Edip-documentation Documentation

Release latest

November 29, 2015

Contents

Contents:

CHAPTER 1

E.D.I.P presentation

Program Edip

Version 1.0.0

Author Eddie Brüggemann <mrcyberfighter@gmail.com>

Language C++

Release 24/11/2015

License GPLv3

1.1 E.D.I.P the Easy Digital Imaging Processing program

`Edip` is an easy to use image effects processing programme with severals functionnalities.

`Edip` permit to:

- **Rotate** your image over 90° to the **left** or to the **right**.
- **Mirror** your image, so that:
  - The left side from your image is copy, mirror likewise, on the right side.
  - The right side from your image is copy, mirror likewise, on the left side.
  - The upper part of your image is copy, mirror likewise, on the bottom part.
  - The bottom part of your image is copy, mirror likewise, on the upper part.
- **Zoom** on a **selected center** or the default **image center**.
- **Flipping** your image.
  - From left to right.
  - From right to left.
  - From up to down.
  - From down to up.

`Edip` permit to change the image color intensity and some phenomenal color spaces settings.

- **Color intensity**
  - You can set the **red** intensity.
  - You can set the **green** intensity.

  - You can set the **blue** intensity.
  - You can set the **global** intensity.
- **Phenomenal color spaces settings**
  - You can set the Lightness from your image.
  - You can set the *Hue* from your image.
  - You can set the *Saturation* from your image.
  - You can set the *Brightness* from your image.

`Edip` permit to apply severals built-in filters to your image by simply selecting the wanted filter in the list and pressing the apply button.

The built-in filters are divide into **3** differents categories:

- **normal filters**

  You can apply over **30** differents filters to your image.

- **Grayscaling filters**

  You can grayscale your image according **6** differents algorithms.

- **Colorscaling** filters

  You can colorscale your image in 6 differents colors (**red**, **green**, **blue**, **yellow**, **pink**, **turquoise**) according differents algorithms for every color.

In addition `Edip` provide many fully configurable filters which you can apply to your image:

- **Kernel** filters.
- **Photography** filters.
- **Morphological** filters.
- **Canny** filter.
- **Color contours** filter.
- **Multiply colors**.
- **Multiply global intensity**.
- **Set image in color tone**.

`Edip` provide some basic draw functionalities to draw onto your image in the wanted color with the wanted thickness or to fill your form.

- **Polylines**.
- **Rectangle**.
- **Circle**.
- **Ellipse**.
- **Polygone**: *normal* or *strikethrough*.
- **Star**: *normal*, *strikethrough*, *flower likewise*, *stroked*.
- **Text**.

CHAPTER 2

To know about imaging processing

2.1 The pixels mystery

An (raw) image is composed from pixels as you know.

Every pixel can be encoded on:

2.1.1 Basic pixel encoding

On a single value between **0** and **255**: This is the case of gray images.

In this case every pixel can take a value from **0** (**black**) to **255** (**white**).

To create every grayscale range value.

On **3** or **4** channels of colors for every pixel.

These channels are:

- The **red** channel which can take a value between **0** (**black**) and **255** (fully **red**).
- The **green** channel which can take a value between **0** (**black**) and **255** (fully **green**).
- The **blue** channel which can take a value between **0** (**black**) and **255** (fully **blue**).
- And a optional fourth channel can add an opacity value: which can take a value between **0** (fully **transparent**) and **255** (fully **opaque**):

  What mean that the image can contains fully transparent pixels, what permit to display a form which according per example your desktop background.

For every pixel: so the pixels are encoded on **3** or **4** values between **0** and **255** representing the **red**, **green**, **blue** and optionnaly **alpha** channel values.

What give us the **RGB(A)** color space.

This is not very intuitive and this is not the way humans think about colors.

But the **red**, **green** and **blue** values combined together give us a wide gamut of different colors

```
256 × 256 × 256 = 16777216 (differents possible colors).
```

Note: In fact, the human visual system is also based on the **trichromatic** perception of colors.

With cone cell sensitivity located around the red, gren and blue spectrum.

This is the common pixel encoding in digital imaging.

examples:

A fully red pixel will take following values:

Channel	Red	Green	Blue
Value	255	0	0

Resulting color:

Note: Red is a base color because only one channel is set.

A fully green pixel will take following values:

Channel	Red	Green	Blue
Value	0	255	0

Resulting color:

Note: Green is a base color because only one channel is set.

A fully blue pixel will take following values:

Channel	Red	Green	Blue
Value	0	0	255

Resulting color:

Note: Blue is a base color because only one channel is set.

A fully yellow pixel will take following values: We mix red and green to obtain yellow color.

Channel	Red	Green	Blue
Value	255	255	0

Resulting color:

Note: Yellow is a composite color because 2 differents channels are set.

A fully pink pixel will take following values: We mix red and blue to obtain pink color.

Channel	Red	Green	Blue
Value	255	0	255

Resulting color:

Note: Pink is a composite color because 2 differents channels are set.

A fully turquoise pixel will take following values: We mix the green and blue to obtain the turquoise color.

Channel	Red	Green	Blue
Value	0	255	255

Resulting color:

Note: Turquoise is a composite color because 2 differents channels are set.

A emboss gray pixel encoded on RGB (Red, Green, Blue) look's like this:

Channel	Red	Green	Blue
Value	127	127	127

Resulting color:

Note: Gray is a composite color because the 3 channels are set.

Gray can be encoded on a single value: a 1 byte encoding.

2.1.2 Other pixels encoding

Otherwise they are other image coding system's: the **phenomenal color spaces**.

Which are based on the concept of

- **Hue**.
- **Saturation**.
- And **brightness**.

These properties are more intuitive for humans.

The **phenomenal color spaces** have been introduced because they correspond to the way humans tend to naturally organized colors.

With atributes like:

- **Tint**.
- **Colorfullness**.
- And **brightness**.

- **Hue**: represent the dominant color.

  The name you give to a color correspond to the hue value.

  – The hue value varies from 0 to 180 by representing the dominant color on a circle (0-360 degrees) which value is divide per 2 for encoding convienence.

    or in others words the hue represent the tint of the color.

- **Saturation**: represent how vivid the color is.

  The saturation varies from **0** to **255** (pure spectrum color).

  The saturation represent the vivid of the colors:

- Low value : pastel color.
- High value : rainbow color.

$$Saturation = \frac{max(red, green, blue) - min(red, green, blue)}{max(red, green, blue)}$$

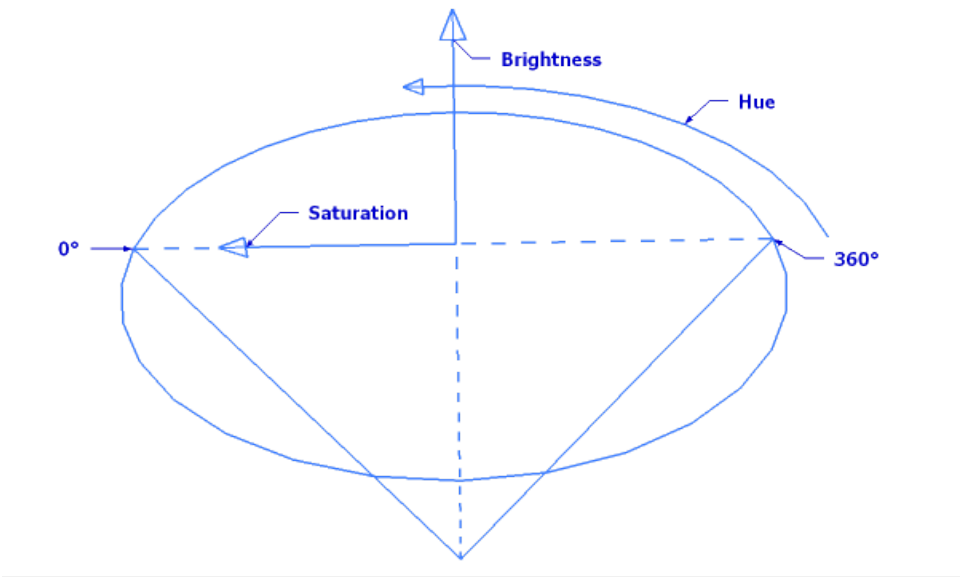
- **Brightness**: represent the luminosity of a color.

  This is an subjectiv attribute.

Note: Others **phenomenal color spaces** values use the concept of value or lightness as a way to characterize the relativ color intensity.

Note: This colors attributes try to mimic the intuitive human pereption of colors.

- Example: the **HSB** color space.



- The hue is the cone top circle.
- The brightness is the cone height.
- The saturation value depends from the length of the arrow which must ne oriented into the direction of the hue value.

2.2 The Convolution kernels

A kernel is a set of weights which determine how each output pixel is calculated from a neighborhood of input pixels.

Applying a kernel filter consist of moving the kernel over each pixel of an image and multiplying each corresponding pixel by it's associated weight.

According to that the central pixel correspond to a pixel of interest and the others to the neighborhood pixels.

The kernel is an odd matrix of values from any (odd) size limited from 3x3 to 31x31 and can be from different forms:

2.2.1 Kernels forms

- A **square** type kernel matrix:

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

- A **diamond** type kernel matrix:

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

- A **cross** type kernel matrix:

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

- An **X** type kernel matrix:

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1

The art which the values are multiply depends from the operator which use the matrix, and the central pixel value depend from it.

The total summe of the matrix values represent the weight of the matrix and so we often adjust the value of the central pixel to make the total weight equal to:

1 or 0

but not always because it depends from how the value are interpolate:

2.2.2 Classical kernel matrixes

- A **sharpen** kernel matrix:

-1	-1	-1
-1	9	-1
-1	-1	-1

has a weight of **+1** due of the **8** x **-1** values and the central **+9** value.

- A **Find Edges** kernel matrix:

-1	-1	-1
-1	8	-1
-1	-1	-1

has a weight of **0** due of the **8** x **-1** values and the central **+8** value.

- A **Emboss** kernel matrix:

-2	-1	0
-1	1	1
0	1	2

has a weight of **+1** due of the **(-2 + -1 + 1)** and **(2 + 1 + 1)** values and the central **1** value.

- A **Mean** kernel matrix:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

has a weight of **+1** du of the fact that all values are equal to **1.0** divide per kernel size.

- A **Gaussian** kernel matrix: is based on a gaussian vector: a vector of values seed from a sigma value. example from a gausssian vector from size **7** with **1.2** as sigma value:

```
[ 0.015, 0.083, 0.236, 0.333, 0.236, 0.083, 0.015 ]
```

- A **Kirsch** kernel matrix:

  Have an orientation:

  - East orientation:

-3	-3	5
-3	0	5
-3	-3	5

  - West orientation:

5	-3	-3
5	0	-3
5	-3	-3

  - North orientation:

-3	-3	-3
-3	0	-3
5	5	5

  - South orientation:

5	5	5
-3	0	-3
-3	-3	-3

- A **Sobel** kernel matrix:
  - horizontally:

-1	-2	-1
0	0	0
1	2	1

  - Vertically:

-1	0	1
-2	0	2
-1	0	1

- A **Laplacian** kernel matrix:

0.5	1.0	0.5
1.0	6	1.0
0.5	1.0	0.5

2.3 Thresholding

The use of thresholding can be used to obtain a *binary image* (black and white image) by setting a value: the threshold value.

In relationship to this value, all pixels values (encoded on a single value) greater as this value are set to white.

All pixels littler are set to black.

What produce a binary image map of the source image.

Note: Thresholding can be invert so that greater values are set to black and littler are set to white.

What permit to get a **White** and a **Black** version of the filters producing a *binary image*.

It exist algorithms which can determine the best threshold value based on the analysis of a grayscale image:

- **OTSU** algorithm. (As in the *Binary White OTSU* or *Binary Black OTSU* filters.)
- **TRIANGLE** algorithm. (As in the *Binary White TRIANGLE* or *Binary Black TRIANGLE* filters.)

Note: You can take the average value from this 2 algorithms computed threshold values

As in the *Binary White Average* or

the *Binary Black Average* filters.

2.4 Binary images

Binary images are strictly **black** and **white** images with as convention that:

The **foreground** from the image is represented in **white**.

The **background** from the image is represented in **black**.

CHAPTER 3

Files

3.1 Supported image file formats

`Edip` support *normally* following images files formats to load into the program for processing:

- **bmp**, **dib** (*bitmap image file.*)
- **jpg**, **jpeg**, **jpe** (*Join photographic Expert Group.*)
- **png** (*Protable Network Graphics.*)
- **tiff**, **tif** (*Tag(ged) Image File (Format).*)
- **ras**, **sr** (*Sun Raster*).
- **pbm**, **pgm**, **ppm** (*Portable Image Format.*)
- **webp**.

The image file is loaded into the program and converted into **RGB**, (internally **BGR**) format or **RGBA** (internally **BGRA**) if an alpha channel is available in the image.

Note: For supporting an image filetype the required library need to be installed except for:

- **bmp**, **dib** (*bitmap image file.*)
- **pbm**, **pgm**, **ppm** (*Portable Image Format.*)
- **ras**, **sr** (*Sun Raster*).

Which are always supported.

3.2 Load images files

You can load an image file to process it into `Edip` by clicking on the *Open* button in the buttonbar, through the menu item *Files* → *Open file* or by hitting the shortcut `Ctrl + O`.

You can start the program with am image file path as argument:

```
$ edip image_filepath
```

You can open an recent image file by using the menu item *Files* → *Recent files*.

3.3 Saving image files

You can save an image file by pressing the button *Save* in the button bar,

trough the menu item *Files* → *Save file* or by hitting the shortcut `Ctrl + S`.

Then you can set the output size of the image which default to the originally image size.

And setting some options in relatinship to the format if available.

3.3.1 Image file format saving options

png image file saving options

- **Compression**: from level from 0 to 9. A higher value means a smaller size and longer compression time. Default value is 3.
- **Strategy**:
  - Default.
  - Filtered.
  - Huffman only.
  - RLE.
  - Fixed.
- **Bilevel**:

  You can only store an image as bilevel if the image is a *binary* (black and white) image.

  Otherwise the result will be undefined.

jpeg image file saving options

- **Quality**: it can be a quality from 0 to 100 (the higher is the better). Default value is 95.
- **Features**:
  - **Progressiv** (disabled per default).
  - **Optimize** (disabled per default).
- **RST Restart**: JPEG restart interval, 0 - 65535, default is 0 - no restart.
- **LUMA quality**: Separate luma quality level, 0 - 100, default is 0 - don't use.
- **CHROMA quality**: Separate chroma quality level, 0 - 100, default is 0 - don't use.

xpm image file saving options

- **Binary**: For PPM, PGM, or PBM, it can be a binary format flag, 0 or 1. Default value is 1.

webp image file saving options

- **Quality**: For WEBP, it can be a quality from 1 to 100 (the higher is the better). By default and for quality above 100 the lossless compression is used.

CHAPTER 4

Images Files Merging

`Edip` provide severals files merging algorithms which permit to combine 2 input files to an single output file through an image mixing algorithm.

You can choose 2 differents input image files, in any supported format, with giving eventual settings, to merge it into a single ouput image file.

The differents images must have the same size to be interpolated, so `Edip` resize the output image to the same size, to the littler image file size from the twos per default.

But you can set the output size you want for your output image.

Then you can save the result of the image files merging to any supported format, by clicking on the *Save file* button.

4.1 Files merging Blend algorithm

Interpolate 2 images file with the constant alpha to generate an output image:

```
Result = Image1 * alpha - (1-alpha) * Image2
```

- You must set the value of **alpha** which default to **0.5** between **0.01** and **1.0**.

The images are converted to the same size (to the littler image) was is required for blend operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.2 Files merging Screen algorithm

Superimpose two inverted images on the top of each other:

```
Result = MAX - ((MAX-Image1)  * (MAX-Images2) / MAX)
```

The images are converted to the same size (to the littler image) was is required for screen operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.3 Files merging Darker algorithm

Interpolate 2 images file and produce an output image with the darkest pixels values:

```
Result = min( Image1, Image2 )
```

The images are converted to the same size (to the littler image) was is required for darker operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.4 Files merging Lighter algorithm

Interpolate 2 images file and produce an output image with the lightest pixels values:

```
Result = max( Image1, Image2 )
```

The images are converted to the same size (to the littler image) was is required for lighter operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.5 Files merging Add algorithm

Add 2 images, dividing the result by scale and adding offset to generate an output image:

```
Result = (Image1 + Image2) / scale + offset
```

- You must set the value of **scale** which default to **1.0** between **0.01** and **8.0**.
- You must set the value of **offset** which default to **0** between **0** and **255**.

The images are converted to the same size (to the littler image) was is required for add operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.6 Files merging Add modulo algorithm

Add 2 images and make the euclidian division of the result per the maximal:

```
Result = ( Image1 + Image2 ) % MAX
```

The images are converted to the same size (to the littler image) was is required for add modulo operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.7 Files merging Subtract algorithm

Subtract 2 images, dividing the result by scale and adding offset to generate an output image:

```
Result = (Image1 - Image2) / scale + offset
```

- You must set the value of **scale** which default to **1.0** between **0.01** and **8.0**.
- You must set the value of **offset** which default to **0** between **0** and **255**.

The images are converted to the same size (to the littler image) was is required for sub operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

4.8 Files merging Subtract modulo algorithm

Subtract 2 images and make the euclidian division of the result per the maximal:

```
Result = ( Image1 - Image2 ) % MAX
```

The images are converted to the same size (to the littler image) was is required for sub modulo operation.

Note: You can use the button *Preview result* to preview your output image.

If the output size is greater than 800x600 pixels the preview image is resize to be included in the image preview window.

CHAPTER 5

Simple Settings

You will find the simple settings on the right sidebar from `Edip`.

You can switch between *Intensity* and *Colorspace* with the corresponding buttons.

5.1 Intensity

Here you can change the color intensities with the corresponding scales.

- **Red**: from -255 to 255.

  Set the wanted **red** intensity change and press the *Apply* button.

  - If the selected value is **greater** than **0** it will **add** the selected value to the **red** channel from all pixels of your image.
  - If the selected value is **littler** than **0** it will **subtract** the selected value from the **red** channel from all pixels of your image.

- **Green**: from -255 to 255.

  Set the wanted **green** intensity change and press the *Apply* button.

  - If the selected value is **greater** than **0** it will **add** the selected value to the **green** channel from all pixels of your image.
  - If the selected value is **littler** than **0** it will **subtract** the selected value from the **green** channel from all pixels of your image.

- **Blue**: from -255 to 255.

  Set the wanted **blue** intensity change and press the *Apply* button.

  - If the selected value is **greater** than **0** it will **add** the selected value to the **blue** channel from all pixels of your image.
  - If the selected value is **littler** than **0** it will **subtract** the selected value from the **blue** channel from all pixels of your image.

- **Global**: from -255 to 255.

  Set the wanted intensity change and press the *Apply* button.

  - If the selected value is **greater** than **0** it will **add** the selected value to the **all** channels from all pixels of your image.

  - If the selected value is **littler** than **0** it will **subtract** the selected value from the **all** channels from all pixels of your image.

5.2 Colorspace

Here you can set:

- The Lightness from your image.
- The *Hue* from your image.
- The *Saturation* from your image.
- The *Brightness* from your image.

CHAPTER 6

Simple Filters

6.1 Edip built-in filters

`Edip` provide a lot of predefine, non-configurable built-in filters, applicable like this:

6.1.1 Pencil Sketch

The Pencil Sketch filter produce a non-photorealistic line drawing image.

6.1.2 Stylisation

The stylisation filter is an edge-aware filters which effect is not focused on photorealism but abstract regions of low contrast while preserving, or enhancing, high-contrast features.

6.1.3 Detail Enhance

The Detail Enhance filter enhances the details of a particular image.

6.1.4 Edges Preserving

The Edge Preserving filter is a smoothing filters used in many different applications.

6.1.5 Stroke Edges

The Stroke Edges filter add black strokes on some edges of the image.

6.1.6 Invert Intensity

The Invert Intensity filter invert the intensity.

This mean that the maximal value become the minimal value and so soon in the pixel values range.

6.1.7 Light Intensity

The Light Intensity filter invert the intensity from the pixels channels average value.

So that it output an grayscale image with light effect.

6.1.8 Recolor RC (Red-Cyan)

Simulate conversion from RGB (Red, Green, Blue) to RC (red, cyan).

Blues and greens are replaced with cyans.

The effect is similar to Technicolor Process 2 (used in early color movies) and CGA Palette 3 (used in early color PCs).

6.1.9 Recolor RGV (Red-Green-Value)

Simulate conversion from RGB (Red, Green, Blue) to RGV (red, green, value).

Blues are desaturated.

The effect is similar to Technicolor Process 1 (used in early color movies).

6.1.10 Recolor CMV (Cyan-Magenta-Value)

Simulate conversion from RGB (Red, Green, Blue) to CMV (cyan, magenta, value).

Yellows are desaturated.

The effect is similar to CGA Palette 1 (used in early color PCs).

6.1.11 Extrema Maximal

The Extrema Maximal filter compute the maximal values from 2 differents channels and set the result on a channel.

6.1.12 Extrema Minimal

The Extrema Minimal filter compute the minimal values from 2 differents channels and set the result on a channel.

6.1.13 Sharpen

The Sharpen filter sharpen the image by adding some white pixels at some edges and points of the image.

This filter is based on a *Sharpen convolution matrix*.

6.1.14 Sharpen More

The Sharpen More filter sharpen the image by adding some white pixels at some edges and points of the image.

It has more impact on the image as the Sharpen filter but does the same stronger.

This filter is based on a *Sharpen convolution matrix*.

6.1.15 Find Edges

The Find Edges filter will result in black background image with the contours and some pixels, of the source image, displayed in color.

This filter is based on a *Find Edges convolution matrix*.

6.1.16 Mean Blur

The Mean Blur filter swindle the image. The result image is blurred.

This filter is based on a *Mean convolution matrix*.

6.1.17 Mean Blur More

The Mean Blur More filter swindle the image. The result image is blurred.

It has more impact on the image as the Mean Blur filter but does the same stronger.

This filter is based on a *Mean convolution matrix*.

6.1.18 Blur

The Blur filter swindle the image.

What can remove some noise of the image or smooth an pixelize image.

6.1.19 Blur Median

The Blur Median filter swindle the image.

What can remove some noise of the image or smooth an pixelize image.

This filter is well know for salt and pepper noise removing.

6.1.20 Blur Gaussian

The Blur Gaussian filter swindle the image.

What can remove some noise of the image or smooth an pixelize image.

It use *gaussian values* for creating the blurring effect applying onto the image.

6.1.21 Denoising

The denoising filter perform image denoising using Non-local Means Denoising with several computational optimizations.

Noise expected to be a gaussian white noise.

Denoising is done by converting image to CIELAB colorspacenand then separately denoise L and AB components with different parameters.

6.1.22 Erode

The Erode filter eroding the image content contours.

This filter erodes the image by minimizing the pixels neighborhood.

6.1.23 Dilate

The Dilate filter dilate the image content contours.

This filter dilates the image by maximizing the pixels neighborhood.

6.1.24 Wave Horizontally

The Wave Horizontally filter output an horizontally waved image.

6.1.25 Wave Vertically

The Wave Vertically filter output an horizontally waved image.

6.1.26 Wave Twice

The Wave Twice filter output an horizontally and vertically waved image.

6.1.27 Contours Sobel White

The Contours *Sobel* White filter result in an image with a white background and colored drawning wise foreground.

All the contours strokes of the image will be displayed on a white background.

6.1.28 Contours Sobel Black

The Contours *Sobel* Black filter result in an image with a black background and colored drawning wise foreground.

All the contours strokes of the image will be displayed on a black background.

6.1.29 Contours Sobel Emboss

The Contours *Sobel* Emboss filter result in an image with an *emboss gray* background and colored drawning wise foreground.

All the contours strokes of the image will be displayed on a *emboss gray* background.

6.1.30 Emboss Sobel

The Emboss *Sobel* filter is an emboss filter which result looking like a kopper engraving.

Based on the Sobel operator which detect edges horizontally and or vertically.

6.1.31 Emboss Laplacian

The Emboss *Laplacian* filter is an emboss filter which result looking like a kopper engraving.

Based on the Laplacian operator which detect edges horizontally and vertically.

6.1.32 Binary White OTSU

The Binary White OTSU filter create an black and white image, from the source image, with white background and black foreground.

Based on the OTSU image specific threshold computing algorithm.

Simply try every algorithm to find the appositely one for your image, to get the best binary image from it.

6.1.33 Binary White TRIANGLE

The Binary White TRIANGLE filter create an black and white image, from the source image, with white background and black foreground.

Based on the TRIANGLE image specific threshold computing algorithm.

Simply try every algorithm to find the appositely one for your image, to get the best binary image from it.

6.1.34 Binary White AVERAGE

The Binary White AVERAGE filter create an black and white image, from the source image, with white background and black foreground.

Based on the OTSU and TRIANGLE image specific threshold computing algorithm, average.

Simply try every algorithm to find the appositely one for your image, to get the best binary image from it.

6.1.35 Binary Black OTSU

The Binary Black OTSU filter create an black and white image, from the source image, with black background and white foreground.

Based on the OTSU image specific threshold computing algorithm.

Simply try every algorithm to find the appositely one for your image, to get the best binary image from it.

6.1.36 Binary Black TRIANGLE

The Binary Black TRIANGLE filter create an black and white image, from the source image, with black background and white foreground.

Based on the TRIANGLE image specific threshold computing algorithm.

Simply try every algorithm to find the appositely one for your image, to get the best binary image from it.

6.1.37 Binary Black AVERAGE

The Binary Black AVERAGE filter create an black and white image, from the source image, with black background and white foreground.

Based on the OTSU and TRIANGLE image specific threshold computing algorithm, average.

Simply try every algorithm to find the appositely one for your image,nto get the best binary image from it.

6.1.38 Binary Contours White

The Binary Contours White filter create an black and white image, from the source image, with white background and black foreground.

With as foreground all the contours strokes of the image.

Like an black and white drawning.

6.1.39 Binary Contours Black

The Binary Contours Black filter create an black and white image, from the source image, with black background and white foreground.

With as foreground all the contours strokes of the image.

Like an black and white drawning.

6.2 Simple Grayscaling

`Edip` can grayscaling your image, to set it enterely in gray tone, according differents bases:

- Gray Scaling **Average**.
- Gray Scaling **Maximal**.
- Gray Scaling **Minimal**
- Gray Scaling **Red**.
- Gray Scaling **Green**.
- Gray Scaling **Blue**.

`Edip` will process all the pixels of your images to compute the wanted gray values.

To know that either a *gray pixel* can be encoded on a single value between **0** and **255** or

setting all channels on the same value will result in a *gray* pixel too.

For example let say the current pixel `Edip` is procesing has the following channels values:

Channel	Red	Green	Blue
Value	32	128	192

- Gray Scaling **Average**:

  `Edip` will compute the *average from all channels* and set the result as **gray value**.

  So `Edip` compute the average value:

```
(32 + 128 + 192) / 3 = 117.
```

  So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	117	117	117

  Or be encoded on a single value equal to **117**.

- Gray Scaling **Maximal**:

  `Edip` will take the *highest value* from all the channels and set the it as **gray value**.

  So `Edip` compute the **maximal value** from the channels:

```
32 < 128 < 192 = 192.
```

  So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	192	192	192

  Or be encoded on a single value equal to **192**.

- Gray Scaling **Minimal**:

  `Edip` will take the *lowest value* from all the channels and set the it as **gray value**.

  So `Edip` compute the **minimal value** from the all channels:

```
192 > 128 > 32 = 32.
```

  So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	32	32	32

  Or be encoded on a single value equal to **32**.

- Gray Scaling **Red**:

  `Edip` will take the value from the *red channel* and set the it as **gray value**.

  So `Edip` recover the **red** channel value which is equal to **32**:

```
Red = 32.
```

  So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	32	32	32

  Or be encoded on a single value equal to **32**.

- Gray Scaling **Green**:

`Edip` will take the value from the *green channel* and set the it as **gray value**.

So `Edip` recover the **green** channel value which is equal to **32**:

```
Green = 128.
```

So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	128	128	128

Or be encoded on a single value equal to **128**.

- Gray Scaling **Blue**:

  `Edip` will take the value from the *green channel* and set the it as **gray value**.

  So `Edip` recover the **blue** channel value which is equal to **192**:

```
Blue = 192.
```

  So the resulting pixel will be transform to have following values:

Channel	Red	Green	Blue
Value	192	192	192

  Or be encoded on a single value equal to **192**.

Note: `Edip` will compute every pixel from your image like explain above according the choosen grayscaling base.

6.3 Simple Colorscaling

`Edip` can colorscaling your image, to set it enterely in a main color tone, according differents bases:

- In *red* tone:
  - Red Scaling **Average**.
  - Red Scaling **Maximal**.
  - Red Scaling **Minimal**.
  - Red Scaling **Green**.
  - Red Scaling **Blue**.
- In *green* tone:
  - Green Scaling **Average**.
  - Green Scaling **Maximal**.
  - Green Scaling **Minimal**.
  - Green Scaling **Red**.
  - Green Scaling **Blue**.
- In *blue* tone:
  - Blue Scaling **Average**.
  - Blue Scaling **Maximal**.
  - Blue Scaling **Minimal**.

  - Blue Scaling **Green**.
  - Blue Scaling **Blue**.
- In *yellow* tone:
  - Yellow Scaling **Average**.
  - Yellow Scaling **Maximal**.
  - Yellow Scaling **Minimal**.
  - Yellow Scaling **Red**.
  - Yellow Scaling **Green**.
  - Yellow Scaling **Blue**.
- In *pink* tone:
  - Pink Scaling **Average**.
  - Pink Scaling **Maximal**.
  - Pink Scaling **Minimal**.
  - Pink Scaling **Red**.
  - Pink Scaling **Green**.
  - Pink Scaling **Blue**.
- In *turquoise* tone:
  - Turquoise Scaling **Average**.
  - Turquoise Scaling **Maximal**.
  - Turquoise Scaling **Minimal**.
  - Turquoise Scaling **Red**.
  - Turquoise Scaling **Green**.
  - Turquoise Scaling **Blue**.

Note: The pixels values computing according the differents bases are the same at by *grayscaling*.

CHAPTER 7

Advanced Configurable Filters

7.1 Kernel filters

7.1.1 Sharpen kernel filter

You can apply a fully configurable *Sharpen kernel* filter to the image.

- **Configuration options**:
  - **Kernel type**:
    - *Square*
    - *Diamond*
    - *Cross*
    - *X*
  - **Kernel size**:

    The size of the kernel: per example a size of **3** correspond to a **3x3 matrix**.

    You can only use **odd** values between **3** and **31** included.
  - **Values factor**:

    The factor to multtply each kernel component with.
  - **Center factor**:

    The factor to multiply the kernel center with.

Note: You can preview the kernel you have configurated by using the *Preview kernel* button.

7.1.2 Emboss kernel filter

You can apply a fully configurable *Emboss kernel* filter to the image.

- **Configuration options**:
  - **Kernel type**:
    - *Square*
    - *Diamond*

        * *Cross*
        * *X*
    - **Kernel size**:

      The size of the kernel: per example a size of **3** correspond to a **3x3 matrix**.

      You can only use **odd** values between **3** and **31** included.

Note: You can preview the kernel you have configurated by using the *Preview kernel* button.

7.1.3 Mean kernel filter

You can apply a fully configurable *Mean kernel* filter to the image.

- **Configuration options**:
    - **Kernel type**:
        * *Square*
        * *Diamond*
        * *Cross*
        * *X*
    - **Kernel size**:

      The size of the kernel: per example a size of **3** correspond to a **3x3 matrix**.

      You can only use **odd** values between **3** and **31** included.

Note: You can preview the kernel you have configurated by using the *Preview kernel* button.

7.1.4 Gaussian kernel filter

You can apply a fully configurable *Gaussian kernel* filter to the image.

- **Configuration options**:
    - **Kernel type**:
        * *Square*
        * *Diamond*
        * *Cross*
        * *X*
    - **Kernel size**:

      The size of the kernel: per example a size of **3** correspond to a **3x3 matrix**.

      You can only use **odd** values between **3** and **31** included.
    - **Sigma value**:

      Seed value used to generate the *gaussian vector* on which the kernel will be based.

Note: You can preview the kernel you have configurated by using the *Preview kernel* button.

7.1.5 Kirsch kernel filter

You can apply a fully configurable *Kirsch kernel* filter to the image.

- **Configuration options**:
  - Direction:
    - *East*
    - *West*
    - *North*
    - *South*
  - **Kernel size**:

    The size of the kernel: per example a size of **3** correspond to a **3x3 matrix**.

    You can only use **odd** values between **3** and **31** included.
  - **Values factor**:

    The factor to multtply each kernel component with.

Note: You can preview the kernel you have configurated by using the *Preview kernel* button.

7.2 Photography filters

7.2.1 Pencil Sketch

The Pencil Sketch filter produce a non-photorealistic line drawing image.

- **Configuration options**:
  - **Sigma S**:

    Value between **0.0** and **200.0**.
  - **Sigma R**:

    Value between **0.0** and **1.0**.
  - **Shade** factor:

    Value between **0.001** and **0.100**.

7.2.2 Stylisation

The stylisation filter is an edge-aware filters which effect is not focused on photorealism but abstract regions of low contrast while preserving, or enhancing, high-contrast features.

- **Configuration options**:

  - **Sigma S**:

    Value between **0.0** and **200.0**.

  - **Sigma R**:

    Value between **0.0** and **1.0**.

7.2.3 Detail Enhance

The Detail Enhance filter enhances the details of a particular image.

- **Configuration options**:
  - **Sigma S**:

    Value between **0.0** and **200.0**.

  - **Sigma R**:

    Value between **0.0** and **1.0**.

7.2.4 Edge Preserving

The Edge Preserving filter is a smoothing filters used in many different applications.

- **Configuration options**:
  - **Sigma S**:

    Value between **0.0** and **200.0**.

  - **Sigma R**:

    Value between **0.0** and **1.0**.

  - **Filter**:
    - **Recurse** filter.
    - **NormConv** Filter.

7.2.5 Denoising

The denoising filter perform image denoising using Non-local Means Denoising with several computational optimizations.

Noise expected to be a gaussian white noise.

Denoising is done by converting image to CIELAB colorspace and then separately denoise L and AB components with different parameters.

- **Configuration options**:
  - **Luminance factor**:

    Parameter regulating filter strength. Big values perfectly removes noise but also removes image details, smaller values preserves details but also preserves some noise.

  - **Color denoising factor**:

    The same as **Luminance factor** but for color components.

7.3 Morphological filters

Morphological operations process images according to shapes.

They apply a defined **structuring element** (named *kernel*) to the image obtaining a new image where the current pixel is computed by comparing it to his neighborhood pixels.

Note: Depending on the **structuring element** selected (which is fully configurable in `Edip`)

a morphological operation is more sensitive to one specific shape or the other.

7.3.1 Morphological operations

Dilatation

Dilatation add pixels from the background to the boundaries of the object in an image.

In dilatation, the value of the output pixel is the maximum of all pixels in the neighborhood.

Erosion

Erosion remove pixels from the foreground.

Using erosion the value of the output pixel is the minimum of all pixels in the neighborhood.

Opening

Opening is an *erosion* followed by a *dilatation*.

Opening remove small objects from an image while preserving the larger one.

Closing

Closing is a *dilatation* followed by an *erosion*.

Closing is used to remove samll holes while preserving the larger one.

Tophat

Tophat ouput is the difference between the source image and it's *opening*.

Blackhat

Blackhat ouput is the difference between the source image and it's *closing*.

7.4 Canny filter

The **Canny** contours detection operator use 2 different *threshold* values:

Low threshold: The low threshold should be chosen in a way that it included all edges pixels that are considered as to belong to a significant image contour.

This should detect more edges that what is ideally needed in the case of the **Canny** algorithm.

High threshold: His role is to define the edges that belong to all important contours.

It should exclude all edges considered as outlier.

The **Canny** algorithm combines these two edges maps in order to produce an optimal map of contours.

It operate by keeping only edge points of the low threshold edge map for which a continuous path of edges exist, linking all edges points of the high threshold map are kept, while isolated chains of edges points in the low threshold map are removed.

Note: This strategy, based on the use of **two threshold** to obtain a binary map, is called **histereris thresholding**.

In addition, the **Canny** algorithm uses an extra strategy to improve the quality of the edge map: all edges points for which the gradient magnitude is not a maximum in the gradient direction are removed.

The gradient orientation is always perpendicular to the edge. Therefor, the local maximum of the gradient is this direction correspond to the points of maximum strength of the contours.

This explains why thin edges are obtained in tha **Canny** contours map.

7.5 Color contours filter

This produce the same result as the **Canny** filter except that you can set:

- The foreground color: the color of the detected edges.
- The background color.

CHAPTER 8

Advanced Configurable Color Filters

8.1 Mulitply Colors

You can set following factors to multiply each different color channel by it corresponding factor value.

- Multiply **red** channel:

  Factor to multiply the **red** channel.

- Multiply **green** channel:

  Factor to multiply the **green** channel.

- Multiply **blue** channel:

  Factor to multiply the **blue** channel.

- Multiply **alpha** channel:

  Factor to multiply the **alpha** channel.

8.2 Multiply global intesity

Mulitply each color channel (**red**, **green** and **blue**) by the given factor.

8.3 Set image in color tone

You can select a color to set the entire image in this color tone.

The color tone is applied to the image by following steps:

1. Computing the image mean value for every channel.
2. Dividing each given color by the corresponding mean value from each channel.
3. Multiply each channel by the result from step 2.

CHAPTER 9

Drawing functionalities

`Edip` provide following drawing functionalities.

- You can set the color of the form to draw.
- You can set the stroke thickness of the form to draw or choose to fill your form.

9.1 Polylines

Draw a line or joined multilines, by clicking on the image with the left mouse button.

Press the right mouse button to stop drawning.

Note: If you clic on the startpoint of mutilines drawning it will join the lines to a polygon.

9.2 Rectangle

Draw an rectangle, by clicking on the image with the left mouse button, to set an anchor corner.

Then move the mouse to size it at your convenience, and click again to draw the rectangle onto your image.

Press the right mouse button to stop drawning.

9.3 Circle

Draw a circle, by clicking on the image with the left mouse button, to set his center.

Then move the mouse to size it at your convenience, and click again to draw the circle onto your image.

Press the right mouse button to stop drawning.

9.4 Ellipse

Draw an ellipse, by clicking on the image with the left mouse button, to set his center.

Then move the mouse to size it at your convenience, and click again to draw the ellipse onto your image.

Press the right mouse button to stop drawning.

9.5 Polygon

Draw a convex regular polygon: first configure what type of polygon you want to draw.

Then click on the image with the left mouse button, to set his center.

And move the mouse to size it at your convenience, and click again to draw the polygon onto your image.

Press the right mouse button to stop drawning.

Note: You can press the keys '$' or '*' to turn the polygon and hit the '=' key to draw it.

9.6 Star

Draw a star: first configure what type of star you want to draw.

Then click on the image with the left mouse button, to set his center.

And move the mouse to size it at your convenience, and click again to draw the star onto your image.

Press the right mouse button to stop drawning.

Note: You can press the keys '$' or '*' to turn the star and hit the '=' key to draw it.

9.7 Text

Insert some text on your image: enter your text and configure the font settings.

Then slide the text to the wanted position, and click with the left mouse button, to set the text position.

Press the right mouse button to stop editing.

CHAPTER 10

Credits

10.1 Family

Note: Thank's to my beloved mother, my family and to the doctors.

Stay away from drugs: drugs destroy your brain and your life.

10.2 OpenCV

The **OpenCV** library provide over 300 algorithms for computer vision purpose.

The **OpenCV** libary was introduced in 1999 and adopted as primary development tool for computer vision by researcher and developers in computer vision.

OpenCV was originally developed at **Intel** by a team lead by **Gary Bradsky** as an initiative to advanced research in vision and promote the development of rich vision, CPU-intensiv applications.

1. 2006 first major release of **OpenCV**.
2. 2009 second major release of **OpenCV**.
3. 2012 reshape from **OpenCV** to a non-profit foundation opencv.org

Note: Thank's to all the contributors of the **OpenCV** libary for providing us this exceptional tool.

10.3 Sources

- `OpenCV Computer Vision Application Programming Cookbook` from **Robert Langarnière** at *Packt Publishing*.
- `Learning Image Processing with OpenCV` from **multi-authors** at *Packt Publishing*.
- `OpenCV Computer Vision with Python` from **Joseph Howse** at *Packt Publishing*.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search