Tabletop Hologram

Jeremy McCulloch, Adam Potter, Sreya Vangara

Overview

Inspired by virtual reality games, we wanted to create an interactive projection that effectively optimized the characteristic speed of an FPGA to real-time render 3D objects for the perspective of a moving user. Using the Pepper's Ghost system of making simple projections, we aim to create a tabletop simulation of an object that is rendered from the perspective of a user. Eventually, the rendering will update following the user's movements, simulating a 3D object, and if possible, the user will be able to interact with the projection.

Minimum Functionality

Our minimum functionality goal is to produce a tabletop projection. This projection should automatically render with the correct dimensions based on an initial user perspective. This takes into account an assumed height and distance from the projection

Target Functionality

Our target functionality goal is to produce a realistic illusion of a 3D object. We will add user tracking with computer vision, and as the user moves around the projection, it should update its rendering so the model is oriented in the direction of the user. This will give the illusion of walking around a 3D object

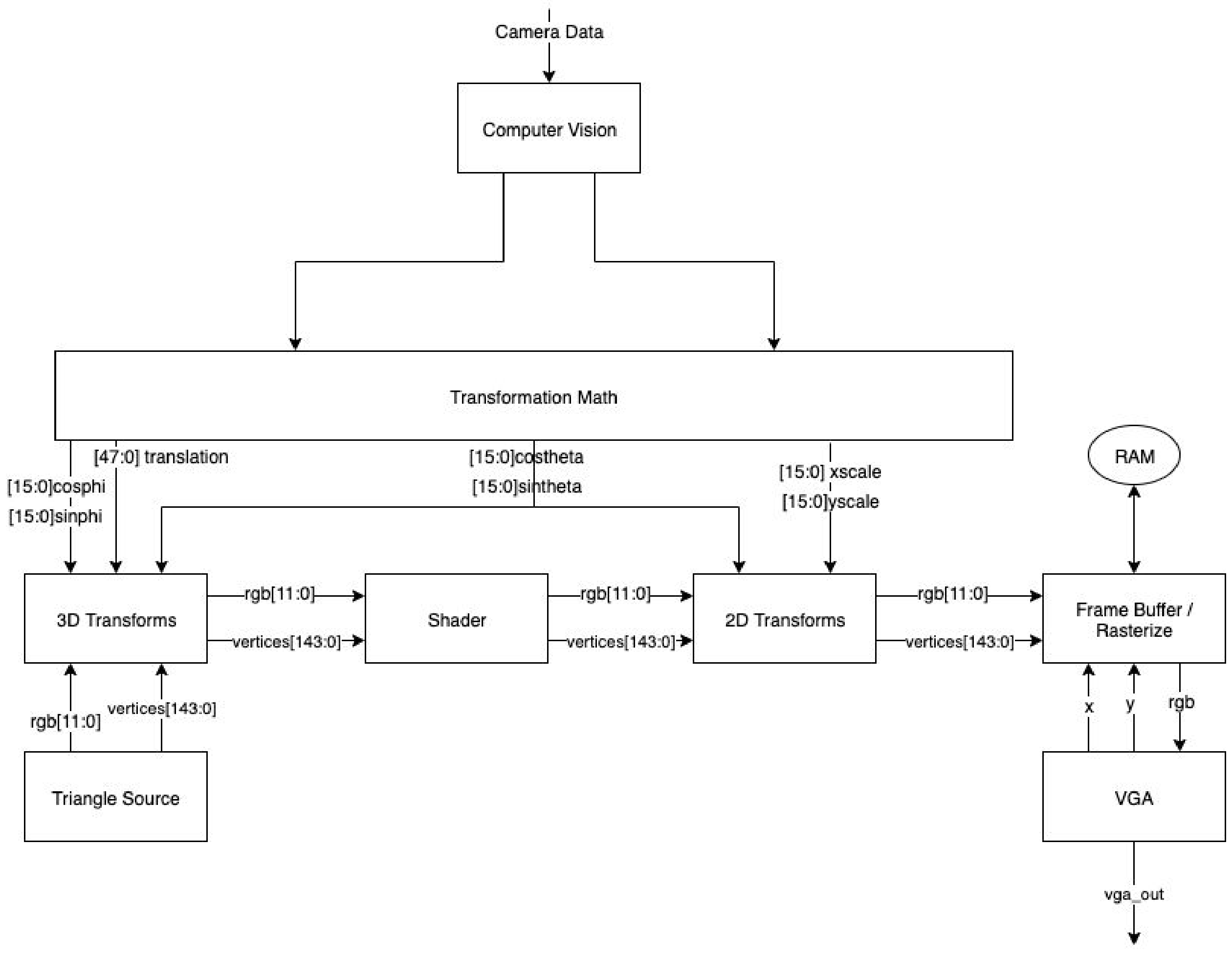
Reach Functionality

Our reach functionality goal is for the user to interact with the rendered object through computer vision. Using some object, light source, or gesture recognition the user could rotate or zoom the rendering of the object to interact with it.

OR

Our reach functionality goal is for the projection to move. Essentially, we would project a video, instead of a static object. This video would rotate with the user as they traverse around the table, adjusting for different angles. This creates the illusion of a three dimensional object because the user has access to many perspectives of the object.

Implementation (Block Diagram)



Computer Vision

The input of this subsystem will be data directly from the camera, and the output will be the x and y coordinates of the user's head. Before implementing this subsystem, we will prototype a computer vision algorithm in OpenCV that consists of dilations, erodes, and thresholds. This may require the user to wear some piece of headwear of a distinct color that makes the thresholding easier for the FPGA.

Shader / Transforms

The coordinate system for the model has x and y axes corresponding to x and y axes of the camera. The positive z axis is up in the world.

We perform a number of transformations on the model to render it. First, we translate and rotate it so that the user's eyes are at the origin and the user is looking along the negative z axis, positive y is down for the user, and positive x is right for the user. Next, we shade the model by multiplying the rgb value of each triangle by dot product of its normal vector and the direction the user is looking. Finally, we rotate the image so that it is facing the user and stretch it so it looks correct even though it is displayed on a flat table. These transformations are all two dimensional and do not modify the z coordinate.

3D Transforms

We first calculate the translation to be:
$\overline{v} = [-x, -y, -H]$ where x and y are the x and y coordinates returned by the camera and H is a tunable constant corresponding to the difference in height between the user's eyes and the table.

Next we calculate the rotation matrix to be the product of 3 matrices $R = R_1 R_2 R_3$ where
$R_1 = [0, 1, 0; 0, 0, 1; 1, 0, 0]$
$R_2 = [cos\phi, 0, sin\phi; 0, 1, 0; -sin\phi, 0, cos\varphi]$
$R_3 = [cos\theta, -sin\theta, 0; sin\theta, cos\theta, 0; 0, 0, 1]$
And
$\theta = -atan2(y, x)$
$\phi = acos(\frac{H}{\sqrt{x^2+y^2+H^2}})$

The angle theta corresponds to how much we have to rotate around the z axis so that the users eyes are in the xz plane. After this rotation is applied, the angle phi represents how much we have to rotate around the y axis so the users eyes are along the x axis. Finally, R1 permutes the axes so that the user is looking along the negative z direction, positive x is to their right, and positive y is up for them.

Shader

To calculate the normal vector for a triangle we find $\widehat{n} = \frac{(\overline{v_2}-\overline{v_1}) \times (\overline{v_3} - \overline{v_1})}{|v_2-v_1| \cdot |v_3-v_1|}$. The direction the user is looking is $-\widehat{z}$, so we care about $\widehat{n} \cdot \widehat{z} = \frac{(x_2-x_1)(y_3-y_1) - (x_3-x_1)(y_2-y_1)}{|v_2-v_1| \cdot |v_3-v_1|}$. Since this step requires many products and a square root, it may need to be buffered. Once we find $\widehat{n} \cdot \widehat{z}$, we use this to scale each component of RGB.

2D Transforms

Affine Transform:
$y' = \frac{y}{sin\phi}$, $x' = x$
$A = [1\ 0;\ 0,\ \frac{1}{sin\theta}]$
Rotation: rotate by angle $-\theta$
$R = [cos\theta, sin\theta; -sin\theta, cos\theta]$
We neglect the distortion due to perspective. We can consider this if we see that it is significant.

Frame Buffer / Rasterize

We create a frame buffer which is the size of the projector and stores rgb values and z coordinates. This allows us to determine which triangles are closest to the user and to render those.

To rasterize a triangle, we compute the following for each point p = (x,y) that is between the smallest x and y coordinates and the largest x and y coordinates.
$A_i = \frac{(v_{i+1} - v_i)\times(p - v_i)}{(v_2 - v_1)\times(v_3 - v_1)}$
If $A_i > 0$ for all i, then the point is within the triangle.
In this case, we calculate $z = A_1 z_1 + A_2 z_2 + A_3 z_3$ and use this to calculate a z coordinate and an rgb value.
Finally, if the point is in the triangle, we compare z to the z coordinate stored in the framebuffer. If the new z coordinate is greater, we replace the z coordinate and rgb values with the new one we calculated.
Each pixel requires two clock cycles since we both read from memory and write to it, which will limit our frame rate.

External Components

Item	**Source**	**Cost**
Camera	6.111 stockroom	$0
VGA Projector	MacGregor	$0
Projector and Camera Mount	Wood structure with laser cut plates for mounting or 80/20 structure	$0-$35 (can use spare materials from shops)

Hardware Limitations

FPGA Memory: There is a chance that we run out of memory and gates while implementing the logic and model rendering, however based on previous projects, for example 3D Pong, it seems probable that we can accomplish our outlined project without fully exhausting our Nexys 4 DDR.

FPGA Pipeline Lag: Iterating over the entirety of an image multiple times may result in more clock cycles than fits in 60 FPS. However, with pipelining this problem turns into reaction lag. If the pipeline is too long then the lag between the user moving and the projector reacting with be noticeable. Preferably this lag is kept below 250ms.

Projector Focal Length: Many commercial projectors are made to operate between 40” and 120” at a 16:9 ratio. At each distinct projection size based on distance from projected surface, the projector needs to be refocused by changing the position of a lens. This means our projector can only produce an image so small before we have to reduce the size of the input image and lose resolution.

Projector Resolution: If the projected object is too small is reference to the full frame of the projector, the resolution might make rendered details blurred and quality of the hologram is lost.

Projector Brightness: Few projectors project more than 4000 lumens onto a screen meaning that for the hologram to be fully visible, the ambient light will need to be lowered. This doesn’t require all the lights in a room to be off, but light should at least be dimmed.

Camera Resolution: Depending on the lens and the mounting location of the camera, the resolution may prohibit how small of an object can be detected. This is unlikely for the user’s head, but may make smaller sources for interacting with the object, like gestures, difficult to detect.

Camera Frame Rate: As the computer vision updates the location of the user’s head, the delay will be at least as long as the period of the camera frame rate (30 fps = 33ms delay). This also limits the speed at which objects can move in the camera’s reference frame and still be detected, though we don’t predict that to be an issue.