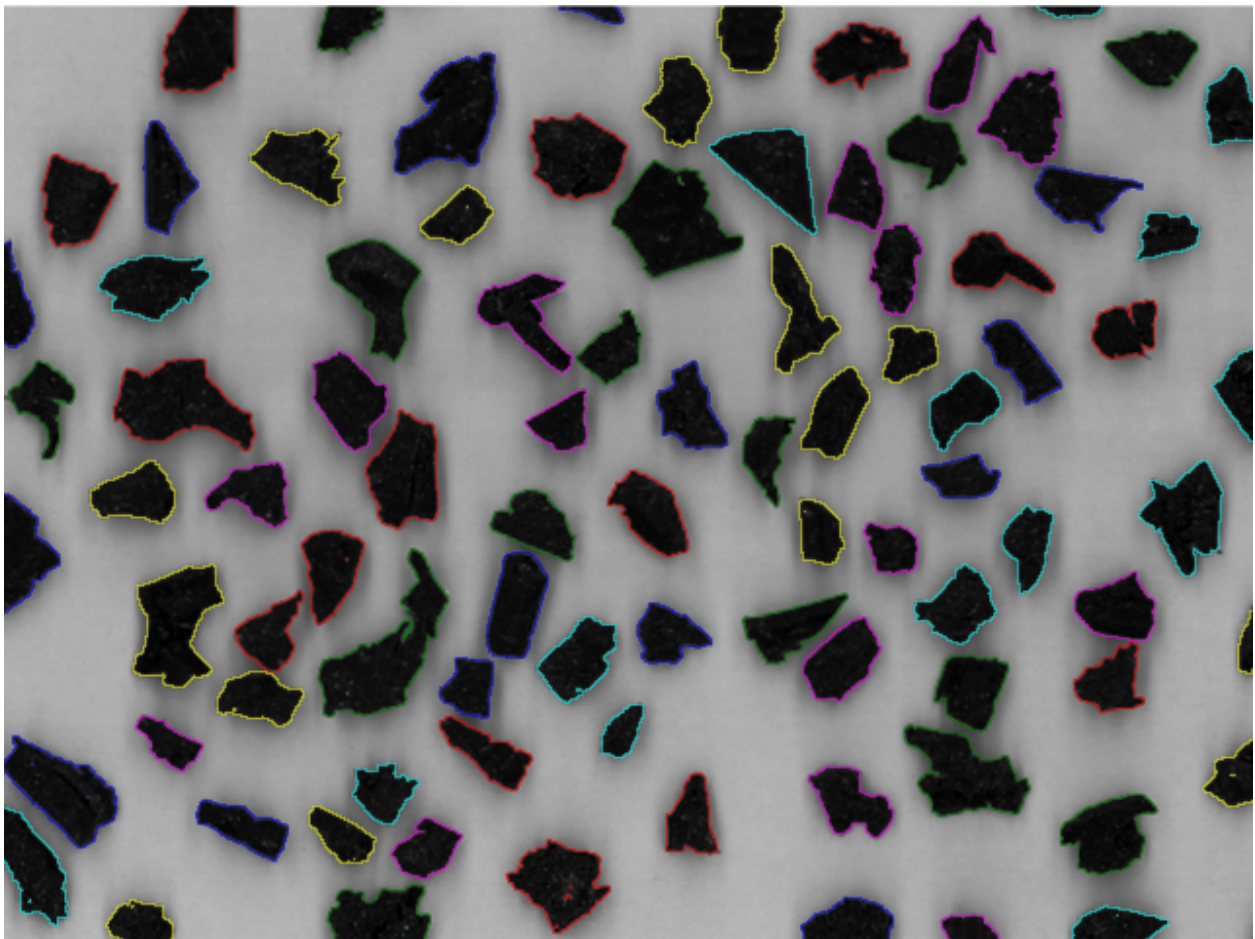
Outputs

Boundary (Type: RegionList)

The objects reduced to their outer boundary.

Comments

Here is an example of an outer boundary:



15.3.203 Paste

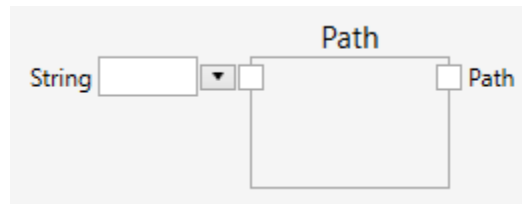
You can use `Ctrl-V` or the **Paste (Ctrl-V)** command from the pipeline menu to paste previously copied nodes and their inside connections from the clipboard.

The nodes are pasted at the location of the mouse cursor.

Since outside connections are not copied, the pasted fragments may need to be reconnected to the rest of the pipeline.

15.3.204 Path

Make a path given a string.



Inputs

String (Type: String)

A string that should be converted to a path.

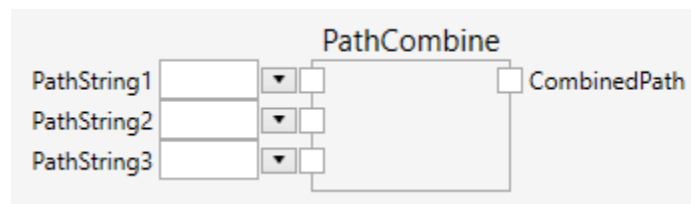
Outputs

Path (Type: Ngi.Path)

The path.

15.3.205 PathCombine

Combines three strings into a path.



Inputs

PathString1 (Type: String)

The first path to combine.

PathString2 (Type: String)

The second path to combine.

PathString3 (Type: String)

The third path to combine.

Outputs**CombinedPath (Type: Ngi.Path)**

The combined path.

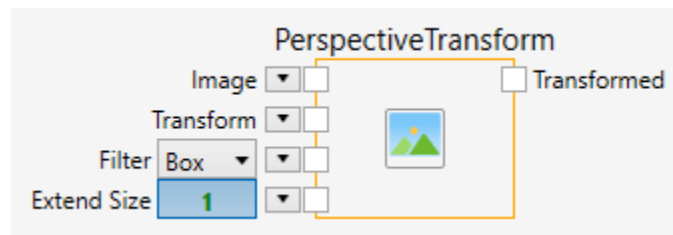
Comments

PathString1 should be an absolute path (for example, “d:” or “\archives”). If PathString2 or PathString3 is also an absolute path, the combine operation discards all previously combined paths and resets to that absolute path.

Empty strings are omitted from the combined path.

15.3.206 PerspectiveTransform

Geometrically transforms an image by a perspective transform.

**Inputs****Image (Type: Image)**

The input image.

Transform (Type: Ngi.PerspectiveMatrix)

The perspective transformation matrix.

Filter (Type: String)

The geometric interpolation. Available values are NearestNeighbor, Box, Triangle, Cubic, Bspline, Sinc, Lanczos and Kaiser. The accuracy of the interpolation increases from NearestNeighbor to Kaiser, but the performance decreases. The default is set to Box.

Extend Size (Type: bool)

The geometric transformation may produce pixels that are outside of the input image bounds. If the parameter is set, the size of the output image is adapted to make room for these pixels. If the parameter is cleared, the size of the output image is kept the same as the input image.

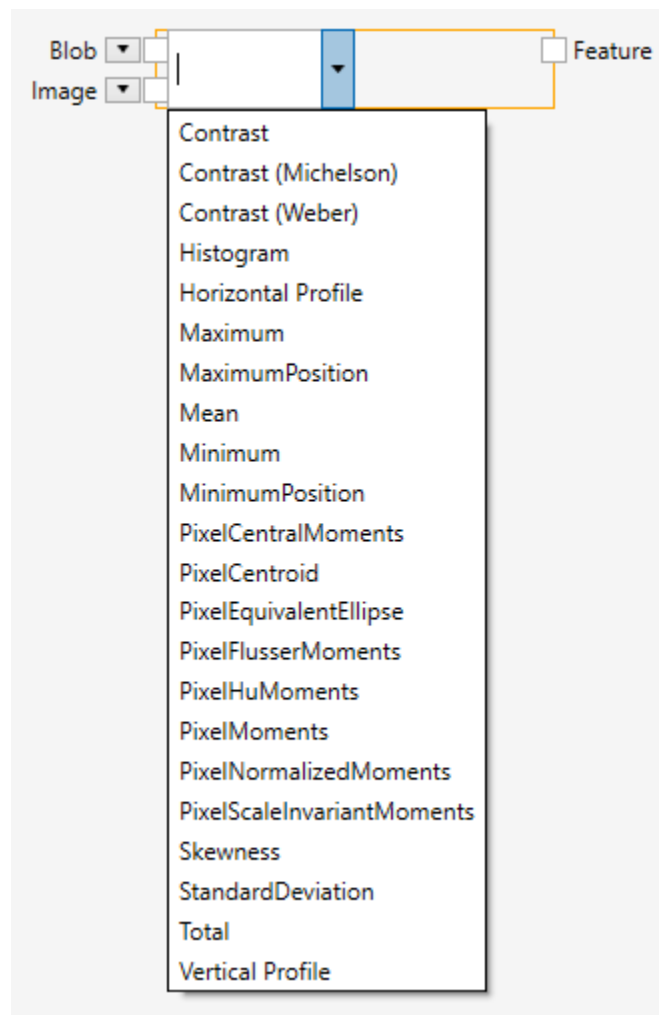
Outputs

Transformed (Type: Image)

The output image.

15.3.207 Pixel Based Blob Feature

Measures pixel-based features of a region and an associated image.



Inputs

Blob (Type: Region)

The region that defines the blob.

Image (Type: Image)

The associated image.

Outputs

Feature

The blob analysis result data. The type of the result depends on the selected feature.

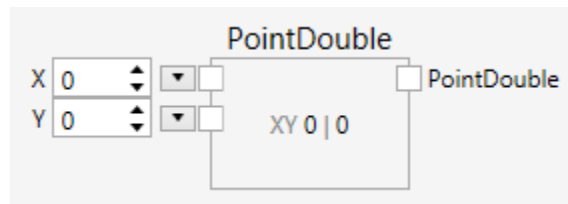
Comments

The **Pixel Based Blob Feature** node calculates the feature for a single region and an associated image.

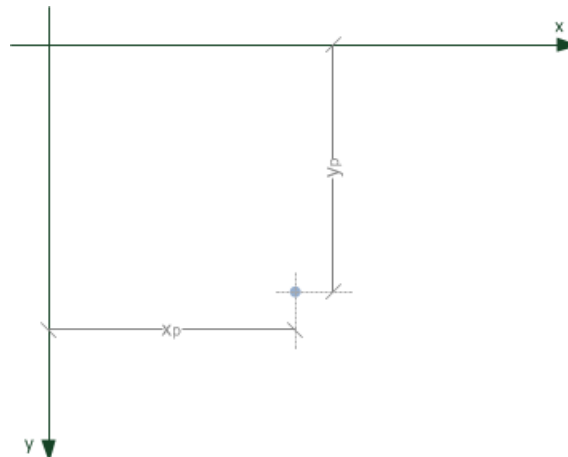
The features can be selected inside the node. The full set of features is described in the Blob Analysis chapter of the nVision User Guide.

15.3.208 Point

Creates a point.



A point is a geometric entity, representing a location in the two-dimensional cartesian plane.



Inputs

X (Type: Double)

The horizontal coordinate.

Y (Type: Double)

The vertical coordinate.

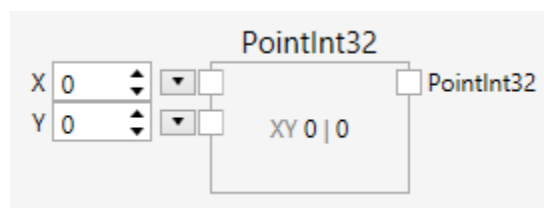
Outputs

PointDouble (Type: PointDouble)

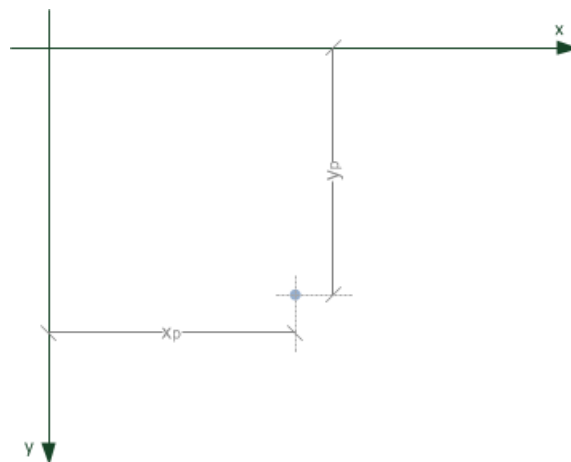
The point.

15.3.209 Point

Creates a point.



A point is a geometric entity, representing a location in the two-dimensional cartesian plane.



Inputs

X (Type: Int32)

The horizontal coordinate.

Y (Type: Int32)

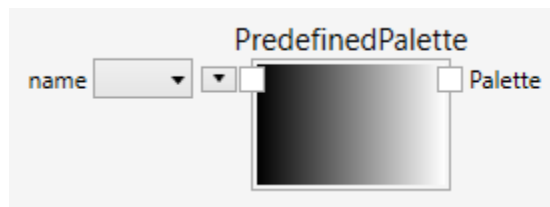
The vertical coordinate.

Outputs**PointInt32 (Type: PointInt32)**

The point.

15.3.210 PredefinedPalette

Creates a palette with a predefined transfer function.

**Inputs****name (Type: String)**

The type of the palette. Choices are Linear, Inverse, Square, SquareInverse, Cube, CubeInverse, CubicRoot, CubicRootInverse, Red, Green, Blue, Cyan, Magenta, Yellow, BlueOrange, Cool, CyanHot, MagentaHot, OrangeHot, RedHot, YellowHot, Ice, Fire, GreenFireBlue, Rainbow and Spectrum.

Outputs**Palette (Type: Palette)**

The output palette.

Comments

This function creates a palette with a predefined function. It can be used to create monochrome or color palettes. The following palettes can be chosen:

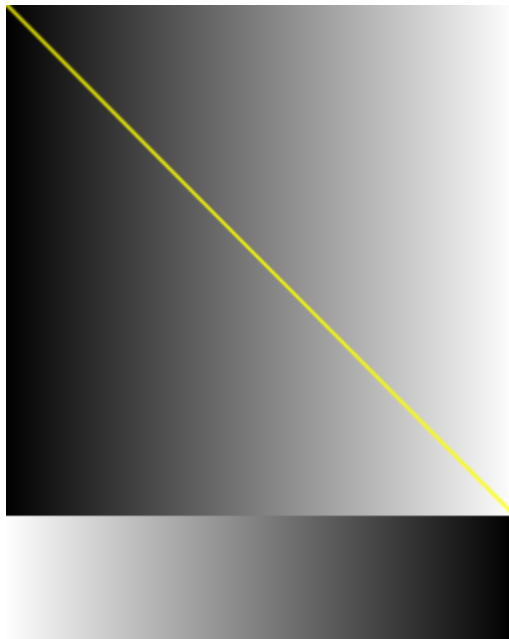
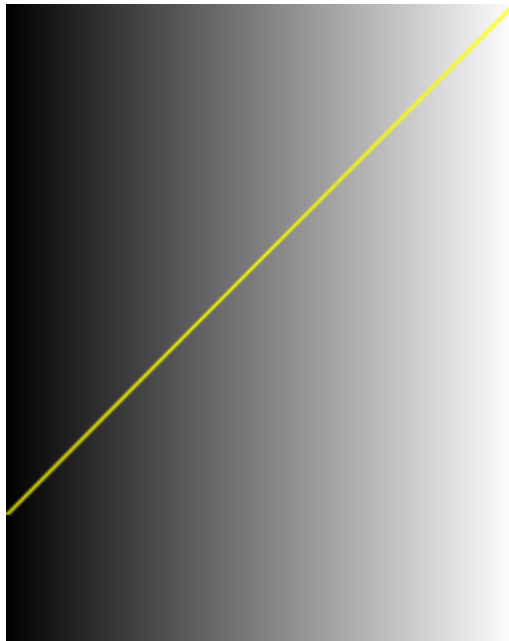
Linear

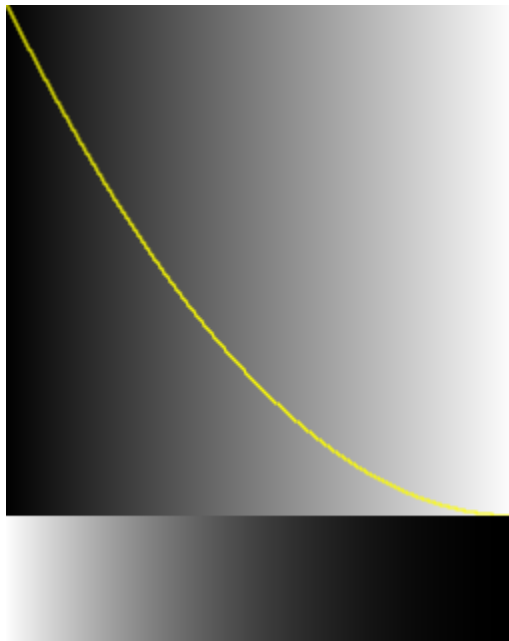
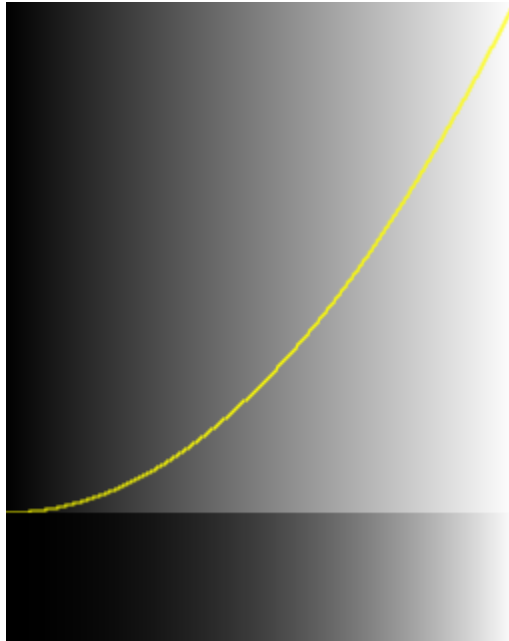
Inverse

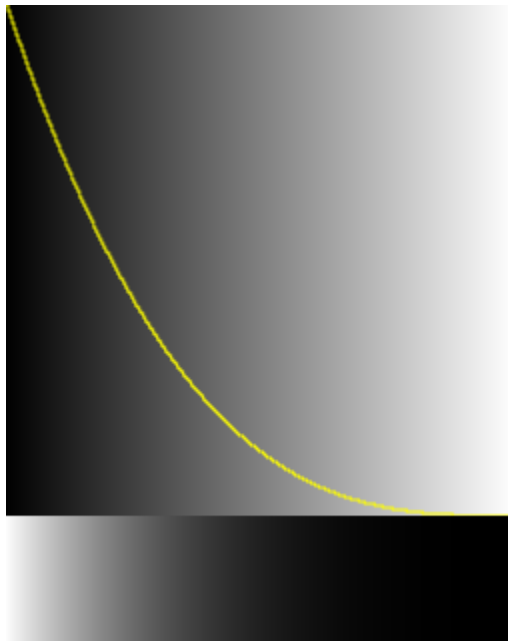
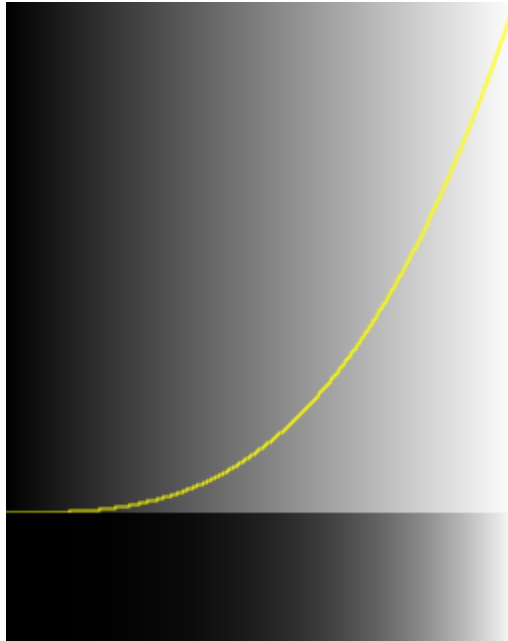
Square

SquareInverse

Cube

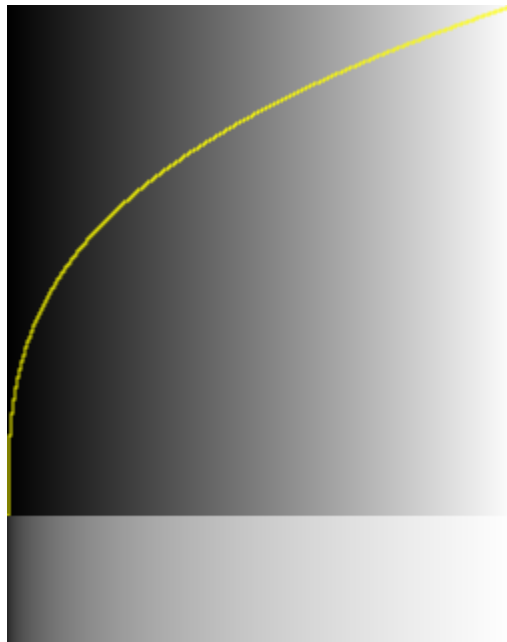




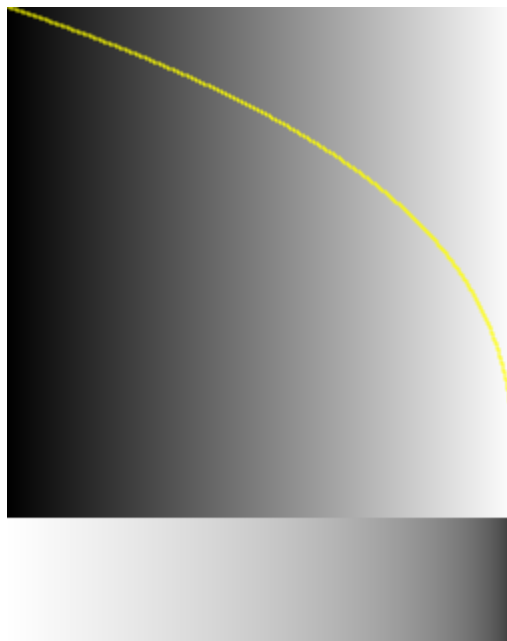


CubeInverse

CubicRoot



CubicRootInverse



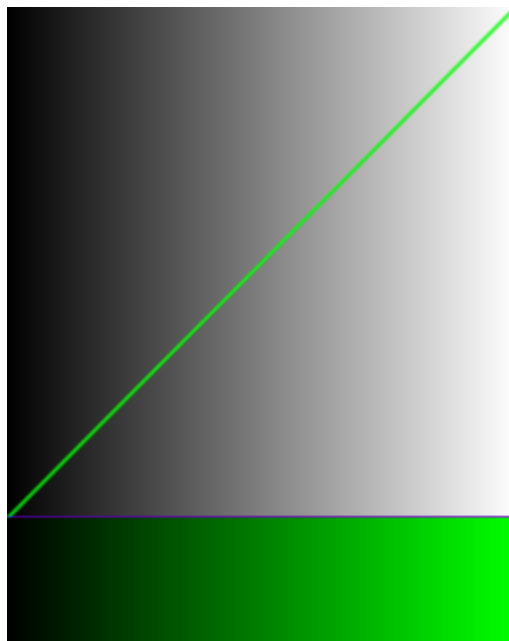
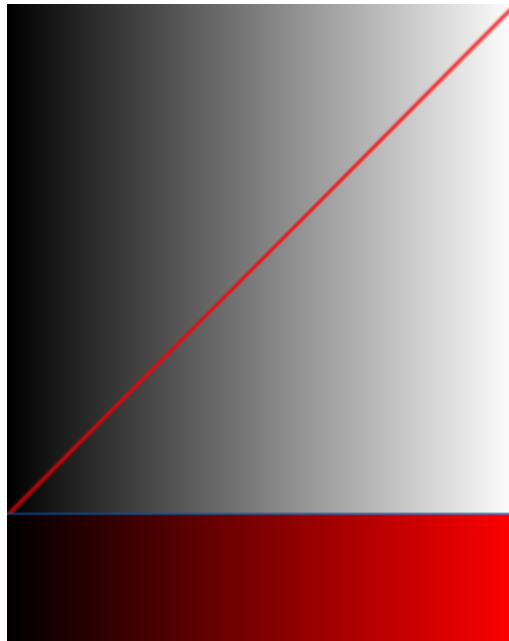
Red

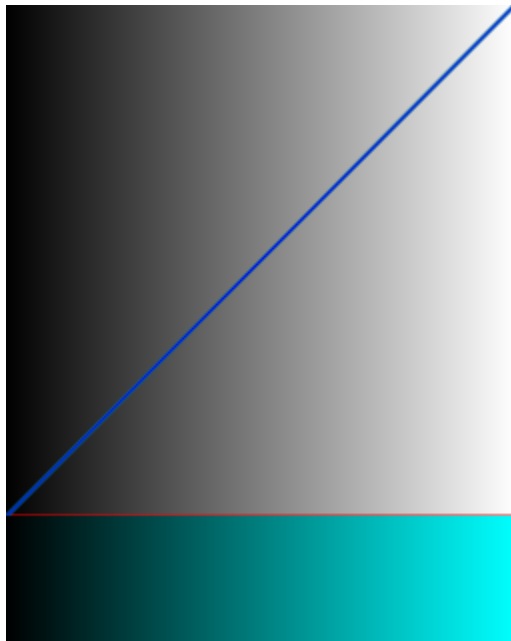
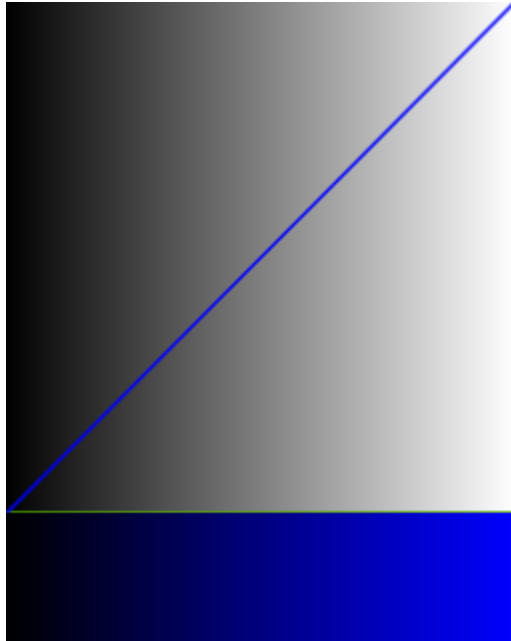
Green

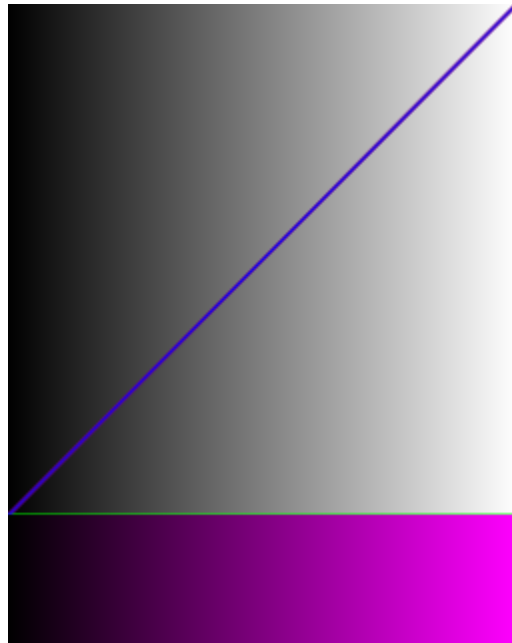
Blue

Cyan

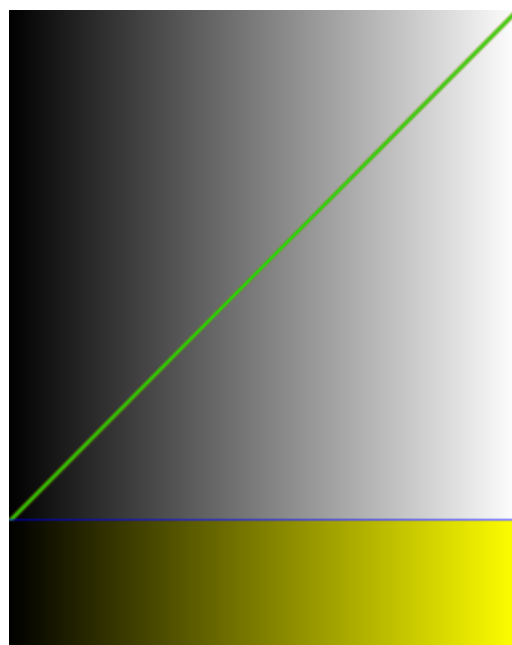
Magenta







Yellow



BlueOrange

Cool

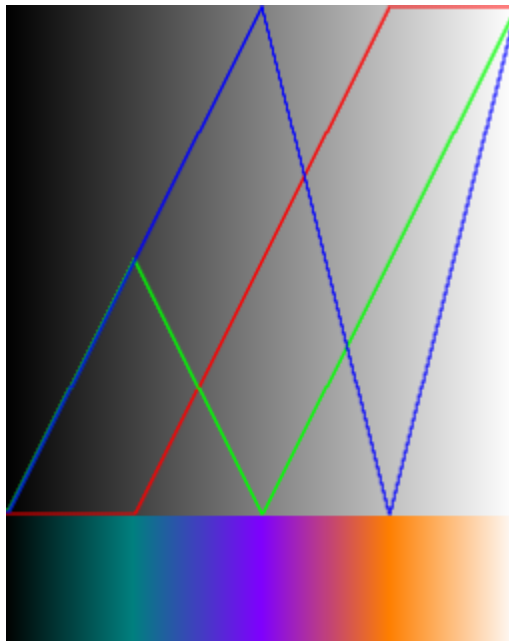
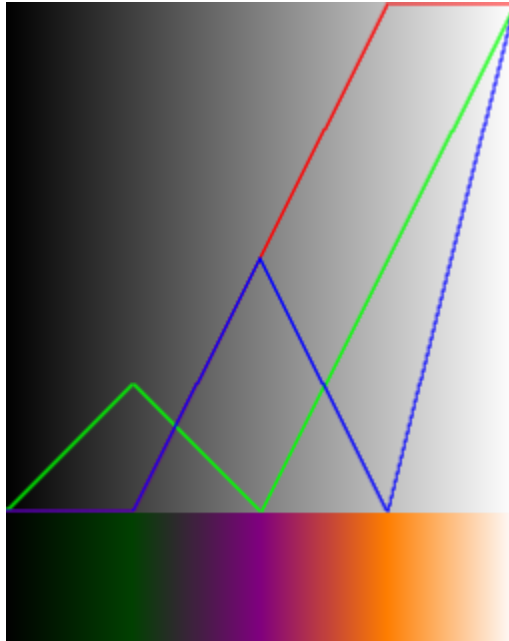
CyanHot

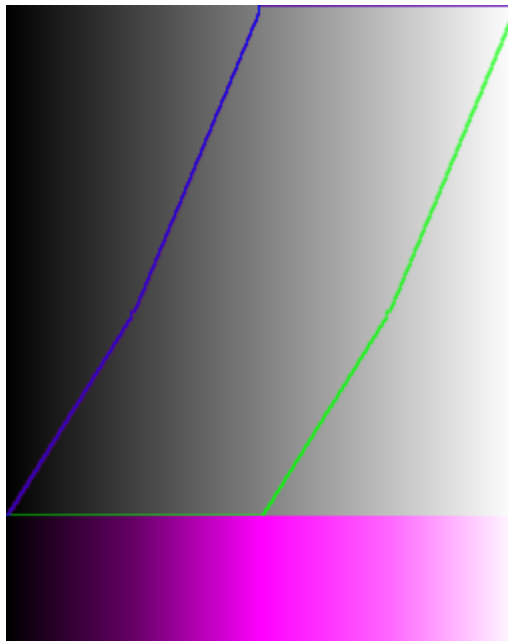
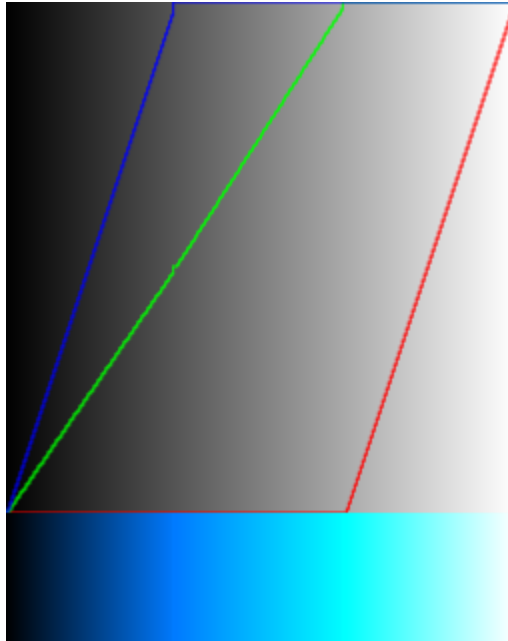
MagentaHot

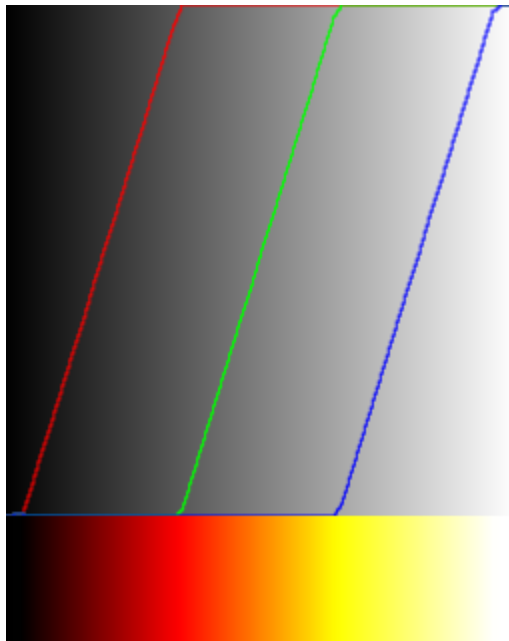
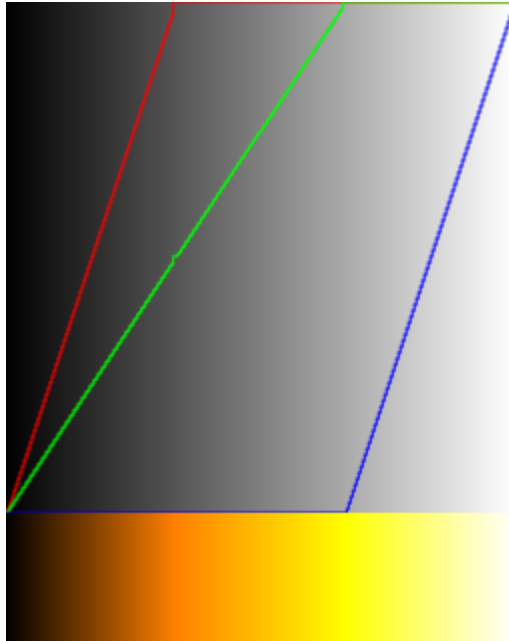
OrangeHot

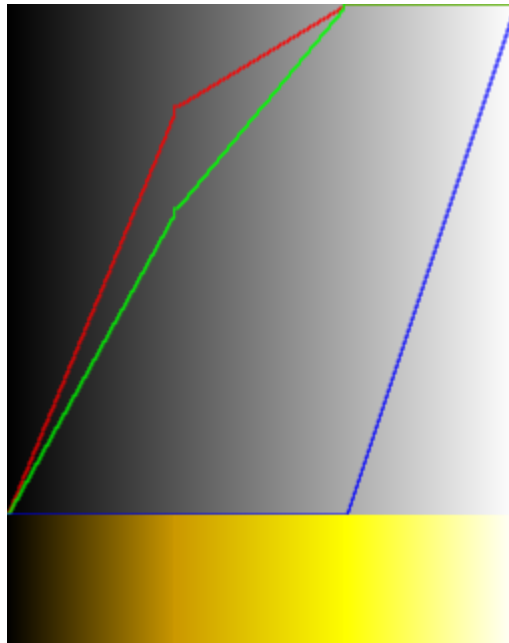
RedHot

YellowHot

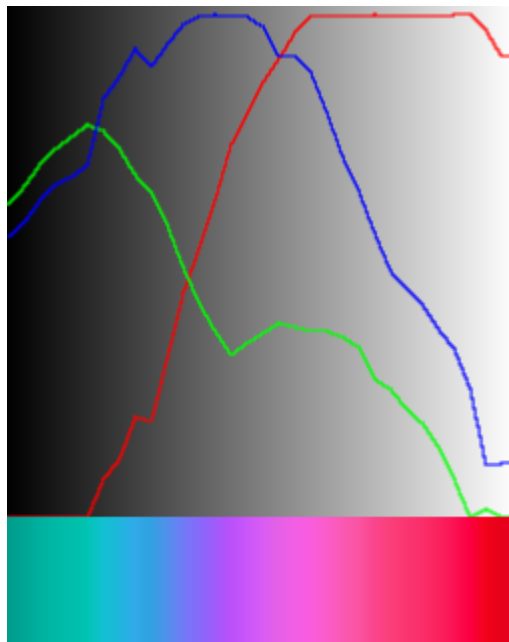








Ice

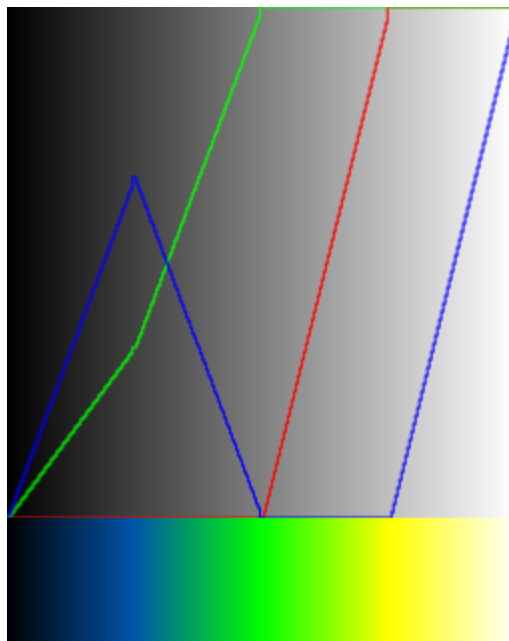
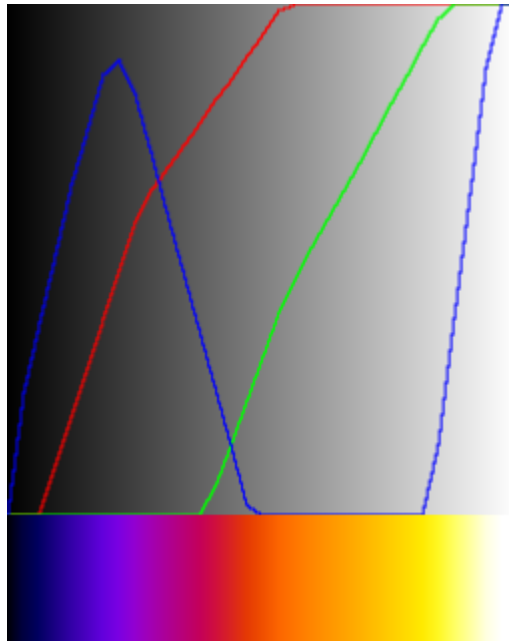


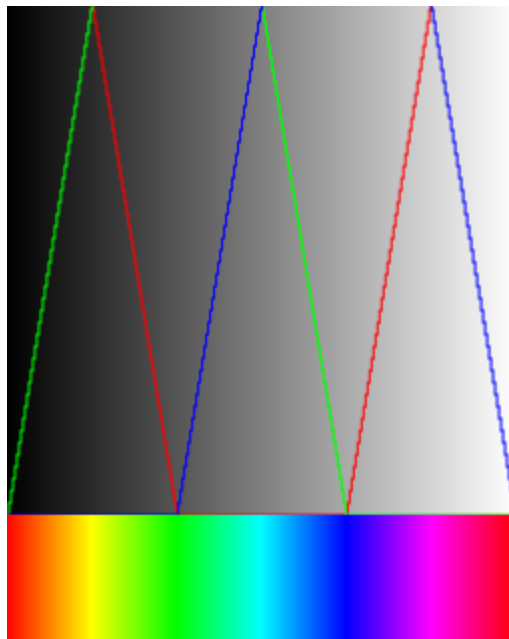
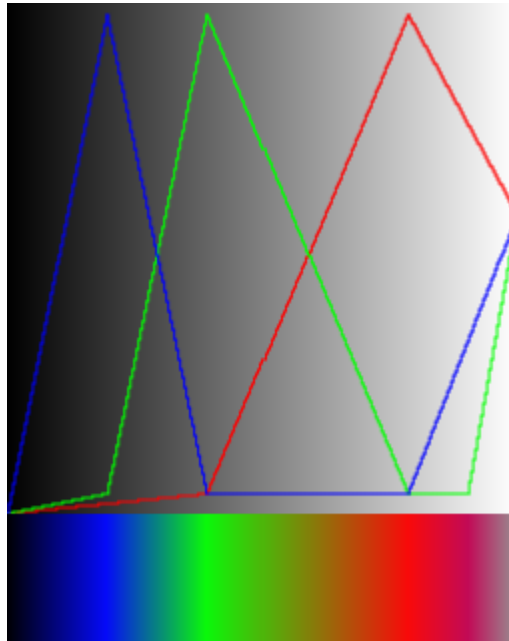
Fire

GreenFireBlue

Rainbow

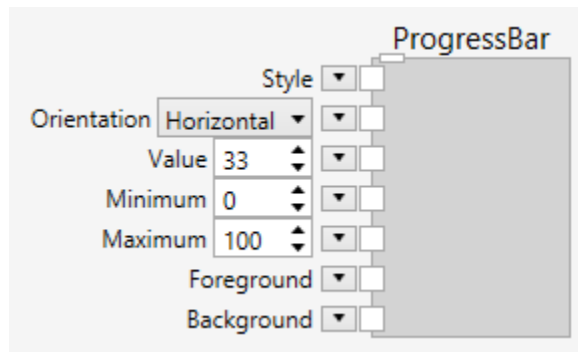
Spectrum





15.3.211 HMI ProgressBar

A **ProgressBar** can be used to move to or visualize a position.



Inputs

Style (Type: Style)

Optional styling of the **ProgressBar**.

The **ProgressBar** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Orientation (Type: string)

Orientation of the **ProgressBar**, `Horizontal` or `Vertical`.

Value (Type: double)

The position.

Minimum (Type: double)

The minimum value.

Maximum (Type: double)

The maximum value.

Foreground (Type: SolidColorBrush)

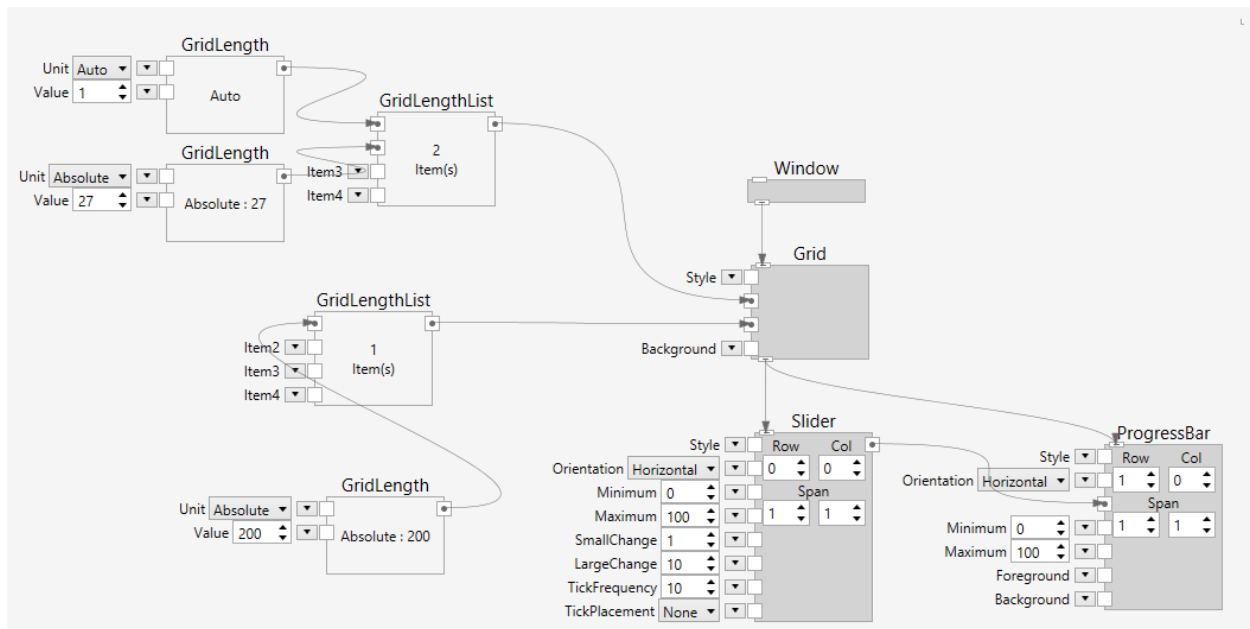
The foreground.

Background (Type: SolidColorBrush)

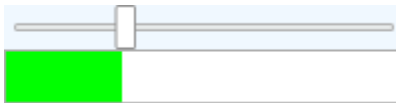
The background.

Example

Here is an example that shows a **ProgressBar** coupled to a **Slider**. This definition:

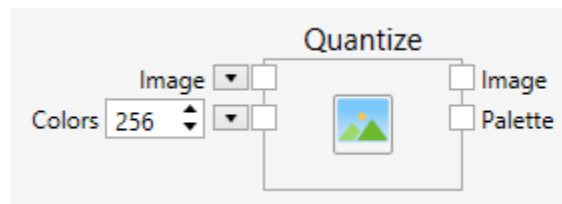


creates the following user interface:



15.3.212 Quantize

Quantize an image down to 256 or less colors.



Inputs

Image (Type: Image)

The input image.

Colors (Type: Int32)

The number of colors in the output image.

Outputs

Image (Type: Image)

The output image.

Palette (Type: Palette)

The output color palette.

Comments

This function reduces the number of colors in an image to the specified number of 256 or less. The result is a one-channel image and an associated color palette.

Color image quantized to 256 colors:

Color image quantized to 16 colors:

Color image quantized to 4 colors:

Sample

Here is an example:

15.3.213 RadialGradient

Presets an image with a radial gradient pattern.

Inputs

Center (Type: PointDouble)

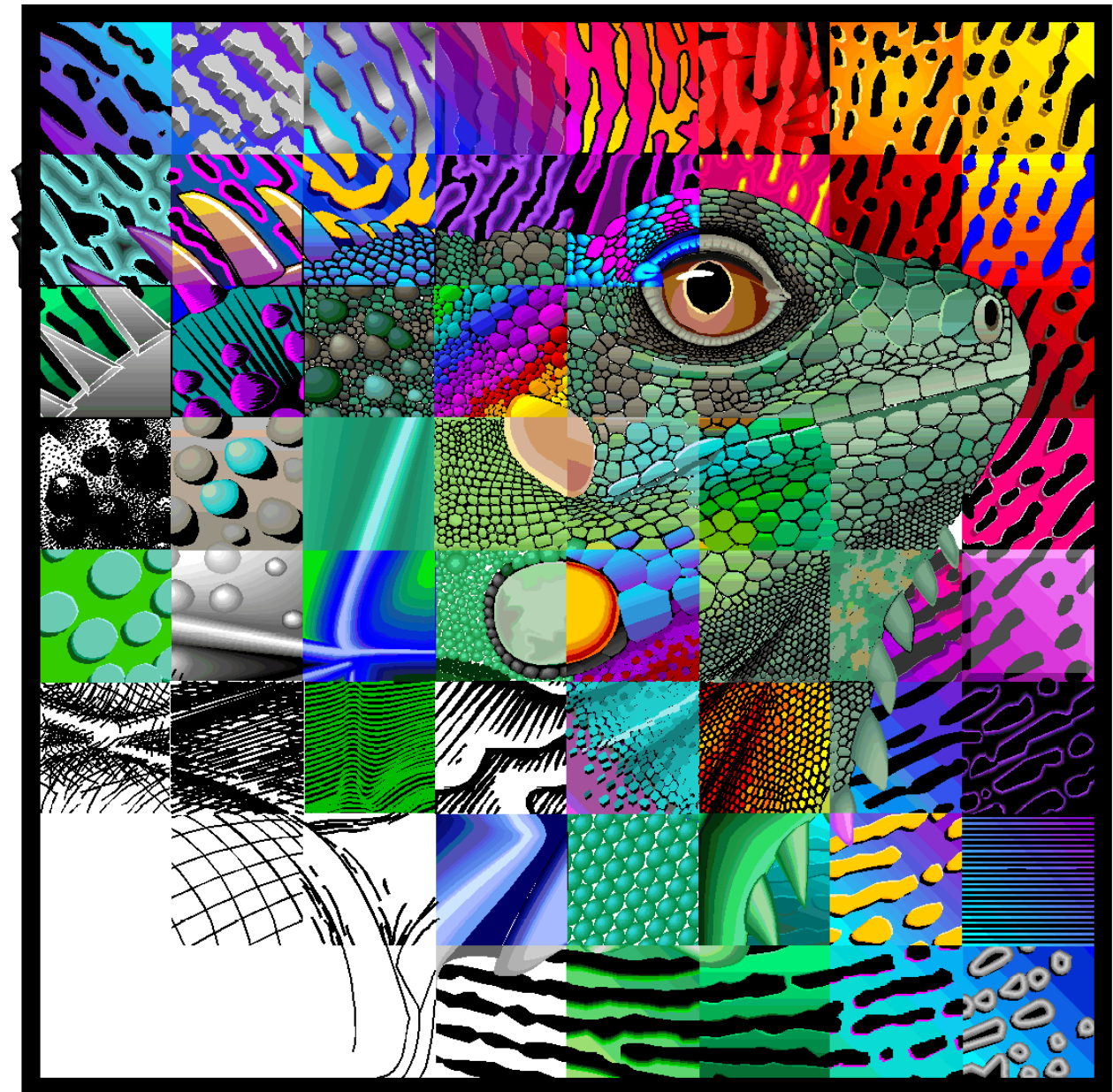
The center of the gradient. The default is 512, 512.

Radius (Type: Double)

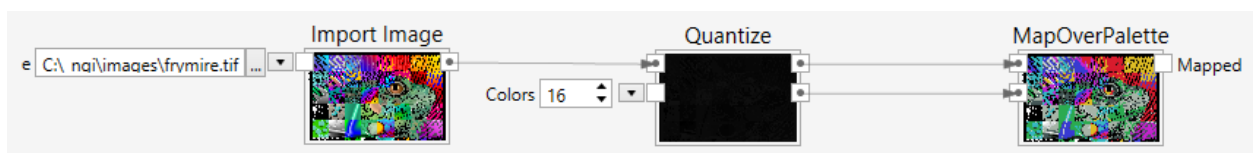
The radius of the gradient. The default is 512.

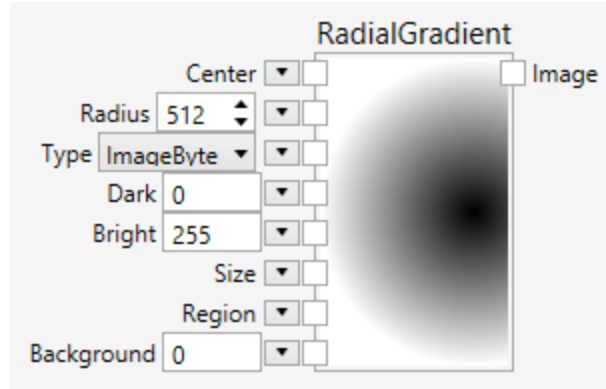
Type (Type: String)

The image type. The following types can be chosen: ImageByte, ImageUInt16, ImageUInt32, ImageDouble, ImageRgbByte, ImageRgbUInt16, ImageRgbUInt32, ImageRgbDouble









Dark (Type: String)

The value of a dark checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Bright (Type: String)

The value of a bright checkerboard field. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Size (Type: Extent3d)

The size of the image that is created. The default value is 1024 x 1024 x 1 pixels.

Region (Type: Region)

An optional region that constrains the preset operation to inside the region only. Pixels outside the region are colored with the Background color.

Background (Type: String)

The preset value outside of the region. This value is converted into the appropriate pixel type, depending on the Type parameter. For monochrome images you should enter one number, for rgb color images you should enter three numbers separated by a space.

Outputs

Image (Type: Image)

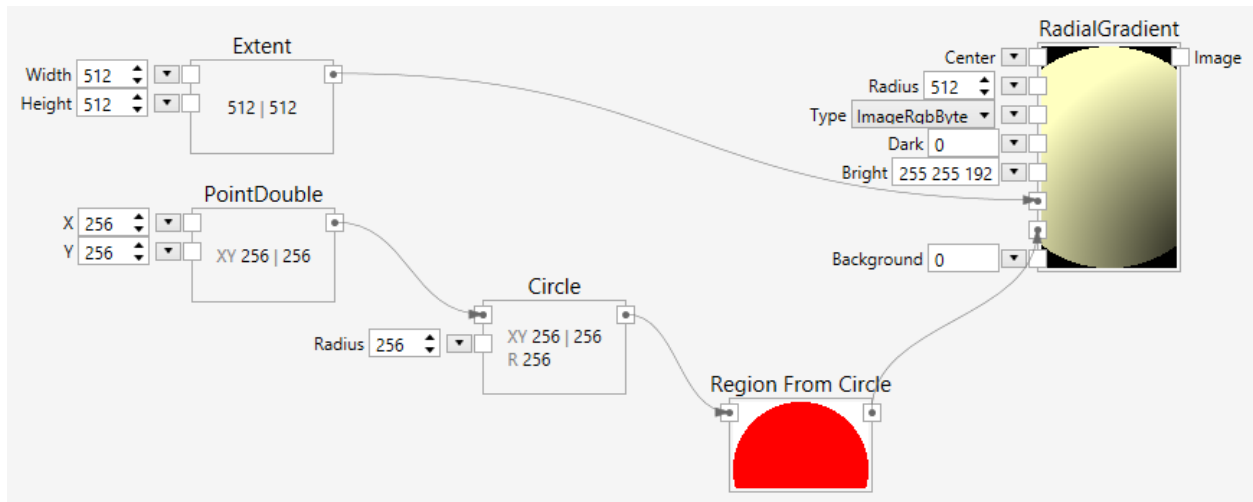
The image preset with a radial gradient pattern.

Comments

The **RadialGradient** node presets an image with a radial gradient pattern. The size of the created image, the colors for dark and bright checkerboard fields, the region of interest and the background color can all be specified.

Sample

Here is an example that shows how to use the **RadialGradient** node.



Here is the image created with the sample:

15.3.214 HMI RadioButton

Buttons are used to switch or visualize state, as well as to trigger actions. The **RadioButton** is used in groups, where only one of the buttons can be checked.

A **RadioButton** is used in a group with other **RadioButton**s. Only one of them is checked at a time. If another **RadioButton** in the same group is checked, the previously checked one is unchecked.

At the bottom of the **RadioButton** node is a pin that allows it to connect one child.

Inputs

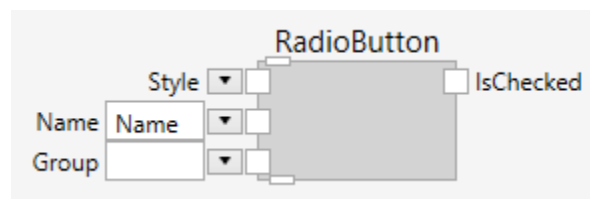
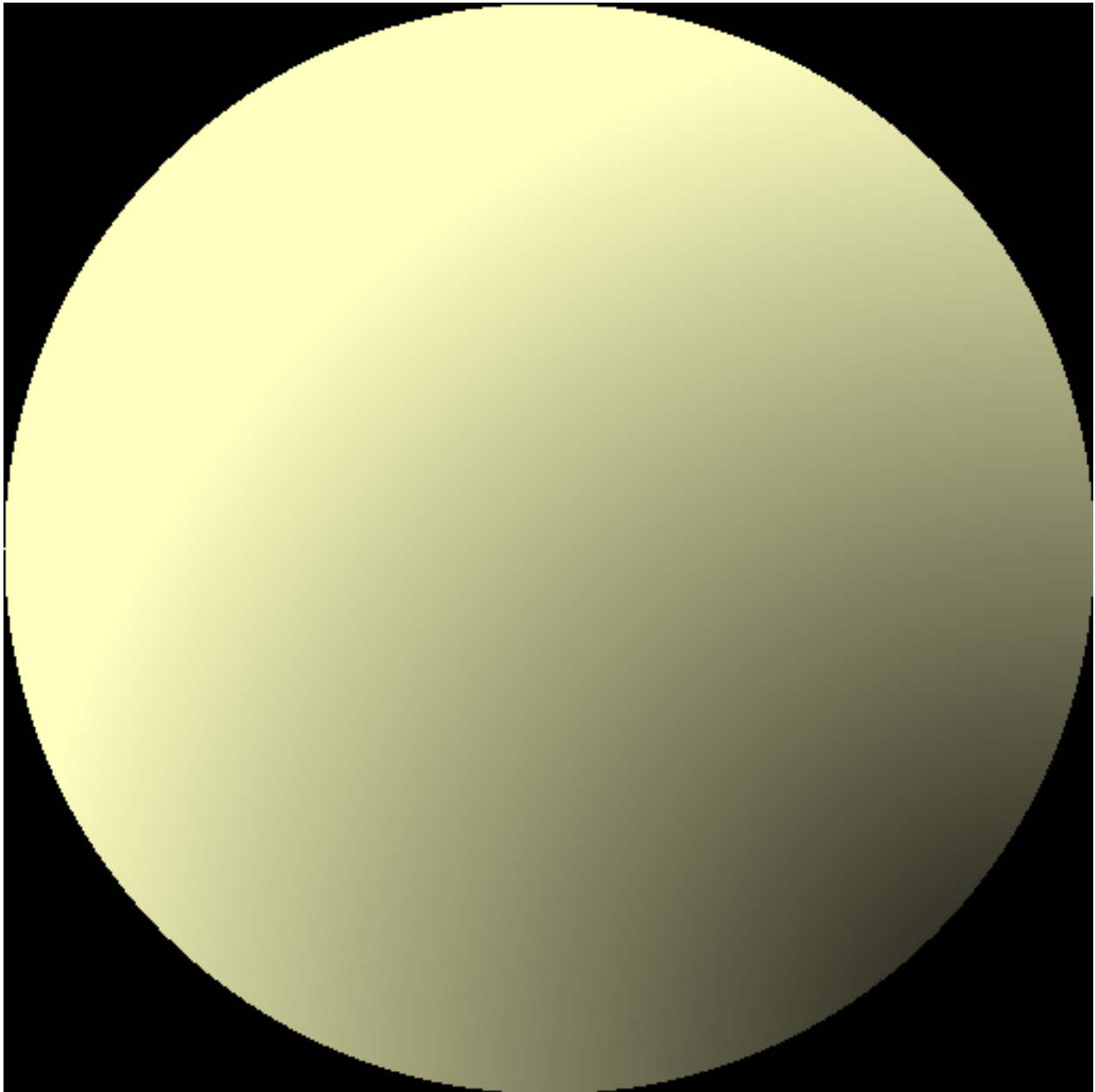
Style (Type: *Style*)

Optional styling of the **RadioButton**.

The **RadioButton** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Name (Type: *string*)

The text that is displayed within the button. This text is only displayed, when no child is connected. If a child is connected, the visual definition of the child is used.



Group (Type: string)

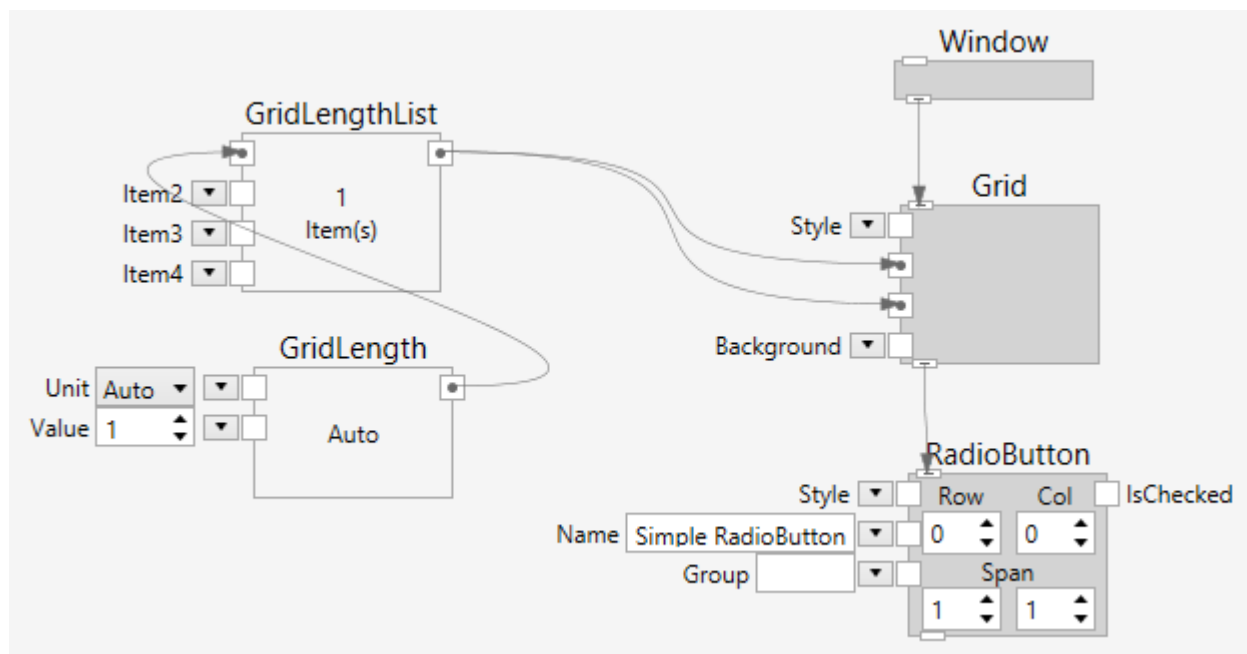
The group of the **RadioButton**. **RadioButtons** with the same group text are in the same group.

Outputs**IsChecked (Type: boolean)**

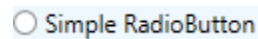
True if the radio button is checked, False otherwise.

Example

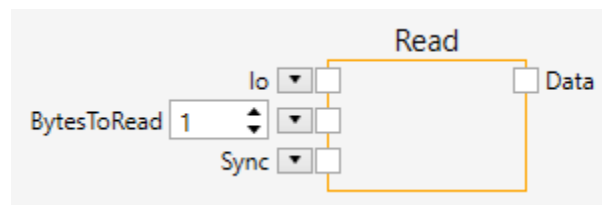
Here is an example that shows a simple **RadioButton**. This definition:



creates the following user interface:

**15.3.215 Read**

Read data from an interface.



This node reads data from an interface (such as a serial port).

Inputs

Io (Type: IoResource)

The IO interface.

BytesToRead (Type: Int32)

The number of bytes to read from the interface.

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs

Data (Type: DataList)

The data that has been read in the form of a DataList (a list of bytes).

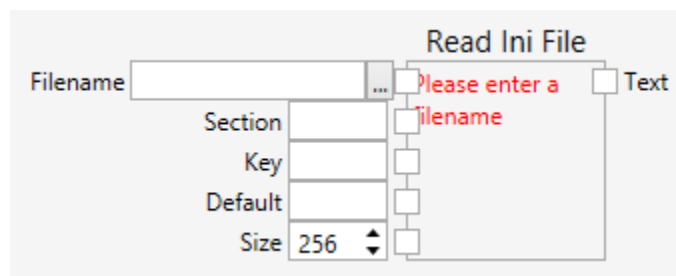
Comments

The **Sync** input and the **Data** output can be used to establish an order of execution.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

15.3.216 Read Ini File

Reads values from an Ini file. Ini files are used to store configuration information.



Inputs

Filename

The path to the file.

Section

The section name in the Ini file.

Key

The key name inside the section.

Default

The default value, if the key could not be found.

Outputs

Text

The value read from the Ini file.

Comments

Wikipedia says the following about Ini files: The Ini file format is an informal standard for configuration files for some platforms or software. Ini files are simple text files with a basic structure composed of sections, properties, and values.

The basic element contained in an Ini file is the key or property. Every key has a name and a value, delimited by an equals sign (=). The name appears to the left of the equals sign.

```
name=value
```

Keys may (but need not) be grouped into arbitrarily named sections. The section name appears on a line by itself, in square brackets ([and]). All keys after the section declaration are associated with that section. There is no explicit “end of section” delimiter; sections end at the next section declaration, or the end of the file. Sections may not be nested.

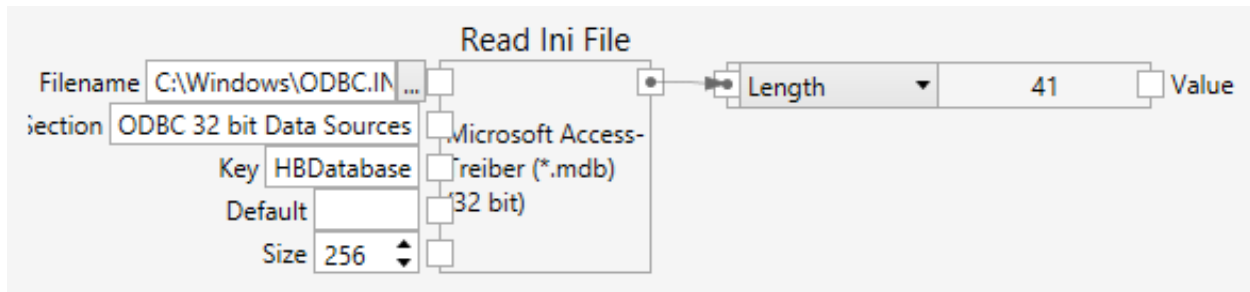
```
[section]
a=10
b=20
```

Section and property names are not case sensitive in the Windows implementation.

The Read Ini File checks the time the file was last written to, and compares it with the time it last read from the file. If the file has been written to after it has been last read, the node reads the file again.

Sample

Here is a sample pipeline that illustrates **ReadIniFile**:

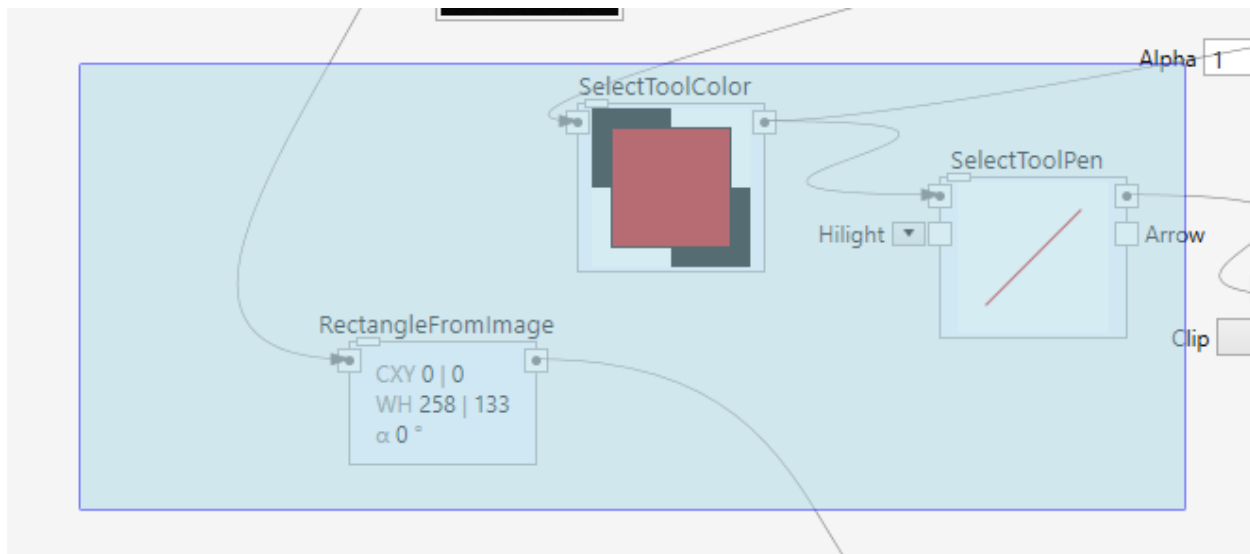


15.3.217 Group

You can use the **Group** command from the pipeline menu to move the selected nodes and their inside connections to a newly created subpipeline. The outside connections are automatically turned into input and output ports of the subpipeline.

You can select nodes by clicking them with the left mouse button. Hold down the **Ctrl** key if you want to add to (or subtract from) the selection.

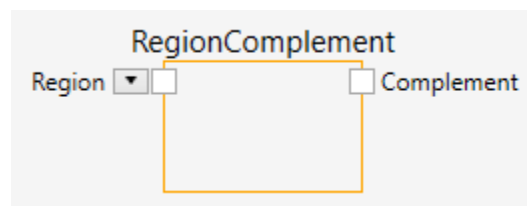
You can also select a set of nodes inside a rectangle that you drag while holding down the **Ctrl** key.



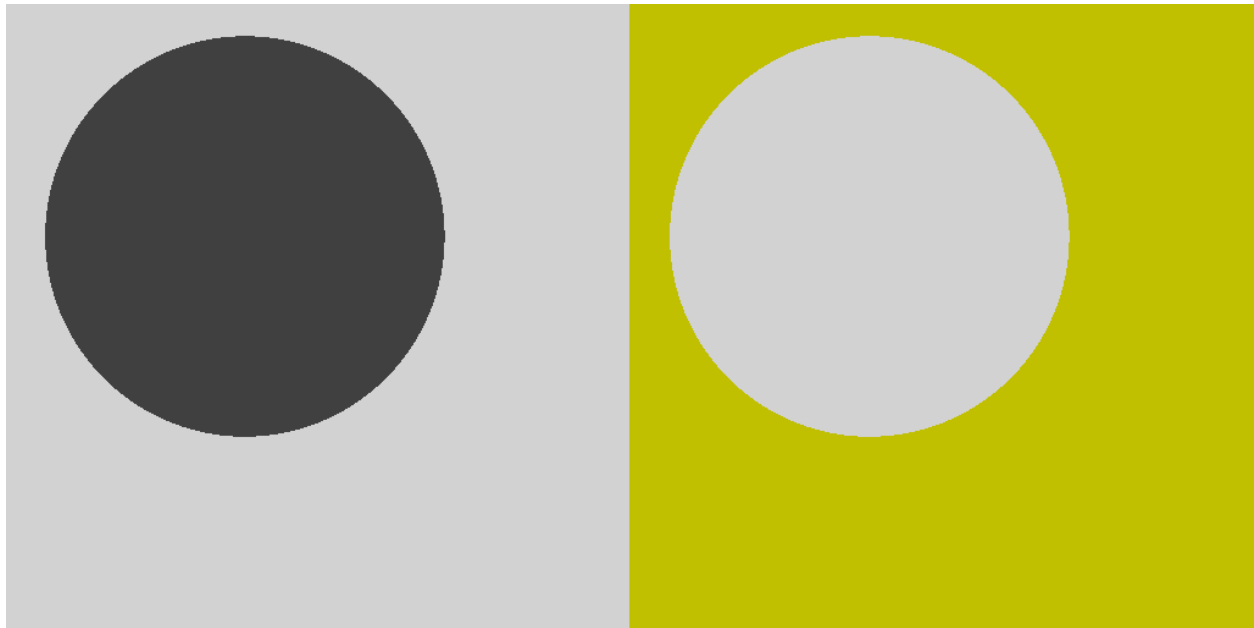
If the selected nodes have vertical widget or HMI connections to the outside, the command has no effect.

15.3.218 RegionComplement

Calculates the complement of a region.



Calculates the complement (yellow) of a region (dark).



A -> Complement

Inputs

Region (Type: Region)

The input region.

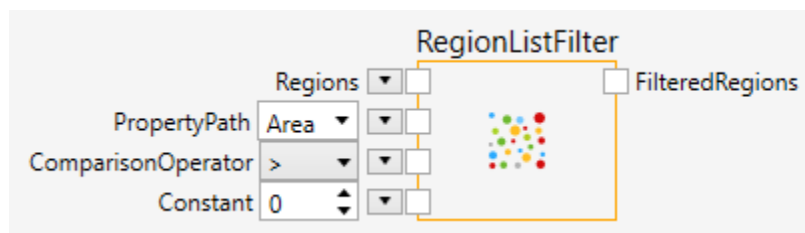
Outputs

Result (Type: Region)

The result region.

15.3.219 RegionListFilter

Filters the regions in a list according to a criterium.



Inputs

Regions (Type: `RegionList`)

The list of regions.

PropertyPath (Type: `string`)

Specifies the property that is used for filtering. All the blob analysis features (Area, Perimeter, etc.) can be used for filtering.

The full set of features is described in the Blob Analysis chapter of the nVision User Guide.

Some features have sub-features, such as the Centroid. For these features, you can use the dot (`.`), to specify the desired feature, such as `RegionCentroid.X`.

ComparisonOperator (Type: `string`)

Specifies the operator (`<`, `<=`, `>`, `>=`, `==` or `!=`).

Constant (Type: `Double`)

The constant used for the comparison.

Outputs

FilteredRegions (Type: `RegionList`)

The resulting list of regions, which contains the regions that satisfy the criterium.

Comments

You can chain `RegionListFilter` nodes to filter for more than one criterium.

You can combine results of several `RegionListFilter` nodes with the set operations on available for lists (Union, Intersect, Except) to create more complex filters.

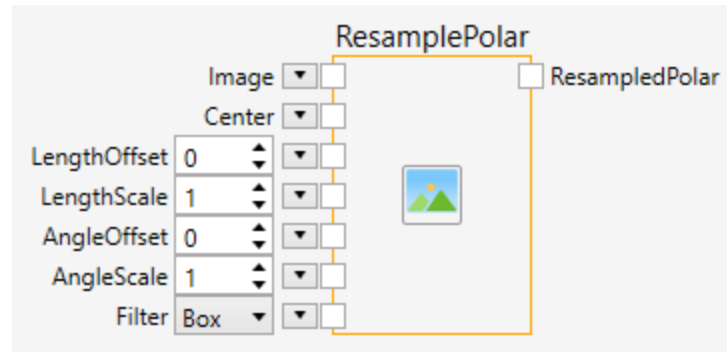
15.3.220 ResamplePolar

Performs a transformation from a polar to a rectangular coordinate system.

Inputs

Image (Type: `Image`)

The input image.

**Center (Type: `Ngui.PointDouble`)**

The polar center.

LengthOffset (Type: `Double`)

The radius from the center. This is mapped to the bottom of the resulting image.

LengthScale (Type: `Double`)

The scaling factor (radius to y-direction).

AngleOffset (Type: `Double`)

The starting angle in radians. This is mapped to the left of the resulting image.

AngleScale (Type: `Double`)

The scaling factor (angle to x-direction).

Filter (Type: `String`)

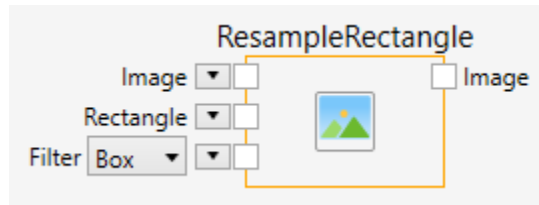
The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

Outputs**Resampled Polar (Type: `Image`)**

The output image (has the same size as the input image).

15.3.221 ResampleRectangle

Resamples a rectangular portion of an image.



Inputs

Image (Type: Image)

The input image.

Rectangle (Type: Ngi.Rectangle)

The rectangle. A rectangle can be oriented at any angle.

Filter (Type: String)

The geometric interpolation. Available values are NearestNeighbor, Box, Triangle, Cubic, Bspline, Sinc, Lanczos and Kaiser. The accuracy of the interpolation increases from NearestNeighbor to Kaiser, but the performance decreases. The default is set to Box.

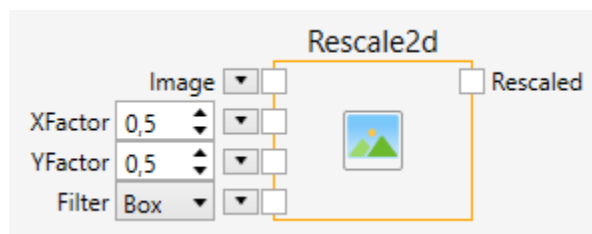
Outputs

Image (Type: Image)

The output image (has the same size as the rectangle, but rounded to an integer).

15.3.222 Rescale2D

Rescales an image.



Inputs

Image (Type: Image)

The input image.

XFactor (Type: Double)

The horizontal scaling factor.

YFactor (Type: Double)

The vertical scaling factor.

Filter (Type: String)

The geometric interpolation. Available values are `NearestNeighbor`, `Box`, `Triangle`, `Cubic`, `Bspline`, `Sinc`, `Lanczos` and `Kaiser`. The accuracy of the interpolation increases from `NearestNeighbor` to `Kaiser`, but the performance decreases. The default is set to `Box`.

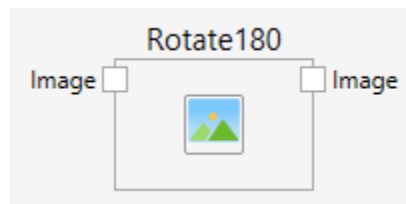
Outputs

Image (Type: Image)

The output image. The size of the output image is determined by the width and height of the input image multiplied by the horizontal and vertical scaling factors.

15.3.223 Rotate 180

Rotates an image by 180 degrees.



Inputs

Image (Type: Image)

The input image.

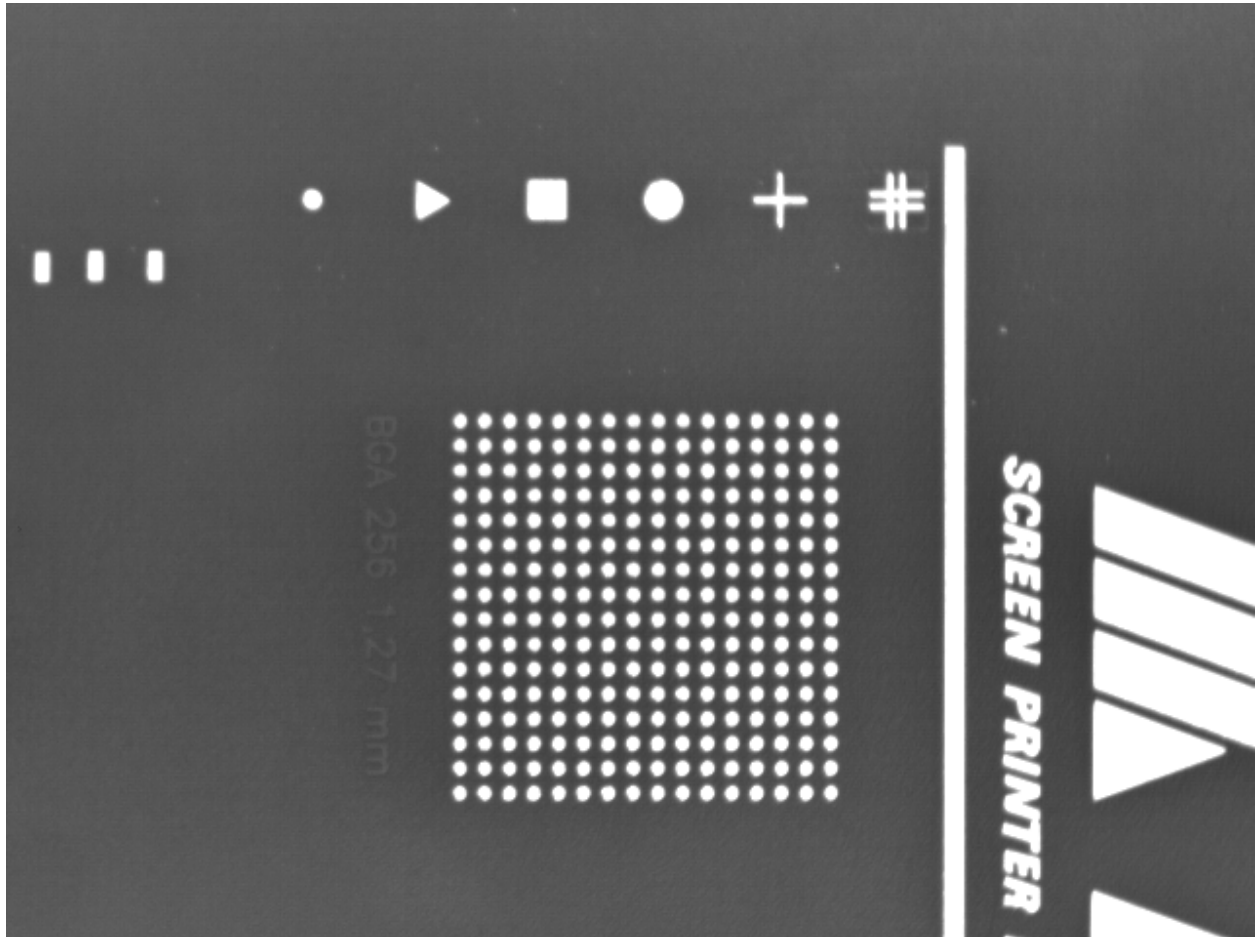
Outputs

Image (Type: Image)

The output image.

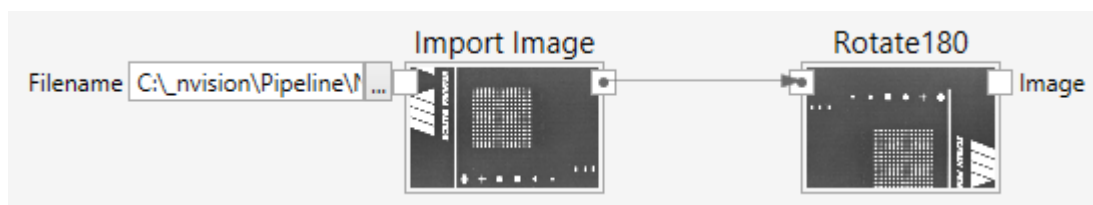
Comments

This function rotates an image by 180 degrees.



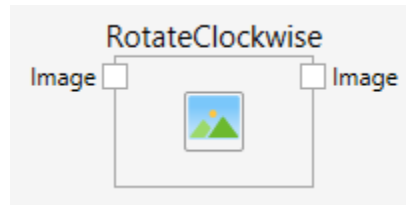
Sample

Here is an example that rotates an image.



15.3.224 Rotate Clockwise

Rotates an image clockwise by 90 degrees.



Inputs

Image (Type: Image)

The input image.

Outputs

Image (Type: Image)

The output image.

Comments

This function rotates an image by 90 degrees in clockwise direction.

Sample

Here is an example that rotates an image.

15.3.225 Rotate Counter Clockwise

Rotates an image counter-clockwise by 90 degrees.

Inputs

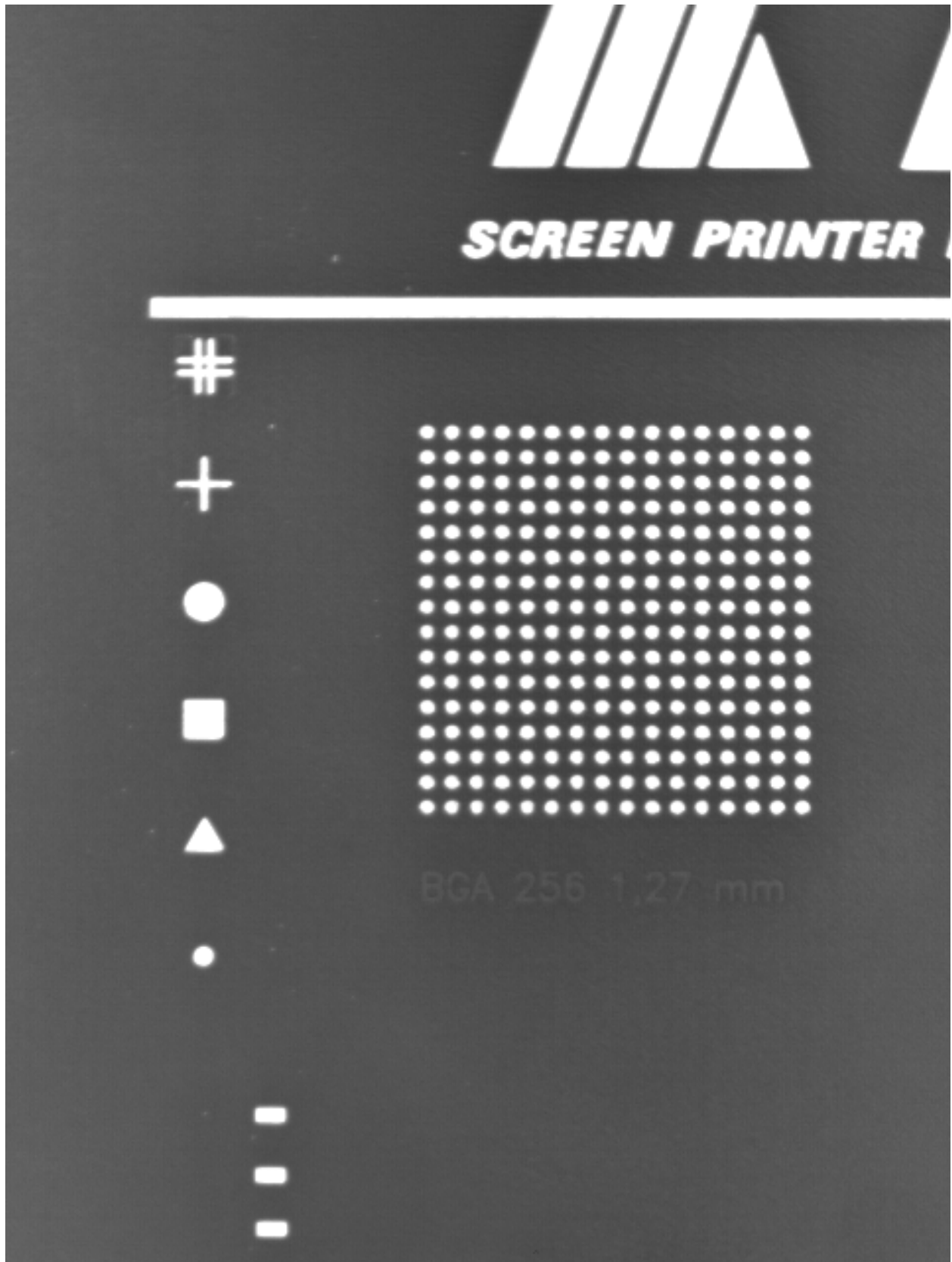
Image (Type: Image)

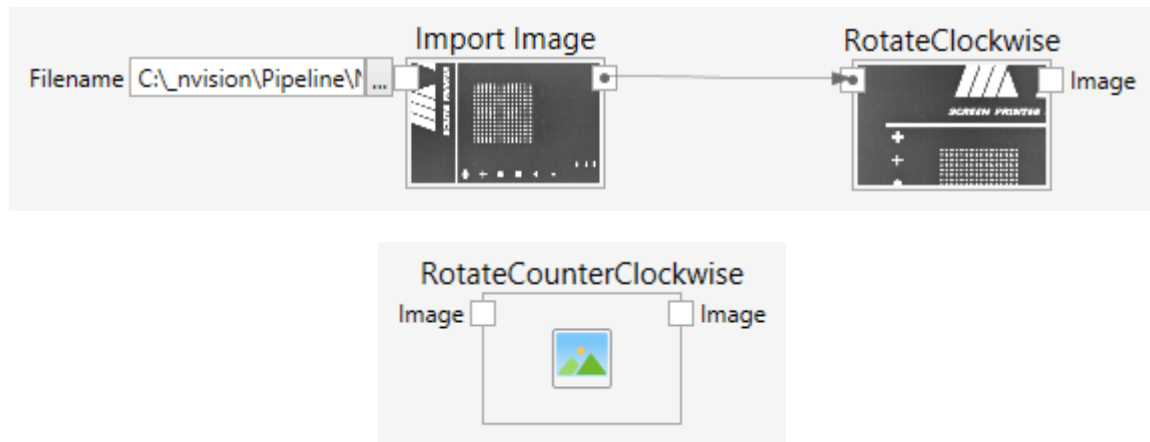
The input image.

Outputs

Image (Type: Image)

The output image.





Comments

This function rotates an image by 90 degrees in counter-clockwise direction.

Sample

Here is an example that rotates an image.

15.3.226 Save File

Saves text to a file.

Inputs

Text (Type: String)

The text that is written to the file.

Directory (Type: String)

A directory. This is preset with the user's documents folder.

Filename (Type: String)

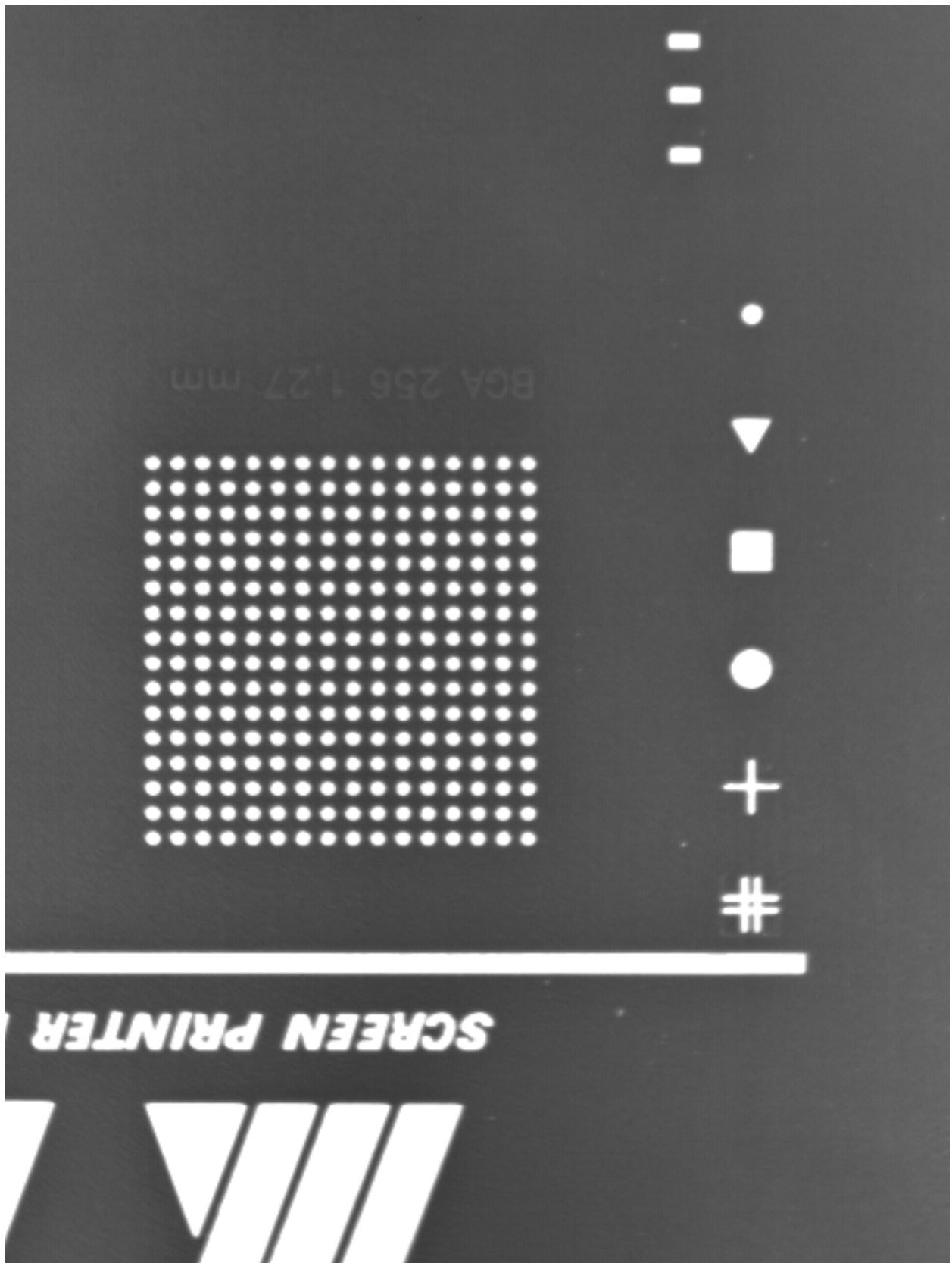
A filename. This is preset with "data.txt".

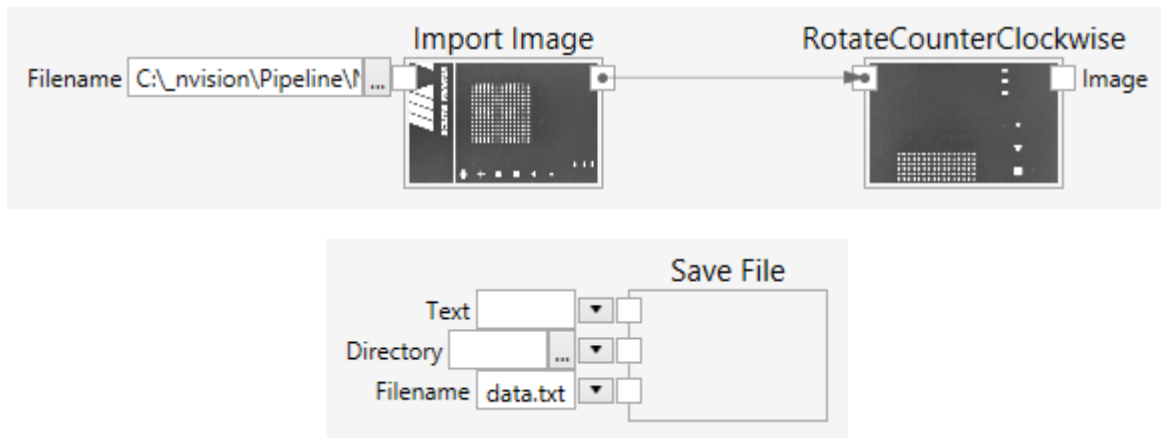
Comments

The **Save File** node writes text to a file.

The directory and filename of the file can be specified. If no directory is used, the node uses the user's documents folder. If no filename is used, the node uses the name "data.txt". If the directory does not exist, it will be created.

The node only executes, if the **Export On/Off** mode has been turned on,





or if the **Export Once** button is clicked.

Otherwise, the node does not execute and has no effect.

15.3.227 HMI Scrollbar

A **ScrollBar** can be used to scroll to or visualize a position.

Inputs

Style (Type: `style`)

Optional styling of the **ScrollBar**.

The **Border** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Orientation (Type: `string`)

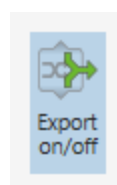
Orientation of the **ScrollBar**, *Horizontal* or *Vertical*.

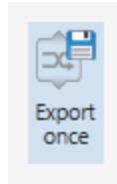
Minimum (Type: `double`)

The minimum value.

Maximum (Type: `double`)

The maximum value.





images/HMIScrollbarNode.png

SmallChange (Type: double)

The amount that is subtracted from or added to the value when the *Left* or *Right* keyboard cursor keys are pressed.

LargeChange (Type: double)

The amount that is subtracted from or added to the value when the channel is clicked left or right of the thumb.

Outputs

Value (Type: double)

The position.

Example

Here is an example that shows a **ScrollBar**. This definition:
creates the following user interface:

15.3.228 HMI Separator

The **Separator** allows you to specify a separator inside a toolbar.

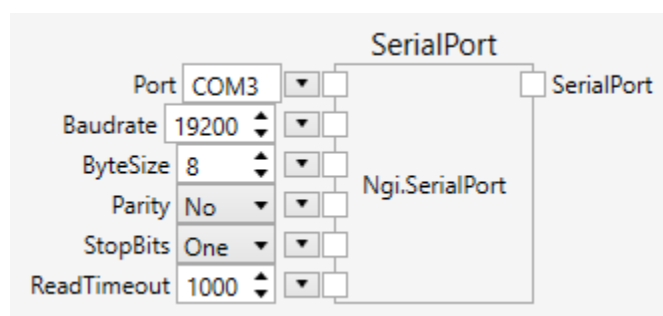
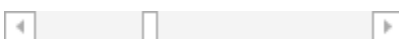
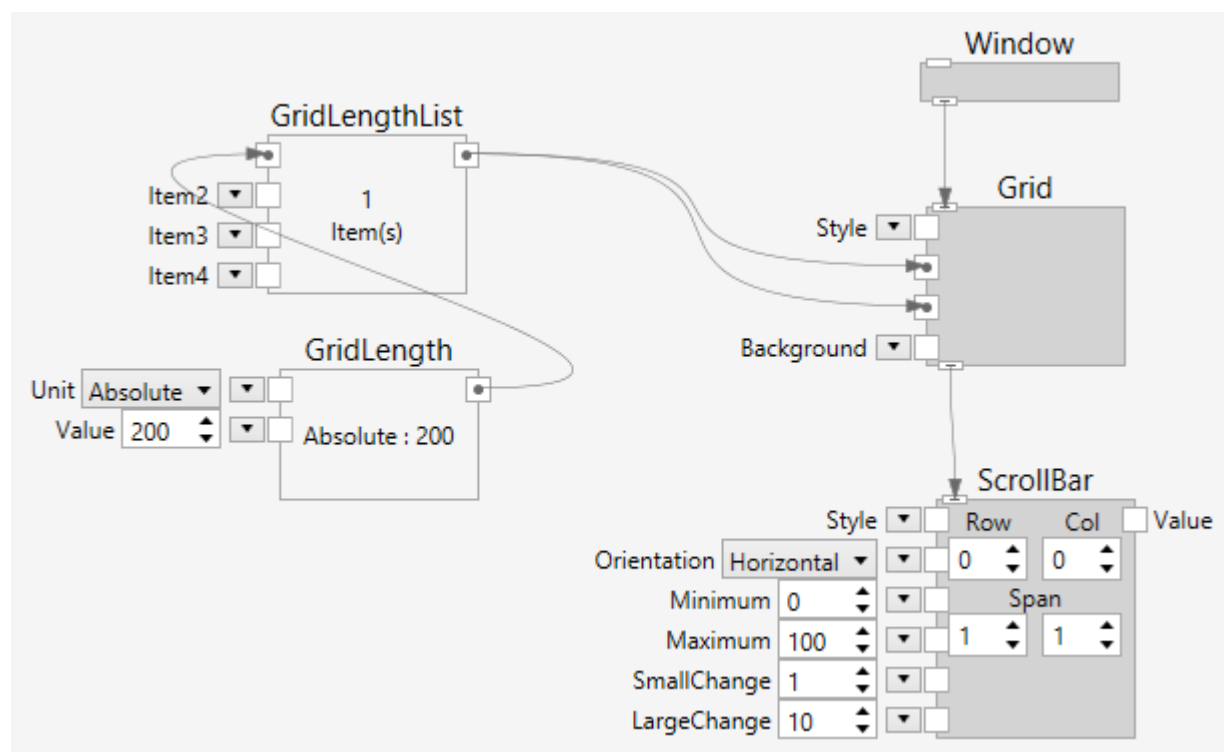
Inputs

Style (Type: style)

Optional styling of the **Separator**.

15.3.229 SerialPort

Establish a connection to a serial port, which allows you to communicate over a serial interface.



This node initializes the serial port interface. You can connect the output to a **Read** node to read data from the serial port, or a **Write** node to write data to the serial port.

Inputs

Port (Type: string)

The identifier of the serial port (COM1, COM2, etc.).

Baudrate (Type: UInt32)

The baudrate of the serial port (9600, 14400, 19200, etc.).

ByteSize (Type: UInt32)

The number of bits in the bytes transmitted and received (4 - 8).

Parity (Type: string)

The parity (No, Odd, Even, Mark or Space).

StopBits (Type: string)

The number of stop bits (One, OnePointFive or Two).

ReadTimeout (Type: UInt32)

The read timeout in milliseconds (0: polling, > 0: use the timeout for read operations).

Outputs

SerialPort (Type: SerialPort)

An instance of the serial port.

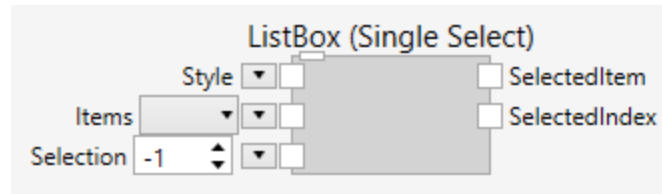
Comments

The output of the **SerialPort** node is usually connected to **Read** or **Write** nodes, in order to read from or write to the serial port, respectively.

Since establishing a connection to a serial port is often an operation with a somewhat global character, the **SerialPort** node is sometimes put into the pipeline globals or the system globals.

15.3.230 HMI ListBox (Single Select)

A single select **ListBox** can be used to select one from a number of entries.



Inputs

Style (Type: `Style`)

Optional styling of the **ListBox**.

The **ListBox** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Items (Type: `List<string>`)

A list of text items that is displayed in the **ListBox**.

Index (Type: `Int32`)

The initial selection of the **ListBox**. This is a zero-based index. You can use `-1` to specify *no selection*. This value is used only when the *Items* pin is connected.

Outputs

SelectedItem (Type: `string`)

The selected item.

SelectedIndex (Type: `Int32`)

The zero-based index of the selected item, or `-1` for *no selection*.

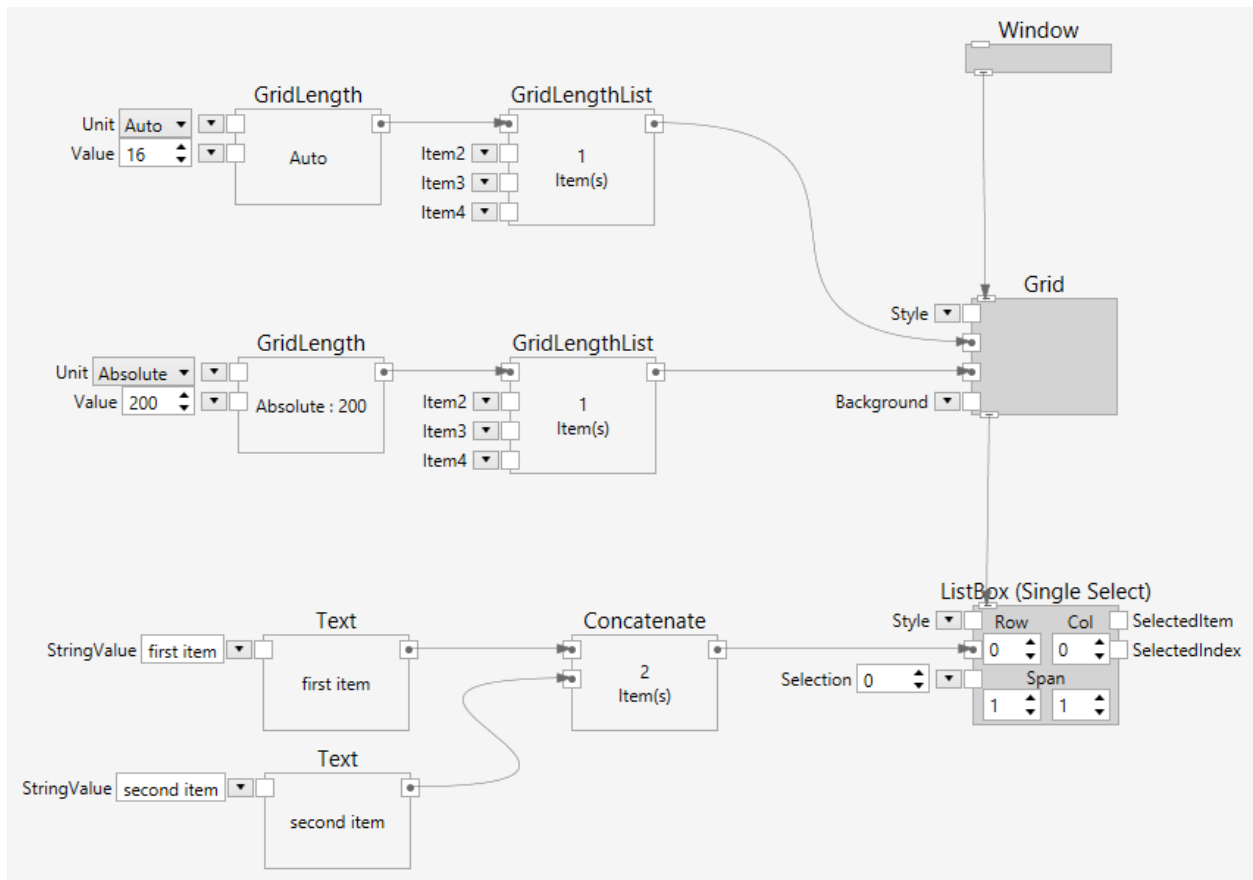
Example

Here is an example that shows a **ListBox**. This definition:

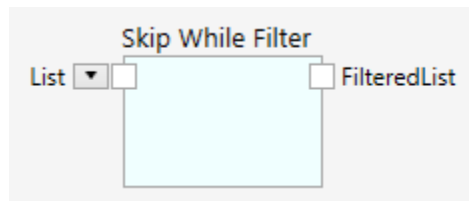
creates the following user interface:

15.3.231 Skip While

Skip While skips items from a list as long as a predicate is true. The predicate is specified itself as a graphical pipeline. The predicate is evaluated for each item of the list, beginning with the first item. As long as the predicate evaluates to true, list items are skipped. As soon as the predicate evaluates to false, the iteration is stopped, and the remaining items are put in the output list.



first item
second item



Outside View

At the outside, the **Skip While** node appears like a single node, with inputs and outputs.

Inputs

List (Type: List<T>)

The input list.

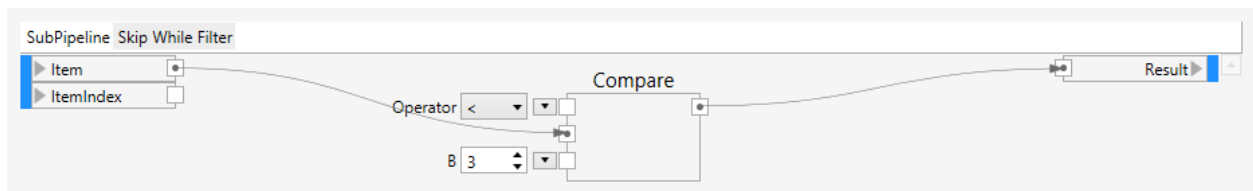
Outputs

FilteredList (Type: List<T>)

The output list, consisting of a part of the input list, as determined by consecutive execution of the predicate.

Inside View

Inside, the **Skip While** node looks like a normal branched pipeline. This pipeline is used to specify the predicate, which is evaluated for each item, until the predicate turns false.



Inputs

Item (Type: T)

One item of the input list.

ItemIndex (Type: Int32)

The index of the item of the input list.

Outputs

Result (Type: Boolean)

The boolean predicate.

Comments

Skip While nodes can be nested in similar ways as subpipelines.

The nesting is shown with a breadcrumbs control at the top.

First Level Second Level Third Level

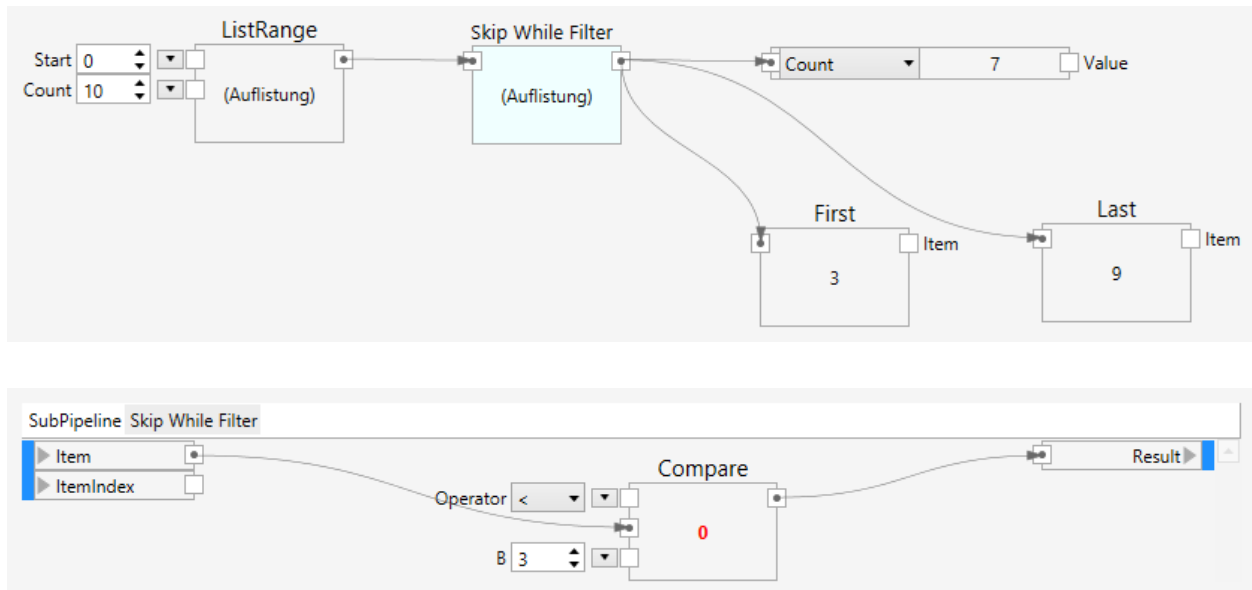
Double-click a **Skip While** node to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

By default, a **Skip While** node has one input and one output.

You can add additional inputs with the **Add Input (Ctrl-I)** command. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

Sample

Here is an example:



It skips items from the first list (0, 1, ..., 9) while the predicate (< 3) is true and returns the resulting list (3, 4, ..., 9).

15.3.232 HMI Slider

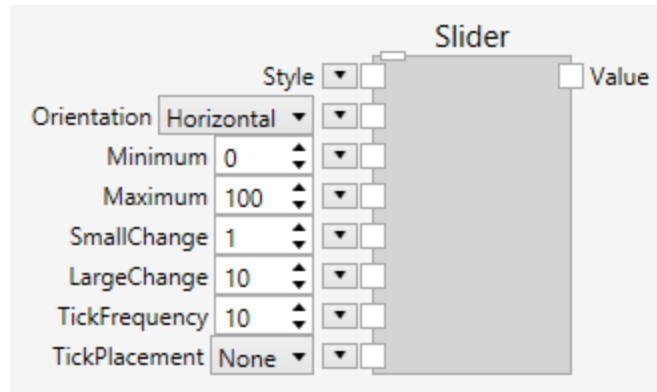
A **Slider** can be used to move to or visualize a position.

Inputs

Style (Type: `Style`)

Optional styling of the **Slider**.

The **Slider** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

**Orientation (Type: string)**

Orientation of the **Slider**, `Horizontal` or `Vertical`.

Minimum (Type: double)

The minimum value.

Maximum (Type: double)

The maximum value.

SmallChange (Type: double)

The amount that is subtracted from or added to the value when the *Left* or *Right* keyboard cursor keys are pressed.

LargeChange (Type: double)

The amount that is subtracted from or added to the value when the channel is clicked left or right of the thumb.

TickFrequency (Type: double)

The distance between tick marks.

TickPlacement (Type: string)

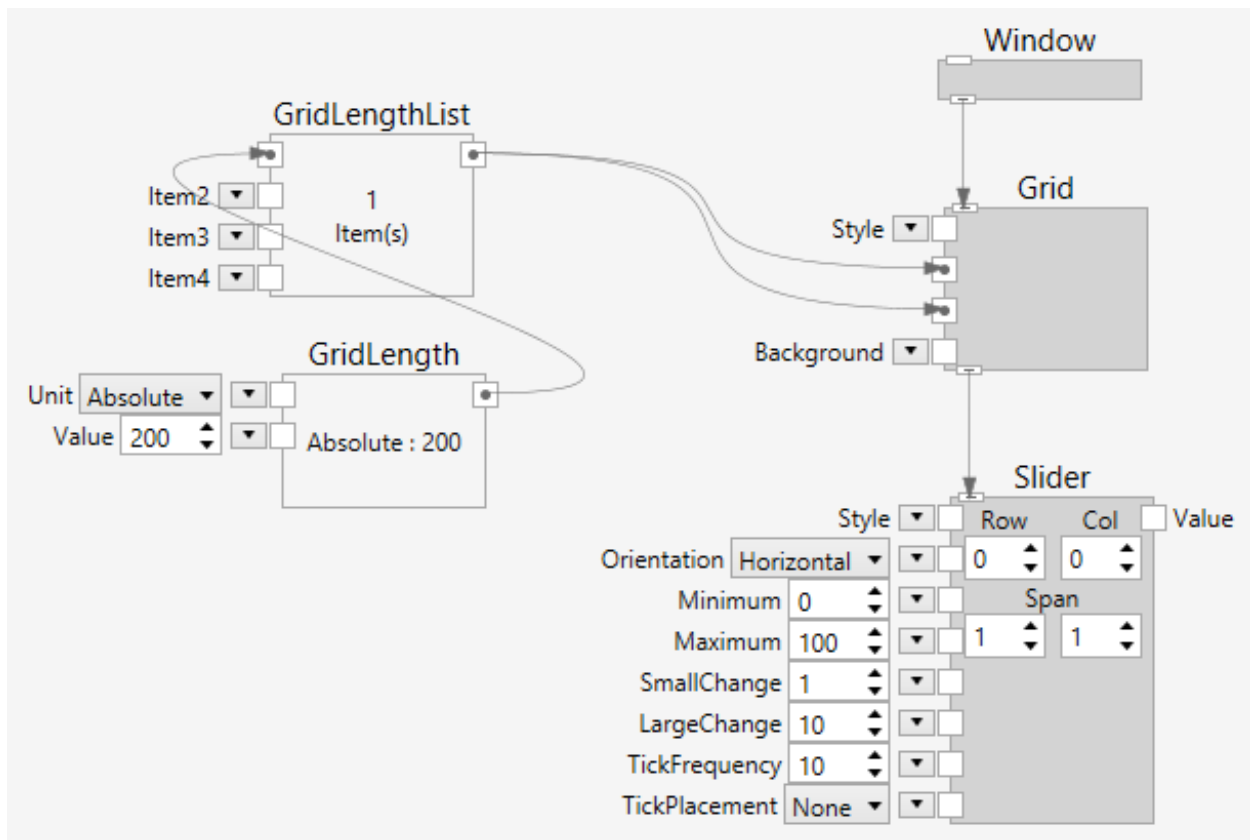
The placement of the ticks, `Both`, `BottomLeft`, `None` or `TopRight`.

Outputs**Value (Type: double)**

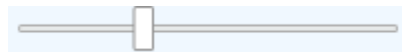
The position.

Example

Here is an example that shows a **Slider**. This definition:

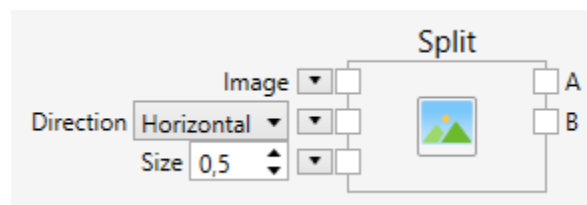


creates the following user interface:



15.3.233 Split

Splits an image in two halves.



Inputs

Image (Type: Image)

The input image.

Direction (Type: String)

The direction to split the image. This can be `Horizontal`, `Vertical` or `Planar` to split in the X, Y, or Z directions, respectively.

Size (Type: Double)

The split position as a number between 0 and 1 (both bounds are exclusive). A value of 0.5 splits the image in half.

Outputs**A (Type: Image)**

The first output image.

B (Type: Image)

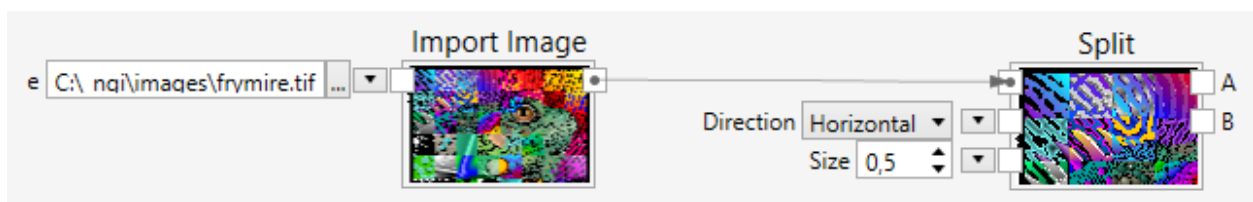
The second output image.

Comments

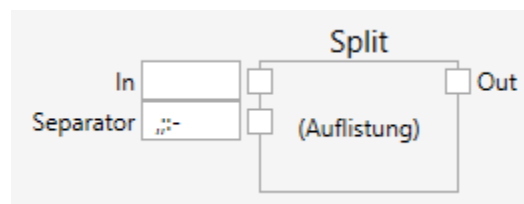
This function splits an image in two parts, in either the horizontal, vertical or planar directions.

Sample

Here is an example that extracts the three color channels from an RGB image.

**15.3.234 Split**

Splits text into a list of strings that contains the substrings that are delimited by elements of a specified text.



Inputs

In

The text that shall be split into parts.

Separator

The separator characters. The default separator characters contain the space, comma, semicolon, colon and dash characters (, ; : -).

If this input is empty, the string will be split at all whitespace characters.

To split the string at line ends, you can use `\n`.

To split the string at tabs, you can use `\t`.

Outputs

Out

A list of strings that are the substrings of the original strings.

Comments

The **Split** node is often used to break text into pieces that is read from file.

Since the output is a list, you can use the **GetListItem** node to pick out a specific item from the list.

Sample

Here is a sample pipeline that illustrates **Split**:

15.3.235 HMI StackPanel

The **StackPanel** stacks its child controls in the vertical or horizontal direction.

At the bottom of the **StackPanel** node is a pin that allows it to connect several children.

Inputs

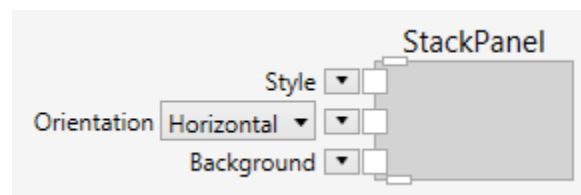
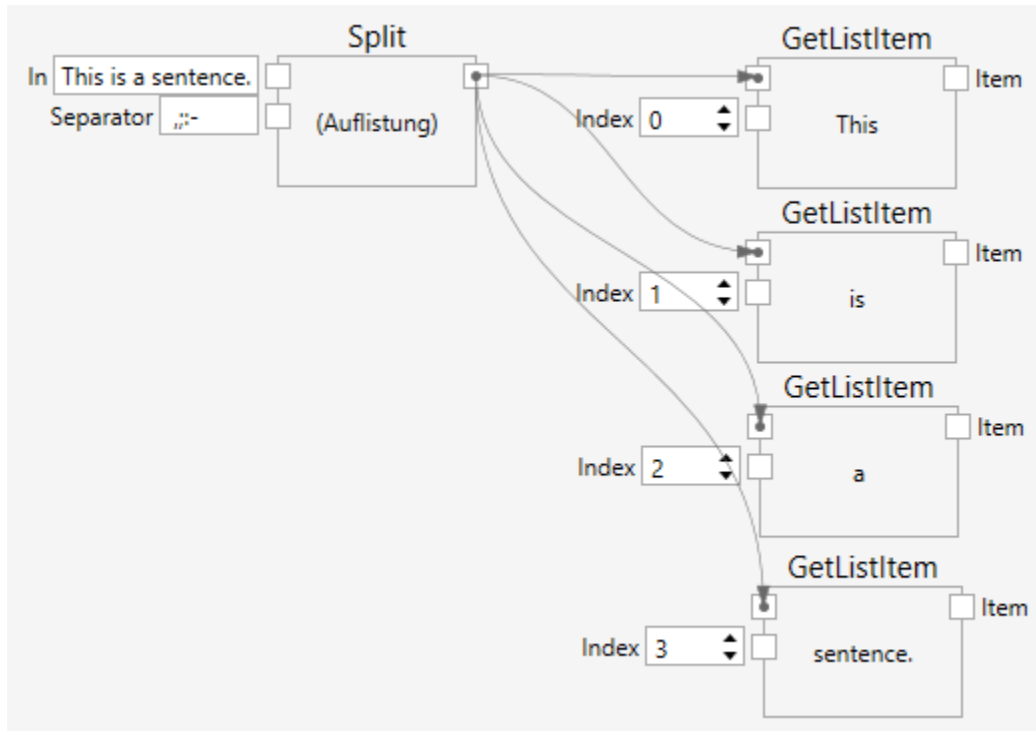
Style (Type: *Style*)

Optional styling of the **StackPanel**.

The **StackPanel** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Background* styles.

Orientation (Type: *string*)

Orientation of the **StackPanel**, *Horizontal* or *Vertical*.

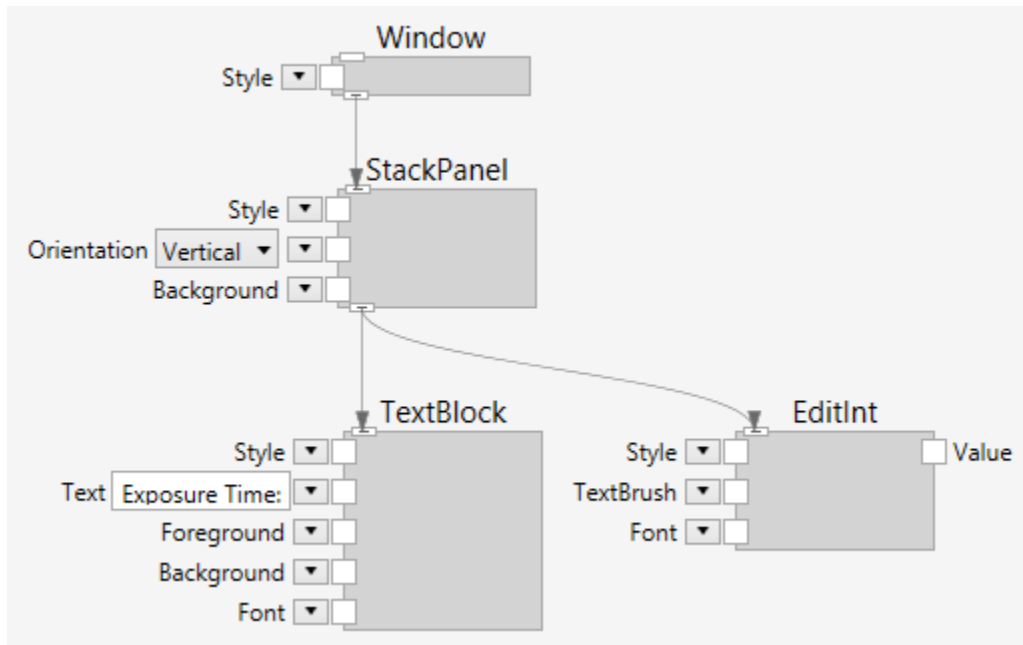


Background (Type: SolidColorBrushByte)

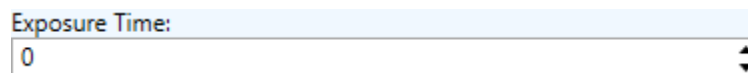
The background of the **StackPanel**.

Example

Here is an example that stacks a text label and an edit control vertically. This definition

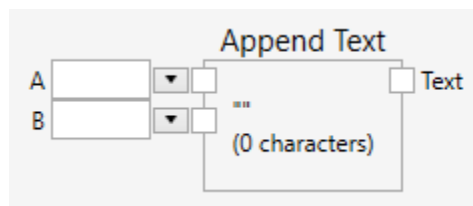


creates the following user interface:



15.3.236 Concat Text

Concatenates two or more text values.



Inputs

A (Type: String)

The first operand.

B (Type: String)

The second operand.

C...Z (Type: String)

Additional operands, added on demand.

Outputs**Text (Type: String)**

The concatenation of all operands.

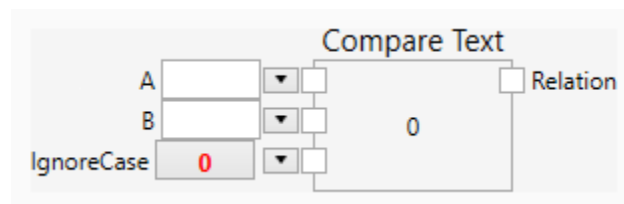
Comments

This node concatenates multiple text operands.

The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.237 Compare Text

Compares two texts, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.

**Inputs****A (Type: String)**

The first text to compare.

B (Type: String)

The second text to compare.

IgnoreCase (Type: Boolean)

True to ignore case during the comparison, false otherwise.

Outputs

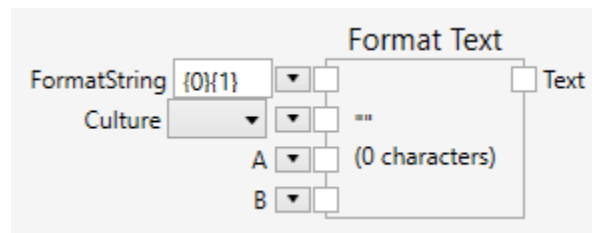
Relation (Type: Int32)

An integer that indicates the relationship between the two texts:

value	condition
< 0	A precedes B in the sort order.
0	A occurs in the same position as B in the sort order.
> 0	A follows B in the sort order.

15.3.238 Format Text

Formats a text using two or more text values.



Inputs

FormatString (Type: String)

todo

Culture (Type: “)

A (Type: String)

The first operand.

B (Type: String)

The second operand.

C...Z (Type: String)

Additional operands, added on demand.

Outputs

Text (Type: String)

todo

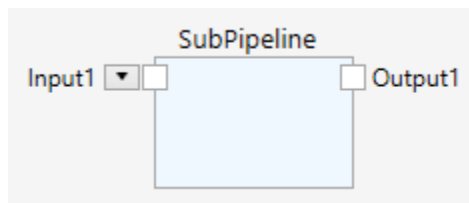
Comments

todo.

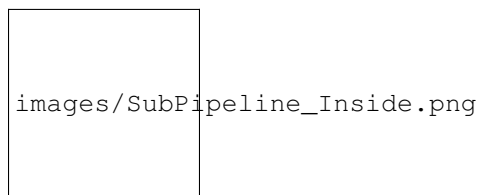
The ports for the inputs are added dynamically on demand. The node starts with the two inputs A and B. If both are connected, a third input named C is added. At a maximum, 26 inputs are possible. Inputs can also be deleted by removing the connection to them.

15.3.239 SubPipeline

A subpipeline groups several nodes and their connections. This is a means to organize code.



At the outside, a subpipeline appears like a single node, with inputs and outputs. Inside, a subpipeline looks like a normal branched pipeline.



A subpipeline can be nested, that is you can have subpipelines inside other subpipelines.

The nesting of subpipelines is shown with a breadcrumbs control at the top.

First Level Second Level **Third Level**

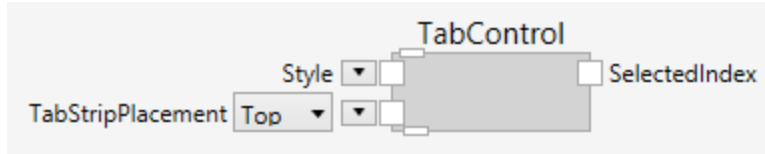
Double-click a subpipeline to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

By default, a subpipeline has one input and one output. You can add inputs and outputs with the **Add Input (Ctrl-I)** and the **Add Output (Ctrl-O)** commands. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

15.3.240 HMI TabControl

The **TabControl** creates a tab control and displays its children in tabs.

At the bottom of the **TabControl** node is a pin that allows it to connect multiple children (use **TabItem** here).



Inputs

Style (Type: `Style`)

Optional styling of the **TabControl**.

The **TabControl** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

TabStripPlacement (Type: `string`)

The placement of the tabstrip. The values `Bottom`, `Left`, `Top` and `Right` are possible.

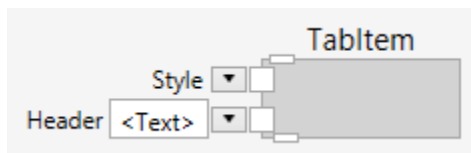
Outputs

SelectedIndex (Type: `Int32`)

The index of the selected tab.

15.3.241 HMI TabItem

The **TabItem** puts an expandable and collapsible area around its child.



At the bottom of the **TabItem** node is a pin that allows it to connect one child.

Inputs

Style (Type: `Style`)

Optional styling of the **Expander**.

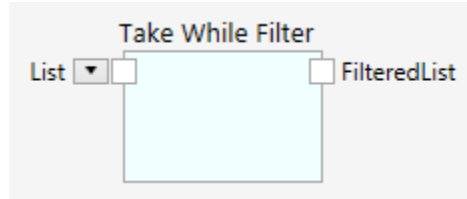
The **Expander** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding* and *Background*, *Foreground* and *Font* styles.

Header (Type: `string`)

The title of the **TabItem**.

15.3.242 Take While

Take While takes items from a list as long as a predicate is true. The predicate is specified itself as a graphical pipeline. The predicate is evaluated for each item of the list, beginning with the first item. As long as the predicate evaluates to true, list items are added to the output list. As soon as the predicate evaluates to false, the iteration is stopped.



Outside View

At the outside, the **Take While** node appears like a single node, with inputs and outputs.

Inputs

List (Type: List<T>)

The input list.

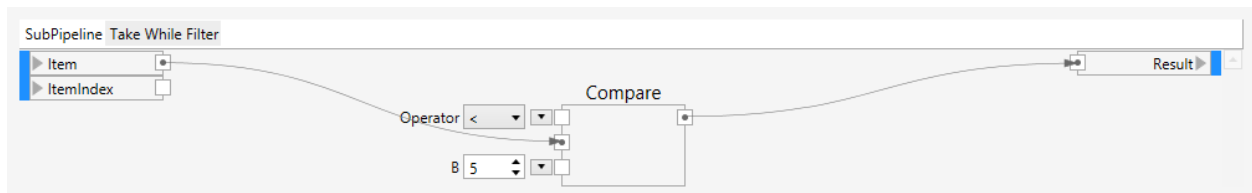
Outputs

FilteredList (Type: List<T>)

The output list, consisting of a part of the input list, as determined by consecutive execution of the predicate.

Inside View

Inside, the **Take While** node looks like a normal branched pipeline. This pipeline is used to specify the predicate, which is evaluated for each item, until the predicate turns false.



Inputs

Item (Type: T)

One item of the input list.

ItemIndex (Type: Int32)

The index of the item of the input list.

Outputs**Result (Type: Boolean)**

The boolean predicate.

Comments

Take While nodes can be nested in similar ways as subpipelines.

The nesting is shown with a breadcrumbs control at the top.

First Level Second Level Third Level

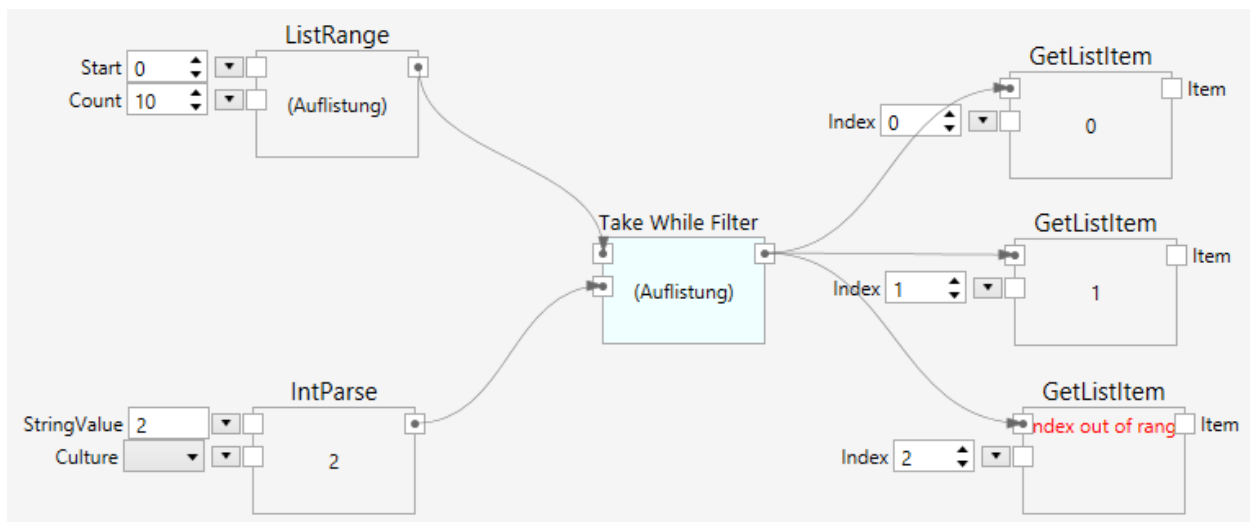
Double-click a **Take While** node to enter it, and use the breadcrumbs control to exit it and go up one or more levels.

By default, a **Take While** node has one input and one output.

You can add additional inputs with the **Add Input (Ctrl-I)** command. You can edit the port names by clicking on them and typing a new name. The type of the ports is determined by the first connection made to them, either from the outside or from the inside.

Sample

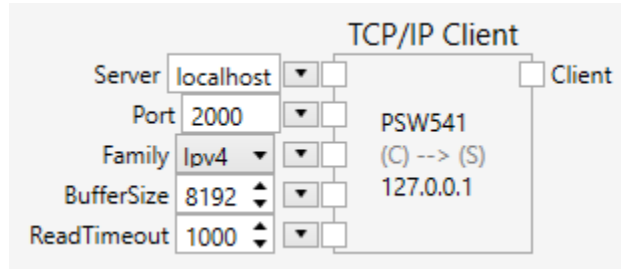
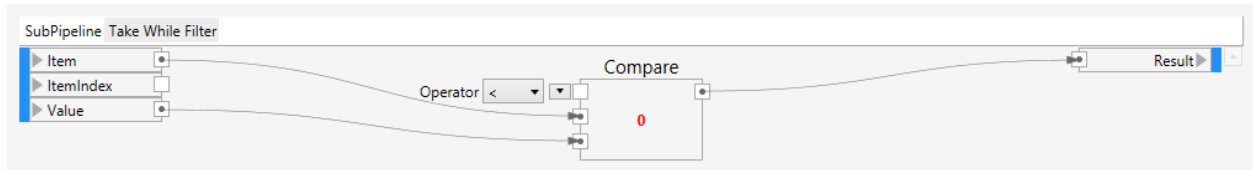
Here is an example:



It takes items from the first list (0, 1, ..., 9) while the predicate (< 2) is true and returns the resulting list (0, 1).

15.3.243 TCP/IP Client

Creates a TCP/IP Client.



Inputs

Server (Type: String)

The server name that should be connected. Strings such as `www.impuls-imaging.com`, `81.169.145.169`, `localhost`, `127.0.0.1`, etc., can be used.

Port (Type: String)

The port or service name. Strings such as `21`, `http`, etc. can be used. Possible values for the service name are listed in the file `%WINDIR%\system32\drivers\etc\services`.

Family (Type: String)

The address family for the connection. `Ipv4` or `Ipv6` are possible.

BufferSize (Type: String)

The size of the read and write buffers.

ReadTimeout (Type: String)

The timeout of a read operation.

Outputs

Client (Type: TcpipClient)

The TCP/IP client. You can use the `PeerName` property to check if the client is connected to a server.

Comments

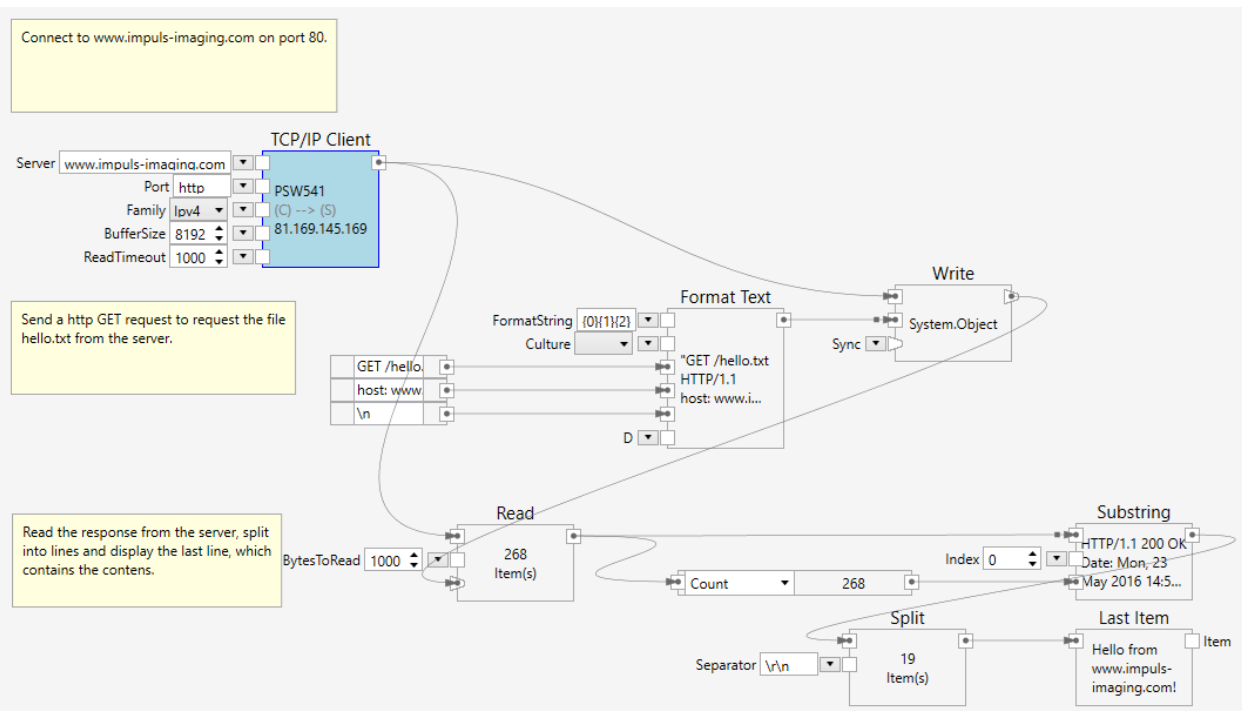
The **TCP/IP Client** node creates a client that communicates with a remote TCP/IP server over a network connection.

The remote server can be another PC, a PLC or another device such as a robot.

When a connection has been made, data can be sent or received through the connection. The **Write** nodes is used to send data, the **Read** node is used to receive data.

Example

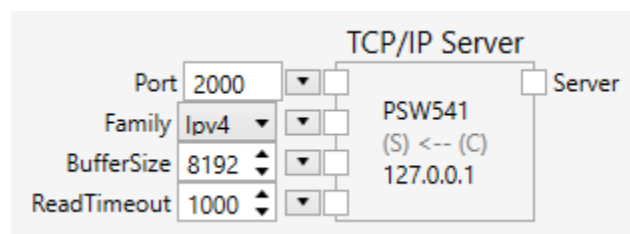
Here is an example that shows how to used the **TCP/IP Client** node (click on the image to load the example):



The example sends a GET command to `www.impuls-imaging.com` and retrieves the contents of the file `hello.txt`.

15.3.244 TCP/IP Server

Creates a TCP/IP Server.



Inputs

Port (Type: String)

The port or service name. Strings such as 21, http, etc. can be used. Possible values for the service name are listed in the file %WINDIR%\system32\drivers\etc\services.

Family (Type: String)

The address family for the connection. Ipv4 or Ipv6 are possible.

BufferSize (Type: String)

The size of the read and write buffers.

ReadTimeout (Type: String)

The timeout of a read operation.

Outputs

Server (Type: TcpipServer)

The TCP/IP server.

Comments

The **TCP/IP Server** node creates a server that communicates with a remote TCP/IP client over a network connection.

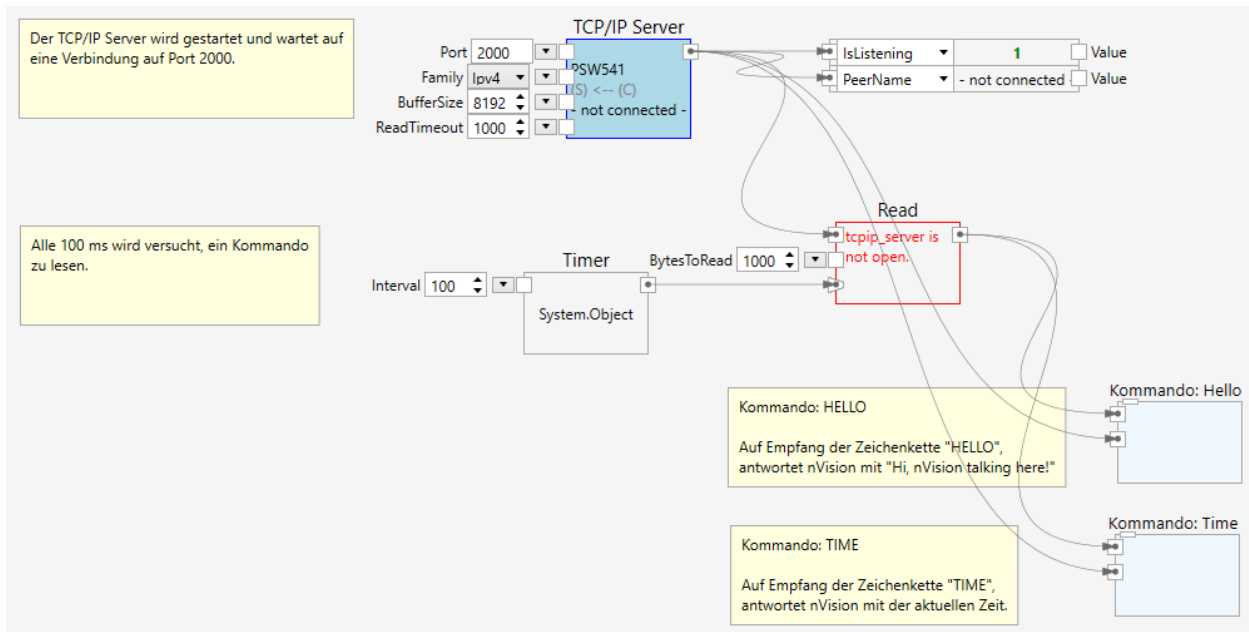
The remote client can be another PC, a PLC or another device such as a robot.

When a connection has been made, data can be sent or received through the connection. The **Write** nodes is used to send data, the **Read** node is used to receive data.

Establishing a connection is a two-step process, which executes asynchronously. First, the server is created and listens for connections. Once a client connects, the **TCP/IP Server** executes again and completes the connection and updates the connection.

Example

Here is an example that shows how to used the **TCP/IP Server** node (click on the image to load the example):

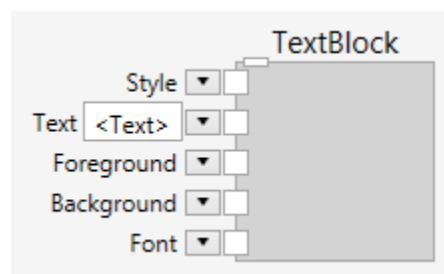


The example starts a TCP/IP server which listens for a client connection. The server implements two commands: HELLO and TIME. It polls for the command every 100 ms. If the server receives HELLO, it responds with *Hi, nVision talking here!*, and if the server receives TIME, it responds with the actual date and time. Anything else is ignored.

You can send the commands with any utility program that allows you to send text strings, such as `telnet`.

15.3.245 HMI TextBlock

A **TextBlock** can be used to display text.



Inputs

Style (Type: Style)

Optional styling of the **TextBlock**.

The **TextBlock** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Text (Type: string)

The text.

Foreground (Type: SolidColorBrush)

The foreground.

Background (Type: SolidColorBrush)

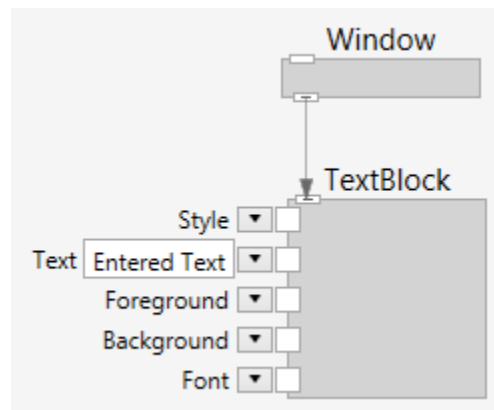
The background.

Font (Type: Font)

The font.

Example

Here is an example that shows a **TextBlock**. This definition:



creates the following user interface:

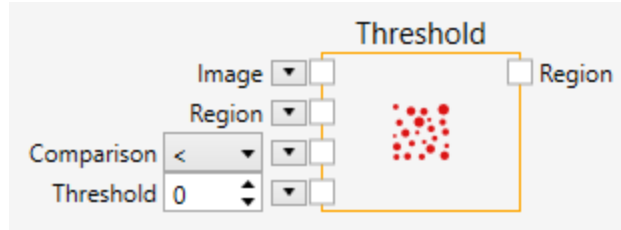
Entered Text

15.3.246 Threshold

Binary thresholds an image at a threshold value and creates a region as a result.

Inputs**Image (Type: Image)**

The input image.



Region (Type: Region)

Constrains the function to a region of interest.

Comparison (Type: String)

The thresholding operation: <, <=, >, >=, == or !=.

Threshold (Type: Double)

The threshold value.

Outputs

Region (Type: Region)

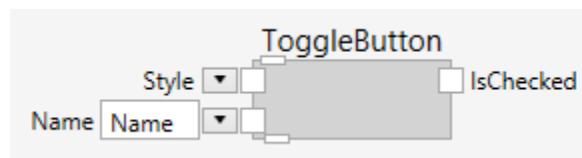
A region that contains all the pixels of the input image that satisfy the thresholding operation.

Comments

The region may or may not be spatially contiguous. If you want to separate the region into spatially distinct regions, use the **Connected Components** command.

15.3.247 HMI ToggleButton

Buttons are used to switch or visualize state, as well as to trigger actions. The **ToggleButton** has two states, up or down.



A **ToggleButton** toggles between the up and down states when it is clicked.

At the bottom of the **ToggleButton** node is a pin that allows it to connect one child.

Inputs

Style (Type: `style`)

Optional styling of the **ToggleButton**.

The **ToggleButton** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin*, *Padding*, *Foreground*, *Background*, *Font* and *Padding* styles.

Name (Type: `string`)

The text that is displayed within the button. This text is only displayed, when no child is connected. If a child is connected, the visual definition of the child is used.

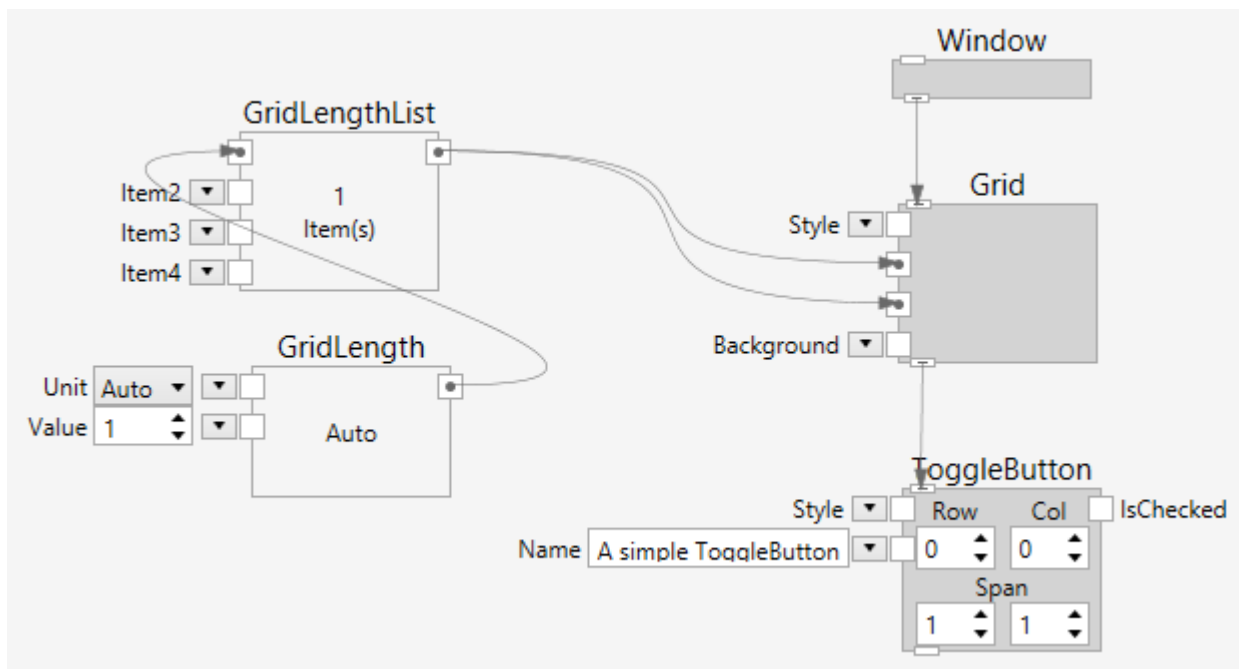
Outputs

IsChecked (Type: `boolean`)

True if the button is down, False otherwise.

Example

Here is an example that shows a simple toggle button. This definition:

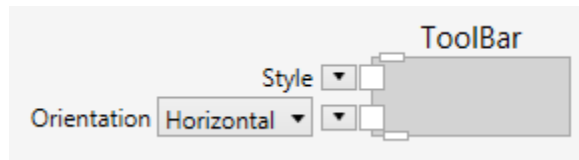


creates the following user interface:

A simple ToggleButton

15.3.248 HMI ToolBar

The **ToolBar** allows you to create a horizontal or vertical toolbar.



At the bottom of the **ToolBar** node is a pin that allows it to connect several children.

Inputs

Style (Type: `Style`)

Optional styling of the **ToolBar**.

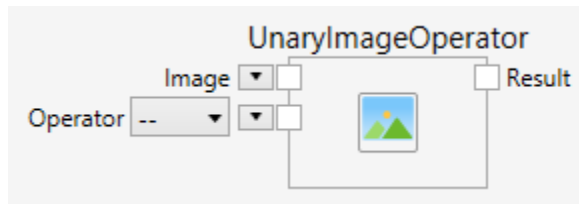
The **ToolBar** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment*, *Margin* and *Background* styles.

Orientation (Type: `string`)







Orientation of the **ToolBar**, `Horizontal` or `Vertical`.

15.3.249 Unary Image Operator

Applies a unary point operation to an image. Unary point operations are applied pixel by pixel, the same way for every pixel.

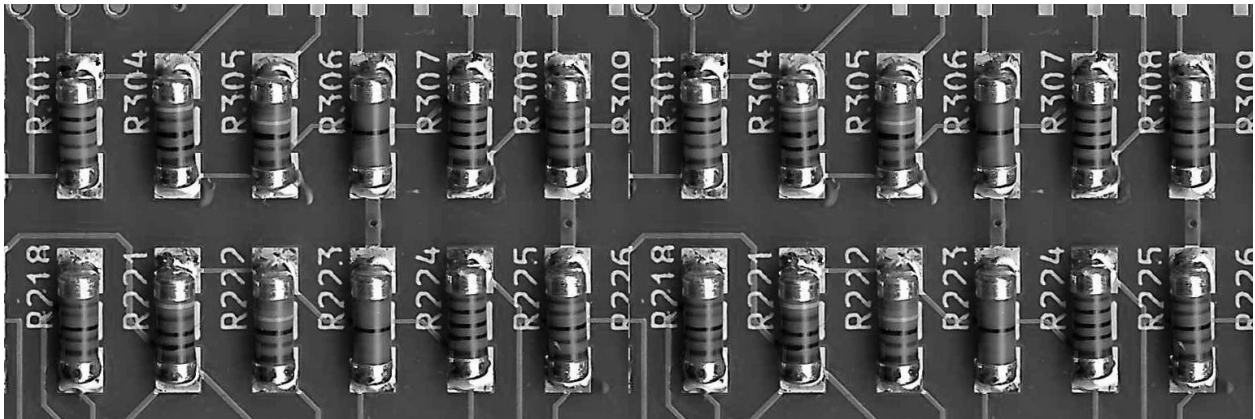


The following operators are available:

-  Absolute
-  Decrement
-  Increment
-  Negate
-  Not
-  Square Root

Absolute

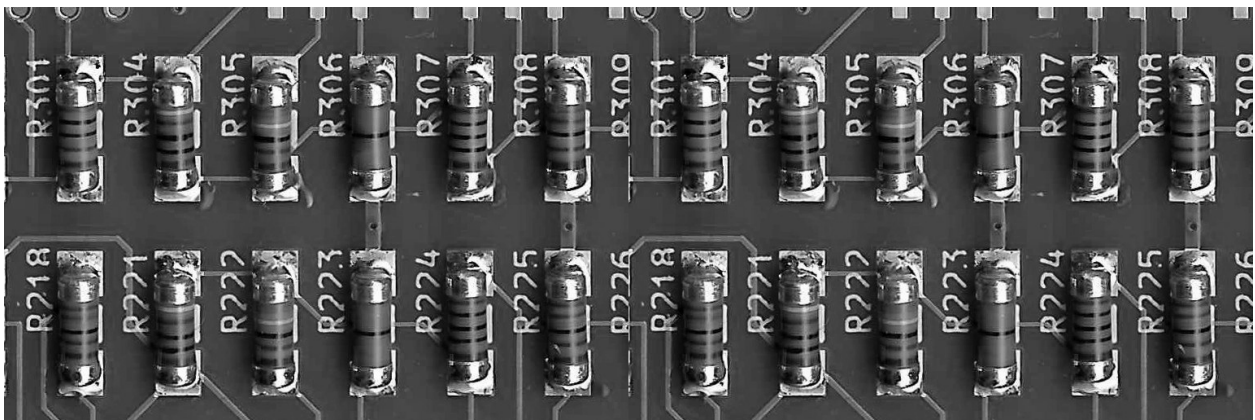
Absolute.



`abs (Image) -> Result`

Decrement

Decrement by 1.



`Image - 1 -> Result`

Increment

Increment by 1.

`Image + 1 -> Result`

Negate

Negate (2's complement).

- `Image -> Result`

Not

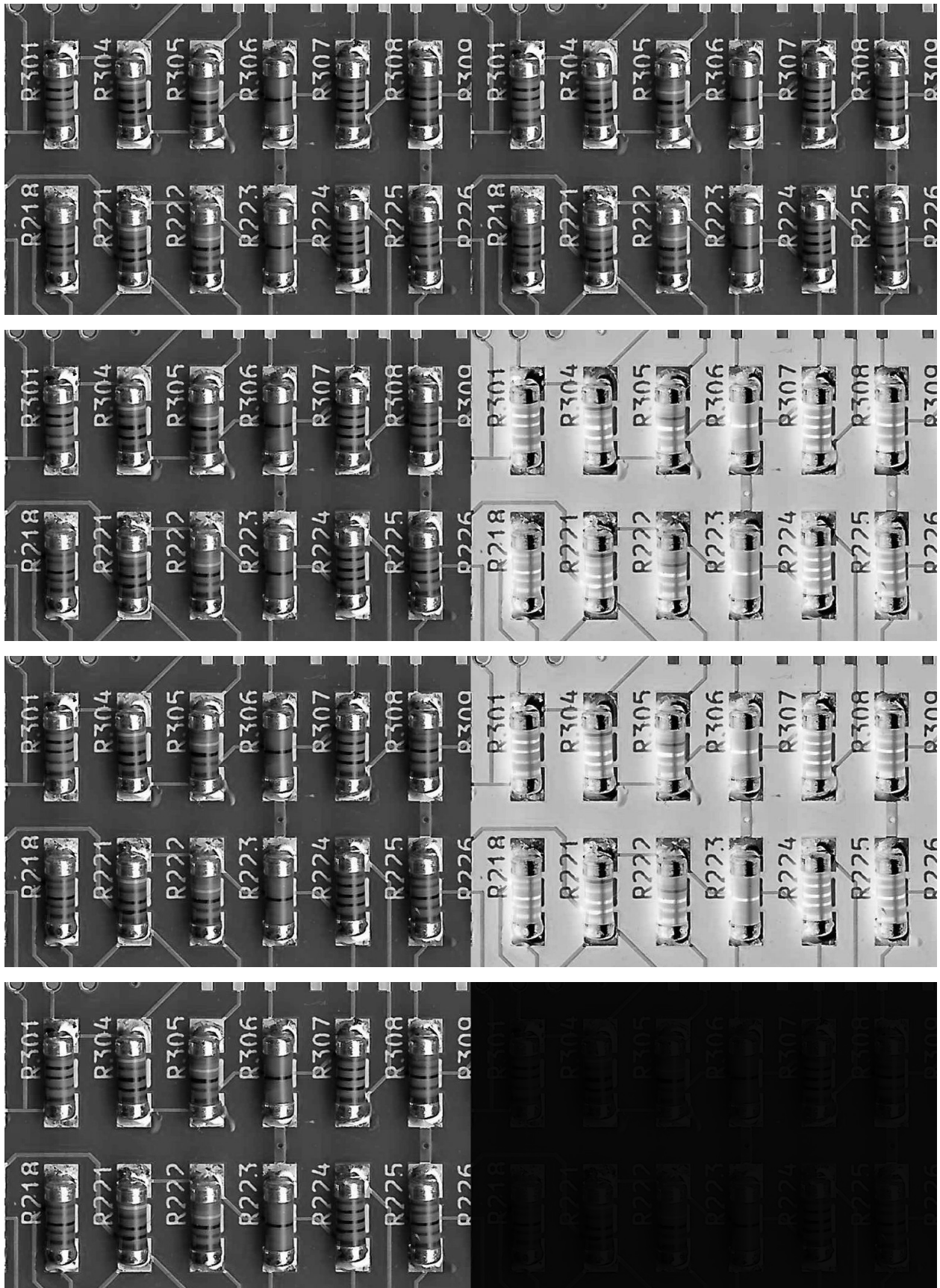
Not (1's complement).

`~ Image -> Result`

Square Root

Square root.

`sqrt(Image) -> Result`



Inputs

Image (Type: Image)

The input image.

Operator (Type: string)

Specifies the operator.

operator	operation
++	increment
--	decrement
-	negate (2's complement)
!	not (1's complement)
abs	absolute
sqrt	square root

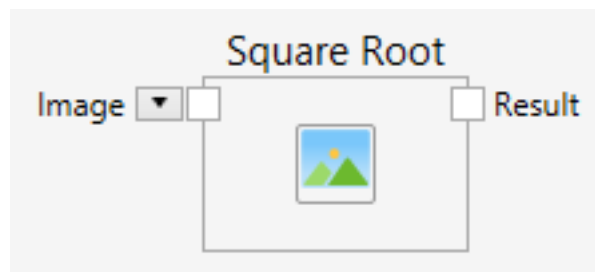
Outputs

Result (Type: Image)

The result image.

15.3.250 Square Root

Calculates the square root of every pixel.



Inputs

Image (Type: Image)

The input image.

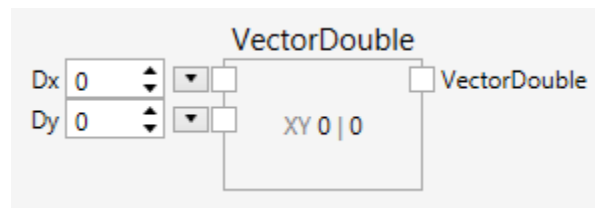
Outputs

Result (Type: Image)

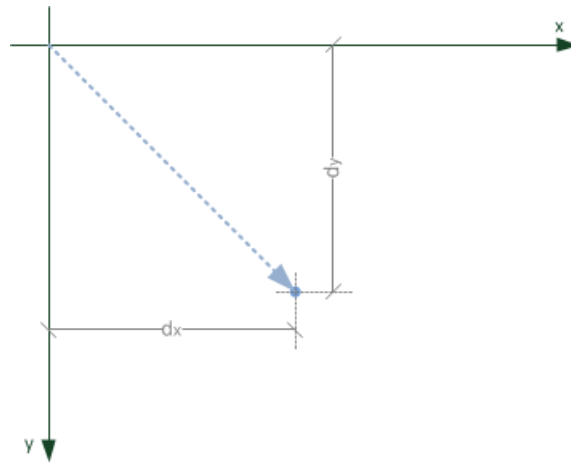
The result image.

15.3.251 Vector

Creates a vector.



A vector is a geometric entity, representing a directed length in the two-dimensional cartesian plane.



Inputs

Dx (Type: Double)

The horizontal coordinate.

Dy (Type: Double)

The vertical coordinate.

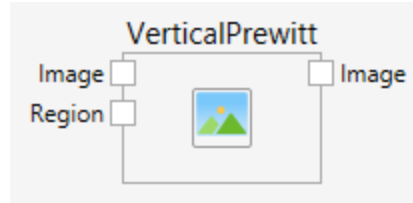
Outputs

VectorDouble (Type: VectorDouble)

The vector.

15.3.252 Vertical Prewitt

Filters an image using a vertical prewitt kernel.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a vertical prewitt filter. The corresponding kernel is a 3x3 matrix with the following values:

```
1 0 -1
1 0 -1
1 0 -1
```

The filter attenuates vertical edges while at the same time smoothing in the vertical direction.

Here are a few results of the vertical prewitt filter with increasing kernel sizes:

Original:

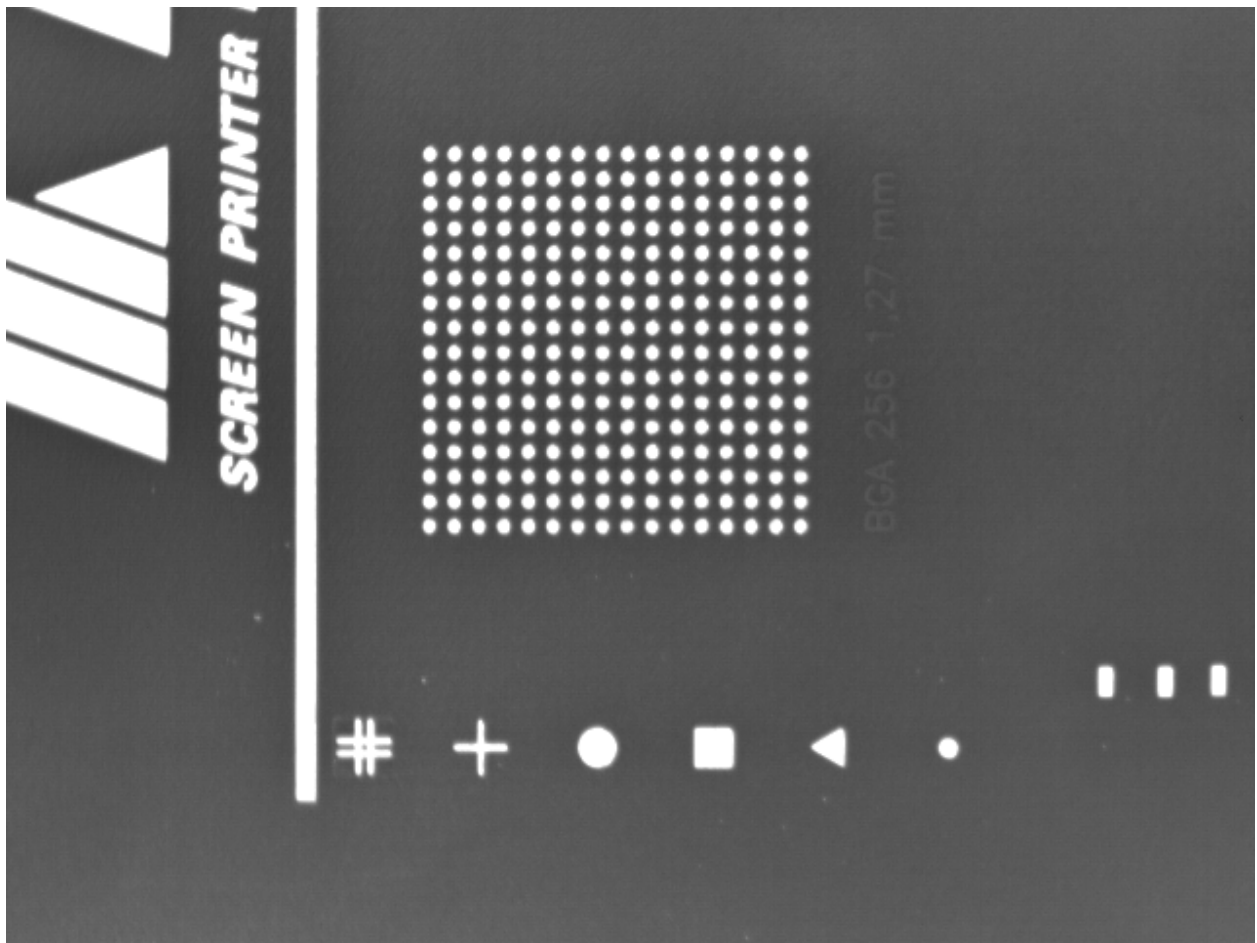
Result:

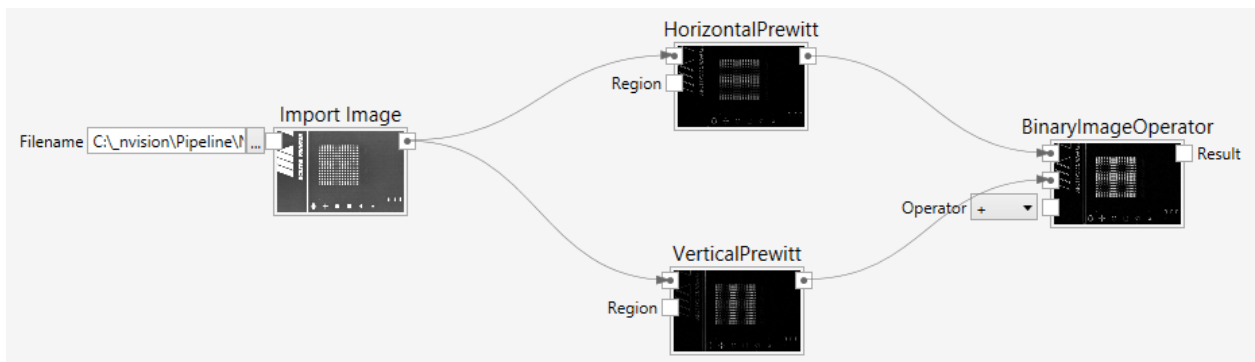
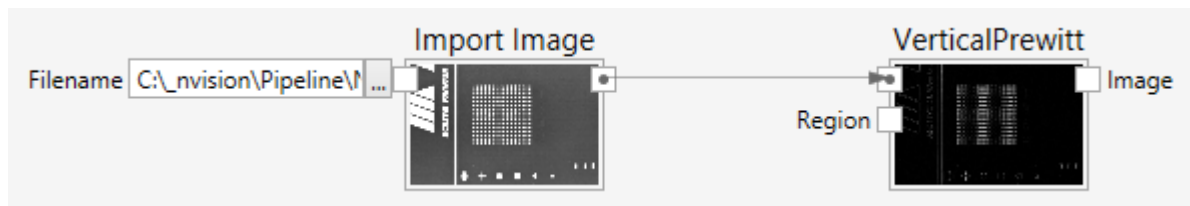
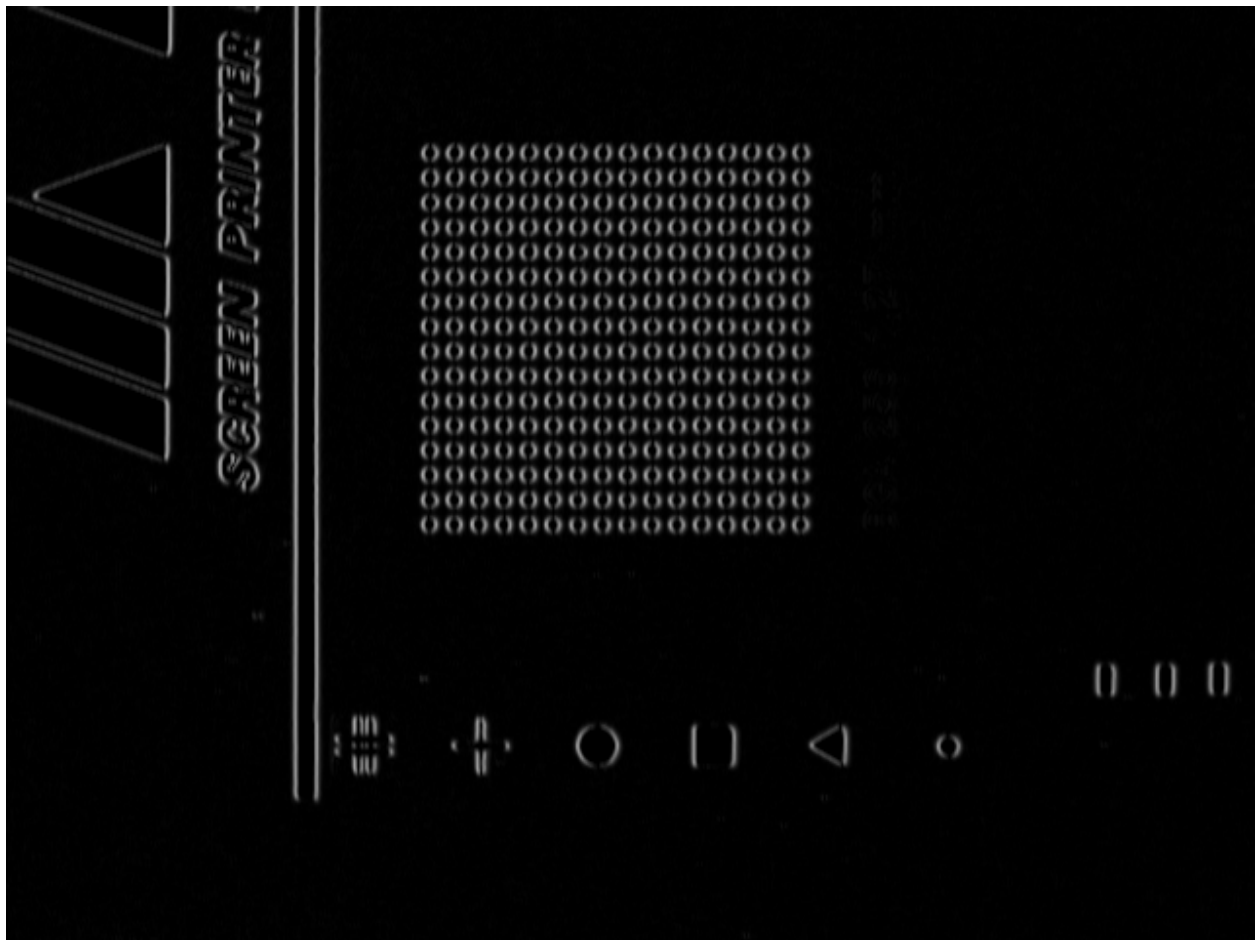
Sample

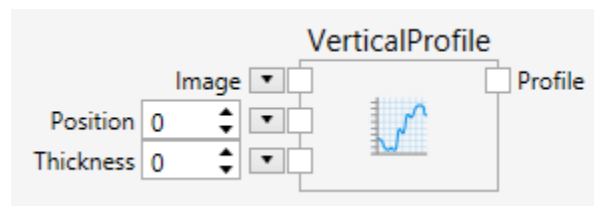
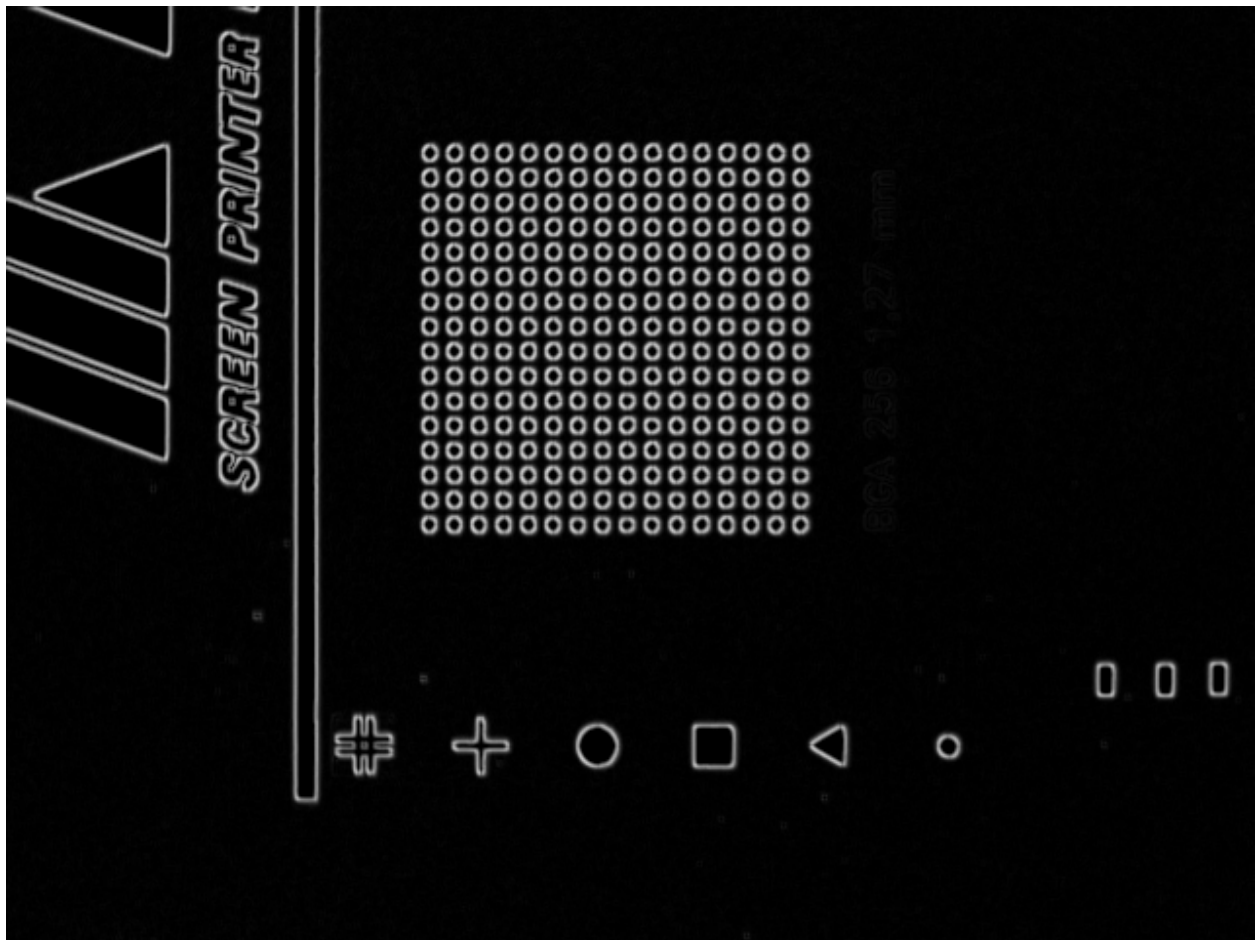
Here is an example that shows how to use the vertical prewitt filter.

Often, you want to combine the horizontal and the vertical prewitt filter to create an edge map. A simple way to do this is shown in the following sample:

Here is an example image of such an edge map:







15.3.253 Vertical Profile

Calculates a vertical profile of an image to show the greyscale/color distribution.

The Vertical Profile node gives you information about the greyscale/color distribution along a vertical line. The line can have a certain thickness.

Inputs

Image (Type: Image)

The input image.

Position (Type: Int32)

The horizontal coordinate of the vertical line.

Thickness (Type: Int32)

The thickness in pixel of the vertical line.

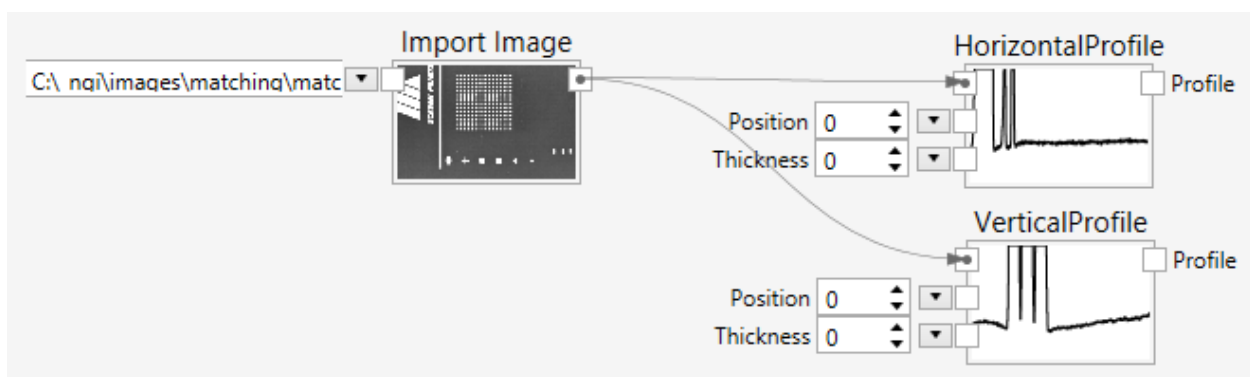
Outputs

Profile (Type: Profile)

The profile.

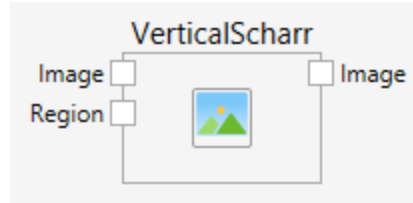
Sample

Here is an example:



15.3.254 Vertical Scharr

Filters an image using a vertical scharr kernel.



Inputs

Image (Type: Image)

The input image.

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a vertical scharr filter. The corresponding kernel is a 3x3 matrix with the following values:

```

3  0  -3
10 0 -10
3  0  -3

```

The filter attenuates vertical edges while at the same time smoothing in the vertical direction.

Here are a few results of the horizontal sobel filter with increasing kernel sizes:

Original:

Result:

Sample

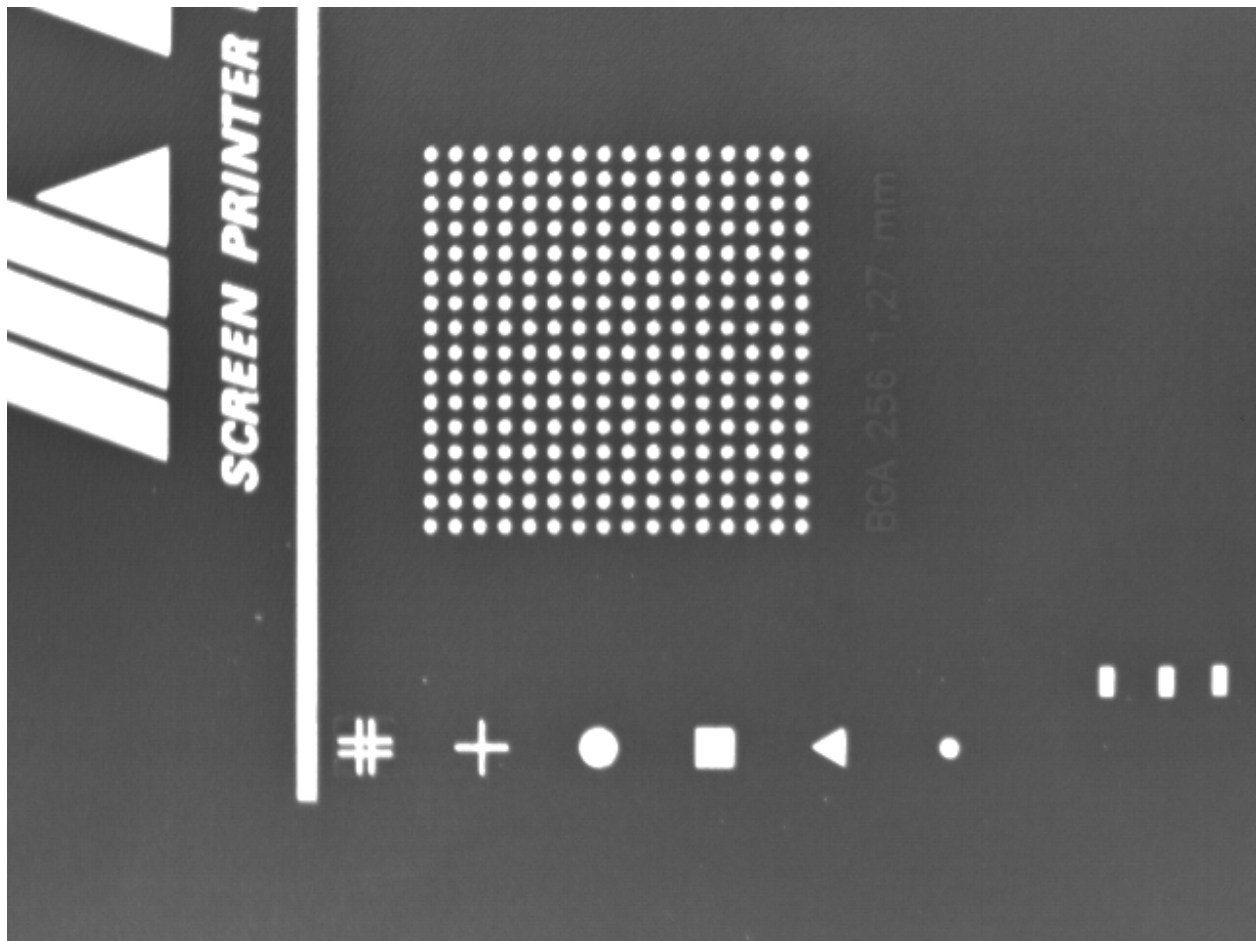
Here is an example that shows how to use the vertical scharr filter.

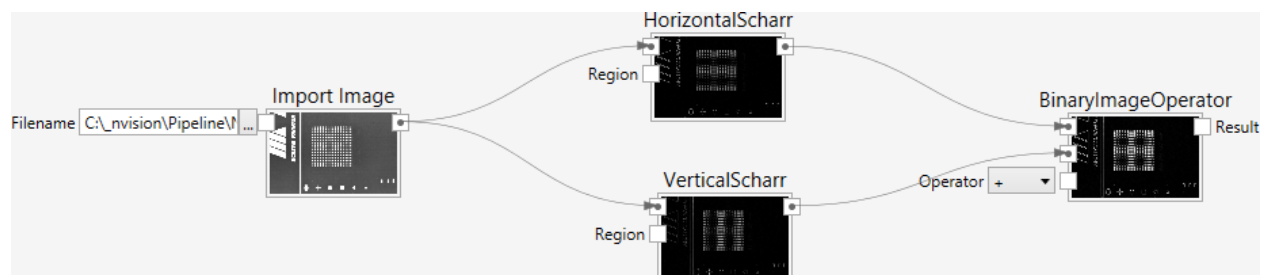
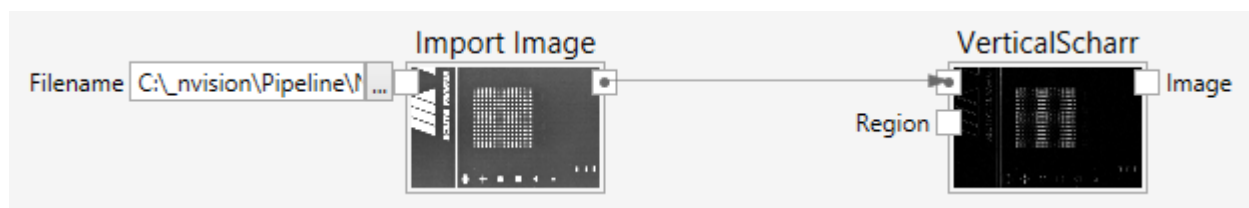
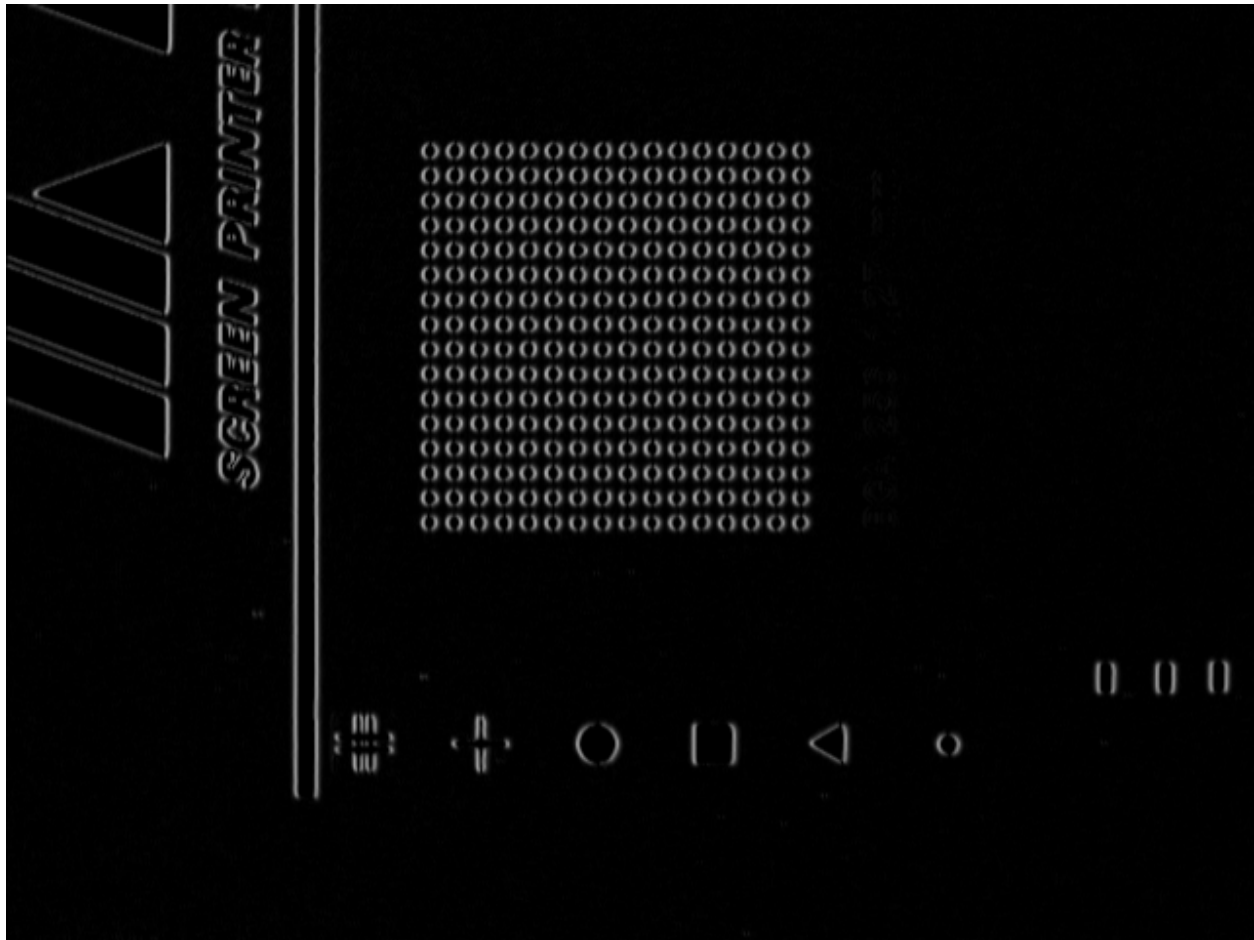
Often, you want to combine the horizontal and the vertical scharr filter to create an edge map. A simple way to do this is shown in the following sample:

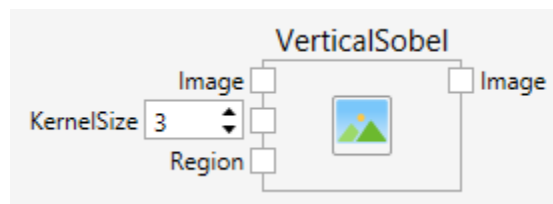
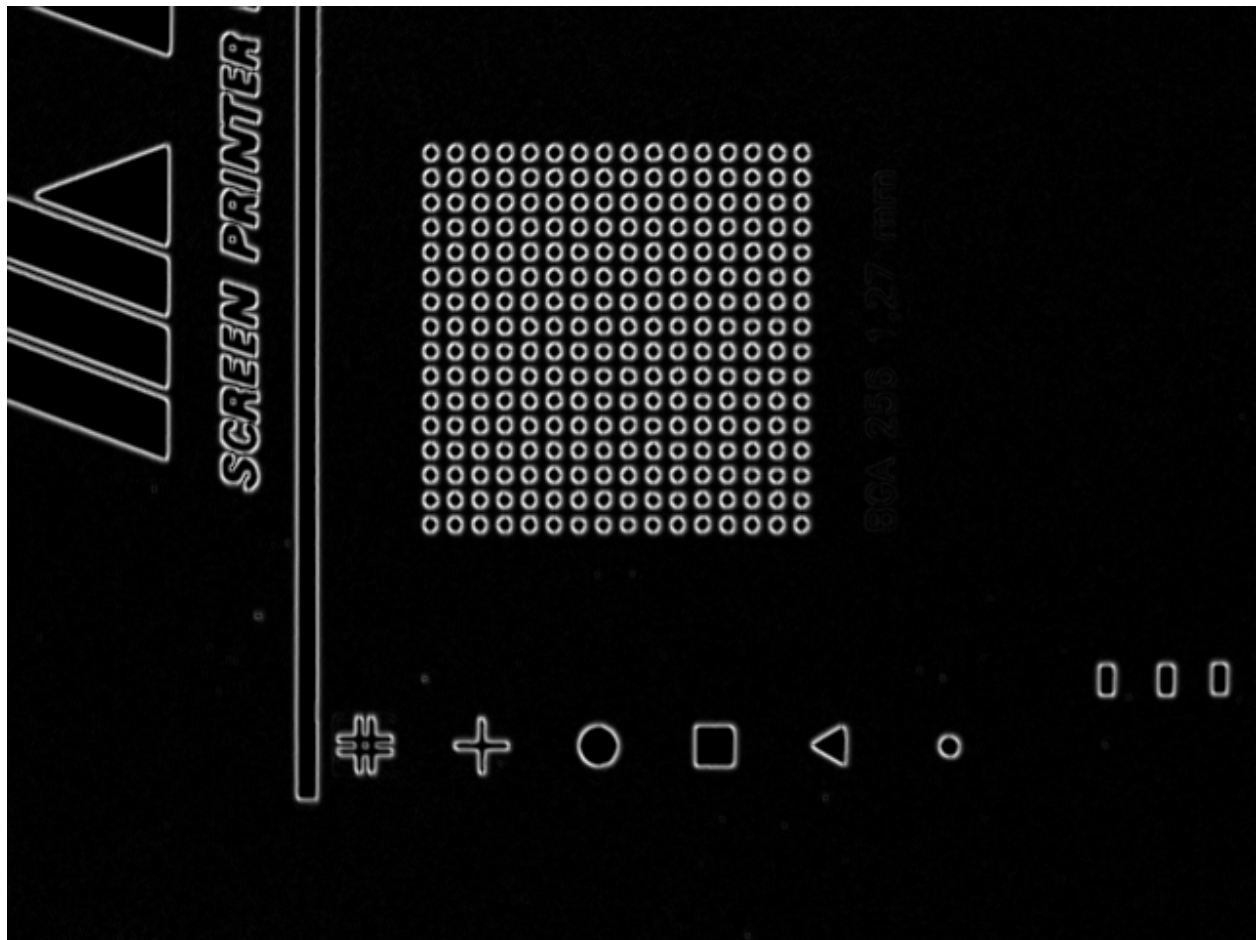
Here is an example image of such an edge map:

15.3.255 Vertical Sobel

Filters an image using a vertical sobel kernel of size 3x3 or 5x5.







Inputs

Image (Type: Image)

The input image.

KernelSize (Type: Int32)

The kernel size (3 -> 3x3, 5 -> 5x5).

Region (Type: Region)

Specifies an optional area of interest.

Outputs

Image (Type: Image)

The output image.

Comments

The function applies a vertical_sobel filter. The corresponding kernel is either a 3x3 matrix with the following values:

```

-1  0  1
-2  0  2
-1  0  1

```

or a 5x5 kernel with the following values:

```

-1  - 2  0  2  1
-4  -8  0  8  4
-6 -12  0 12  6
-4  -8  0  8  4
-1  - 2  0  2  1

```

The effect of a vertical sobel filter is that it amplifies vertical edges and attenuates everything else. The strength of the sobel filter depends on the size of the kernel.

Here are a few results of the vertical sobel filter with increasing kernel sizes:

Original:

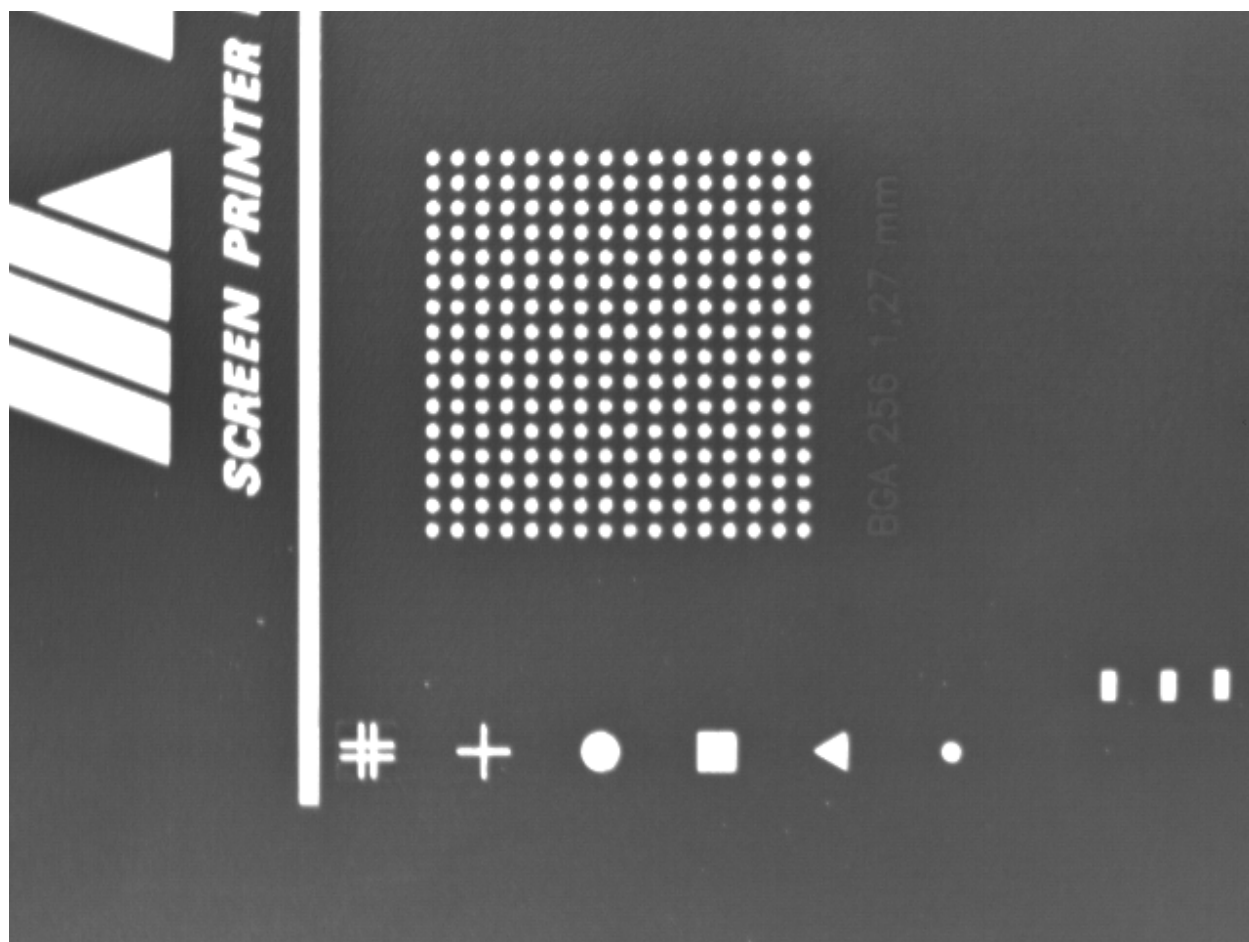
KernelSize = 3:

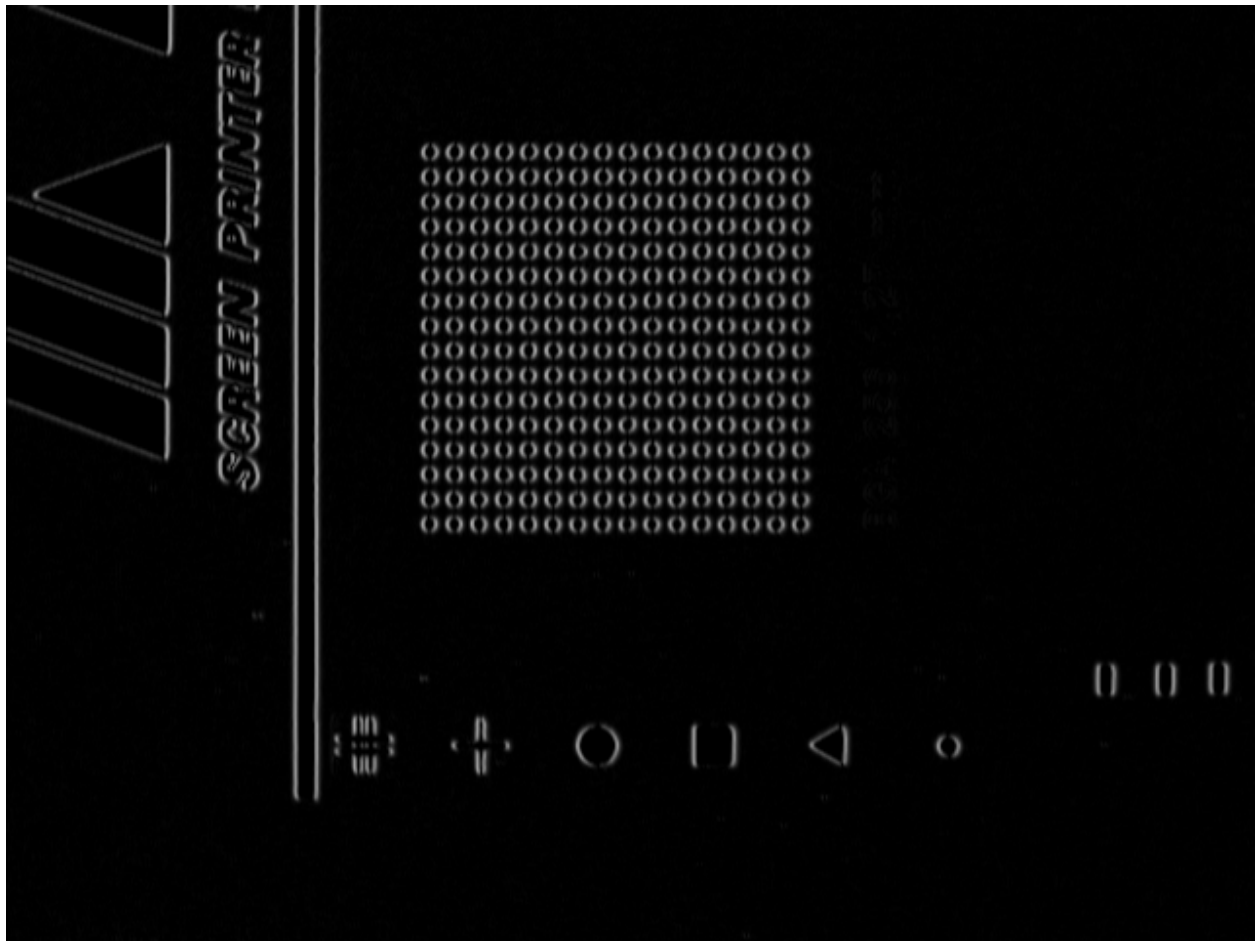
KernelSize = 5:

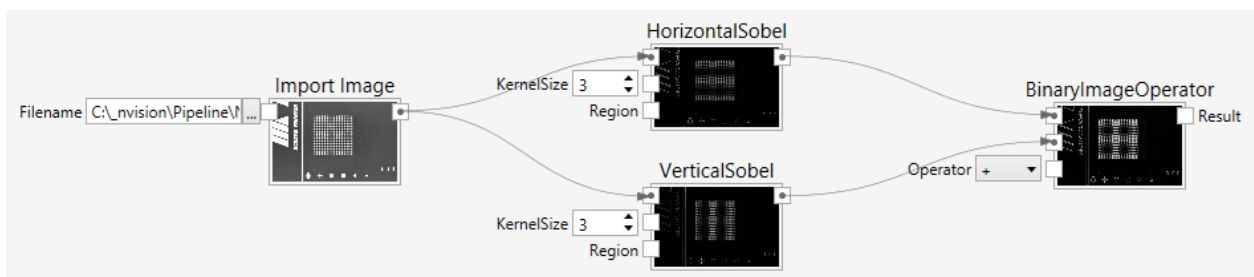
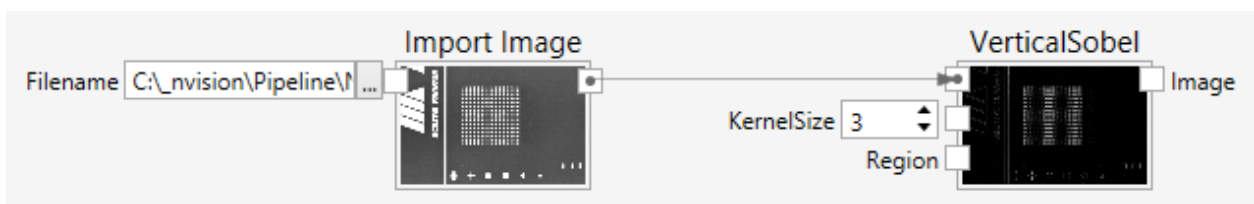
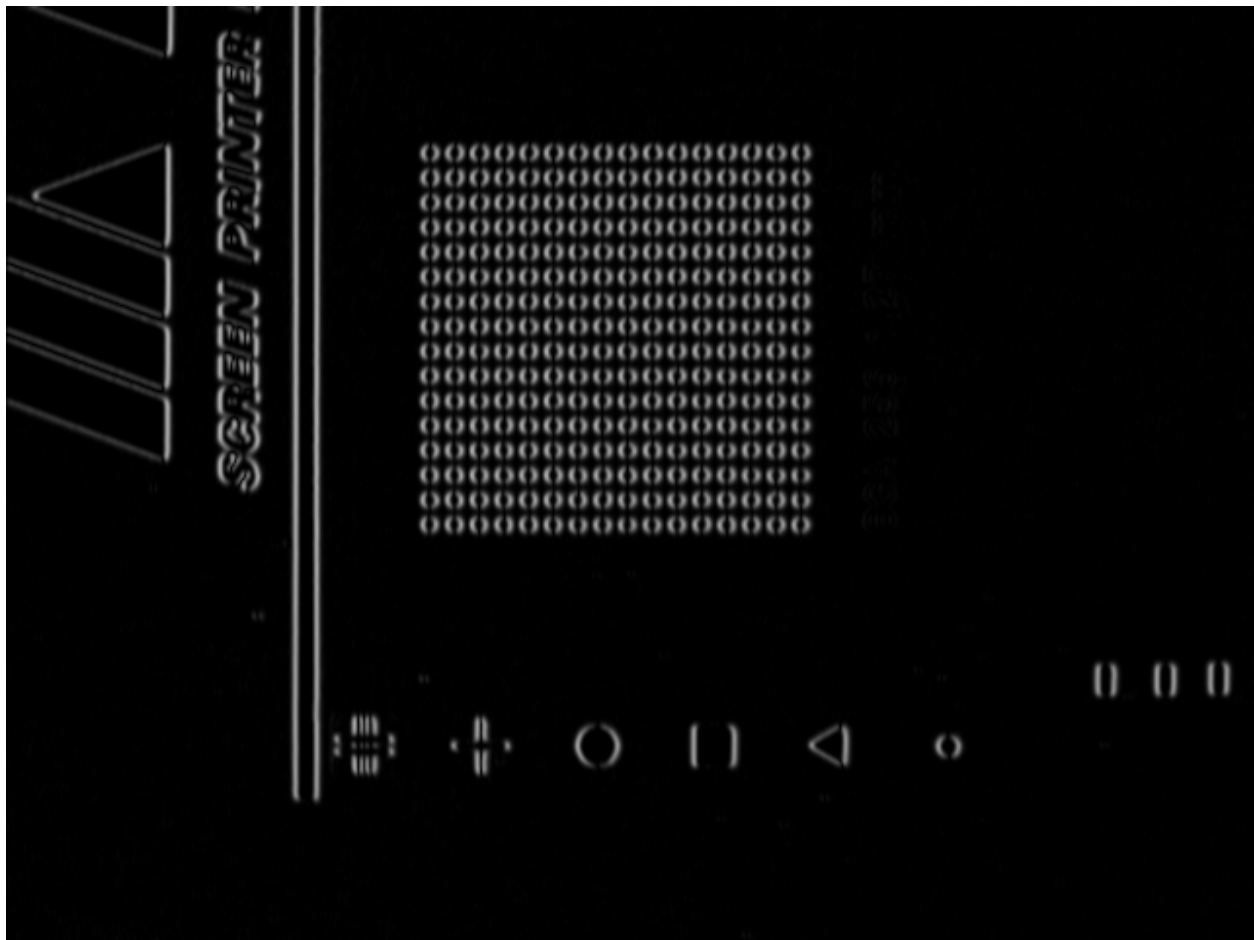
Sample

Here is an example that shows how to use the vertical sobel filter.

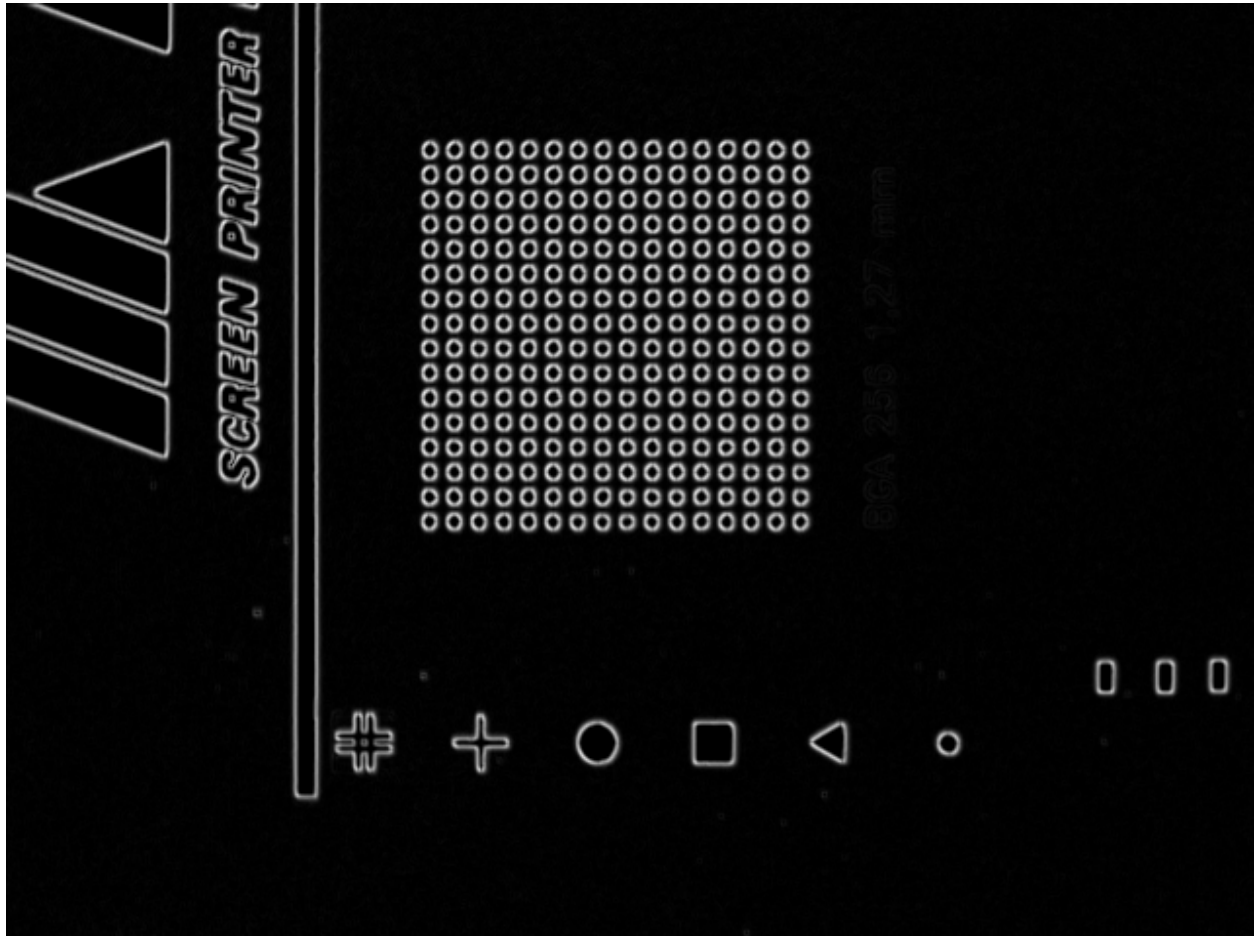
Often, you want to combine the horizontal and the vertical sobel filter to create an edge map. A simple way to do this is shown in the following sample:





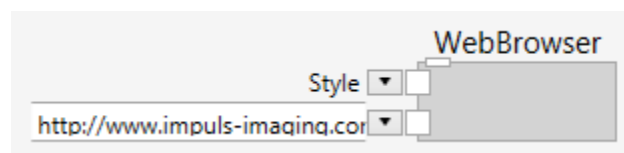


Here is an example image of such an edge map:



15.3.256 HMI WebBrowser

A **WebBrowser** can be used to display a webpage.



Inputs

Style (Type: `style`)

Optional styling of the **WebBrowser**.

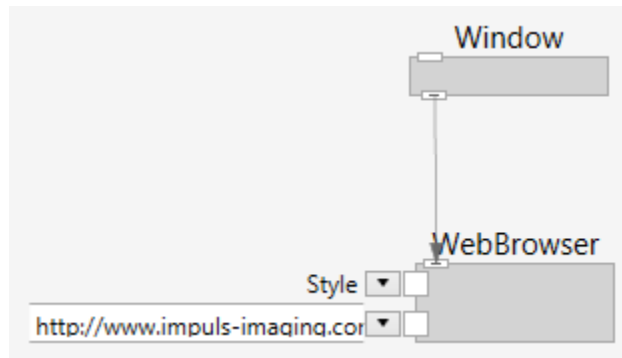
The **WebBrowser** respects the *Width*, *Height*, *HorizontalAlignment*, *VerticalAlignment* and *Margin* styles.

Source (Type: string)

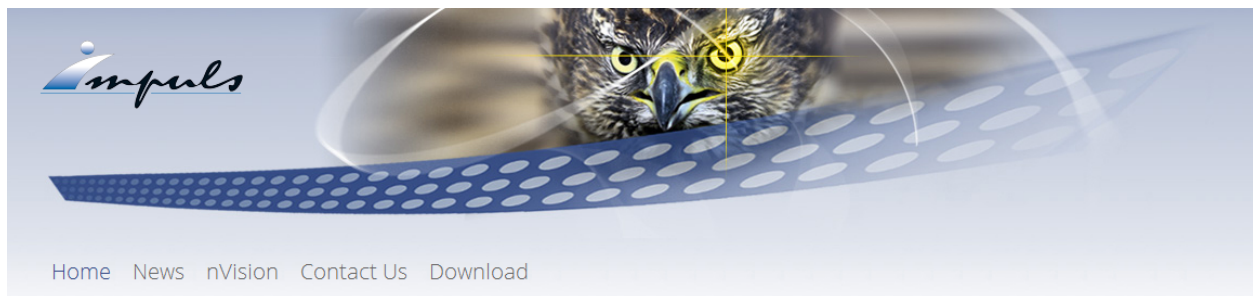
The URL of the webpage.

Example

Here is an example that shows a **WebBrowser**. This definition:

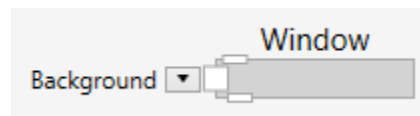


creates the following user interface:



15.3.257 HMI Window

The **Window** node is the root of the HMI (Human Machine Interface) system.



The **Window** node creates the connection to the **nVision** workspace area.

If a **Window** node is available in the pipeline, it takes over the display. This means that the default of displaying the first pipeline output is overridden, and the window along with its children is displayed instead.

At the bottom of the **Window** node is a pin that allows it to connect one child to the window. This child can take all the space in the window. Usually, this will be a control from the layout group of controls.

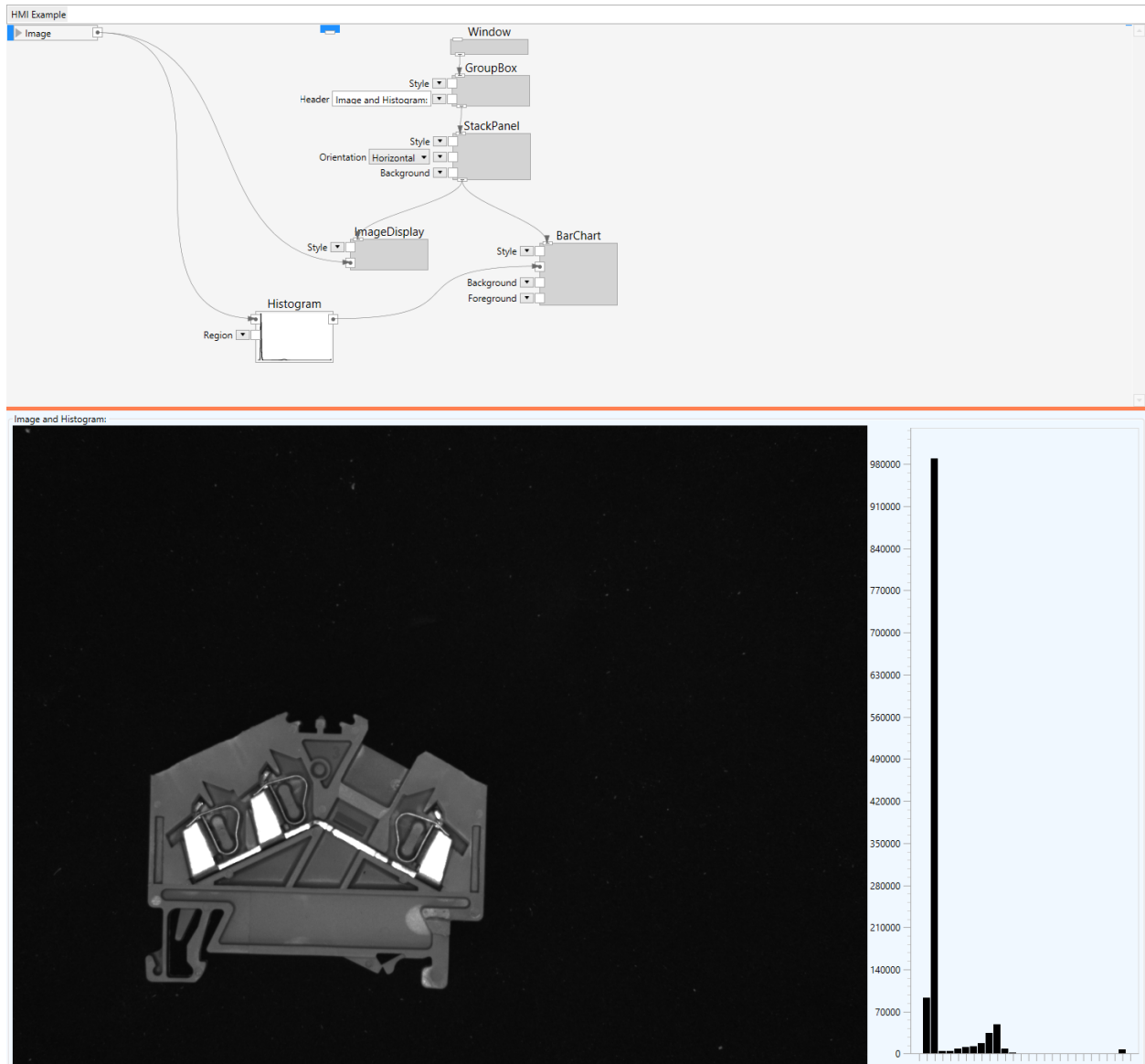
Inputs

Background (Type: SolidColorBrushByte)

The background of the **Window**.

Example

Here is an example:



Below the **Window** node is a **GroupBox**. Inside the **GroupBox** is a **StackPanel** and inside the **StackPanel** is an **ImageDisplay** which displays an image and a **BarChart** which displays the image histogram, stacked in horizontal direction.

15.3.258 Write

Write data to an interface.



This node writes data to an interface (such as a serial port).

Inputs

Io (Type: IoResource)

The IO interface.

Data (Type: DataList)

The data to write in the form of a DataList (a list of bytes).

Sync (Type: object)

This input can be connected to any other object. It is used to establish an order of execution.

Outputs

Sync (Type: object)

Synchronizing object. It is used to establish an order of execution.

Comments

The **Sync** input and output can be used to establish an order of execution.

Sometimes, if this is necessary for technical reasons, you can add a **Delay** node in order to introduce additional delay between synchronized nodes.

nVision uses some third party software components that come with various licenses. This section lists the software packages and outlines their licenses.

16.1 Boost

Boost provides free peer-reviewed portable C++ source libraries. It comes with the following license:

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the “Software”) to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.2 zlib

zlib is a compression/decompression library. It comes with the following license:

zlib.h – interface of the ‘zlib’ general purpose compression library version 1.2.3, July 18th, 2005

Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler

This software is provided ‘as-is’, without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org Mark Adler madler@alumni.caltech.edu

16.3 bzip

bzip is a compression/decompression library. It comes with the following license:

This program, “bzip2” and associated library “libbzip2”, are copyright (C) 1996-2000 Julian R Seward. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, Cambridge, UK.

16.4 FreeImage

FreeImage is a library which supports reading and writing from popular graphics images formats. nVision uses it under the terms of the FreeImage Public License (FIPL):

1. Definitions.
 2. (a) “Contributor” means each entity that creates or contributes to the creation of Modifications.
 3. (a) “Contributor Version” means the combination of the Original Code, prior Modifications used by a Contributor, and the Modifications made by that particular Contributor.
 2. (a) “Covered Code” means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.
 3. (a) “Electronic Distribution Mechanism” means a mechanism generally accepted in the software development community for the electronic transfer of data.
 4. (a) “Executable” means Covered Code in any form other than Source Code.
 5. (a) “Initial Developer” means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.
 6. (a) “Larger Work” means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.
 7. (a) “License” means this document.
 8. (a) “Modifications” means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:
 1. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.
 2. Any new file that contains any part of the Original Code or previous Modifications.
- 1.10. “Original Code” means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.
- 1.11. “Source Code” means the preferred form of the Covered Code for making modifications to it, including all modules it contains, plus any associated interface definition files, scripts used to control compilation and installation of an Executable, or a list of source code differential comparisons against either the Original Code or another well known, available Covered Code of the Contributor’s choice. The Source Code can be in a compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.
- 1.12. “You” means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 6.1. For legal entities, “You” includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of fifty percent (50%) or more of the outstanding shares or beneficial ownership of such entity.
2. Source Code License.
 3. (a) The Initial Developer Grant. The Initial Developer hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:
 1. to use, reproduce, modify, display, perform, sublicense and distribute the Original Code (or portions thereof) with or without Modifications, or as part of a Larger Work; and

2. under patents now or hereafter owned or controlled by Initial Developer, to make, have made, use and sell (“Utilize”) the Original Code (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Original Code (or portions thereof) and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

2.2. Contributor Grant. Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license, subject to third party intellectual property claims:

1. to use, reproduce, modify, display, perform, sublicense and distribute the Modifications created by such Contributor (or portions thereof) either on an unmodified basis, with other Modifications, as Covered Code or as part of a Larger Work; and
2. under patents now or hereafter owned or controlled by Contributor, to Utilize the Contributor Version (or portions thereof), but solely to the extent that any such patent is reasonably necessary to enable You to Utilize the Contributor Version (or portions thereof), and not to any greater extent that may be necessary to Utilize further Modifications or combinations.

3. Distribution Obligations.

4. (a) Application of License. The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2. The Source Code version of Covered Code may be distributed only under the terms of this License or a future version of this License released under Section 6.1, and You must include a copy of this License with every copy of the Source Code You distribute. You may not offer or impose any terms on any Source Code version that alters or restricts the applicable version of this License or the recipients’ rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5.

5. (a) Availability of Source Code. Any Modification which You create or to which You contribute must be made available in Source Code form under the terms of this License either on the same media as an Executable version or via an accepted Electronic Distribution Mechanism to anyone to whom you made an Executable version available; and if made available via Electronic Distribution Mechanism, must remain available for at least twelve (12) months after the date it initially became available, or at least six (6) months after a subsequent version of that particular Modification has been made available to such recipients. You are responsible for ensuring that the Source Code version remains available even if the Electronic Distribution Mechanism is maintained by a third party.

2. (a) Description of Modifications. You must cause all Covered Code to which you contribute to contain a file documenting the changes You made to create that Covered Code and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Code provided by the Initial Developer and including the name of the Initial Developer in (a) the Source Code, and (b) in any notice in an Executable version or related documentation in which You describe the origin or ownership of the Covered Code.

3. (a) Intellectual Property Matters

1. Third Party Claims. If You have knowledge that a party claims an intellectual property right in particular functionality or code (or its utilization under this License), you must include a text file with the source code distribution titled “LEGAL” which describes the claim and the party making the claim in sufficient detail that a recipient will know whom to contact. If you obtain such knowledge after You make Your Modification available as described in Section 3.2, You shall promptly modify the LEGAL file in all copies You make available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Covered Code that new knowledge has been obtained.

2. Contributor APIs. If Your Modification is an application programming interface and You own or control patents which are reasonably necessary to implement that API, you must also include this information in the LEGAL file.

3.5. **Required Notices.** You must duplicate the notice in Exhibit A in each file of the Source Code, and this License in any documentation for the Source Code, where You describe recipients' rights relating to Covered Code. If You created one or more Modification(s), You may add your name as a Contributor to the notice described in Exhibit A. If it is not possible to put such notice in a particular Source Code file due to its structure, then you must include such notice in a location (such as a relevant directory file) where a user would be likely to look for such a notice. You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Code. However, You may do so only on Your own behalf, and not on behalf of the Initial Developer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. **Distribution of Executable Versions.** You may distribute Covered Code in Executable form only if the requirements of Section 3.1-3.5 have been met for that Covered Code, and if You include a notice stating that the Source Code version of the Covered Code is available under the terms of this License, including a description of how and where You have fulfilled the obligations of Section 3.2. The notice must be conspicuously included in any notice in an Executable version, related documentation or collateral in which You describe recipients' rights relating to the Covered Code. You may distribute the Executable version of Covered Code under a license of Your choice, which may contain terms different from this License, provided that You are in compliance with the terms of this License and that the license for the Executable version does not attempt to limit or alter the recipient's rights in the Source Code version from the rights set forth in this License. If You distribute the Executable version under a different license You must make it absolutely clear that any terms which differ from this License are offered by You alone, not by the Initial Developer or any Contributor. You hereby agree to indemnify the Initial Developer and every Contributor for any liability incurred by the Initial Developer or such Contributor as a result of any such terms You offer.

3.7. **Larger Works.** You may create a Larger Work by combining Covered Code with other code not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Code due to statute or regulation then You must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the LEGAL file described in Section 3.4 and must be included with all distributions of the Source Code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the notice in Exhibit A, and to related Covered Code.

6. Versions of the License.

7. (a) **New Versions.** Floris van den Berg may publish revised and/or new versions of the License from time to time. Each version will be given a distinguishing version number.

8. (a) **Effect of New Versions.** Once Covered Code has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Covered Code under the terms of any subsequent version of the License published by Floris van den Berg No one other than Floris van den Berg has the right to modify the terms applicable to Covered Code created under this License.

2. (a) **Derivative Works.** If you create or use a modified version of this License (which you may only do in order to apply it to code which is not already Covered Code governed by this Li-

cense), you must (a) rename Your license so that the phrases “FreeImage”, “FreeImage Public License”, “FIPL”, or any confusingly similar phrase do not appear anywhere in your license and (b) otherwise make it clear that your version of the license contains terms which differ from the FreeImage Public License. (Filling in the name of the Initial Developer, Original Code or Contributor in the notice described in Exhibit A shall not of themselves be deemed to be modifications of this License.)

3. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN “AS IS” BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Covered Code which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL DEVELOPER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY RESULTING FROM SUCH PARTY’S NEGLIGENCE TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT EXCLUSION AND LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a “commercial item,” as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of “commercial computer software” and “commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by Dutch law provisions (except to the extent applicable law, if any, provides otherwise), excluding its conflict-of-law provisions. With respect to disputes in which at least one party is a citizen of, or an entity chartered or registered to do business in, the The Netherlands: (a) unless otherwise agreed in writing, all disputes relating to this License (excepting any

dispute relating to intellectual property rights) shall be subject to final and binding arbitration, with the losing party paying all costs of arbitration; (b) any arbitration relating to this Agreement shall be held in Almelo, The Netherlands; and (c) any litigation relating to this Agreement shall be subject to the jurisdiction of the court of Almelo, The Netherlands with the losing party responsible for costs, including without limitation, court costs and reasonable attorneys fees and expenses. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

Except in cases where another Contributor has failed to comply with Section 3.4, You are responsible for damages arising, directly or indirectly, out of Your utilization of rights under this License, based on the number of copies of Covered Code you made available, the revenues you received from utilizing such rights, and other relevant factors. You agree to work with affected parties to distribute responsibility on an equitable basis.

EXHIBIT A.

“The contents of this file are subject to the FreeImage Public License Version 1.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://home.wxs.nl/~flvdberg/freeimage-license.txt>

Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

16.5 DynamicExpresso

Dynamic Expresso is an interpreter for simple C# statements. It is licensed under the MIT license:

MIT License

Copyright (c) 2015 Davide Icardi

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.6 KBCsv

KBCsv is an efficient, easy to use .NET parsing and writing library for the CSV (comma-separated values) format, written by Kent Boogaart. It is licensed under the MIT license:

The MIT License (MIT)

Copyright (c) 2014 Kent Boogaart

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.7 NetworkView

NetworkView is a WPF custom control that is used to display and edit networks, graphs and flow-charts. It is licensed under the Code Project Open License (CPOL):

This License governs Your use of the Work. This License is intended to allow developers to use the Source Code and Executable Files provided as part of the Work in any application in any form.

The main points subject to the terms of the License are:

- Source Code and Executable Files can be used in commercial applications;
- Source Code and Executable Files can be redistributed; and
- Source Code can be modified to create derivative works.
- No claim of suitability, guarantee, or any warranty whatsoever is provided. The software is provided “as-is”.
- The Article(s) accompanying the Work may not be distributed or republished without the Author’s consent
- This License is entered between You, the individual or other entity reading or otherwise making use of the Work licensed pursuant to this License and the individual or other entity which offers the Work under the terms of this License (“Author”).

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CODE PROJECT OPEN LICENSE (“LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HEREIN, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE AUTHOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO ACCEPT AND BE BOUND BY THE TERMS OF THIS LICENSE, YOU CANNOT MAKE ANY USE OF THE WORK.

1. Definitions.

1. “**Articles**” means, collectively, all articles written by Author which describes how the Source Code and Executable Files for the Work may be used by a user.

2. **“Author”** means the individual or entity that offers the Work under the terms of this License.
3. **“Derivative Work”** means a work based upon the Work or upon the Work and other pre-existing works.
4. **“Executable Files”** refer to the executables, binary files, configuration and any required data files included in the Work.
5. **“Publisher”** means the provider of the website, magazine, CD-ROM, DVD or other medium from or by which the Work is obtained by You.
6. **“Source Code”** refers to the collection of source code and configuration files used to create the Executable Files.
7. **“Standard Version”** refers to such a Work if it has not been modified, or has been modified in accordance with the consent of the Author, such consent being in the full discretion of the Author.
8. **“Work”** refers to the collection of files distributed by the Publisher, including the Source Code, Executable Files, binaries, data files, documentation, whitepapers and the Articles.
9. **“You”** is you, an individual or entity wishing to use the Work and exercise your rights under this License.
2. **Fair Use/Fair Use Rights.** Nothing in this License is intended to reduce, limit, or restrict any rights arising from fair use, fair dealing, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
3. **License Grant.** Subject to the terms and conditions of this License, the Author hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
 1. You may use the standard version of the Source Code or Executable Files in Your own applications.
 2. You may apply bug fixes, portability fixes and other modifications obtained from the Public Domain or from the Author. A Work modified in such a way shall still be considered the standard version and will be subject to this License.
 3. You may otherwise modify Your copy of this Work (excluding the Articles) in any way to create a Derivative Work, provided that You insert a prominent notice in each changed file stating how, when and where You changed that file.
 4. You may distribute the standard version of the Executable Files and Source Code or Derivative Work in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution.
 5. The Articles discussing the Work published in any form by the author may not be distributed or republished without the Author’s consent. The author retains copyright to any such Articles. You may use the Executable Files and Source Code pursuant to this License but you may not repost or republish or otherwise distribute or make available the Articles, without the prior written consent of the Author.

Any subroutines or modules supplied by You and linked into the Source Code or Executable Files of this Work shall not be considered part of this Work and will not be subject to the terms of this License.

4. **Patent License.** Subject to the terms and conditions of this License, each Author hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, import, and otherwise transfer the Work.
5. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
 1. You agree not to remove any of the original copyright, patent, trademark, and attribution notices and associated disclaimers that may appear in the Source Code or Executable Files.

2. You agree not to advertise or in any way imply that this Work is a product of Your own.
3. The name of the Author may not be used to endorse or promote products derived from the Work without the prior written consent of the Author.
4. You agree not to sell, lease, or rent any part of the Work. This does not restrict you from including the Work or any part of the Work inside a larger software distribution that itself is being sold. The Work by itself, though, cannot be sold, leased or rented.
5. You may distribute the Executable Files and Source Code only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy of the Executable Files or Source Code You distribute and ensure that anyone receiving such Executable Files and Source Code agrees that the terms of this License apply to such Executable Files and/or Source Code. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute the Executable Files or Source Code with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License.
6. You agree not to use the Work for illegal, immoral or improper purposes, or on pages containing illegal, immoral or improper material. The Work is subject to applicable export laws. You agree to comply with all such laws and regulations that may apply to the Work after Your receipt of the Work.
6. **Representations, Warranties and Disclaimer.** THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.
7. **Indemnity.** You agree to defend, indemnify and hold harmless the Author and the Publisher from and against any claims, suits, losses, damages, liabilities, costs, and expenses (including reasonable legal or attorneys' fees) resulting from or relating to any use of the Work by You.
8. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL THE AUTHOR OR THE PUBLISHER BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK OR OTHERWISE, EVEN IF THE AUTHOR OR THE PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
9. **Termination.**
 1. This License and the rights granted hereunder will terminate automatically upon any breach by You of any term of this License. Individuals or entities who have received Derivative Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 6, 7, 8, 9, 10 and 11 will survive any termination of this License.
 2. If You bring a copyright, trademark, patent or any other infringement claim against any contributor over infringements You claim are made by the Work, your License from such contributor to the Work ends automatically.

3. Subject to the above terms and conditions, this License is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, the Author reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

10. **Publisher.**

The parties hereby confirm that the Publisher shall not, under any circumstances, be responsible for and shall not have any liability in respect of the subject matter of this License. The Publisher makes no warranty whatsoever in connection with the Work and shall not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. The Publisher reserves the right to cease making the Work available to You at any time without notice

11. **Miscellaneous**

1. This License shall be governed by the laws of the location of the head office of the Author or if the Author is an individual, the laws of location of the principal place of residence of the Author.
2. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this License, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
3. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
4. This License constitutes the entire agreement between the parties with respect to the Work licensed herein. There are no understandings, agreements or representations with respect to the Work not specified herein. The Author shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Author and You.

16.8 TaskDialog

Task Dialog is a TaskDialog wrapper with fallback emulator. It is licensed under the Code Project Open License (CPOL):

This License governs Your use of the Work. This License is intended to allow developers to use the Source Code and Executable Files provided as part of the Work in any application in any form.

The main points subject to the terms of the License are:

- Source Code and Executable Files can be used in commercial applications;
- Source Code and Executable Files can be redistributed; and
- Source Code can be modified to create derivative works.
- No claim of suitability, guarantee, or any warranty whatsoever is provided. The software is provided “as-is”.
- The Article(s) accompanying the Work may not be distributed or republished without the Author’s consent

- This License is entered between You, the individual or other entity reading or otherwise making use of the Work licensed pursuant to this License and the individual or other entity which offers the Work under the terms of this License (“Author”).

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CODE PROJECT OPEN LICENSE (“LICENSE”). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HEREIN, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE AUTHOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO ACCEPT AND BE BOUND BY THE TERMS OF THIS LICENSE, YOU CANNOT MAKE ANY USE OF THE WORK.

1. **Definitions.**

1. **“Articles”** means, collectively, all articles written by Author which describes how the Source Code and Executable Files for the Work may be used by a user.
 2. **“Author”** means the individual or entity that offers the Work under the terms of this License.
 3. **“Derivative Work”** means a work based upon the Work or upon the Work and other pre-existing works.
 4. **“Executable Files”** refer to the executables, binary files, configuration and any required data files included in the Work.
 5. **“Publisher”** means the provider of the website, magazine, CD-ROM, DVD or other medium from or by which the Work is obtained by You.
 6. **“Source Code”** refers to the collection of source code and configuration files used to create the Executable Files.
 7. **“Standard Version”** refers to such a Work if it has not been modified, or has been modified in accordance with the consent of the Author, such consent being in the full discretion of the Author.
 8. **“Work”** refers to the collection of files distributed by the Publisher, including the Source Code, Executable Files, binaries, data files, documentation, whitepapers and the Articles.
 9. **“You”** is you, an individual or entity wishing to use the Work and exercise your rights under this License.
2. **Fair Use/Fair Use Rights.** Nothing in this License is intended to reduce, limit, or restrict any rights arising from fair use, fair dealing, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
3. **License Grant.** Subject to the terms and conditions of this License, the Author hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:
1. You may use the standard version of the Source Code or Executable Files in Your own applications.
 2. You may apply bug fixes, portability fixes and other modifications obtained from the Public Domain or from the Author. A Work modified in such a way shall still be considered the standard version and will be subject to this License.
 3. You may otherwise modify Your copy of this Work (excluding the Articles) in any way to create a Derivative Work, provided that You insert a prominent notice in each changed file stating how, when and where You changed that file.

4. You may distribute the standard version of the Executable Files and Source Code or Derivative Work in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution.
5. The Articles discussing the Work published in any form by the author may not be distributed or republished without the Author's consent. The author retains copyright to any such Articles. You may use the Executable Files and Source Code pursuant to this License but you may not repost or republish or otherwise distribute or make available the Articles, without the prior written consent of the Author.

Any subroutines or modules supplied by You and linked into the Source Code or Executable Files of this Work shall not be considered part of this Work and will not be subject to the terms of this License.

4. **Patent License.** Subject to the terms and conditions of this License, each Author hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, import, and otherwise transfer the Work.
5. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
 1. You agree not to remove any of the original copyright, patent, trademark, and attribution notices and associated disclaimers that may appear in the Source Code or Executable Files.
 2. You agree not to advertise or in any way imply that this Work is a product of Your own.
 3. The name of the Author may not be used to endorse or promote products derived from the Work without the prior written consent of the Author.
 4. You agree not to sell, lease, or rent any part of the Work. This does not restrict you from including the Work or any part of the Work inside a larger software distribution that itself is being sold. The Work by itself, though, cannot be sold, leased or rented.
 5. You may distribute the Executable Files and Source Code only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy of the Executable Files or Source Code You distribute and ensure that anyone receiving such Executable Files and Source Code agrees that the terms of this License apply to such Executable Files and/or Source Code. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute the Executable Files or Source Code with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License.
 6. You agree not to use the Work for illegal, immoral or improper purposes, or on pages containing illegal, immoral or improper material. The Work is subject to applicable export laws. You agree to comply with all such laws and regulations that may apply to the Work after Your receipt of the Work.
6. **Representations, Warranties and Disclaimer.** THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

7. **Indemnity.** You agree to defend, indemnify and hold harmless the Author and the Publisher from and against any claims, suits, losses, damages, liabilities, costs, and expenses (including reasonable legal or attorneys' fees) resulting from or relating to any use of the Work by You.
8. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL THE AUTHOR OR THE PUBLISHER BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK OR OTHERWISE, EVEN IF THE AUTHOR OR THE PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9. **Termination.**

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of any term of this License. Individuals or entities who have received Derivative Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 6, 7, 8, 9, 10 and 11 will survive any termination of this License.
2. If You bring a copyright, trademark, patent or any other infringement claim against any contributor over infringements You claim are made by the Work, your License from such contributor to the Work ends automatically.
3. Subject to the above terms and conditions, this License is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, the Author reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

10. **Publisher.**

The parties hereby confirm that the Publisher shall not, under any circumstances, be responsible for and shall not have any liability in respect of the subject matter of this License. The Publisher makes no warranty whatsoever in connection with the Work and shall not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. The Publisher reserves the right to cease making the Work available to You at any time without notice

11. **Miscellaneous**

1. This License shall be governed by the laws of the location of the head office of the Author or if the Author is an individual, the laws of location of the principal place of residence of the Author.
2. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this License, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
3. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
4. This License constitutes the entire agreement between the parties with respect to the Work licensed herein. There are no understandings, agreements or representations with respect to the Work not specified herein. The Author shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Author and You.