Motion Detection OpenVX* Sample

Developer Guide

Intel Computer Vision SDK - Samples

Contents

Legal Information

Introduction

This sample teaches how to use OpenVX* to develop Motion detection application. It focuses on the particular technique. The implementation illustrated in this document is required by the customer. Specifically, it implements a simplified motion detection algorithm based on Background Subtraction MOG2, dilate, erode and connected component labeling.

If you need a detailed step-by-step introduction to the *basics* of OpenVX* development, see Auto Contrast sample, available in this SDK `(<SDK_ROOT>/samples/auto_contrast).`

The following advanced OpenVX* topics are also covered in other samples:

- Expressing user-defined logic as a part of a graph via **user nodes.** `(<SDK_ROOT>/samples/census_transform).`
- **Virtual and non-virtual data** objects and their role in the graph. `(<SDK_ROOT>/samples/video_stabilization).`
- Using of Intel experimental feature called **Targets API** for heterogeneous computing. `(<SDK_ROOT>/samples/hetero_basic` and `video_stabilization` for common infrastructure reused in theis sample`).`
- Obtaining and interpreting OpenVX* **performance information.** `(<SDK_ROOT>/samples/video_stabilization` and `lane_detection).`
- Basic **interoperability with OpenCV** through data sharing. OpenCV is used for reading the data from an image file, and writing result to another image file.

  `(<SDK_ROOT>/samples/video_stabilization).`

Brief Introduction to OpenVX*

OpenVX* is a new standard from Khronos*, offering a set of optimized primitives low-level image processing and computer visions primitives. OpenVX* is a specification across multiple vendors and platforms. Relatively high abstraction of OpenVX* notions of resources and execution enables hardware vendors to optimize implementation with a strong focus on a particular platform.

Computer vision algorithms are commonly expressed using dataflow graphs. OpenVX* also structures *nodes* (functions with *parameters*) and data dependencies in directed acyclic *graphs*. Any graph must be verified by the OpenVX* runtime before execution. The same graph can be executed multiple times, with different data inputs.

Motion detection Algorithm as an OpenVX* Graph

The graph consists of Image scaling (optional), Background subtraction MOG2, Dilate, Erode, Dilate and Connected component labeling nodes. The graph consumes an image with RGB format and produces a label image and an array of bounding rectangles of detected moving objects.

Image scaling is optional, it can scale down the size of input image to save computations for subsequent processing. It can greatly improve performance without obvious loss in accuracy.

Background subtraction is a technique which extracts an image's foreground for further processing. It's widely used for motion detection in videos from static cameras. Mixture Of Gaussian method models each pixel as a mixture of Gaussians and uses an on-line approximation to update the model. The pixels which do not match to the model are called the foreground pixels. The Background subtraction MOG2 node takes a RGB (CPU and GPU) or grayscale (GPU) image as input, and generates a binary image, with foreground (detected moving objects) marked as white, and background marked as black. The GPU node can use both RGB and grayscale images as input. Compare with RGB, grayscale has better performance without obvious degradation in quality. When the node runs on GPU, the input can be a grayscale image.

The 3 morphology nodes (Dilate, Erode and Dilate) are required by customers. They are used for image de-noising: make foreground pixels connected, at the same time, suppress the noise to some extent. The input and output to these nodes are both binary images.

Connected component labeling is used to detect connected regions in binary images, it assigns a distinct label to each connected component. It's used here to group the foreground pixels. The Connected component labeling node includes a size filter which filters out the noise components with component size as the threshold. The bounding rectangles of detected moving objects are derived from the label image and returned. Connect component labeling is a regular algorithm in motion detection, after background subtraction step. OpenCV may have functions which have similar functionality. This algorithm is adopted as the code is simple, and it's easier for GPU implementation in the future.

The complete OpenVX* graph is presented in Figure 1. See `motion_detection.cpp` where the following code can be found:

- creation of `vx_context` and `vx_graph` instances;
- creation of all non-virtual data objects;
- the main loop to process an input video file, frame-by-frame;
- creation of all virtual data objects;
- population of the graph with nodes to build the pipeline presented on Figure 1.

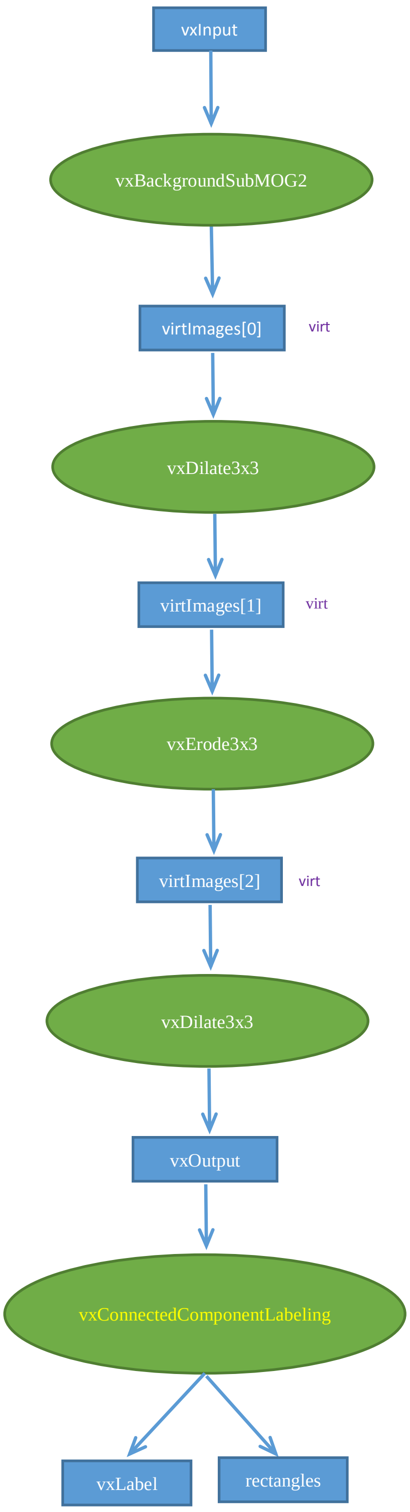

Figure 1: Complete OpenVX* graph implemented in the sample.
Legend: nodes are green, data objects are blue, "virt" indicate virtual objects.
Names for the data objects correspond to the names of variables in the source code.

Building the Sample

See the root README file for all samples and another README file located in video_stabilization sample directory for complete instructions about how to build the sample.

Running the Sample and Understanding the Output

The sample is a command line application. The behavior is controlled by providing command line parameters. To get the complete list of command line parameters, run:

```
$ ./motion_detection --help
```

The complete command line arguments are:

```
$ ./motion_detection --input inputFileName --output outputFileName --max-frames
max_frame_number --threshold thresholdValue --merge mergeFlag --scale scaleFlag
```

Here inputFileName specifies input file name, and outputFileName specifies output file name. max_frame_number specifies maximum number of frames to process, 0 is the default value which means all frames will be processed. Image width is denoted as imgWidth and image height is denoted as imgHeight. imgCount represents number of images for processing. threshold is the threshold used by size filter. mergeFlag is a flag used by post processing functions which can merge overlapping rectangles into one. scaleFlag is a flag to indicate if input image can be scaled down before processing. If this flag is enabled and width of input image is larger than 1280, then input image will be scaled down to save computation. The scale factor is 2 for both horizontal and vertical directions.

References

- OpenVX* 1.0.1 Specification
- Intel® Computer Vision SDK Developer Guide
- https://en.wikipedia.org/wiki/Background_subtraction
- https://en.wikipedia.org/wiki/Connected-component_labeling