# Computer Vision For Mobile and Embedded Platforms

Devin Homan

## problem

Mobile and embedded computing platforms are increasingly being used for computer vision tasks. Currently, these platforms either do all computation on the CPU or offload the processing to another computer. Present computer vision libraries, such as OpenCV, do not utilize the GPUs on these platforms. For GPU processing, OpenCV, requires that the GPU support CUDA or OpenCL. Mobile GPUs do not intrinsically support these languages. However, many mobile GPUs have built-in support for the C APIs, OpenGL ES and OpenVG. OpenGL ES is a 3D graphics engine, and OpenVG a 2D vector and raster graphics engine. OpenVG is easy to use and has built-in functionality for photo manipulation that makes it an ideal platform for creating a computer vision system.

Getting OpenVG and OpenGL ES to run, requires EGL. EGL creates the rendering environment. However, getting EGL setup and running correctly is non-trivial and device dependent, which is why CVPI also provides an interface to simplify this process.

## OpenVG

OpenVG provides methods for filtering image data.

**vgConvolve vgSeparableConvolve vgGaussianBlur**

$$I(x,y) = s\left(\sum_{i=0}^{w-1}\sum_{j=0}^{h-1} k(w-i-1,h-j-1)P(x+i-s_x,y+j-s_y)\right)+b$$

Convolution maps a kernel matrix, $k$, to image $P$ for every pixel, $(x,y)$, resulting in image $I$. Kernel and image indexing start from 0. $w$ and $h$ are the kernel's width and height. $s$, $s_x$, $s_y$, and $b$ are constants. Image channels are convolved independently.

**vgLookup vgLookupSingle**

These functions map image channel values using lookup tables. Each channel can hold an 8-bit value between 0 and 255. vgLookup uses a different lookup table, $f$, per channel. `vgLookupSingle` maps a single image channel, $C$, to a 32-bit, 4-channel value using a single lookup table. This is done for every pixel in the image.

$$\begin{pmatrix}R\\G\\B\\A\end{pmatrix}\begin{matrix}\xrightarrow{f_R}\\\xrightarrow{f_G}\\\xrightarrow{f_B}\\\xrightarrow{f_A}\end{matrix}\begin{pmatrix}R'\\G'\\B'\\A'\end{pmatrix}\qquad C\rightarrow\begin{pmatrix}R'\\G'\\B'\\A'\end{pmatrix}$$

**vgColorMatrix**

$$\begin{pmatrix}R'\\G'\\B'\\A'\end{pmatrix}\leftarrow\begin{pmatrix}1&5&9&13\\2&6&10&14\\3&7&11&15\\4&8&12&16\end{pmatrix}\begin{pmatrix}R\\G\\B\\A\end{pmatrix}+\begin{pmatrix}17\\18\\19\\20\end{pmatrix}\rightarrow\begin{pmatrix}1R+5G+9B+13A+17\\2R+6G+10B+14A+18\\3R+7G+11B+15A+19\\4R+8G+12B+16A+20\end{pmatrix}$$

`vgColorMatrix` applies a 4×4 matrix to each image pixel, multiplying the 4×4 matrix by the 4×1 pixel color vector. Another 4×1 vector is added to the product.

# CVPI Image Functions

## *OpenVG Function Wrappers*

**vgConvolveNormal vgConvolveNormalNoShift**
**vgConvolveNoShift vgColorMatrixNormal**

OpenVG's convolution and color matrix functions scale input pixel values to be between 0 and 1. These *Normal functions scale pixel values from 0 to 255. OpenVG's convolution operators shift pixel values towards the origin, *NoShift functions correct for the shift.

## *Image Processing Functions*

The choice of what functionality to implement was largely based on Robert, Perkins, Walker, and Wolfart's *Hypermedia Image Processing Reference*. Only a selection of functions that CVPI provides are listed.

### cvpi_image_add

$$C_{i,j} = s\cdot(a\cdot A_{i,j} + b\cdot B_{i,j}) + t$$

The function `cvpi_image_add` can add and subtract images, $A$ and $B$, of like dimensions resulting in image $C$. $s$, $a$, and $b$ are numeric constants, $t$ is a constant 4×1 vector. The function combines the two images in 1-pixel horizontal strips, uses `vgConvolve` to add adjacent rows, and then copies every other row into the return image. CVPI also has functions for adding and subtracting channels of the same image, utilizing `vgColorMatrix`.

### cvpi_yuyv2yuva

Many low-cost USB web cameras return data in YUYV format; that is, captured pixels are horizontally paired, sharing U and V channels while having their own Y channel. Each channel is 8-bits. `cvpi_yuyv2yuva` splits these combined pixels into separate YUVA pixels, with the A (alpha) channel set to 255. The YUVA image is twice the width but the same height as the YUYV input. The function utilizes `vgColorMatrix` for channel reordering.

### cvpi_image_magnitude

When two image convolutions are performed on the same image, such as a Sobel operation, the magnitude between the two resulting images is desirable. `cvpi_image_magnitude` calculates the magnitude value between pixel channel values in the same locations within the two images. The input data is scaled such that the sum of the squares of the data does not exceed 255. The data is then re-scaled after the square root is taken. Precision is decreased but data points can stay within the 8-bit color channels.

### cvpi_image_threshold cvpi_image_threshold_sector cvpi_image_threshold_adaptive_mean

Thresholding is used to filter out information within a certain value range. `cvpi_image_threshold` allows the user to specify two range values and whether to keep or remove information within the range, while doing the opposite for values outside the range. `cvpi_image_threshold_sector` partitions the image and finds a statistic for the partition, such as the mean, using a user supplied function. `cvpi_image_threshold_adaptive_mean` uses `vgConvolve` to find the local mean for each pixel. The resulting image is then subtracted from the original.

### cvpi_image_logical_*

CVPI can perform any logical operation between two images: AND, OR, NOR, XOR, XNOR, NAND, Complement, Inverse Complement. Channel values are treated as binary, depending on the function's parameters, 0 could be treated as false and non-zero treated as true. True values in one image are set to 1, and they are set to 2 in the other image. False values are set to 0. The two images are added. The resulting image has channel pixel values of 0, 1, 2, and 3. A lookup table is used to perform the particular logical operation. True and false values are mapped to user defined output values.

## *Morphology*

Image morphology works similarly to the logical operations; channel values are treated as true or false. CVPI has functions for dilating and eroding images; increasing and decreasing regions with true values. `cvpi_image_dialate` and `cvpi_image_thicken` increase true regions, while `cvpi_image_erode` and `cvpi_image_thin` decrease true regions.

### cvpi_image_histogram_equalization

Histogram equalization is used for image contrast adjustment. CVPI creates a per-channel histogram table and then a cumulative distribution table from the histogram table. The table is passed to `vgLookup` to change the channel values, image wide.

### cvpi_image_coordinate_table

Once the data has been filtered, the data coordinates might be desired for CPU processing, such as for a regression. `cvpi_image_coordinate_table` returns an array of coordinate pairs for the locations of non-zero values.

### cvpi_avuy2argb

There are no programs available for viewing YUVA; so the data is transformed to red, green, and blue. CVPI can create a BMP (Bitmap) header for ARGB. Bitmap is a widely supported image header specification.

## EGL Interface


CVPI provides a template interface for setting up and taking down EGL.

**cvpi_egl_settings_create**
**cvpi_egl_settings_check**

`cvpi_egl_settings_create` creates a structure of settings information that can be passed to `cvpi_egl_settings_create` or to `cvpi_egl_settings_check`. `cvpi_egl_settings_check` is used to check for faulty settings in the structure. Documentation and error reporting is lacking on Raspberry Pi's EGL implementation.

**cvpi_egl_instance_setup**
**cvpi_egl_instance_takedown**

`cvpi_egl_instance_setup` creates a `cvpi_egl_instance` structure that can be passed to `cvpi_egl_instance_takedown`. The structure gives the take-down procedure the information needed to undo the setup procedure. The `cvpi_egl_instance_setup` template performs a set of steps common to setting up EGL. EGL API implementation specific steps are included using conditional macros. However, creating the rendering window requires calling device-specific routines that cannot be accounted for. These routines return data pointers that EGL interprets as `EGLNativePixmapType` or `EGLNativeWindowType`. The EGL interface requires that the user pass in function pointers that, when executed, create and destroy the native type, respectively for setup and take-down. The Raspberry Pi has multiple ways of creating `EGLNativeWindowType` and `EGLNativePixmapType`. Raspberry Pi's function documentation is sparse, and the EGL standard does not specify native type creation methods. CVPI provides example code for `EGLNativePixmapType`.

# Interactive Demo

The computer running is a stock Raspberry Pi model B. It is using a web-camera to capture video. The program interfaces with the camera using Video4Linux. The frames are then processed by CVPI. CVPI is performing simple motion detection by subtracting the newest frame with the one previous, and using the result to mask the current frame. Unchanged channel pixels are set to 0. Changed channel pixels are set to the current value, so stationary objects are removed. CVPI converts the image to RGBA and adds a Bitmap header. The program used to display the frames, `feh`, cannot read Bitmaps, so ImageMagick `convert` is used to convert frames to JPEG. For each cycle, the program retrieves 51 frames from the camera, performs the motion detection (resulting in 50 images), saves the images, converts the saved images, and displays them at 10 frames per second.

## CONTACT

Devin Homan
devinwh7@gmail.com
907-862-0777
github.com/dwhoman/CVPI

## references

Khronos Group Inc. OpenVG Specification: Version 1.1, December 2008. www.khronos.org/registry/vg/specs/openvg-1.1.pdf.
Khronos Group Inc. Khronos Native Platform Graphics Interface: EGL Version 1.4, December 2013. www.khronos.org/registry/egl/specs/eglspec.1.4.pdf.
Robert, Perkins, Walker, and Wolfart. Hypermedia image processing reference, 2000.homepages.inf.ed.ac.uk/rbf/HIPR2/wksheets.htm.
Raspberry pi videocore apis, December 2013. elinux.org/Raspberry_Pi_VideoCore_APIs.
Schimek, Dirks, and Verkuil. Video for linux two api specification, 2006. videotechnology.com/dwg/v4l2.html.
R. Szeliski. Computer Vision: Algorithms and Applications. Springer, London, 2011.
Wikipedia. Histogram equalization — Wikipedia, the free encyclopedia, 2015. [Online; accessed 01-April-2015].
Wikipedia. Sobel operator — Wikipedia, the free encyclopedia, 2015. [Online; accessed 01-April-2015].
Converting between yuv and rgb, April 2010. msdn.microsoft.com/en-us/library/aa917087.aspx.