James Robertson
CPSC 4040 – 001

Interactive Projective Warper – Project Report

Overview

For my final project, I created an interactive image warping program that produces similar output to the “Transform -> Distort” function from Adobe Photoshop. The program can support up to 10 layered images as well as an alpha channel. Each layer can be individually selected and have its corners or entire screen position translated by the user via left clicking and moving the mouse. The image will then be projective warped based on the new corner positions. With all those features combined, a user can create an image composed of multiple other images that are distorted to fit a certain perspective.

Implementation

The program uses OpenGL, GLUT, and OpenImageIO to pull pixel data from image files and display them to the screen. Each image displayed on the screen is generated from a warped version of the raw pixel data, where the warped pixmap is generated via inverse mapping from a computed forward mapping matrix. Since the displayed pixels are always based on the current matrix, we can simply initialize a layer’s matrix as the identity matrix and update it as we move corners around.

The method I used to compute the forward mapping matrix can be found here: *http://graphics.cs.cmu.edu/courses/15-463/2008_fall/Papers/proj.pdf.*

In summary, we need to solve for 8 coefficients in the transformation matrix since a projective mapping has 8 degrees of freedom. We assume the remaining coefficient is 1 and solve the following 8x8 system via gaussian elimination. Variables *a-h* are our coefficients to solve, the xy-coordinates are the current positions of the output corners, and the uv-coordinates are the raw image’s corner positions.

$$\begin{pmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3y_3 & -v_3y_3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

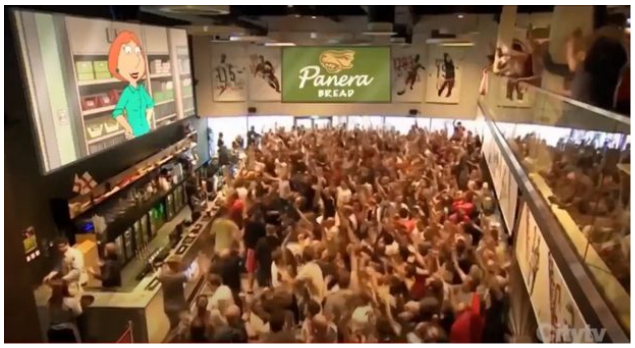
To simplify the computation of matrices and pixel positions, the open-source matrix library *Eigen* was used to store, manipulate, and solve matrices and vectors. The solution to the 8-component vector has its a-*f* coefficients swizzled into a 3x3 forward mapping matrix that we then use to inverse map and compute the warped pixmap during runtime.

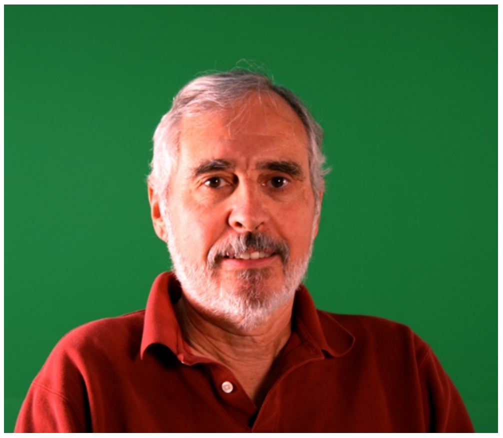
James Robertson
CPSC 4040 – 001

Results

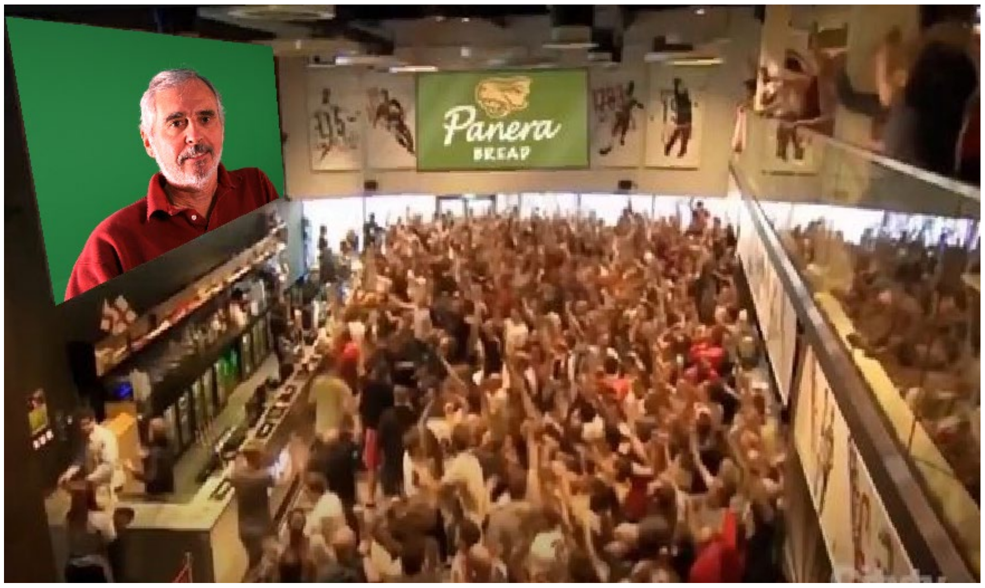
Below are some images composed solely using the projective warping program and premade images:

Input Images:





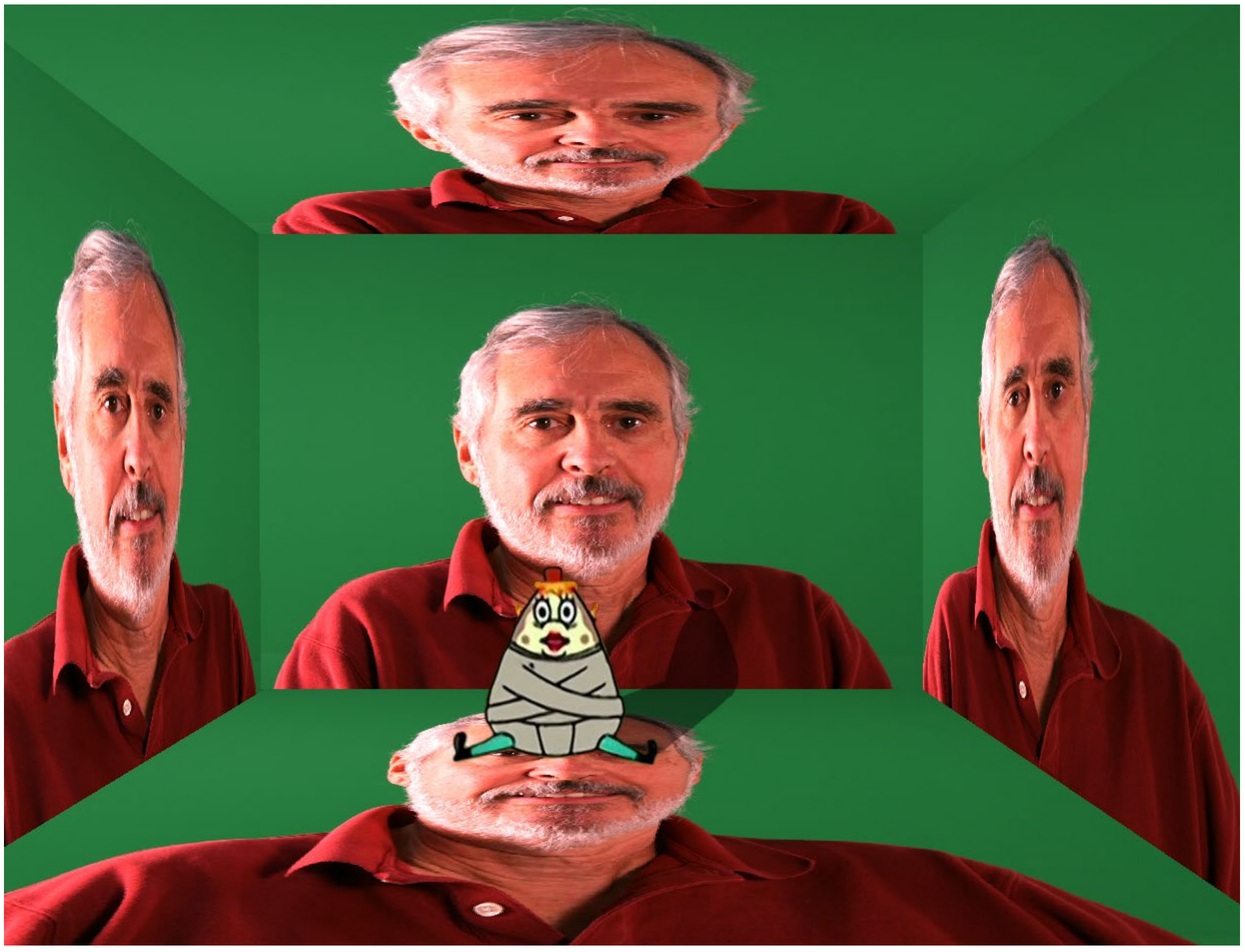
Custom Output:



James Robertson
CPSC 4040 – 001

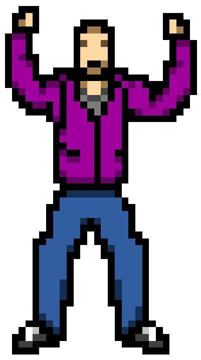
Input Images:

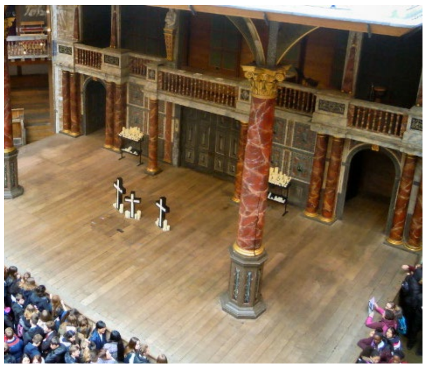
Custom Output:



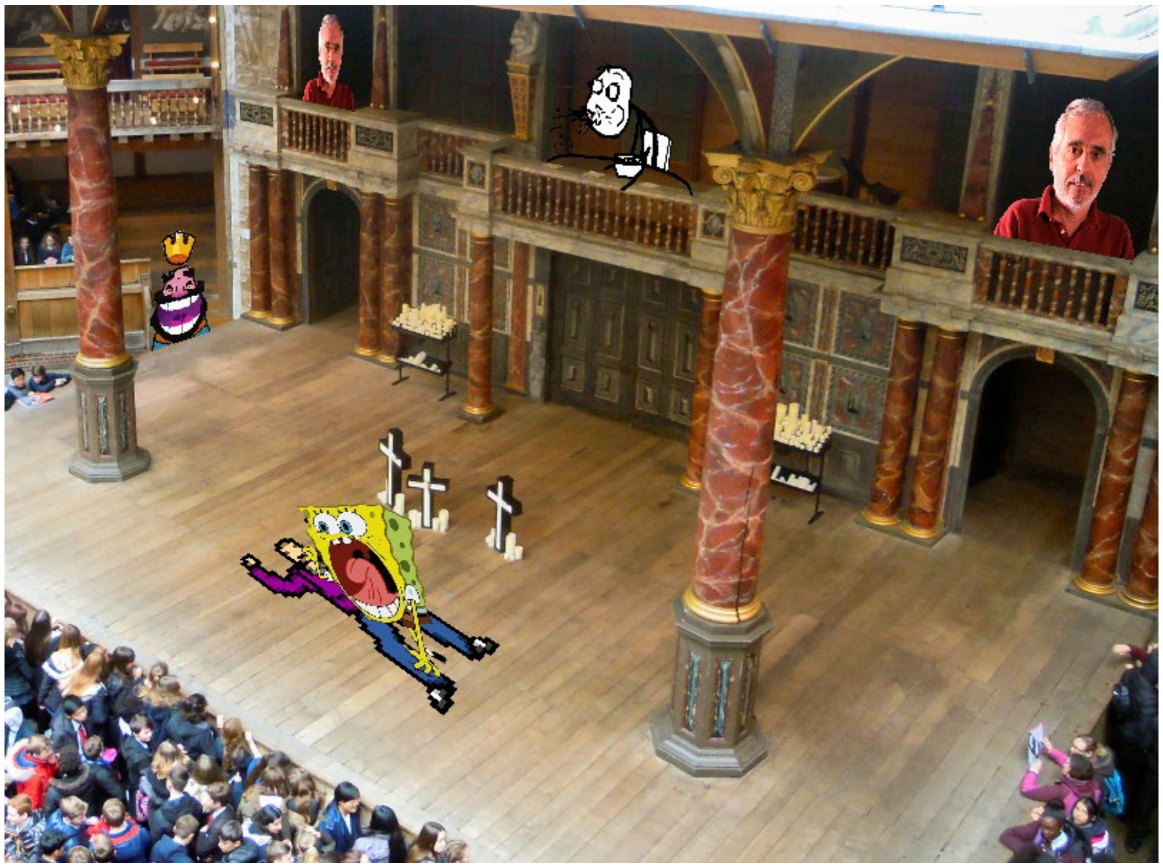
James Robertson
CPSC 4040 – 001

Input Images:





Custom Output:



James Robertson
CPSC 4040 – 001

Improvements:

If I was given the time to expand this further, there are a few key features I'd implement. For one, the current implementation is quite slow since we have to compute the matrix *and* inverse map every pixel every frame that a corner is moved. There's not much to do to get around this except to make a GPU accelerated implementation. This could be done with something like a CUDA/OpenCL kernel implementation, or possibly using GLSL shaders with OpenGL. Mapping the pixels to an OpenGL texture at runtime and doing a traditional render would also speed things up significantly, since I used the deprecated and rather slow function, glDrawPixels().

I'd also like to improve the workflow a little bit, and transfer most of the functions that are done using keyboard keys to UI elements instead. At a minimum, this could be accomplished with some of GLUT's menu/submenu features.