Computer Vision in the Functional Paradigm

Capstone Design
Gabriel Thomas Wylie Marques
Rutgers, School of Engineering
gmarques@eden.rutgers.edu

Contents

Abstract

It is highly unlikely than an exhaustive unified architecture for robotics and computer vision will exist in a single computer language. The difficulty of the problems in these fields requires a breadth of mathematics and algorithms to solve. While the mathematics applied is diverse, covering from fields such as differential geometry and topology to probabilities and stochastic processes, the implementation primarily remains in low level languages. Therefore, we investigate paths around this limitation through the development of a prototype augmented reality program utilizing the higher language of the functional paradigm, incorporating elements of low-level imperative language C.

1 The Growing Trend of Machine Learning

There has been general trend in robotics and computer vision toward increasing development in the field of machine learning. Machine learning has been seen emerging in many different applied fields. Facial recognition is one of the most successful applications of image analysis and machine learning. Most consumer cameras have facial detection features as to better find and focus on human faces. Further application allows the immediate removal of red-eye from the images. Facial recognition has prevalent usage in the field of law enforcement where it can be used for subject tracking and crime prevention. [11]

The advances in facial recognition corresponded to the advances in the design of classifiers for facial features. There are various categories of algorithms used for solving this problem. Holistic techniques utilize the whole face as an input, where are feature-based techniques look for structure from facial features, there is also a hybrid of both. Examples of the holistic technique include eigenface and support vector machines. Examples of feature-based methods included Hidden Markov models and Convolution Neural Networks.

The advancement of algorithms in the field of machine learning is being driving by the increasing trend of applying mathematical programming to the problem. MP being a field of mathematical optimization rather than an actual programming discipline. The intersection of ML and MP will usually find researchers developing solutions to very specific problems in ML using the general models of MP. The MP side being concerted with highly accurate solutions across a wide array of problems, where as ML seeks good solutions to a narrow array of problems. [1].

2 Introduction

2.1 Motivation for the Use of Haskell

Haskell is a lazy functional programming language. This means its objects are mathematical functions and being lazy essentially means the program only evaluates an expression when needed. As a programming language Haskell is an academic playground that allows for incredible amounts of experimentation.

The value in the lazy functional style of programming Haskell provides is in its use to solve difficult problems. There are papers discussing the paradigms of functional programing where it has been used to model quantum systems, and circuits, with stunningly short code. Providing a better understanding of how a quantum computer may work, and maybe one far off day provide a basis for their programming. [7]

We could go on for quite a while, but to shorten the read on why we chose Haskell, we itemize the following points.

- Template Haskell allows for meta-programming, programs that write other programs (or themselves).
- In Haskell functions can take functions that take functions and so on (higher order functions).
- Haskell can be used to perform symbolic computations, theorem proving.
- Haskell can implement lazy non-deterministic algorithms.
- The code is easily vectorized, and in some cases automatically.
- Haskell can be written to function in any computational structure one would like. Such as reactive programming, i.e. signal processors or data-flow programs.

3 Fusing Languages

3.1 Minimal Working Base

The most arduous obstacle in the implementation of this project was the bindings to the OpenCV libraries which are written in C. The minimal working base consists of a sampling of important vision functions implemented from the libraries: cv.h, highgui.h, and our own C written implementations for speed. This will be the backbone of the application, communication between C and Haskell.

The general methodology to deal with a C-function that allocates some data and returns it as a pointer to that memory on the heap has been outlined [6]. In the case that the foreign object returned by the function is a standard C type, the Haskell FFI contains the correct types to handle it, and GHC's garbage collector can deal with it. When the foreign object turns out to be some unfamiliar struct, we wrap the returned Ptr into a newForeignPtr, using a FinalizerPtr as a function call to the Garbage collector on the C heap. This effectively acts as a type of smart pointer, telling GHC's garbage collect to not worry about it as it is handled by the C heap.

However the evaluation of the runtime code resulted in massive memory leaks. This was essentially determined to be caused by use of the ForeignPtr type, which the FFI addendum explicitly states is not guaranteed to call the finalizers to release the memory. As a result the accumulation of allocated IplImages from the camera occurs.

As a result of this development the functions were unwrapped and phantom data types were used with base Haskell Ptrs, essentially forming the equivalent of the C void* pointer. The results leaves naked pointers to the C-Heap floating around the haskell code. Yet despite this step memory leaks still occured. Further analysis revealed the C portion of the code to be leaking, for some odd reason to release a pointer passed to it by Haskell a C-function needs the following structure:

```
void free_memory (object* mem) {
  object *ref = mem;
  free( ref );
}
```

To possibly remedy this problem, it was considered creating a Haskell equivalent data-type for IplImage and CvPoint. The reasoning behind this methodology is that the datatype would be able to be allocated on the Haskell side, marshaled to the c-function as a pointer of the corresponding type, and modified safely. This would release the problems associated with the pointers to memory in the C-Heap being handled by one of GHC's four garbage collectors. The C2HS interfacing tool, is useful for automating the rather mundane bits of writing a FFI [3]. However despite C2HS being able to determine the appropriate values for alignment and size, the internal structure of the IplImage type is more complex than the interfacing tool is able to handle. As such it was abandoned for the IplImage type, but provided an working CvPoint type, which is used in the implementation of the marker detection in the next section to marshal back the points found by it. The CvPoint type conversion created in C2HS is below. It allows us to retrieve but not set the elements of the foreign struct CvPoint.

```
data CvPoint = CvPoint {x :: Int, y :: Int} deriving (Eq, Show)

instance Storable CvPoint where
  sizeOf _ = {# sizeof CvPoint #}
  alignment _ = alignment (undefined :: CInt)
  peek p = CvPoint
    <$> fmap fromIntegral ({#get CvPoint->x #} p)
    <*> fmap fromIntegral ({#get CvPoint->y #} p)
```

4 Augmented Reality

The following subsections layout the process describing the realization for a prototype augmented reality program, built from a mix of Haskell and the C vision library, OpenCV.

4.1 Camera Calibration

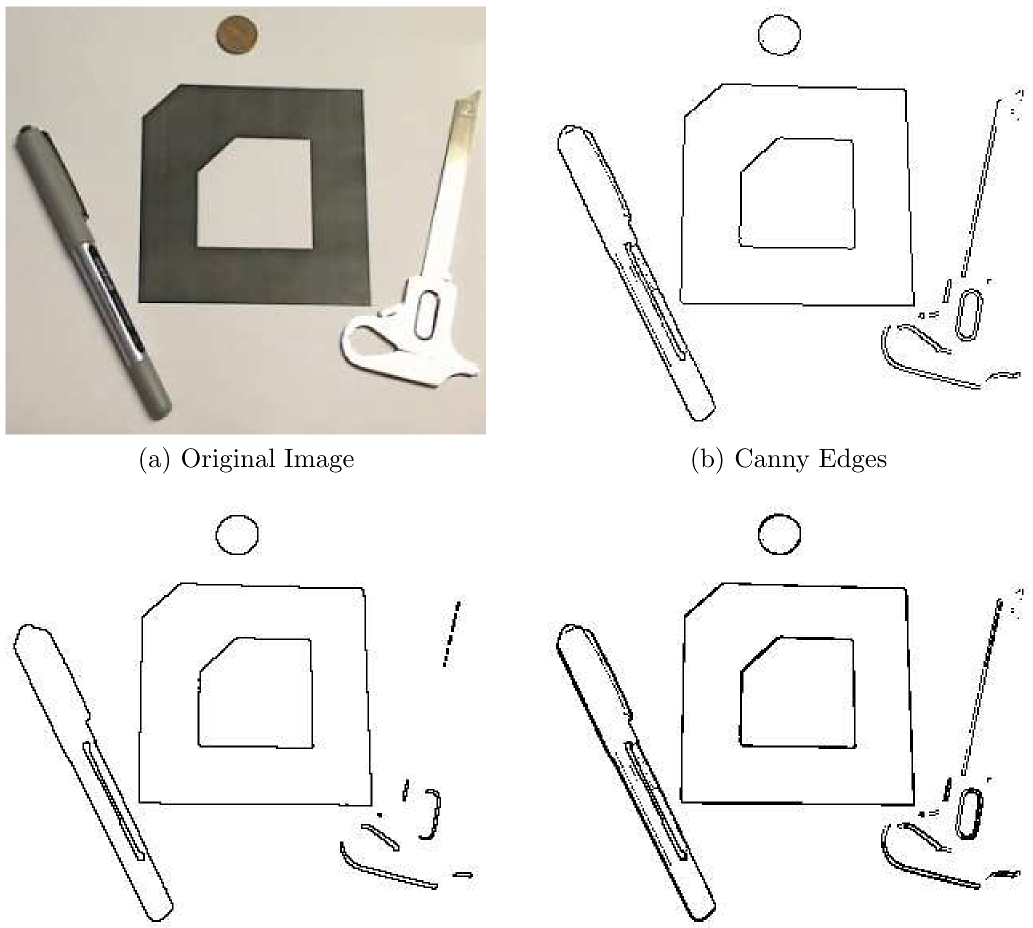
To properly determine important aspects of the vision pipeline, allowing us to determine where in the world the camera is, we first require the intrinsic matrix as it holds the fixed internal camera parameters. The camera utilized for this project was the Logitech Quickcam Pro 9000. The intrinsic parameters were first calculated using 50 poses of a 9 x 6 chessboard grid at a resolution of 640 by 480 pixels per a frame, from a program acquired through standard material [2]. The program stored the camera parameters and returned the root mean square of the re-projection error with a value of 42.352. The following intrinsic matrix was acquired:

$$\begin{bmatrix} 5.25405029e+02 & 0.0 & 3.07575745e+02 \\ 0.0 & 5.25583801e+02 & 2.22051102e+02 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The values in the above recovered intrinsic matrix were stored in the transformation module as a constant matrix for use by the homography decomposition algorithm and OpenGL projection matrix.

4.2 Marker Detection

4.2.1 Edge Detection



(a) Original Image

(b) Canny Edges

(c) Morph. Edges

(d) Bitwise OR

Figure 1: The original image (a) with the three edge extraction methods, all edge images have been negated to provide better viewing under printing.

To find a marker located in an image we must first find the edges of objects in that image. Three methods of edge detection were ultimately chosen for comparison, canny edges, a morphological gradient, and the bitwise or of the two methods. Figure 1 displays the original image with the results from the three methods. Figure 1 (b) displays the canonical method of edge detection, canny edges. As compared to the other two methods canny produces thinner edges more reliably under parameters such as color difference. This can be observed in the caliper present in the image. Due to the high sheen of its surface the white level is comparable to that of the surrounding paper. The morphological operation in sub-figure (c) is insensitive to the slight variation in color and does not pick up the same edge that canny does. This discrepancy is due to the thinness of the dark edge produced by the shadow, which is thin enough to be consumed by the erode operation in the morphological gradient. However, this erosion - dilation difference that the morphological gradient produces is effective at filtering out the type of edge noise that is present in the canny image. The final method in Figure 1 (d) is the bit-wise or operation between the two methods. This was proposed to work on the principle that one method may work when the other may not, the method was successful in that it provided better over all detection of the edges, however in doing so the or method introduced the noise present in the canny method. The best outcome was determined to be that of the morphological gradient as the other methods turned out be very sensitive to movement of the image.

```
IplConvKernel* SE = cvCreateStructuringElementEx(2,
                    2, 1, 1, CV_SHAPE_RECT, NULL);
cvSmooth(Gray, Morph, CV_GAUSSIAN, 3, 0, 0, 0);
cvThreshold(Morph, Morph, 0, 255,
            CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
cvMorphologyEx(Morph, Morph, T, SE, CV_MOP_GRADIENT, 1);
```

4.2.2 Finding Contours

We try to find structure from the edges created by the previous subsection utilizing the OpenCV function cvFindContours, which finds all contours from the given binary input image. The next step is the approximation of the contours into polygonal curves by the cvApproxPoly function. From the the space of all polygonal curves we imposed constraints to select the appropriate object in the scene. This was realized by selecting polygonal curves consisting of only four segments and were convex.

This filters the polygonal space to only four sided objects. We further reduce the polygonal space by utilizing an iterative technique to "bubble" up the polygonal curve in the space that has the largest area and average line segment length. When we say "bubble" we mean that the algorithm iterates the entire space once, only accepting a polygonal curve if its values for area and average line segment length are greater that the current biggest, as such the largest value floats to the top.

There is one additional flag to determine if the selected polygonal curve is the object we desire. The true or false "Found It" flag, if we have found the object then we accept no more candidates for the solution and the loop runs out. To determine if the curve qualifies as "Found It" and the marker is indeed found, the polygonal curve is convex hulled. The four points of this hull correspond to the four points of a possible marker. One distinct property of the marker is one of the corners of the image is filled, where as the others are not.

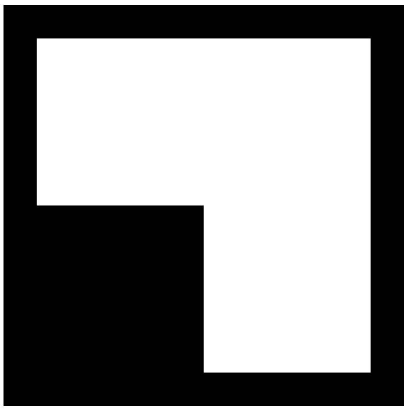
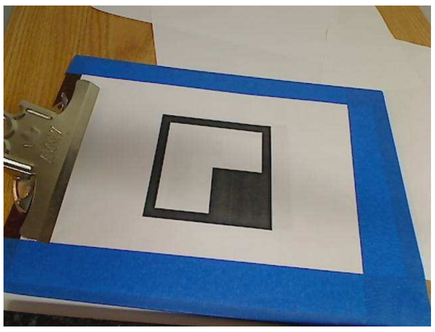
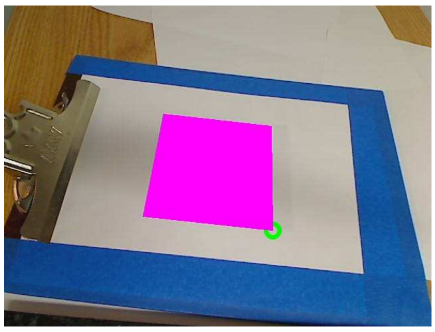


Figure 2: The marker used in this detection scheme.

Using the corner points from the convex hull we compute the midpoint of each of the four line segments and use them to split the object up into four smaller pieces, each of which is used to draw a white polygon using cvDrawConvexPoly onto a black image. This creates a mask by which to copy the local portion of the inverse binary thresholded image. From each masked image, we count the number of non zero elements in found it it using cvCountNonZero. If the amount pixels for a certain corner is greater the a constant threshold and if that corner is greater than the other three corners, we have found out marker. The result of the marker detection algorithm is four points which are used to compute the homography in the next section.



(a) What the camera sees



(b) What the detector finds

Figure 3: Subfigure (a) shows the actual image retreived by the camera before detection. Subfigure (b) displays the four corners found by with the first corner, marked by the green circle, in the list corresponding the the dark quadrant of the marker.

4.2.3 Error Analysis of Marker Detection

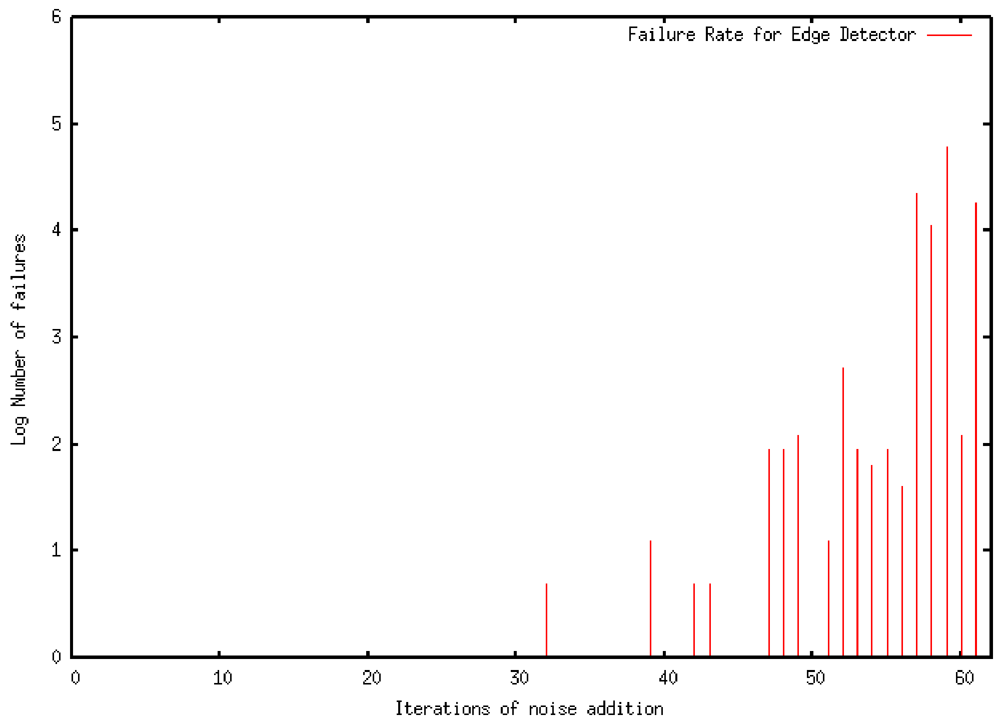


Figure 4: The logarithm of the number of consecutive failure the edge detection method had as a result of smoothed pixel noise

To test the failure rate of the edge detector we introduce sparse uniform pixel noise in the incoming image and then smooth it with a Gaussian blur. To increase the amount of noise the number of times the image is run through the noise addition loop is incremented. If the marker detector fails to find its target then the program grabs another image and runs it through the noise addition loop with the same number of iterations that previously failed. The function

4.3 Homography

In two view geometry, when we have correspondences between two images, we can compute the essential matrix, that relates the views for the case when there is general structure to the scene. When we know our points have planar structure, that is the points in both images lie in the same plane, it is possible to compute the homography. The homography provides a bijective linear map $\mathbf{H}_{3\times 3} : \mathcal{R}^2 \to \mathcal{R}^2$ between planes in $\mathcal{R}^3$. The homography matrix contains nine degrees of freedom, however the ninth can be excluded because the transformation is only defined up to some scale.

When we consider that each point in a projective plane has two degrees of freedom, in both the x and y dimensions, the homography then only requires 4 points to perfectly constrain. Other cases that provide more points for estimation, lend to over-constrained homographies. These situations require optimization algorithms to compute the best transformation of a given set of points, by minimizing some "cost" function, a geometric or algebraic metric by which to compute the error. Since our homography is perfectly constrained there is no error, because the transformation for the points will be exact.

4.3.1 Estimation of the Homography Matrix

From the array of four points returned by the marker detection algorithm, we compute the homography. The algorithm used to compute the homography is known as the normalized Direct Linear Transformation and is given in [5]. As we seek to compute $\mathbf{x_i'} = \mathbf{Hx_i}$ it is useful to realize $\mathbf{x_i'}$ is parallel with $\mathbf{Hx_i}$. From this we have $\mathbf{x_i'} \times \mathbf{Hx_i} = \mathbf{0}$. Taking the matrix form of the cross product yields the skew symmetric matrix:

$$\begin{bmatrix} 0^T & -w_i' x_i^T & y_i' x_i^T \\ w_i' x_i^T & 0^T & -x_i' x_i^T \\ -y_i' x_i^T & x_i' x_i^T & 0^T \end{bmatrix}$$

The normalized direct linear transformation algorithm was implemented purely in Haskell.

```
nDlt :: [(Double, Double)] → [(Double, Double)] → Matrix Double
nDlt [] _ = ident 3
nDlt _ [] = ident 3
nDlt xs ys | length xs /= length ys = ident 3
           | otherwise = (inv t') <> mat <> t
  where mat = scale (-1) $ ((reshape 3) ∘ nullVector ∘ fromBlocks)
                         $ zipWith dRows nxs nys
        (t, nxs)  = normPnts xs
        (t', nys) = normPnts ys
```

The dRows function constructs the rows of the matrix for SVD.

```
dRows :: Matrix Double → Matrix Double → [Matrix Double]
dRows u' u =
  [fromBlocks [[          v0, scale (-w) v',    scale y v']
              ,[   scale w v',           v0, scale (-x) v']
              ,[scale (-y) v',    scale x v',            v0]]]
  where (!v', !v)    = (trans u', trans u)
        (!x, !y, !w) = (u @@> (0,0), u @@> (1,0), u @@> (2,0))
        v0           = (1><3) [0,0,0]
```

The centroid function computes the center of a list of points.

```
centroid :: [(Double, Double)] → (Double, Double)
centroid [] = (0, 0)
centroid xs = (cx / n, cy / n)
  where (cx, cy) = foldl' addTup (0, 0) xs
        n = fromIntegral $ length xs
        addTup (x, y) (x', y') = (x + x', y + y')
```

The shiftC function shifts a list of points to a new center.

```
shiftC :: [(Double, Double)] → (Double, Double) → [(Double, Double)]
shiftC [] _ = []
shiftC xs (x, y) = map (move (x, y)) xs
  where move (cx, cy) (u, v) = (u - cx, v - cy)
```

The normPnts function normalizes a list of points, and provides the similarity transform.

```
normPnts :: [(Double, Double)] → (Matrix Double, [Matrix Double])
normPnts [] = (ident 3, [])
normPnts xs = (simT, (map (λv → simT <> v) vs))
  where (cx, cy) = centroid xs
        ns       = shiftC xs (cx, cy)
        vs       = map (λ(x, y) → (3><1) [x, y, 1]) xs
        s        = let acc = sum $ map (λ(x, y) → sqrt $ x^2 + y^2) ns
                       n   = fromIntegral $ length ns
                   in (sqrt 2) / (acc / n)
        simT     = (3><3) [ s,  0,  (-s)*cx
                          , 0,  s,  (-s)*cy
                          , 0,  0,        1 ]
```

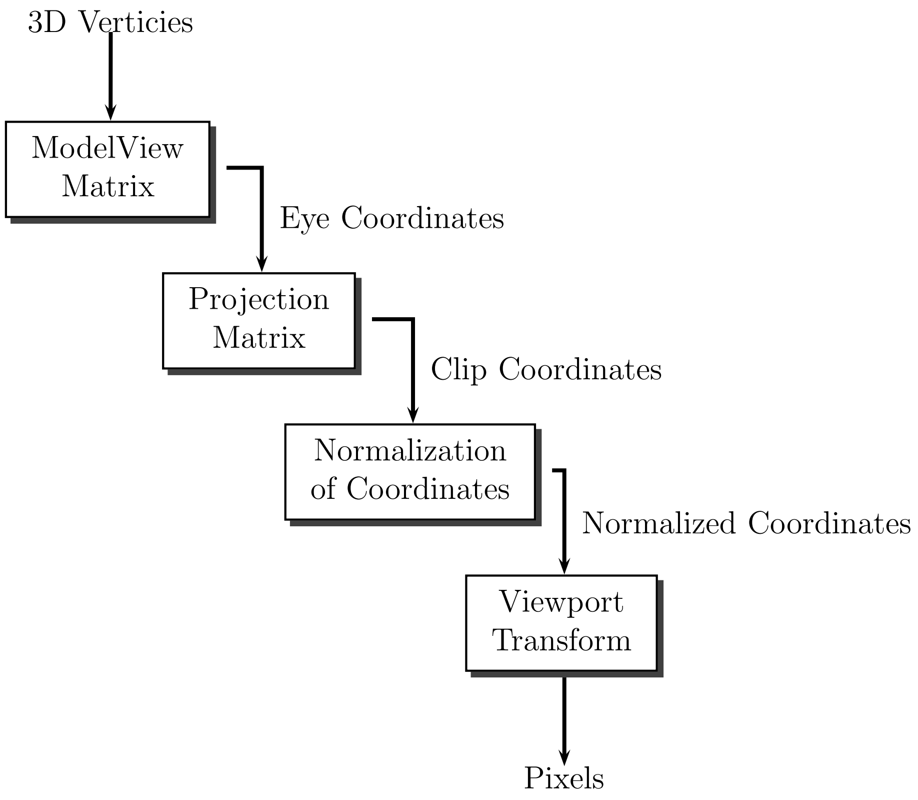
4.3.2 Homography Decomposition

The homography matrix is decomposed into the extrinsic parameters with the help of the intrinc parameters. The rotation matrix and translation vector extracted here will be used to create the OpenGL modelview matrix.

```
svdDecomHom :: Matrix Double → (Matrix Double, Matrix Double)
svdDecomHom hom | l == 0 = (ident 3, ((3><1) [0, 0, 0]))
                | otherwise = (rOrth, tVec)
  where [h1,h2,h3] = toColumns hom
        l          = sqrt $ (sum ∘ toList ∘ (zipVectorWith (*) h1)) h1
        r1         = cInv <> h1
        r2         = cInv <> h2
        r3         = (zipVectorWith (*) r1 r2) - (zipVectorWith (*) r2 r1)
        tVec       = fromColumns $ [cInv <> h3]
        rm         = fromColumns [r1, r2, r3]
        (u, _, v)  = fullSVD rm
        rOrth      = u <> (trans v)
        cInv       = inv intrin
```

4.4 OpenGL Rendering

The OpenGL graphics pipeline, used to transform 3D vertices into on screen pixels, is as follows:



Rendering in OpenGL was accomplished utilizing the Haskell OpenGLRaw bindings package. Some difficulties arose when attempting to find how to match the OpenGL camera orientation to that of the actual camera. The trick to properly modeling the camera in OpenGL is to realize that the camera model used in computer vision is facing the +Z direction with the Y+ axis facing down. OpenGL's camera on the other hand faces in the -Z direction with the +Y axis facing up. The way OpenGL mimics the intrinsic parameters of a camera is through a Projection matrix as and the viewport transformation as found in [8]. Finally, when directly loading a matrix from memory into OpenGL using the glLoadMatrix function, one has to remember that OpenGL performs right multiplication, equivalent to transposing the matrix.

To generate the modelview matrix, the rotation and translation of the camera, for OpenGL to use we use the following function. Notice it flips the X and Y translations, this is needed to make our OpenGL camera agree with the real camera.

```
modelMatrix :: (Matrix Double, Matrix Double) → [Double]
modelMatrix (rot, tvec) = out
  where lst = toRows $ fromBlocks [[rot, nTv]]
        adV = ({-trans ◦ -}fromRows) (lst ++ [z])
        out = concat $ map toList $ toColumns (m <> adV)
        z = 4 |> [0, 0, 0, 1]
        nTv = (3><1) [(-(tvec @@> (0,0)))
                     , (-(tvec @@> (1,0)))
                     , ((tvec @@> (2,0)))]
```

The createGLMatrix, modelPipe, and computeHomography functions work as follows. The function createGLMatrix is given a list of 16 points from which to create a pointer to the C allocated memory to return.

```
createGLMatrix :: [Double] → IO (Ptr GLdouble)
createGLMatrix elements = newArray (map realToFrac elements)
```

The function modelPipe is actually just the function composition ($f \circ g$) of two of our previous functions.

```
modelPipe :: Matrix Double → [Double]
modelPipe = modelMatrix ◦ svdDecomHom
```

The function computeHomography is passed a list of tuples, to which it applies the normalized DLT algorithm with a list of internally stored marker reference points, returning the matrix found.

```
computeHomography :: [(Float, Float)] → Matrix Double
computeHomography [] = ident 3
computeHomography xs | elem (0,0) xs = ident 3
                     | otherwise = nDlt world image
  where
    image = map (λ(x,y) → (realToFrac x, realToFrac y)) xs
    world = [(0,0),(0,1),(1,1),(1,0)]
```

The call that renders the object consists of the setting the intrinsic parameters with glViewport and gluPerspective. The call to gluPerspective generates the proper projection matrix given the parameters for field of view, aspect ration, the z near, and the z far. The composition of modelPipe and computeHomography is applied to yield the modelView matrix ready for OpenGL. The last transformation is rotating around the Y-axis by one half turn to match the direction the GL camera views with the direction the actual camera views as in the code below.

```
glViewport vx vx vw vh
glMatrixMode gl_PROJECTION
glLoadIdentity
```

```
gluPerspective pfovy paspect pnear pfar

glMatrixMode gl_MODELVIEW
glLoadIdentity
model ← createGLMatrix $ (modelPipe ∘ computeHomography) pnts
glLoadMatrixd model

glRotatef 180 0 1 0
glScalef 0.01 0.01 0.01
```

4.4.1 Rendering Video into OpenGL

To complete the augmented reality experience, we have to actually make it display over something, the real world. This is a lot easier than it sounds, as OpenGL is directly compatible with the way OpenCV stores the image data. The first step is the creation of a partial application of a function. The queryFrame function in OpenCV only takes one argument, the pointer to the allocated CvCapture object. Here queryFrame takes two, it was redesigned to take the pointer to previous frame for deletion. Not necessary when using the C function in a C loop, however it is necessary when working between these two languages. The next important action in the following code is the modification of the IORef that contains the pointer to the grabbed Image, IORefs are mutable objects in Haskell. The modifyIORef takes a function to modify the value it currently stores, here we update it with a new Ptr to a C allocated image.

```
  let partialFrame x = CV.queryFrame x cam
  glClear $ fromIntegral  $  gl_COLOR_BUFFER_BIT
  modifyIORef imgRef $! (unsafePerformIO ∘ partialFrame)
```

Next we grab the pointer to the image from the IORef. The function flipIm is exactly cvFlip, since the data that comes in from OpenGL will be of opposite orientation, it is necessary to flip the image first. The next relevant bit of information is the application of the function imgData, this returns the pointer to the IplImage imageData, exactly in the way it would be accessed in OpenCV. The OpenGL call glDrawPixels takes this pointer along with some basic parameters and draws it to screen.

```
  img ← readIORef imgRef
  CV.flipIm img img 0
  cop ← CV.cloneImage img
  pnts ← CV.markerDetect cop
```

```
dat ← CV.imgData img

glViewport vx vy vw vh

glMatrixMode gl_PROJECTION
glLoadIdentity
gluPerspective pfovy paspect pnear pfar

glMatrixMode gl_MODELVIEW
glLoadIdentity

glDrawPixels 640 480 gl_BGR gl_UNSIGNED_BYTE dat
```

4.4.2 Results

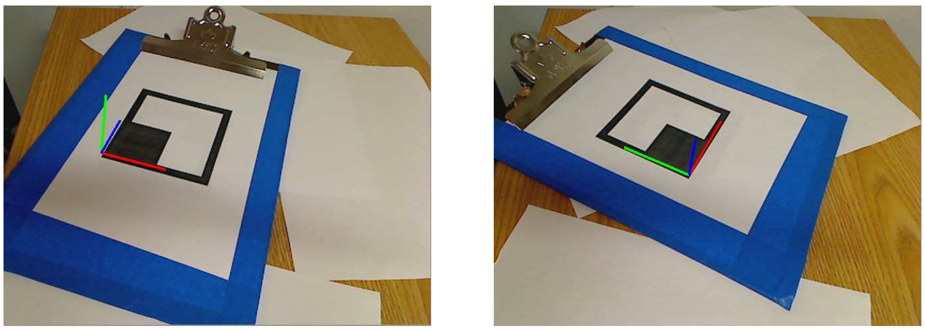

(a) View One (b) View Two

Figure 5: Two different views of the OpenGL rendered 3D coordinate axes.

5 Hand Detection

5.1 Skin Detection

The skin detection algorithm written entirely in C and was meant to be a means by which to manipulate objects rendered by the augmented reality program. It effectively uses bounds on two color spaces, RGB space and YCbCr colorspaces per pixel, and resizing before and after. This technique was laid out previously by Mahmoud in two papers [4] [10].

Given a pixel $P \in \mathcal{I}$, where $\mathcal{I}$ is our BGR image retrieved from camera, we seek to determine if P is representative of skin or not.

This requires the analysis of $\mathcal{I}$ in two colorspaces $\mathcal{I}_{\text{RGB}}, \mathcal{I}_{\text{YCrCb}}$. We also seek to mask the image so that only skin remains.

The image is first downscaled using cubic interpolation, then, by pixel per pixel iteration for any given Any pixel M^i_j in the binary mask image is created through a piecewise function of the pixels $R^i_j, G^i_j, B^i_j, Y^i_j, Cr^i_j, Cb^i_j$ from the colorspaces of the image I such that:

$$M^i_j = \begin{cases} 1: & R^i_j > 95 \ \wedge \ G^i_j > 40 \ \wedge \ B^i_j > 20 \ \wedge \ R^i_j - G^i_j > 15 \\ & \wedge \max R^i_j, G^i_j, B^i_j - \ \min R^i_j, G^i_j, B^i_j > 15 \\ & \wedge \ 135 > Cb^i_j > 85 \ \wedge \ 180 > Cr^i_j > 135 \\ & \wedge \ Y^i_j > 80 \ \wedge \ G^i_j < R^i_j \ \wedge \ B^i_j < R^i_j \\ 0: & \text{Otherwise} \end{cases}$$

The skin detection algorithm still leaves a lot of noise in the background, so we choose to use two morphological operations to eat it away. This involves resizing the imape up using cubic interpolation and then two iterations of a morphological opening and on iteration of morphological close are applied, followed by a gaussian smoothing.

```
cvMorphologyEx(N,N,0,0,CV_MOP_OPEN,2);
cvMorphologyEx(N,N,0,0,CV_MOP_CLOSE,1);
cvSmooth(N,N,CV_GAUSSIAN,3,0,0,0);
```

5.2 Contouring and Convexity Defects

To detect the actual open hand, we first find the contours in the image. The function cvApproxPoly is once again used to build polygonal curves from the contours. These polygonal curves are used to generate convex hulls of the objects in the scene. For each convex hull/curve pair we find the convexity defects. That is the points on the contour that are farthest away from the convex hull. On a open human hand this corresponds to the places in between fingers. The depths of the convexity defects, stored in the cvConvexityDefect structure, is used to compute the average depth of the deepest convexity defects for a found object. We seek the object with the largest average depth of the convexity defects along with the most area. The result is that open hands are found more likely than other objects in the scene.

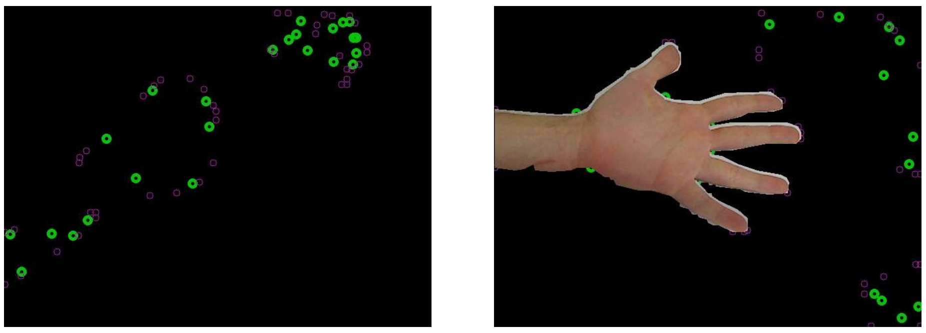

(a) Closed Fist (b) Open Hand

Figure 6: A closed fist (a) does not get detected while an open hand (b) does. The marks in the images that are not on the hands are from other hulls that satisfied the color requirements but not the geometrical ones.

6 Conclusion

Fusing the Languages of Haskell and C was arduous, It takes a large amount of skill to be able to appropriately handle, and avoiding memory leaks on the C side was a daunting task. Surprisingly Haskell had no problem running real-time code, indicating that the language would provide as suitable base for computation of vision algorithms. There were still problems with the numerical stability of the image detection. Filters could be applied to fix this. However more interesting is the thought of using Haskell to bring the rewrite all the linux code in a DSL, or domain specific language, that could be converted to a language like verilog. There are already package for Haskell that allow it to program FPGAs such as:

> Atom is a Haskell DSL for designing hard realtime embedded software. Based on guarded atomic actions (similar to STM), Atom enables highly concurrent programming without the need for mutex locking. In addition, Atom performs compile-time task scheduling and generates code with deterministic execution time and constant memory use, simplifying the process of timing verification and memory consumption in hard realtime applications. Without mutex locking and run-time task scheduling, Atom eliminates the need and overhead of RTOSs for many embedded applications.

There are still more examples such as LAVA, which provides formal system design and can compile to FPGA. [9]. An FPGA implementation of vision algorithms provides a powerful computational base. Languages like Verilog are reactive, meaning that instead of dealing with state, they have a continuous flow of data. Functional Reactive Programming (FRP) is an active field of research in Haskell, with applications to robotics and vision. The stream processing performed utilizing arrows is the core of FRP and is simultaneously used by the EDSL (Embedded Domain Specif Language) in Haskell that can run on FPGAs. Implementation of the augmented reality program in arrows not only optimizes the runtime of the code on a computer, but simultaneously eases the transition to FPGA.

References

[1] Kristin P. Bennett and Emilio Parrado-Hernández. The interplay of optimization and machine learning research. *J. Mach. Learn. Res.*, 7:1265–1281, December 2006.

[2] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning opencv, 1st edition.* O'Reilly Media, Inc., first edition, 2008.

[3] Manuel Chakravarty. C to haskell, or yet another interfacing tool. In Pieter Koopman and Chris Clack, editors, *Implementation of Functional Languages*, volume 1868 of *Lecture Notes in Computer Science*, pages 131–148. Springer Berlin / Heidelberg, 2000. 10.1007/107222988.

[4] Moheb Girgis, Tarek Mahmoud, and Tarek Abd-El-Hafeez. An approach to image extraction and accurate skin detection from web pages. *World Academy of Science, Engineering and Technology*, 27, 2007.

[5] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[6] Simon Peyton Jones. Tackling the awkward squad: monadic input/output... *Change*, (May):47–96, 2005.

[7] Jerzy Karczmarczuk. Structure and interpretation of quantum mechanics: a functional framework. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, Haskell '03, pages 50–61, New York, NY, USA, 2003. ACM.

[8] Ming Li. Correspondence analysis between the image formation pipelines of graphics and vision.

[9] Zhonghai Lu, Ingo Sander, and Axel Jantsch. A case study of hardware and software synthesis in forsyde. In *Proceedings of the 15th international symposium on System Synthesis*, ISSS '02, pages 86–91, New York, NY, USA, 2002. ACM.

[10] Tarek M. Mahmoud. A new fast skin color detection technique. *World Academy of Science, Engineering and Technology*, 43, 2008.

[11] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35:399–458, December 2003.

7 Code

7.1 ImgDecon Header

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>

CvPoint midpoint(CvPoint a, CvPoint b);

int imgWidth (IplImage* I);

int imgHeight (IplImage* I);

uchar* ImgMatrix (IplImage* Arr, int Chan);

void freeVector (uchar* Vec);

int imgDepth (IplImage* I);

IplImage *NewImage (int width, int height, int depth, int channels);

CvPoint2D32f* FindCorrespondences (IplImage* I, const int n);

void freePointArr (CvPoint2D32f* Arr);

void freeSeq (CvSeq* Seq);

void freeImg (IplImage* Img);

//void freeCapture (CvCapture* Cap);
//IplImage* qFrame(CvCapture* Capture);

uchar* ImgData (IplImage* I);

uchar* castCPtr (char* Ptr);

//void freeMarker (marker* M);
IplImage* releaseData (IplImage* I);
//marker* newMarker ();

IplImage* qFrame(IplImage* I ,CvCapture* Capture);
```

```
IplImage* frameGrab (CvCapture* Capture);

IplImage* morphOrCanny ( IplImage* I );

IplImage* cloneImage ( IplImage* I );

IplImage* edgeCanny ( IplImage* I );

IplImage* edgeMorph ( IplImage* I );

void markerDetect ( IplImage* I, CvPoint* points );

float glToFloat (float c);

float doubleToFloat (double c);

double floatToDouble (float c);
```

7.2 ImgDecon C File

```
#include "ImgDecon.h"

int imgWidth (IplImage* I) {
  return (I->width);
}

int imgHeight (IplImage* I) {
  return (I->height);
}

uchar* ImgMatrix (IplImage* Arr, int Chan) {
  assert( Chan < 3 && Chan >= 0 );
  int size = (imgWidth(Arr))*(imgHeight(Arr));
  uchar* data;
  data = (uchar*) malloc( sizeof(uchar)*(size+1) );

  int x;
  int y;
  int z = 0;
  for (x = 0; x < Arr->width; x++) {
    for (y = 0; y < Arr->height; y++) {
      *(data+z) = ((uchar*)(Arr->imageData + Arr->widthStep*y))[x*3+Chan];
      z++;
    }
  }
  return data;
}

uchar* ImgData (IplImage* I) {
 return I->imageData;
}

void freeVector (uchar* Vec) {
  uchar* Vector = Vec;
  free( &Vector );
}

int imgDepth (IplImage* I) {
 return (I->depth);
}

IplImage *NewImage (int width, int height
          , int depth, int channels) {
  IplImage *Im = cvCreateImage(cvSize(width,height)
        , depth, channels);
  return Im;
}

CvPoint2D32f* FindCorrespondences (IplImage* I
        , const int n) {
  int width  = I->width;
  int height = I->height;

  int nFeatures = n;

  CvSize size = cvSize(width, height);

  IplImage* Eigen = cvCreateImage(size, 8, 1);
  IplImage* Temp  = cvCreateImage(size, 8, 1);

  CvPoint2D32f* Features = (CvPoint2D32f*)
      malloc( sizeof(CvPoint2D32f)*n );

  cvGoodFeaturesToTrack(I, Eigen, Temp, Features,
      &nFeatures, .01, .01, NULL, 3, 0, 0.04);
  return Features;
}

void freePointArr (CvPoint2D32f* Arr) {
  free (Arr);
}

void freeSeq (CvSeq* Seq) {
  // Do absolutely nothing
}

void freeImg (IplImage* Img) {
  IplImage *Temp = Img;
  cvReleaseImage(&Temp);
}


IplImage* qFrame(IplImage* I ,CvCapture* Capture) {
  IplImage* Img = I;
  CvCapture* Cap = Capture;
  Img = cvQueryFrame(Cap);
  if (Img == 0)
    Img = qFrame(Img, Cap);
  return Img;
}

IplImage* frameGrab (CvCapture* Capture) {
  IplImage* I = 0;
  CvCapture* Cap = Capture;
  I = cvQueryFrame(Cap);
  // infinite recurse
  if (I == 0)
    I = frameGrab (Cap);
  return I;
}


uchar* castCPtr (char* Ptr) {
  return (uchar*)Ptr;
}

IplImage* releaseData (IplImage* I) {
  IplImage* Img = I;
  cvReleaseImage(&Img);
  IplImage* New = 0;
  return New;
}

//================================================
// Marker Detection Algorithm
//================================================

IplImage* cloneImage ( IplImage* I ) {
  if (I != 0){
    IplImage *New = cvCreateImage(cvSize(I->width, I->height)
        , IPL_DEPTH_8U, I->nChannels);
    cvCopy(I, New, NULL);
    return New;
  }
  else {
    IplImage *Def = cvLoadImage("Empty.jpg", 1);
    return Def;
  }
}

IplImage* edgeCanny ( IplImage* I ) {
  //if ( I != 0 ) {
    IplImage* Img = I;
    IplImage* Gray = cvCreateImage(cvSize(Img->width, Img->height)
                      , IPL_DEPTH_8U, 1);
    IplImage* Edges = cvCreateImage(cvSize(Gray->width, Gray->height)
                      , IPL_DEPTH_8U, 1);
    cvCvtColor(I, Gray, CV_BGR2GRAY);
    //I = cvCreateImage(cvSize(Gray->width, Gray->height)
                      , IPL_DEPTH_8U, 1);
    cvCanny(Gray,Edges, 250, 255, 3);
    cvReleaseImage(&Gray);
    cvReleaseImage(&Img);
    //I = Edges;
    return Edges;
  //}
    //else
}

IplImage* morphOrCanny ( IplImage* I ) {
    IplImage* T = 0;
    IplImage* Img = I;
    IplImage* Gray = cvCreateImage(cvSize(Img->width, Img->height)
                      , IPL_DEPTH_8U, 1);
    IplImage* Edges = cvCreateImage(cvSize(Gray->width, Gray->height)
                      , IPL_DEPTH_8U, 1);
    IplImage* Morph = cvCreateImage(cvSize(Gray->width, Gray->height)
                      , IPL_DEPTH_8U, 1);
    IplImage* OR = cvCreateImage(cvSize(Gray->width, Gray->height)
                        , IPL_DEPTH_8U, 1);

    cvCvtColor(I, Gray, CV_BGR2GRAY);
    cvCanny(Gray,Edges, 250, 255, 3);

    IplConvKernel* SE = cvCreateStructuringElementEx(2, 2, 1,
                                    1, CV_SHAPE_RECT, NULL);
    cvSmooth(Gray, Morph, CV_GAUSSIAN, 3, 0, 0, 0);
    cvThreshold(Morph, Morph, 0, 255
                , CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
    cvMorphologyEx(Morph, Morph, T, SE, CV_MOP_GRADIENT, 1);

    cvOr(Morph, Edges, OR, 0);

    cvReleaseImage(&Edges);
    cvReleaseImage(&Morph);
    cvReleaseImage(&Gray);
    cvReleaseImage(&Img);
    return OR;
}

IplImage* edgeMorph ( IplImage* I ) {
    IplImage* T = 0;
    IplImage* Img = I;
    IplImage* Gray = cvCreateImage(cvSize(Img->width, Img->height)
                      , IPL_DEPTH_8U, 1);
    IplImage* Morph = cvCreateImage(cvSize(Gray->width, Gray->height)
                      , IPL_DEPTH_8U, 1);

    cvCvtColor(I, Gray, CV_BGR2GRAY);

    IplConvKernel* SE = cvCreateStructuringElementEx(2, 2,
                            1, 1, CV_SHAPE_RECT, NULL);
    cvSmooth(Gray, Morph, CV_GAUSSIAN, 3, 0, 0, 0);
    cvThreshold(Morph, Morph, 0, 255
              , CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
    cvMorphologyEx(Morph, Morph, T, SE, CV_MOP_GRADIENT, 1);

    cvReleaseStructuringElement(&SE);
    cvReleaseImage(&Gray);
    cvReleaseImage(&Img);
    return Morph;
}

void markerDetect ( IplImage* I, CvPoint* hpoints ) {
  int k;
  for (k = 0; k < 4; k++) {
   hpoints[k].x = 0;
   hpoints[k].y = 0;
  }

  //assert (I->nChannels == 1);

  CvPoint* points = (CvPoint*) malloc(sizeof(CvPoint)*4);

  if (I != 0) {



    IplImage* T = 0;
    IplImage* Image = I;
    IplImage* Gray = cvCreateImage(cvSize(Image->width, Image->height)
                                  , IPL_DEPTH_8U, 1);
    IplImage* Morph = cvCreateImage(cvSize(Gray->width, Gray->height)
                                  , IPL_DEPTH_8U, 1);

    cvCvtColor(Image, Gray, CV_BGR2GRAY);

    IplConvKernel* SE = cvCreateStructuringElementEx(2, 2
                , 1, 1, CV_SHAPE_RECT, NULL);
    cvSmooth(Gray, Gray, CV_GAUSSIAN, 3, 0, 0, 0);
    cvThreshold(Gray, Gray, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
    cvMorphologyEx(Gray, Morph, T, SE, CV_MOP_GRADIENT, 1);

    cvReleaseStructuringElement(&SE);
```

```
CvMemStorage* Storage = cvCreateMemStorage(0);

IplImage* Img = Morph;

CvSeq* Cont;

IplImage* M1 = cvCreateImage(cvSize(Image->width, Image->height)
           , IPL_DEPTH_8U, 1);
IplImage* M2 = cvCreateImage(cvSize(Image->width, Image->height)
           , IPL_DEPTH_8U, 1);
IplImage* M3 = cvCreateImage(cvSize(Image->width, Image->height)
           , IPL_DEPTH_8U, 1);
IplImage* M4 = cvCreateImage(cvSize(Image->width, Image->height)
           , IPL_DEPTH_8U, 1);
IplImage* Temp = cvCreateImage(cvSize(Image->width, Image->height)
          , IPL_DEPTH_8U, 1);

int j=0;
int i;
int CountThresh = 300;
int FoundIt = 0;
CvContourScanner Scanner = cvStartFindContours( Img,
                                           Storage,
                                           sizeof(CvContour),
                                           CV_RETR_CCOMP,
                                           CV_CHAIN_APPROX_SIMPLE,
                                           cvPoint(0,0) );
float CurrentTop = 0;
float TopPerm = 0;

while ( (Cont = cvFindNextContour( Scanner )) != NULL ) {

CvSeq* Hull;
CvSeq* Approx;

Approx = cvApproxPoly(Cont,
                      sizeof(CvContour),
                      Storage,
                      CV_POLY_APPROX_DP,
                      3,
                      3);
float area = 0;
float permAvg = 0;
```

```
if (Approx->total == 4) {
  area = fabs(cvContourArea(Approx, CV_WHOLE_SEQ, 0));
  permAvg = cvContourPerimeter(Approx)/4;
}
if (Approx->total == 4 && !FoundIt &&
        cvCheckContourConvexity(Approx)
        && area > CurrentTop && permAvg > TopPerm )  {


  CurrentTop = area;
  TopPerm = permAvg;

  Hull = cvConvexHull2(Approx,
                       Storage,
                       CV_CLOCKWISE,
                       1);

  cvCvtSeqToArray(Hull, points, CV_WHOLE_SEQ);;

  CvPoint m01 = midpoint(points[0], points[1]);
  CvPoint m12 = midpoint(points[1], points[2]);
  CvPoint m23 = midpoint(points[2], points[3]);
  CvPoint m30 = midpoint(points[3], points[0]);

  CvPoint cen = midpoint(m01, m23);

  cvZero(M1);
  cvZero(M2);
  cvZero(M3);
  cvZero(M4);

  CvPoint A1[] = {cen, m30, points[0], m01};
  CvPoint A2[] = {cen, m01, points[1], m12};
  CvPoint A3[] = {cen, m12, points[2], m23};
  CvPoint A4[] = {cen, m23, points[3], m30};

  cvFillConvexPoly(M1, A1, 4, CV_RGB(255,255,255), 8, 0);
  cvFillConvexPoly(M2, A2, 4, CV_RGB(255,255,255), 8, 0);
  cvFillConvexPoly(M3, A3, 4, CV_RGB(255,255,255), 8, 0);
  cvFillConvexPoly(M4, A4, 4, CV_RGB(255,255,255), 8, 0);

  cvZero(Temp);
  cvCopy(Gray, Temp, M1);
```

```
int C1 = cvCountNonZero(Temp);

cvZero(Temp);
cvCopy(Gray, Temp, M2);
int C2 = cvCountNonZero(Temp);

cvZero(Temp);
cvCopy(Gray, Temp, M3);
int C3 = cvCountNonZero(Temp);

cvZero(Temp);
cvCopy(Gray, Temp, M4);
int C4 = cvCountNonZero(Temp);

if (C1 > CountThresh &&
    C1 > C2 &&
    C1 > C3 &&
    C1 > C4) {
   hpoints[0] = points[0];
   hpoints[1] = points[1];
   hpoints[2] = points[2];
   hpoints[3] = points[3];
   FoundIt = 1;
}
if (C2 > CountThresh &&
    C2 > C1 &&
    C2 > C3 &&
    C2 > C4) {
   hpoints[0] = points[1];
   hpoints[1] = points[2];
   hpoints[2] = points[3];
   hpoints[3] = points[0];
   FoundIt = 1;
}
if (C3 > CountThresh &&
    C3 > C1 &&
    C3 > C2 &&
    C3 > C4) {
   hpoints[0] = points[2];
   hpoints[1] = points[3];
   hpoints[2] = points[0];
   hpoints[3] = points[1];
   FoundIt = 1;
```

```
      }
      if (C4 > CountThresh &&
          C4 > C1 &&
          C4 > C2 &&
          C4 > C3) {
         hpoints[0] = points[3];
         hpoints[1] = points[0];
         hpoints[2] = points[1];
         hpoints[3] = points[2];
         FoundIt = 1;
      }
    }
    cvSubstituteContour( Scanner, NULL );
  }
  cvReleaseImage(&Image);
  cvReleaseImage(&Temp);
  cvReleaseImage(&Gray);
  cvReleaseImage(&M1);
  cvReleaseImage(&M2);
  cvReleaseImage(&M3);
  cvReleaseImage(&M4);
  cvReleaseImage(&Img);
  cvReleaseMemStorage(&Storage);
  }
  free(points);
}


CvPoint midpoint(CvPoint a, CvPoint b) {
  CvPoint Mid;
  float hx = (float)(a.x + b.x);
  float hy = (float)(a.y + b.y);
  Mid.x = (int)(hx/2);
  Mid.y = (int)(hy/2);
  return Mid;
}


float glToFloat (float c) {
  return c;
}

float doubleToFloat (double c) {
  return (float)c;
}

double floatToDouble (float c) {
  return (double)c;
}
```

7.3 CvStructures Chs File

```
{-# LANGUAGE ForeignFunctionInterface #-}
{-# LANGUAGE TypeSynonymInstances #-}

module HaskCV.Structs (Point2F, CvPoint, IplImage
            , pointToTuple, imageData, width, height, align) where

#include <cv.h>

import C2HS
import Foreign.Ptr
import Foreign.ForeignPtr
import Foreign.C.Types
import Foreign.C.String
import Foreign.Storable
import System.IO.Unsafe (unsafePerformIO)
import Data.ByteString
import Data.Word (Word8)
import Control.Monad
import Control.Applicative ((<$>),(<*>))


{-# context lib="cv" #-}

type Point a = (a, a)
type Point2D = Point Int


data CvPoint = CvPoint {x :: Int, y :: Int} deriving (Eq, Show)

instance Storable CvPoint where
  sizeOf _ = {# sizeof CvPoint #}
  alignment _ = alignment (undefined :: CInt)
  peek p = CvPoint
    <$> fmap fromIntegral ({#get CvPoint->x #} p)
    <*> fmap fromIntegral ({#get CvPoint->y #} p)
 -- poke p point = do
  --  {#set CvPoint.x #} p ((fromIntegral x)::CInt point)
   -- {#set CvPoint.y #} p ((fromIntegral y)::CInt point)

pointToTuple :: CvPoint -> (Float, Float)
pointToTuple (CvPoint x y) = (fromIntegral x, fromIntegral y)

tupleToPoint :: (Float, Float) -> CvPoint
tupleToPoint (x, y) = CvPoint (cast x) (cast y)
  where
    cast = (toEnum ∘ fromEnum)

type Point2F = CvPoint2D32f

data CvPoint2D32f = CvPoint2D32f {
    x'CvPoint2D32f :: Float
  , y'CvPoint2D32f :: Float
}

instance Storable CvPoint2D32f where
  sizeOf _ = {# sizeof CvPoint2D32f #}
  alignment _ = alignment (undefined :: CFloat)
  peek p = CvPoint2D32f
    <$> liftM cFloatConv ({#get CvPoint2D32f->x #} p)
    <*> liftM cFloatConv ({#get CvPoint2D32f->y #} p)
--  poke p x = do
--    {#set CvPoint2D32f.x #} p (cFloatConv $ x'CvPoint2D32f x)
--    {#set CvPoint2D32f.y #} p (cFloatConv $ y'CvPoint2D32f x)

data IplImage = IplImage {
  -- nSize'IplImage :: Int,
  -- nChannels'IplImage :: Int,
  -- alphaChannel'IplImage :: Int
  -- depth'IplImage :: Int,
  -- dataOrder'IplImage :: Int,
  -- origin'IplImage :: Int,
  align :: Int,
  width :: Int,
  height :: Int,
  -- imageSize'IplImage :: CInt,
  imageData :: ByteString } deriving (Eq, Show)
```

```
instance Storable IplImage where
  sizeOf _ = {#sizeof IplImage #}
  alignment _ = alignment (undefined :: CInt)
  peek p =
    IplImage <$> fmap fromIntegral ({#get IplImage->align #} p)
             <*> fmap fromIntegral ({#get IplImage->width #} p)
             <*> fmap fromIntegral ({#get IplImage->height #} p)
             <*> fmap (unsafePerformIO ∘ packCString)
                  ({#get IplImage->imageData #} p)
```

7.4 HaskCV Hs File

```
{-# LANGUAGE ForeignFunctionInterface, EmptyDataDecls #-}

module HaskCV.Funcs where

import Foreign --(Ptr, ForeignPtr, withForeignPtr, newForeignPtr)
import Foreign.C
import Foreign.Ptr
import Foreign.ForeignPtr
import Foreign.C.Types
import Foreign.C.String
import Foreign.Marshal.Array
import Prelude
import System.IO
import Control.Monad
import Data.Word (Word8)
import Graphics.Rendering.OpenGL.Raw
import Generics.Pointless.MonadCombinators ( mfuse )
import Data.Packed.Matrix
import Data.Packed.Vector
import qualified Data.Packed.Development as Util
import HaskCV.Structs

type CvSize = (Int, Int)
type CvSize = (Int, Int)

-- Phantom Data Type for Our IplImage
data CvImage

-- Phantom Data Type for our CvCapture type
data CvCapture -- = CvCapture (Ptr CvCapture)

-- Phantom Data Type for our CvMemStorage
```

```
data CvMemStorage = CvMemStorage

-- Phantom Data Type for our CvSeq, a growable seqeunce of elements
data CvSeq = CvSeq

foreign import ccall unsafe "highgui.h cvCreateCameraCapture"
  captureFromCam :: Int → IO (Ptr CvCapture)

foreign import ccall unsafe "ImgDecon.h glToFloat"
  floatToGL :: Float → GLfloat


foreign import ccall unsafe "ImgDecon.h doubleToFloat"
  doubleToGLfloat :: Double → GLfloat

foreign import ccall unsafe "ImgDecon.h floatToDouble"
  floatToDouble :: Float → Double

foreign import ccall unsafe "ImgDecon.h castCPtr"
  castCPtrToUC :: Ptr CChar → IO (Ptr CUChar)

foreign import ccall unsafe "ImgDecon.h frameGrab"
  frameGrab :: Ptr CvCapture → IO (Ptr CvImage)


foreign import ccall unsafe "ImgDecon.h imgWidth"
  imgWidth :: Ptr CvImage → IO (Int)

-- Return Height
foreign import ccall unsafe "ImgDecon.h imgHeight"
  imgHeight :: Ptr CvImage → IO (Int)

foreign import ccall unsafe "ImgDecon.h ImgData"
  imgData :: Ptr CvImage → IO (Ptr Word8)

foreign import ccall unsafe "ImgDecon.h cloneImage"
  cloneImage :: Ptr CvImage → IO (Ptr CvImage)

foreign import ccall unsafe "ImgDecon.h edgeCanny"
  edgeCanny :: Ptr CvImage → IO (Ptr CvImage)

foreign import ccall unsafe "ImgDecon.h edgeMorph"
  edgeMorph :: Ptr CvImage → IO (Ptr CvImage)
```

```
foreign import ccall unsafe "ImgDecon.h morphOrCanny"
  morphOrCanny :: Ptr CvImage → IO (Ptr CvImage)

foreign import ccall unsafe "ImgDecon.h markerDetect"
  markerDetectCore :: Ptr CvImage → Ptr CvPoint → IO ()

markerDetect :: Ptr CvImage → IO [(Float, Float)]
markerDetect img =
  allocaArray 4 $ λ points →
  do markerDetectCore img points
     l ← peekArray 4 points
     return $ map pointToTuple l

foreign import ccall unsafe "cv.h cvFlip"
  flipIm :: Ptr CvImage → Ptr CvImage → Int → IO ()

foreign import ccall unsafe "cv.h cvReleaseImage"
  freeImage :: Ptr CvImage → IO ()

foreign import ccall unsafe "ImgDecon.h freeVector"
  freeVector :: Ptr Word8 → IO ()

foreign import ccall unsafe "ImgDecon.h releaseData"
  releaseData :: Ptr CvImage → IO (Ptr CvImage)

foreign import ccall unsafe "ImgDecon.h qFrame"
  queryFrame :: Ptr CvImage → Ptr CvCapture → IO (Ptr CvImage)

foreign import ccall unsafe "cv.h cvCvtColor"
  cvtColor :: Ptr CvImage → Ptr CvImage → Int → IO ()

-- Load an CvImage
foreign import ccall unsafe "highgui.h cvLoadImage"
  loadCvImage :: CString → Int → IO (Ptr CvImage)

-- <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
-- HIGHGUI BINDS
-- <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
-- Create a window
foreign import ccall unsafe "highgui.h cvNamedWindow"
  namedWindow :: CString → Int → IO (Int)
```

```
-- Destroy a window
foreign import ccall unsafe "highgui.h cvDestroyWindow"
  releaseWindow :: CString → IO (Int)

-- Display to a window
foreign import ccall unsafe "highgui.h cvShowImage"
  showCvImageCore :: CString → Ptr CvImage → IO ()

showCvImage :: CString → ForeignPtr CvImage → IO ()
showCvImage window img =
  withForeignPtr img ( λ ptr →
  do if (ptr ≡ nullPtr)
        then return()
        else showCvImageCore window ptr )
-- WaitKey
foreign import ccall unsafe "highgui.h cvWaitKey"
  waitKey :: Int → IO (Int)



-- ≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪
-- IMAGE PROCESSING
-- ≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪
foreign import ccall unsafe "ImgDecon.h FindCorrespondences"
  findCorrespondencesCore :: Ptr CvImage → Int → IO (Ptr Point2F)

findCorrespondences :: ForeignPtr CvImage → Int → IO (ForeignPtr Point2F)
findCorrespondences im n = withForeignPtr im $ λ p→
  do p2  ← findCorrespondencesCore p n
     p3 ← newForeignPtr freePoints p2
     return p3

-- Canny Edge Detection
foreign import ccall unsafe "cv.h cvCanny"
  cannyDetect :: Ptr CvImage → Ptr CvImage → Double →
                 Double → Int → IO ()

{-
cannyDetect :: ForeignPtr CvImage → ForeignPtr CvImage →
               Double → Double → Int → IO ()
cannyDetect img edges thresh1 thresh2 aper =
  withForeignPtr img   ( λ p1 →
  withForeignPtr edges ( λ p2 →
  do cannyDetectCore p1 p2 thresh1 thresh2 aper ) )

-}

-- Downsample
foreign import ccall unsafe "cv.h cvPyrDown"
  downsample :: Ptr CvImage → Ptr CvImage → Int → IO ()

-- Set Channel of Interest
foreign import ccall unsafe "cv.h cvSetImageCOI"
  setImageCOICore :: Ptr CvImage → Int → IO ()

setImageCOI :: ForeignPtr CvImage → Int → IO ()
setImageCOI img coi =
  withForeignPtr img ( λ i →
  do setImageCOICore i coi )

-- ≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪
-- MEMORY MANAGMENT
-- ≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪≪
-- Functions that create memory C-side
-- Create Memstorage
foreign import ccall unsafe "cxcore.h cvCreateMemStorage"
  createMemStorageCore :: Int → IO (Ptr CvMemStorage)

createMemStorage :: Int → IO (ForeignPtr CvMemStorage)
createMemStorage n = createMemStorageCore n >>= λ p →
                     newForeignPtr freeMemStor p

-- Create a Sequence
foreign import ccall unsafe "cxcore.h cvCreateSeq"
  createSeqCore :: Int → Int → Int → Ptr CvMemStorage → IO (Ptr CvSeq)

createSeq :: Int → Int → Int →
              ForeignPtr CvMemStorage → IO (ForeignPtr CvSeq)
createSeq seqFlags headerSize elemSize storage =
  withForeignPtr storage ( λ ps →
  do ptr ← createSeqCore seqFlags headerSize elemSize ps
     nseq ← newForeignPtr freeSeq ptr
     return nseq )
-- Create Image
-- Width Height Depth Channels
foreign import ccall unsafe "cv.h cvCreateImage"
```

```
  createImageCore :: Int → Int → Int → Int → IO (Ptr CvImage)

createImage :: Int → Int → Int → Int → IO (ForeignPtr CvImage)
createImage width height code chan =
  createImageCore width height code chan >>= λ p →
  newForeignPtr freeImg p

foreign import ccall unsafe "ImgDecon.h NewImage"
  newImage :: Int → Int → Int → Int → IO (Ptr CvImage)

--newImage :: (Int, Int) → Int → Int → IO (ForeignPtr CvImage)
--newImage (w, h) d c = newImageCore w h d c >>= λ p →
  --newForeignPtr freeImg p


-- Create Image Header
-- Width Height Depth Channels
foreign import ccall unsafe "cv.h cvCreateImageHeader"
  createImageHeaderCore :: Int → Int → Int → Int → IO (Ptr CvImage)

createImageHeader :: CvSize → Int → Int → IO (ForeignPtr CvImage)
createImageHeader (width, height) depth channels =
  createImageCore width height depth channels >>= λ p →
  newForeignPtr freeImg p

-- DUPLICATION FUNCTIONS
-- cvCopy
-- Copies one array to another
foreign import ccall unsafe "cv.h cvCopy"
  copyImgCore :: Ptr CvImage → Ptr CvImage → Ptr CvImage → IO ()

copyImg :: ForeignPtr CvImage → ForeignPtr CvImage →
           ForeignPtr CvImage → IO ()
copyImg src dst msk =
  withForeignPtr src ( λ s →
  withForeignPtr dst ( λ d →
  withForeignPtr msk ( λ m →
  do copyImgCore s d m ) ) )

-- cvCloneImage
-- Makes a full copy of an image
-- MEMORY MANAGEMENT FINALIZER PTRS
foreign import ccall unsafe "cxcore.h &cvReleaseMemStorage"
```

```
  freeMemStor :: FunPtr (Ptr CvMemStorage → IO ())

foreign import ccall unsafe "ImgDecon.h &freeImg"
  freeImg :: FunPtr (Ptr CvImage → IO ())

foreign import ccall unsafe "highgui.h &cvReleaseCapture"
  freeCap :: FunPtr (Ptr CvCapture → IO ())

foreign import ccall unsafe "ImgDecon.h &freePointArr"
  freePoints :: FunPtr (Ptr Point2F → IO ())

-- MY FINALIZERS
foreign import ccall unsafe "ImgDecon.h &freeVector"
  freeVec :: FunPtr (Ptr CUChar → IO ())

foreign import ccall unsafe "ImgDecon.h &freeSeq"
  freeSeq :: FunPtr (Ptr CvSeq → IO ())


-- <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
-- GENERIC FUNCTIONS
-- <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

-- Return Depth
foreign import ccall unsafe "ImgDecon.h imgDepth"
  imgDepthCore :: Ptr CvImage → IO (Int)

imgDepth :: ForeignPtr CvImage → IO (Int)
imgDepth im = withForeignPtr im ( λ p →
  do depth ← imgDepthCore p
     return depth )
```

7.5 Transformation Hs File

```
module HaskCV.VectorSpace.Endomorphism (nDlt, homTransform
     , disp, intrin, modelMatrix, svdDecomHom, modelPipe
     , frustumParams, modelMatrix, perspectiveGLU, projectionMatrix
     , computeViewport, rotateTranslate, computePerspective
     , computeHomography ) where

{-# LANGUAGE BangPatterns #-}

import Graphics.Rendering.OpenGL.Raw
import Data.Packed.Matrix
```

```haskell
import Numeric.LinearAlgebra
import Numeric.Special.Trigonometric (acot, cot)
import Data.List
-- Camera Intrinsics
intrin :: Matrix Double
intrin = (3><3) [5.25405029e+02,  0.0, (3.07575745e+02)
                , 0.0, 5.25583801e+02, (2.22051102e+02)
                , 0.0,            0.0,            (1.0)] -- repoj 42

-- Nice Matrix Printing
disp :: Matrix Double → IO ()
disp = putStr ∘ dispf 2


homTransform :: [(Double, Double)] → Matrix Double → [(Double, Double)]
homTransform [] _ = []
homTransform xs h =
  map ((λu → (u @@> (0,0), u @@> (1,0)))
       ∘ (λu → scale (1/(u @@> (2,0))) u)
       ∘ (λu → h <> u)
       ∘ (λ(x, y) → (3><1) [x, y, 1])) xs

nDlt :: [(Double, Double)] → [(Double, Double)] → Matrix Double
nDlt [] _ = ident 3
nDlt _ [] = ident 3
nDlt xs ys | length xs /= length ys = ident 3
           | otherwise = (inv t') <> mat <> t
  where mat = scale (-1) $ ((reshape 3) ∘ nullVector ∘ fromBlocks) $ zipWith dRows nxs n
        (t, nxs)  = normPnts xs
        (t', nys) = normPnts ys

dRows :: Matrix Double → Matrix Double → [Matrix Double]
dRows u' u = [fromBlocks [[           v0, scale (-w) v',    scale y v']
                         ,[   scale w v',            v0, scale (-x) v']
                         ,[scale (-y) v',    scale x v',
v0]]]
  where (!v', !v)    = (trans u', trans u)
        (!x, !y, !w) = (u @@> (0,0), u @@> (1,0), u @@> (2,0))
        v0           = (1><3) [0,0,0]

centroid :: [(Double, Double)] → (Double, Double)
centroid [] = (0, 0)
centroid xs = (cx / n, cy / n)
```

```
  where (cx, cy) = foldl' addTup (0, 0) xs
        n = fromIntegral $ length xs
        addTup (x, y) (x', y') = (x + x', y + y')

shiftC :: [(Double, Double)] → (Double, Double) → [(Double, Double)]
shiftC [] _ = []
shiftC xs (x, y) = map (move (x, y)) xs
  where move (cx, cy) (u, v) = (u - cx, v - cy)

normPnts :: [(Double, Double)] → (Matrix Double, [Matrix Double])
normPnts [] = (ident 3, [])
normPnts xs = (simT, (map (λv → simT <> v) vs))
  where (cx, cy) = centroid xs
        ns       = shiftC xs (cx, cy)
        vs       = map (λ(x, y) → (3><1) [x, y, 1]) xs
        s        = let acc = sum $ map (λ(x, y) → sqrt $ x^2 + y^2) ns
                       n   = fromIntegral $ length ns
                   in (sqrt 2) / (acc / n)
        simT     = (3><3) [ s,  0,  (-s)*cx
                          , 0,  s,  (-s)*cy
                          , 0,  0,        1 ]


-- Decomposition of Homography Matrix

svdDecomHom :: Matrix Double → (Matrix Double, Matrix Double)
svdDecomHom hom | l == 0 = (ident 3, ((3><1) [0, 0, 0]))
                | otherwise = (rOrth, tVec)
  where [h1,h2,h3] = toColumns hom
        l          = sqrt $ (sum ∘ toList ∘ (zipVectorWith (*) h1)) h1
        r1         = cInv <> h1
        r2         = cInv <> h2
        r3         = (zipVectorWith (*) r1 r2) - (zipVectorWith (*) r2 r1)
        tVec       = fromColumns $ [cInv <> h3]
        rm         = fromColumns [r1, r2, r3]
        (u, _, v)  = fullSVD rm
        rOrth      = u <> (trans v)
        cInv       = inv intrin

modelMatrix :: (Matrix Double, Matrix Double) → [Double]
modelMatrix (rot, tvec) = out
  where lst = toRows $ fromBlocks [[rot, nTv]]
        adV = ({-trans ∘ -}fromRows) (lst ++ [z])
        out = concat $ map toList $ toColumns adV
        z = 4 |> [0, 0, 0, 1]
        nTv = (3><1) [(-(tvec @@> (0,0)))
                     , (-(tvec @@> (1,0)))
                     , ((tvec @@> (2,0)))]

modelPipe :: Matrix Double → [Double]
modelPipe = modelMatrix ∘ svdDecomHom

perspectiveGLU :: Int → Int → (GLint, GLint, GLdouble, GLdouble)
perspectiveGLU width height = (x0, y0, fovy, aspect)
  where
    !w = fromIntegral width
    !h = fromIntegral height
    !fx = intrin @@> (0, 0)
    !fy = (1)*intrin @@> (1, 1)
    !x0 = conToGLi $ (intrin @@> (0, 2)) - (w/2)
    !y0 = conToGLi $ (1)*((h/2) - (intrin @@> (1, 2)))
    !fovy = realToFrac $ 2 * (atan (h/(2*fy))) * (180 / pi)  -- 2 *
acot (2 * fy / h)
    !aspect = realToFrac $ (w/h) * (fy/fx)


frustumParams :: Int → Int →
            (GLdouble, GLdouble, GLdouble, GLdouble, GLdouble, GLdouble)
frustumParams width height = (left, right, bottom, top, nearVal, farVal)
  where
    w = fromIntegral width
    h = fromIntegral height
    fx = intrin @@> (0, 0)
    fy = intrin @@> (1, 1)
    cx = intrin @@> (0, 2)
    cy = intrin @@> (1, 2)
    left = realToFrac $ -cx
    right = realToFrac $ cx
    bottom = realToFrac $ -cy
    top = realToFrac $ cy
    nearVal = realToFrac $ 0.01
    farVal = realToFrac $ 1000

{-
projectionMatrix :: [Double]
projectionMatrix = f
```

```
  where
    s = toRows $ fromBlocks [[intrin, (3><1) [0, 0, 0]]]
    z = 4 |> [0, 0, (-1), 0]
    !f = concat $ map toList (s ++ [z])
-}


projectionMatrix :: Double → Double → [Double]
projectionMatrix kn kf = concat $ (map toList ∘  toRows ∘ trans)  m
  where
    fx = intrin @@> (0,0)
    fy = intrin @@> (1,1)
    cx = intrin @@> (0,2)
    cy = intrin @@> (1,2)
    w  = 640
    h  = 480
    (x0, y0, fovy, aspect) = perspectiveGLU 640 480
    {-scl = (4><4) [1,  0,  0,  0
                 ,0, (-1), 0, 0
                 ,0, 0, (-1), 0
                 ,0, 0, 0,    1]
    pro = (3><4) [( (cot (fovy/2)) / aspect), 0, 0, 0
                 ,0, (cot (fovy/2)), 0, 0
                 ,0, 0, (-1), 0]
    cli = (3><3) [(w/2), 0, (x0 + w/2)
                 ,(cot (fovy/2)), 0, 0
                 ,0, 0, 1]-}
    m  = (4><4) [(2*fx/w), 0, (1 - (2*cx/w)), 0
                , 0, (2*fy/h), (((2*cy + 2)/h) - 1), 0
                , 0, 0, ((kf + kn)/(kn - kf)), (2*kf*kn/(kn - kf))
                , 0, 0,                    -1,                0]


eulerAnglesFromMatrix :: Matrix Double → (Double, Double, Double)
eulerAnglesFromMatrix rot
  | rot @@> (1,0) > 0.998 =
      ( atan2 (rot @@> (0,2)) (rot @@> (2,2))
      , pi / 2
      , 0
      )
  | rot @@> (1,0) < (-0.998) =
      ( atan2 (rot @@> (0,2)) (rot @@> (2,2))
      , (-pi)/2
      , 0
      )
  | otherwise =
      ( atan2 (-(rot @@> (0,2))) (rot @@> (0,0))
      , atan2 (-(rot @@> (1,2))) (rot @@> (1,1))
      , asin (rot @@> (1,0))
      )


conToGLi = (toEnum ∘ fromEnum)
conToGLs = (toEnum ∘ fromEnum)

computeViewport :: Int → Int → (GLint, GLint, GLsizei, GLsizei)
computeViewport width height = (x, y, conToGLs width, conToGLs height)
  where
    w = fromIntegral width
    h = fromIntegral height
    uc = intrin @@> (0,2)
    vc = intrin @@> (1,2)
    x = conToGLi $ uc - w/2
    y = conToGLi $ vc - h/2

computePerspective :: Int → Int → (GLdouble, GLdouble, GLdouble, GLdouble)
computePerspective width height = (fovy, aspect, near, far)
  where
    w = fromIntegral width
    h = fromIntegral height
    fovy   = realToFrac $ 2 ∗ (atan ((h/2)/(lambda∗f/dy)))/pi ∗ 180
    aspect = realToFrac $ (w/h)∗(dx/dy)
    near   = 0.1
    far    = 200.0
    lambda = 1 -- + k∗rd -- radial distortion
    f  = 1
    dx = 1 / (intrin @@> (0,0))
    dy = 1 / (intrin @@> (1,1))

computeHomography :: [(Float, Float)] → Matrix Double
computeHomography [] = ident 3
computeHomography xs | elem (0,0) xs = ident 3
                     | otherwise = nDlt world image
  where
    image = map (λ(x,y) → (realToFrac x, realToFrac y)) xs
    world = [(0,0),(0,1),(1,1),(1,0)]
```

```
rotateTranslate :: Matrix Double →
              (GLfloat, GLfloat, GLfloat, GLfloat, GLfloat, GLfloat)
rotateTranslate hom = (a, b, g, t0, t1, t2)
  where
    (!rotation, !t) = svdDecomHom hom
    (!alpha, !beta, !gamma) = eulerAnglesFromMatrix $ skew <>  rotation
    [!a, !b, !g] = map (realToFrac ∘ (λx → x/pi ∗ 180)) [alpha
                                                        ,beta
                                                        ,gamma]
    [!t0, !t1, !t2] = map realToFrac
                   [(-(t @@> (0,0)))
                   ,(-(t @@> (1,0)))
                   ,t @@> (2,0)]
    !skew = (3><3) [1,0,0
              ,0,(-1),0
              ,0,0,(-1)]
```

7.6 Main Hs File

```
--
-- Based on code from Jeff Molofee '99 (ported to Haskell GHC 2005)
-- Who Translated the nehe tutorials to haskell
--

module Main where

import qualified Graphics.UI.GLFW as GLFW
import Graphics.Rendering.OpenGL.Raw
import Graphics.Rendering.GLU.Raw (gluPerspective
                                  ,gluLookAt)
import Data.Bits ((.|.))
import System.Exit (exitWith, ExitCode(..))
import Control.Monad (forever, liftM, sequence)
import Data.IORef (IORef, newIORef, readIORef, modifyIORef, writeIORef)
import Foreign (Ptr, finalizeForeignPtr, touchForeignPtr
               , withForeignPtr, plusPtr, peek, alloca, free)
import Foreign.Marshal.Array (newArray)
import qualified Data.ByteString.Internal as BSI
import Util (Image(..), bitmapLoad)
import qualified HaskCV.Funcs as CV
import HaskCV.Structs (IplImage(..))
import HaskCV.VectorSpace.Endomorphism
import System.IO.Unsafe
```

```haskell
import Data.Packed.Matrix
import Numeric.LinearAlgebra

tex :: GLuint
tex = 1

(pfovy, paspect, pnear, pfar) = computePerspective 640 480
(vx, vy, vw, vh) = computeViewport 640 480

initGL :: IO (Ptr CV.CvCapture)
initGL = do
  glShadeModel gl_SMOOTH
  glClearColor 0 0 0 1
  glDepthFunc gl_LEQUAL
  glHint gl_LINE_SMOOTH_HINT gl_NICEST
  CV.captureFromCam 3

-- use glDrawPixels instead?
genGLTextureFromImage :: IORef( Ptr CV.CvImage ) → IO ()
genGLTextureFromImage iref = do
  img ← readIORef iref
  dat ← CV.imgData   img
  w   ← CV.imgWidth  img
  h   ← CV.imgHeight img
  glBindTexture gl_TEXTURE_2D tex
  glTexImage2D  gl_TEXTURE_2D 0 3
    (fromIntegral w) (fromIntegral h) 0 gl_BGR gl_UNSIGNED_BYTE
    dat --pd
  let glLinear = fromIntegral gl_LINEAR
  glTexParameteri gl_TEXTURE_2D gl_TEXTURE_MIN_FILTER glLinear
  glTexParameteri gl_TEXTURE_2D gl_TEXTURE_MIN_FILTER glLinear


drawPoly :: [(Float, Float)] → [IO ()]
drawPoly xs = map (λ(x, y) → glVertex2f (CV.floatToGL x) (CV.floatToGL y)) xs


mPicture :: [(Double, Double)] → IO ()
mPicture [] = return ()
mPicture xs = do
  let  x i = realToFrac $ fst $ xs !! i
  let  y i = realToFrac $ snd $ xs !! i
  glBegin gl_QUADS
  glTexCoord2f   0     0
  glVertex2f     (x 0) (y 0)
  glTexCoord2f   1     0
  glVertex2f     (x 1) (y 1)
  glTexCoord2f   1     1
  glVertex2f     (x 2) (y 2)
  glTexCoord2f   0     1
  glVertex2f     (x 3) (y 3)
  glEnd


drawSqr :: [(Double, Double)] → IO ()
drawSqr xs = do
  glBegin gl_QUADS
  glVertex2f (x 0) (y 0)
  glVertex2f (x 1) (y 1)
  glVertex2f (x 2) (y 2)
  glVertex2f (x 3) (y 3)
  glEnd
  where
    x i = realToFrac $ fst $ xs !! i
    y i = realToFrac $ snd $ xs !! i


findTransform :: [(Float, Float)] → [(Double, Double)]
findTransform [] = []
findTransform xs | cond = [(0,0),(0,0),(0,0),(0,0)]
                 | otherwise = homTransform world hom
  where
    image  = map (λ(x, y) → (realToFrac x, realToFrac y)) xs
    hom    = nDlt world image
    world  = [(0, 0), (0, 600), (600, 600), (600, 0)]
    cond   = elem (0,0) xs

-- no longer used
orthoView :: IO ()
orthoView = do
  glMatrixMode gl_PROJECTION
  glPushMatrix
  glLoadIdentity
  glOrtho 0.0 640.0 480.0 0.0 (-1.0) 1.0
  glMatrixMode gl_MODELVIEW
  glPushMatrix
  glLoadIdentity

resizeScene :: GLFW.WindowSizeCallback
resizeScene w     0      = resizeScene w 1 -- prevent divide by zero
resizeScene width height = do
  glViewport vx vy vw vh
  glMatrixMode gl_PROJECTION
  glLoadIdentity
  gluPerspective pfovy paspect pnear pfar
  glMatrixMode gl_MODELVIEW
  glLoadIdentity
  glFlush

createGLMatrix :: [Double] → IO (Ptr GLdouble)
createGLMatrix elements = newArray (map realToFrac elements)

drawScene :: Ptr CV.CvCapture → IORef (Ptr CV.CvImage) →
                IORef (Ptr CV.CvImage) → IO ()
drawScene cam imgRef picRef = do
  let partialFrame x = CV.queryFrame x cam -- flip $ CV.queryFrame cam
  glClear $ fromIntegral  $  gl_COLOR_BUFFER_BIT
  modifyIORef imgRef $! (unsafePerformIO ∘ partialFrame)

  img ← readIORef imgRef
  CV.flipIm img img 0
  cop ← CV.cloneImage img
  pnts ← CV.markerDetect cop

  dat ← CV.imgData img

  glViewport vx vy vw vh

  glMatrixMode gl_PROJECTION
  glLoadIdentity
  gluPerspective pfovy paspect pnear pfar

  glMatrixMode gl_MODELVIEW
  glLoadIdentity

  glDrawPixels 640 480 gl_BGR gl_UNSIGNED_BYTE dat

  glMatrixMode gl_PROJECTION
  glPopMatrix
```

```
glMatrixMode gl_MODELVIEW
glPopMatrix


-- SO this works fine but we're going to try something else for a second
{-
genGLTextureFromImage imgRef

glBindTexture gl_TEXTURE_2D tex  -- ::GLuint)

glBegin gl_QUADS -- start drawing a polygon (4 sided)
glTexCoord2f   0    0
glVertex3f     0    480 (-1999)    -- bottom left of quad (Front)
glTexCoord2f   1    0
glVertex3f     640  480 (-1999)    -- bottom right of quad (Front)
glTexCoord2f   1    1
glVertex3f     640  0   (-1999)    -- top right of quad (Front)
glTexCoord2f   0    1
glVertex3f     0    0   (-1999)    -- top left of quad (Front)
glEnd
-}

glViewport vx vx vw vh
glMatrixMode gl_PROJECTION
glLoadIdentity
gluPerspective pfovy paspect pnear pfar

glMatrixMode gl_MODELVIEW
glLoadIdentity

model ← createGLMatrix $ (modelPipe ∘ computeHomography) pnts
glLoadMatrixd model

glRotatef 180 0 1 0

glScalef 0.01 0.01 0.01

glPushMatrix
glBegin gl_LINES
glColor3f 1 0 0
glVertex3f 0 0 0
glVertex3f (10) 0 0
glColor3f 0 1 0
```

```
  glVertex3f 0 0 0
  glVertex3f 0 (-10) 0
  glColor3f 0 0 1
  glVertex3f 0 0 0
  glVertex3f 0 0 (10)
  glEnd
  glPopMatrix

  glFlush


shutdown :: GLFW.WindowCloseCallback
shutdown = do
  GLFW.closeWindow
  GLFW.terminate
  _ ← exitWith ExitSuccess
  return True

keyPressed :: GLFW.KeyCallback
keyPressed GLFW.KeyEsc True = shutdown >> return ()
keyPressed _           _    = return ()

main :: IO ()
main = do
     True ← GLFW.initialize
     let dspOpts = GLFW.defaultDisplayOptions {
       GLFW.displayOptions_width  = 640,
       GLFW.displayOptions_height = 480,
       GLFW.displayOptions_numRedBits   = 8,
       GLFW.displayOptions_numGreenBits = 8,
       GLFW.displayOptions_numBlueBits  = 8,
       GLFW.displayOptions_numAlphaBits = 8,
       GLFW.displayOptions_numDepthBits = 1
     }
     True ← GLFW.openWindow dspOpts
     GLFW.setWindowPosition 0 0
     GLFW.setWindowTitle "Augmented Reality"
     imgRef ← newIORef $ unsafePerformIO $ CV.newImage 640 480 8 (3 :: Int)
     picRef ← newIORef $ unsafePerformIO $ CV.newImage 640 480 8 (3 :: Int)
     cam  ← initGL

     GLFW.setWindowRefreshCallback (drawScene cam imgRef picRef)
     GLFW.setWindowSizeCallback resizeScene
     GLFW.setKeyCallback keyPressed
     GLFW.setWindowCloseCallback shutdown
     forever $ do
       drawScene cam imgRef picRef
       GLFW.swapBuffers
```

7.7 Open Hand Detector C File

```
IplImage *OpenHandDetector (IplImage *Input) {
IplImage *Im = Input;

int method = CV_INTER_CUBIC;

IplImage *I = cvCreateImage(cvSize(Im->width/2, Im->height/2), IPL_DEPTH_8U, 3);
cvResize(Im, I, method);

// RGB ALG FIRST
// R > 95 and G > 40 and B > 20
// max(R,G,B) - min(R,G,B) > 15
// |R-G| > 15 and R > G and R > B
IplImage *CRB = cvCreateImage(cvSize(I->width, I->height), IPL_DEPTH_8U, 3);
IplImage *Mask = cvCreateImage(cvSize(I->width, I->height), IPL_DEPTH_8U, 1);
IplImage *N = cvCreateImage(cvSize(Im->width, Im->height), IPL_DEPTH_8U, 1);

cvCvtColor(I, CRB, CV_BGR2YCrCb);

cvZero(Mask);
int x = 0;
int y = 0;
uchar R,G,B,Y,Cr,Cb,Dif,RmG,max,min;
for (x = 0; x < I->width; x++) {
  for (y = 0; y < I->height; y++) {
    B = ((uchar*)(I->imageData + I->widthStep*y))[x*3];
    G = ((uchar*)(I->imageData + I->widthStep*y))[x*3+1];
    R = ((uchar*)(I->imageData + I->widthStep*y))[x*3+2];

    Y  = ((uchar*)(CRB->imageData + CRB->widthStep*y))[x*3];
    Cr = ((uchar*)(CRB->imageData + CRB->widthStep*y))[x*3+1];
    Cb = ((uchar*)(CRB->imageData + CRB->widthStep*y))[x*3+2];

    RmG = R-G;

    if (R > G && R > B)
      max = R;
    else
      if (G > R && G > B)
        max = G;
      else
        max = B;

    if (R < G && R < B)
      min = R;
    else
      if (G < R && G < B)
        min = G;
      else
        min = B;

    Dif = max-min;

    if ( R > 95 && G > 40 && B > 20 && Dif > 15 && RmG > 15 && R > G && R > B ) {
      if (Y > 80 && Cb > 85 && Cb < 135 && Cr > 135 && Cr < 180)  {
          ((uchar*)(Mask->imageData + Mask->widthStep*y))[x] = 255;
      }
    } //End Constraints
  }
} // End Loops

cvResize(Mask, N, method);
cvMorphologyEx(N,N,0,0,CV_MOP_OPEN,2);
cvMorphologyEx(N,N,0,0,CV_MOP_CLOSE,1);
cvSmooth(N,N,CV_GAUSSIAN,3,0,0,0);

IplImage *CImg = cvCreateImage(cvSize(N->width, N->height), IPL_DEPTH_8U, 1);

IplImage* Out = cvCreateImage(cvSize(N->width, N->height), IPL_DEPTH_8U, 3);
cvZero(Out);
cvZero(CImg);

CvMemStorage *Storage = cvCreateMemStorage(0);
CvMemStorage *DefStorage = cvCreateMemStorage(0);

CvSeq* Contours;

CvContourScanner Scanner = cvStartFindContours( N,
                                                Storage,
                                                sizeof(CvContour),
                                                CV_RETR_EXTERNAL,
                                                CV_CHAIN_APPROX_SIMPLE,
                                                cvPoint(0,0) );
CvSeq* Cont;

int i;

while ( (Cont = cvFindNextContour( Scanner )) != NULL ) {
  double len = cvContourPerimeter(Cont);
  double q   = (N->height + N->width)/4;
  double area = fabs(cvContourArea(Cont, CV_WHOLE_SEQ, 0));
  if ( len > q && area > len) {
    CvSeq* Approx;
    CvSeq* Hull;
    CvSeq* HullPnts;
    CvSeq* Defects;

    Approx = cvApproxPoly(Cont,
                          sizeof(CvContour),
                          Storage,
                          CV_POLY_APPROX_DP,
                          2,
                          0);

    Hull = cvConvexHull2(Approx,
                         Storage,
                         CV_CLOCKWISE,
                         1);

    HullPnts = cvConvexHull2(Approx,
                             Storage,
                             CV_CLOCKWISE,
                             0);

    Defects = cvConvexityDefects(Approx, HullPnts, DefStorage);


    CvConvexityDefect* DefPnts = (CvConvexityDefect*)malloc(
                                sizeof(CvConvexityDefect)*(Defects->total));
    cvCvtSeqToArray(Defects, DefPnts, CV_WHOLE_SEQ);

    CvPoint* Points = (CvPoint*)malloc(sizeof(CvPoint)*(Hull->total));
    cvCvtSeqToArray(Hull, Points, CV_WHOLE_SEQ);
    float Agg = 0;
    float Avg = 0;

    for (i=0; i < Defects->total; i++) {
      cvCircle(Out, (*DefPnts[i].depth_point), 5, CV_RGB(0,200,0), 3, 8, 0);
      Agg += fabs(DefPnts[i].depth);
    }

    if (Defects->total != 0)
      Avg = Agg/Defects->total;
    else
      Avg = 0;

    for(i = 0; i < Hull->total; i++) {
      cvCircle(Out, Points[i], 5, CV_RGB(200,0,200), 1, 8, 0);
    }

    free(Points);
    free(DefPnts);
    if (Avg > 30)
    cvDrawContours( CImg,
                    Approx,
                    CV_RGB(255,255,255),
                    CV_RGB(0,0,0),
                    -1,
                    CV_FILLED,
                    8,
                    cvPoint(0,0) );

    cvSubstituteContour( Scanner, NULL );
    }
    else
      cvSubstituteContour( Scanner, NULL );
}

cvReleaseMemStorage(&DefStorage);
cvReleaseMemStorage(&Storage);

cvCopy(Im, Out, CImg);
cvReleaseImage(&N);
cvReleaseImage(&CImg);
cvReleaseImage(&CRB);
cvReleaseImage(&Mask);
cvReleaseImage(&I);
cvReleaseImage(&Im);
return Out;
};
```