PyVX Documentation

Release 0.3.0

Hakan Ardo

November 07, 2015

Contents

PyVX is a set of python bindings for OpenVX. OpenVX is a standard for expressing computer vision processing algorithms as a graph of function nodes. This graph is verified once and can then be processed (executed) multiple times. PyVX allows these graphs to be constructed and interacted with from python. It also supports the use of multiple OpenVX backends, both C and python backends. It also used to contain a code generating OpenVX backend written it python, but it will be moved to a package of it's own (curently it lives on the try1 branch of pyvx).

CHAPTER 1

Status

The C-like Python API `pyvx.api` wraps the entire OpenVX standard version 1.0.1. The pythonic interface `pyvx.pythonic` is still work in progress. Some small examples are working. See the demo directory.

CHAPTER 2

Installation

Then there are a few different ways to install PyVX:

- Use pip:

```
sudo pip install pyvx
```

- or get the source code via the Python Package Index.
- or get it from Github:

```
git clone https://github.com/hakanardo/pyvx.git
cd pyvx
sudo python setup.py install
```

This will install the frontend and the python API. You will also need one or several openvx backend implementations. For each backend you need to build some backend specific wrappers:

```
sudo python -mpyvx.build_cbackend [--default] name /path/to/openvx/install/
```

This will make the openvx backend installed in /path/to/openvx/install/ availible as `pyvx.backend.name` in python. One of the backends should be marked as the default backend using the –default switch.

There is a sample bakcend implementation from Khronos. To use it as the default backend on a 64 bit linux system:

```
cd /usr/local/src
wget https://www.khronos.org/registry/vx/sample/openvx_sample_1.0.1.tar.bz2
tar xf openvx_sample_1.0.1.tar.bz2
cd openvx_sample
python Build.py --os Linux
python -mpyvx.build_cbackend --default sample /usr/local/src/openvx_sample/install/Linux/x64/Release/
```

CHAPTER 3

Modules

The main modules of PyVX are:

pyvx.vx C-like Python API following the OpenVX vxXxx API as strictly as possible.

pyvx.vxu C-like Python API following the immediate OpenVX vxuXxx API as strictly as possible.

pyvx.pythonic A more python friendly version of the OpenVX API.

3.1 pyvx

pyvx.**use_backend**(*backend*)

Specifies which backend to use. It is typically called before any other modules are imported. If it is called later it will reload the modules. However if any symbols was imported from the modules, they will have to be reimported. If it is not called, the default backend will be used.

The *backend* parameter can either be an already imported module or a string which specifyes a module under *pyvx.backend* to load. That is the same string that was passed as the *name* argument of build_cbackend during *installation*. The first form allows for backends that's not part of pyvx to be used.

Typical usage:

```
from pyvx import use_backend
use_backend("sample")
from pyvx import vx
```

3.2 pyvx.vx — C-like Python API

The functions specified by the OpenVX standard are provided in form of two modules, *pyvx.vx* that provide the vxXxxfunctions and *pyvx.vxu* that provide the vxuXxx functions. Pleaserefer to the OpenVX speficication for a description of the API. The modulenames vx and vxu is used instead of a vx/vxu prefix on all symbols. The initialexample on page 12 of the specification would in python look like this:

```
from pyvx import vx

context = vx.CreateContext()
images = [
    vx.CreateImage(context, 640, 480, vx.DF_IMAGE_UYVY),
    vx.CreateImage(context, 640, 480, vx.DF_IMAGE_S16),
    vx.CreateImage(context, 640, 480, vx.DF_IMAGE_U8),
```

```
]
graph = vx.CreateGraph(context)
virts = [
    vx.CreateVirtualImage(graph, 0, 0, vx.DF_IMAGE_VIRT),
    vx.CreateVirtualImage(graph, 0, 0, vx.DF_IMAGE_VIRT),
    vx.CreateVirtualImage(graph, 0, 0, vx.DF_IMAGE_VIRT),
    vx.CreateVirtualImage(graph, 0, 0, vx.DF_IMAGE_VIRT),
]
vx.ChannelExtractNode(graph, images[0], vx.CHANNEL_Y, virts[0])
vx.Gaussian3x3Node(graph, virts[0], virts[1])
vx.Sobel3x3Node(graph, virts[1], virts[2], virts[3])
vx.MagnitudeNode(graph, virts[2], virts[3], images[1])
vx.PhaseNode(graph, virts[2], virts[3], images[2])
status = vx.VerifyGraph(graph)
print status
if status == vx.SUCCESS:
    status = vx.ProcessGraph(graph)
else:
    print("Verification failed.")
vx.ReleaseContext(context)
```

For a compact example on how to call all the functions in the API check out test_vx.py.

The API is kept as close as possible to the C API, but the few changes listed below were made. Mostly due to the usage of pointers in C.

- The vx prefix is removed for each function name. The module name forms a similar role in python.
- The *ReleaseXxx* and *RemoveNode* functions take a normal object (as returned by the corresponding CreateXxx) as argument and not a pointer to a pointer.
- Out arguments passed in as pointers are returned instead. The returned tuple will contain the original return value as it's first value and following it, the output arguments in the same order as they apear in the C signature.
- In/Out arguemnts are passed in as values and then returned in the same manner as the out arguments.
- Any python object implementing the buffer interface can be passed instead of pointers to blocks of data. This includes both *array.array* and *numpy.ndarray* objects.
- Python buffer objects are returned instead of pointers to blocks of data.
- ***QueryXxx* functions have the signature**

```
(status, value) = vx.QueryXxx(context, attribute, c_type, python_type=None)
```

  where *c_type* is a string specifying the type of the attribute, for example "vx_uint32", and *python_type* can be set to *str* for string-valued attributes.

- ***SetXxxAttribute* functions have the signature**

```
status = vx.SetXxxAttribute(context, attribute, value, c_type=None)
```

  where *c_type* is a string specifying the type of the attribute, for example "vx_uint32".

- ***CreateUniformImage* have the signature**

```
image = vx.CreateUniformImage(context, width, height, color, value, c_type)
```

  where value is a python *int* and *c_type* a string specifying it's type. For example "vx_uint32".

- Normal python functions can be used instead of function pointers.

- *LoadKernels* can load python modules if it is passed a string that is the name of an importable python module. In that case it will import *PublishKernels* from it and call *PublishKernels(context)*.
- *CreateScalar* and *WriteScalarValue* take a python int as value.
- Objects are not implicitly casted to/from references. Use `pyvx.vx.reference()` and `pyvx.vx.from_reference()` instead.
- The typedefed structures called vx_xxx_t can be allocated using vx.xxx_t(...). See below.

pyvx.vx.**from_reference**(*ref*)
Cast the "vx_reference" object *ref* into it's specific type (i.e. "vx_image" or "vx_graqph" or ...).

pyvx.vx.**reference**(*reference*)
Cast the object *reference* into a "vx_reference" object.

pyvx.types.**border_mode_t**(*mode*, *constant_value=0*)
Allocates and returns a vx_border_mode_t struct.

pyvx.types.**coordinates2d_t**(*x*, *y*)
Allocates and returns a vx_coordinates2d_t struct.

pyvx.types.**coordinates3d_t**(*x*, *y*, *z*)
Allocates and returns a vx_coordinates3d_t struct.

pyvx.types.**delta_rectangle_t**(*delta_start_x*, *delta_start_y*, *delta_end_x*, *delta_end_y*)
Allocates and returns a vx_delta_rectangle_t struct.

pyvx.types.**imagepatch_addressing_t**(*dim_x=0*, *dim_y=0*, *stride_x=0*, *stride_y=0*, *scale_x=0*, *scale_y=0*, *step_x=0*, *step_y=0*)
Allocates and returns a vx_imagepatch_addressing_t struct.

pyvx.types.**kernel_info_t**(*enumeration*, *name*)
Allocates and returns a vx_kernel_info_t struct.

pyvx.types.**keypoint_t**(*x*, *y*, *strength*, *scale*, *orientation*, *tracking_status*, *error*)
Allocates and returns a vx_keypoint_t struct.

pyvx.types.**perf_t**(*tmp=0*, *beg=0*, *end=0*, *sum=0*, *avg=0*, *min=0*, *num=0*, *max=0*)
Allocates and returns a vx_perf_t struct.

pyvx.types.**rectangle_t**(*start_x*, *start_y*, *end_x*, *end_y*)
Allocates and returns a vx_rectangle_t struct.

3.3 pyvx.vxu — C-like Python API

All the *vxuXxx* functions of the OpenVX standard are availible as *vxu.Xxx*. For example,

```
c = vx.CreateContext()
img = vx.CreateImage(c, 640, 480, vx.DF_IMAGE_U8)
dx = vx.CreateImage(c, 640, 480, vx.DF_IMAGE_S16)
dy = vx.CreateImage(c, 640, 480, vx.DF_IMAGE_S16)
assert vxu.Sobel3x3(c, img, dx, dy) == vx.SUCCESS
vx.ReleaseContext(c)
```

CHAPTER 4

Comments and bugs

There is a mailing list for general discussions and an issue tracker for reporting bugs and a continuous integration service that's running tests.

p

B

C

D

F

I

K

P

R

U