3D RENDERING IN FPGA

CSEE 4840
Embedded System Design
Spring 2014

Earvin Caceres (ec2946)
Gautham Vunnam (gv2226)
Garvit Singh (gs2731)
Anjana Ramesh (ar3303)
# TABLE OF CONTENTS

**Introduction** ................................................................. 2

**System Overview** ................................................................ 4
Software Overview ................................................................ 5
Hardware Overview ............................................................. 5

**3D RENDER** ............................................................................. 5
Shader Module ........................................................................ 6
Algorithm ................................................................................. 6
Hardware Implementation ..................................................... 10
Hardware Designs ................................................................. 12

**MEMORY TESTING AND OPTIMIZATION** ................................. 13

**SOFTWARE MODULES** ........................................................... 16
Software Prototype ............................................................... 16
Controller Specification ......................................................... 16
Matrix Transformation Pipeline ............................................. 22
Driver for interfacing with Hardware .................................... 25

**CONCLUSION AND RESULTS** .................................................. 27
Challenges Faced ................................................................. 27
Lessons Learnt ................................................................. 27
Role of each individual ........................................................ 27

**REFERENCES** ................................................................... 28

**APPENDIX** .......................................................................... 29
SOFTWARE FILES ................................................................. 29
HARDWARE FILES ............................................................... 45
SOFTWARE PROTOTYPE FILES IN PYTHON ......................... 72
Introduction

Our initial plan was to implement a Mario party game - The Ultimate Ball Balancer. The game involved the concept of a 3D plate on screen, with a ball on it. The user could use the controller to balance the ball on the plate. Through the course of the project, our focus shifted. We realized the difficulty of 3D Rendering, especially using Z-buffering and Rasterization mechanisms. We then decided to focus on this concept, and get it working.

The project aims to provide 3D Rendering of any object in the FPGA. Using the PS3 Controller, it is possible for the user to rotate the 3D object. For this, we have written a bunch of software and hardware modules. The project takes care of user inputs via a PS3 Controller. The final result seen by the user is the movement of a 3D plate based on the direction of the controller input. Given below, is the design flow for the 3D Rendering project.

Figure 1: Design flow of 3D Rendering
System Overview

The high-level system overview is shown in Figure 2. The system takes input from the PS3 controller, and the output of the entire system, is the 3D object rotating on screen. The controller inputs are between 0 and 255. The display is implemented for a screen with pixel dimensions of 640x480. The hardware shader module involves a series of fixed-point arithmetic. We got a precision of 16 bits for the result at each step.

Figure 2: System Overview
Software Overview

The software components handle the controller input, its translation into the angle of rotation, which is fed as an input to the Matrix pipeline block. A device driver is written to carry out the software interfacing with hardware.

The software part typically consists of the following components:

- Software prototype
  - We completed the whole project on software before starting on Hardware. This was to ensure we got the mathematics of the transformations correct. Blender is used to draw the 3D model of our choice on software. It generates the vertices of all the triangles that make up the model.
- PS3 Controller
  - Interface the controller, and mapping of the input from the controller to appropriate angle of rotation that is fed into the Matrix Transformation module.
- Matrix Transformation/Pipeline
  - The mathematical calculation of the model that takes angles as inputs, and transforms them into 2D coordinates.
- Software interfacing
  - Software driver to communicate with the hardware.

Hardware Overview

The hardware component handles the crux of our project, which is the Shader and Rasterization module. To make the object seamless while rotating, Z-buffering was implemented. A frame buffer was used to store the 640x480 pixel values on screen, and interface with the VGA module.

- 3D Rendering of the model
  - Shader module - Takes the transformed 2D coordinates, and communicated with the VGA module to print the object on screen.
  - Z-buffering - Fine tuning the object seen on screen, by considering the Z-axis, and how it affects an object when it rotates.
- VGA Module – This block takes pixels as inputs. It performs Rasterization (not the conventional way; explained in detail in the 3D Render module) and displays the object on screen.
3D RENDER

Shader Module

Overview
The purpose of the shader module is to rasterize each triangle of the 3D object being rendered. The multiplication pipeline outputs three vertices representing the points of a triangle that have been projected onto a 2D screen. It determines which pixels reside within that triangle and calculates the depth of each pixel to compare against the depth of the old pixel at the same location. If the new pixel has less “depth” than the existing pixel it will be output to the screen. This preserves the “depth” effect of 3D objects when rendered onto a 2D screen. Multiple triangles are then output to display the object that is being modeled.

This section describes the algorithm used to implement the shader as well as the architecture of how it was implemented in the FPGA.

Algorithm

Overview
The algorithm we used was borrowed from the tutorial in reference [1] and uses a simple method of linear interpolation to rasterize a triangle. The basic idea is illustrated as follows.

Given three points of a triangle, the purpose of this algorithm is to determine the outline of the triangle as well as fill in the pixels horizontally from left-to-right until the entire triangle is shaded in. The depth of each pixel needs to be calculated and compared against an old pixel in that same location to determine whether the pixel needs to be drawn or not.
Sorting

The algorithm starts with 3 vertices P1, P2 and P3, each with (x, y, z) coordinates. The vertices need to be sorted with respect to their y coordinate so that you have either of the two triangle configurations shown below:

![Figure 4: Triangles for rasterization, draw line](image)

Note the difference in location in P2 between Case 1 and Case 2. The reason why this is important is because lines are drawn horizontally from left to right so it’s important to know whether P2 resides to the left or right of the triangle.

To determine which case is being rasterized, the inverse slopes dP1P3 and dP1P2 are calculated and compared. If dP1P2 > dP1P3 then the triangle is in case 1 with P2 on the right. Alternatively, if dP1P2 < dP1P3, the triangle is in case 2 with P2 on the left.

Note that rasterizing the triangles is a two-step process where the triangle is divided into two as shown in case 2 above. Step1 in the figure is rasterized first followed by Step 2.

Gradients and Interpolation

Case 1, with P2 to the right of the triangle, will be used to illustrate the remaining aspects of the algorithm. The same process can easily be applied to case 2.

As mentioned, the triangle is filled horizontally from left-to-right and line-by-line in the positive y-direction until the triangle is filled. The endpoints of each line of the triangle are denoted by “sx” and “ex” in the figure above so we want to be able to determine “sx” and “ex” for each y-coordinate that spans the entire triangle. Furthermore, you can determine “sz” and “ez” which represent the z-value (depth) of the start and end points. Given P1, P2, P3 and location y that will span the entire height of the triangle, we can interpolate the coordinates (x, y and z) for every pixel that makes up the given triangle.
First we define the x, y and z coordinates of P1 P2 P3 as follows

P1 \rightarrow p1x, p1y, p1z
P2 \rightarrow p2x, p2y, p2z
P3 \rightarrow p3x, p3y, p3z

Y \rightarrow \text{current height of the triangle}

Note that Y = p1y when we first start and traverses the triangle until Y = p3y. Starting with Step 1 in case 1 above, the gradient with respect to the current Y value has to first be calculated. It is given as follows

$$gradient = \frac{Y - p1y}{p2y - p1y}$$

Given a value Y, you can determine how close you are to P2 from P1 in terms of a percentage. In other words, if you’re at the starting point and Y = p1y, the “percentage of distance” you are at between P1 and P2 is 0. Likewise if Y = p2y, you’re at the end of the triangle for Step 1 and the percentage of distance is 1 (i.e. 100%). As such, gradient will be a value between 0 and 1.

After calculating the gradient for a specific value Y, you can determine sx and ex as follows

$$sx = p1x + (p3x - p1x) \times gradient$$
$$ex = p1x + (p2x - p1x) \times gradient$$

Note that the z-values (depth) of the starting and ending pixels must be interpolated as well by applying the equation

$$zs = p1z + (p3z - p1z) \times gradient$$
$$ze = p1z + (p2z - p1z) \times gradient$$

Note that for stage 2 of the triangle, the equations change as follows

$$sx = p1x + (p3x - p1x) \times gradient$$
$$ex = p2x + (p3x - p2x) \times gradient$$
$$zs = p1z + (p3z - p1z) \times gradient$$
$$ze = p2z + (p3z - p2z) \times gradient$$

This process lets you calculate sx, ex, sz and ez to process and potentially draw every line in the triangle. This same process can be easily extended to Case 2 but must take into account the location of P2 (to the left of the object).
Scan Line

As described in the process above, you can determine sx, ex, sz and ez for every line (i.e. Y value) in your triangle. Without z-buffering, you can simply draw a line between sx and ez to fill in the entire triangle.

Z-buffering however requires that you do further interpolations to determine the z-values of each pixel in the line being processed. This is done similarly to before where for each x value, a gradient is calculated, followed by a linear interpolation using the zs and ze values calculated for the current line

\[
\text{gradient} = \frac{x - sx}{ex - sx} \\
\text{z} = zs + (ze - zs) \times \text{gradient}
\]

Conclusion

This rasterizing algorithm allows you to identify the location of the pixels within a triangle defined by 3 vertices. It also allows you to determine the z-value of each pixel in the triangle to implement z-buffering.
Hardware Implementation

Overview

The block diagram for the shader module is shown below. Each block 1-5 is described briefly and the code is commented for further detail.
An overview of the hardware is as follows:

1. mem_IO – This is a simple memory mapped I/O interface that is connected to the Avalon bus. This is the main HW/SW interface. It is used to pass vertices from the software multiplier pipeline to the shader module. It also passes color as well as control signals for hiding the screen during frame updates and clearing the screen which includes clearing the frame buffer.

2. shader – This is the main shader module. It accepts vertices from the memory mapped I/O interface. It sorts the vertices and calculates dP1P2 and dP1P3 as described in the algorithm to determine which triangle configuration we have (i.e. Case 1 with P2 on the right or Case 2 with P2 on the left). Once determined, for each Y coordinate of the triangle from top-to-bottom, the shader passes the sorted pixels to the draw_line module which contains the logic to draw the line on the screen with the help of the put_pixel and VGA_framebuffer blocks.

3. draw_line – This module takes sorted pixel values and a specific Y coordinate and outputs sx, ex and zs, ze as described in the algorithm above. Once calculated, it sends these values to put_pixel and signals put_pixel for the next stage in the pipeline.

4. put_pixel – This module accepts the sx, ex, zs and ze coordinates as well as the current Y coordinate and does the interpolation to calculate the z-values for each pixel. Each pixel along with its z value is sent to the VGA_framebuffer module to be printed to the screen.

5. The VGA_framebuffer module contains the frame buffer and z-buffer memories. It takes pixel coordinates, x, y and z, along with a pixel color and start signal to determine if a pixel should be written to the frame buffer and therefore output to the screen. Before writing a pixel to the frame buffer, it checks the z-buffer to see if the current pixel being written should overwrite the existing pixel at the same location. The VGA_framebuffer module also contains logic to handle two signals called hide_screen and clear_screen that are passed in from software from the Avalon bus.
Hardware Designs

Number Representation

We used our software prototype to confirm the types and resolution of the numbers in our application. This table lists the fixed-point representation of the data we used throughout the hardware.

<table>
<thead>
<tr>
<th>Value</th>
<th>Signed/Unsigned</th>
<th>Range</th>
<th>Total Bits</th>
<th>Integer Bits</th>
<th>Fractional Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel/Vertex/Coordinate</td>
<td>Signed</td>
<td>[480, 640]</td>
<td>16</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Gradient</td>
<td>Signed</td>
<td>[-1, 1]</td>
<td>16</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>z-value</td>
<td>Unsigned</td>
<td>[11,12]</td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Pixel color</td>
<td>Unsigned</td>
<td>[0, 3]</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Memory Budget

<table>
<thead>
<tr>
<th>Module</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framebuffer</td>
<td>65536</td>
</tr>
<tr>
<td>z-buffer</td>
<td>360448</td>
</tr>
</tbody>
</table>

The following peripheral blocks were also created to support the pipeline operations:

- divider – Divider that does fixed point signed division
- divider14 – Higher resolution divider that does fixed point signed division with 14-bits used for calculating gradients
- interpolate – does interpolation functionality as described in the algorithm section
- Interpolate7 – higher resolution interpolate function used for the interpolated z-buffer values. The result is a fixed point number with 7 bits used for the fractional part
MEMORY TESTING AND OPTIMIZATION

We started with DDR3 for our memory module, as our initial calculation for rendering the 3D graphics needed a lot of memory.

62 vertices → 48 bits per vertex = 720 bytes
120 faces → (3*6) bits each face = 120*18 = 270 bytes { 6 bits for using the vertex index, as to which vertices form the faces }
640*480 Framebuffer and 16bits to for pixel color(use R & G) → 640*480*16 = 616KB
Z buffer 11bit for each pixel → 11*640*480 = 423KB

After this mad calculation we realized we had to use DDR3.
Below is how we progressed through the testing and implementation stages.

We spent lot of time figuring about the DDR3 specification. The module below shows the structural coding that we came up with after implementing the DDR3 design in QSYS.
Problems we ran into while working with DDR3
We started with DDR3 but we soon realized that it was tricky to get it working. We might have used two masters, one on the HPS and the other on the FPGS side to get it working. We also tried to generate the controller module for the DDR3 controller using megafunction wizard, but even after 8-9 hours the wizard was not able to produce the controller module.
We also felt that the timing synchronization would have been tricky with the DDR3 controller. We were looking into fast slave but we haven’t used fast slave in the course so far and getting it to work with all the specifications would have been cumbersome.

Implementing using on-chip registers+megafunction
So we switched to on-chip memory registers. We cut down the details of the design to save on the memory. We removed the number of vertices, faces that we had in our 3D model to save on the memory. We ran it in the software prototype(blender) with the reduced details and the figure was smooth enough, so we decided to go ahead with it.

We tested if throwing in values on the on-chip memory registers could work for our design and we concluded we could go with it. We tested if we were able to address the memory registers this way and read and write those registers, also we took help from professor’s slides where he implements framebuffer in similar fashion.
Initially the modules that we implemented showed high memory utilization of about 80-85%. We even ran into memory shortage while compiling the modules few times. We figured out megafuncton optimizes the memory utilization by somehow using memory registers in a better manner. So we then implemented our modules including the Z buffer using megafunction. Modules that we compiled with this design in Quartus had comparatively low memory utilization (60%).
SOFTWARE MODULES

Software Prototype

Our initial proof of concept for the project was verified on Software. We used Python to write, verify the entire shader module. The basis for the hardware was this software prototype, and replicated the shader module on hardware, and compared it with software. We cannot reiterate enough how important Software prototyping and ModelSim (to test hardware design) have been to our project.

The software prototype was used with the help of this Tutorial. The tutorial was in C/Java. We implemented it on Python.

Controller Specification

Game controller

The original ball balance game by Nintendo used the accelerometer on its devices to control gameplay. It is the most intuitive way to play the game, compared to using thumb sticks or D-pads, as it helps perfectly map the plate being displayed onscreen with the device the player is holding.

We set out to find a controller that met the following requirements for use as an input device:

- Built in accelerometer
- Connects via USB
- Has a reasonable amount of documentation online

The Wii remote was not ideal even though it had tons of documentation online, as it used Bluetooth to communicate. We figured that writing Bluetooth drivers from scratch would be quite tedious and take too much time away from the core part of the project. At the same time we didn’t want to completely rely on packages provided online, as we didn’t have a full understanding on how they worked.

So we settled for…
This controller seemed to have met all our requirements. However, we later realized that the documentation for the controller was actually quite poor for our case. While we wanted to write a driver from scratch, most of the documentation online for Linux involved using complex joystick packages. Considering that we wished we had better documentation ourselves, we hope that this section will help any future hackers implement a simple software package that can read from the Sony DualShock 3.

**Quirks**

The controller will not automatically report any events (such as button presses) upon being plugged in, while sniffing the USB packets. The controller needs to be changed to an “operational” mode using a HID_REQ_GET_REPORT command.

Windows, Mac OS X and Linux with kernels >= 2.6.21 have the necessary drivers to talk to set it to “operational” mode and talk to it. However, the Linaro distribution does not have the necessary module to do this (even though it has a kernel >= 2.6.21).

We found out that our version of the Kernel was compiled without many of the useful HID modules. The specific module that is required is ‘hid-sony.c’. Upon getting more documentation from Professor Edwards on how to compile our own Kernels, we went through ‘menuconfig’ and selected the module under ‘Device Drives → HID → Additional Drivers’.

Now, we can start reading from the controller.
Now, we needed to know what data the controller was sending to us. We need to do two things to accomplish this:

1. We need to know what the Device ID was
2. How to sniff the packets coming out from it

1) We installed a very useful utility called ‘lsusb’ on our distribution and used it to determine the controller’s Device ID.

2) We used a program called ‘USBTrace’ on Windows to determine what the ‘SetupPacket’ was to start receiving data from the controller by connecting it up to the software.

We then utilized a Python library called PyUSB to allow us to talk to the controller through the USB protocol without too many lines of code. All we needed to do was send the ‘SetupPacket’ to the controller and let the controller inundate us with status data.

The packets contain a wealth of information, from accelerometers to pressure sensitivity of every button press.

**Snippet of the Python Code**

```python
import usb.core
import usb.util
import sys

# Scanning the tree for our DualShock 3 controller
controller = usb.core.find(idVendor=0x054c, idProduct=0x0268)

reattach = False
if controller.is_kernel_driver_active(0):
    reattach = True
    controller.detach_kernel_driver(0)

# Error message for device if not found
if controller is None:
    raise ValueError('Device not found')

# Set the active configuration. With no arguments, the first configuration will be the active one
controller.set_configuration()

while True:
    # Our “SetupPacket”
```
ret = controller.ctrl_transfer(0xa1, 0x1, 0x0101, 0x0, 0x31)

# Prints out all the packets coming out from the controller
print ret

Output

Mapping

Once we could see the statuses generated by the controller, we identified values in array 41, 42, 43 and 44 of ‘ret’ as being the relevant data for X and Y axis tilt. The values went from 0 – 110 for a 90-degree tilt on the X-axis in the anticlockwise direction, and from 255 – 142 for a 90-degree tilt in the opposite direction.

These values are in array indices 42 and 44. Array indices 41 and 43 indicate the quadrant in which the rotation is taking place for either axis, and can be either ‘1’ (clockwise) or ‘2’ (anti-clockwise).

Snippet of Python code to convert readings to degrees

# Check for anti-clockwise rotation on the x-axis
if ret[41] == 2:
    if ret[42] <= 110 and ret[42] >= 0:
        xaxis = ret[42]*90/110

        # Limit maximum to 90 degrees
        if xaxis > 90:
            xaxis = 90

# Check for clockwise rotation on the x-axis
if ret[41] == 1:
    if ret[42] <= 255 and ret[42] >= 142:
xaxis = (ret[42] - 255)*90/113

    # Limit maximum to 90 degrees
if xaxis < -90:
xaxis = -90

# Check for anti-clockwise rotation on the y-axis
if ret[43] == 2:
    if ret[44] <= 110 and ret[44] >= 0:
        yaxis = ret[44]*90/110

        # Limit maximum to 90 degrees
if yaxis > 90:
yaxis = 90

# Check for clockwise rotation on the y-axis
if ret[43] == 1:
    if ret[42] <= 255 and ret[42] >= 142:
        yaxis = (ret[44] - 255)*90/115

        # Limit maximum to 90 degrees
if yaxis < -90:
yaxis = -90

\textit{Interface to Matrix Multipliers}\\

The tilt data in degrees is then fed to C code to use for matrix multiplications on the plate’s vertices. This is very easy to do using Python’s OS module. In this instance, the Python script acts as a wrapper to bridge the controller output to the FPGA through device drivers.

\textit{Snippet of Python Code}\\

\# Passing arguments to the C executable using the command line interface
os.system("./main " + str(yaxis) + " " + str(xaxis) + " " + str(stop) + " " + str(prev_stop))

\textit{Additional Features}\\

Due to the nature of our VGA framebuffer design, the plate wasn’t being displayed perfectly due to glitches in timing. In order to showcase our implementation of Z-Buffering in hardware, we used the ‘X’ button on the controller to freeze a particular render on the screen to be admired.

To do this, we just checked for value ‘64’ on the 3rd array index of ‘ret’. It is ‘0’ when not pressed and ‘64’ when pressed.
**Suggested Improvements**

If we had a little more time in our hands, we would have loved to take full advantage of the Bluetooth functionality on the DualShock 3. There are currently many software packages that enable pairing using Bluetooth on Linux, such as QTSixA and Sixad.

In general, initial pairing can be done over a USB cable. After the connection is established, one can read data from a Bluetooth USB dongle using a HCI module in the Kernel.
Matrix Transformation Pipeline

A 3D object is modeled using a collection of vertices in 3D space (x,y,z). The vertices can be connected to form the surfaces of the 3D object. The surfaces can be filled in using any number of methods according to the model. To project a 3D object onto a 2D screen, the model’s vertices undergo a series of matrix multiplications to transform the 3D vertices into 2D coordinates that can then be displayed on the screen. The rendering pipeline we will use for our project is shown in Figure 3.
In our model, the vertices are represented using four coordinates \((x, y, z, w)\). The addition of the ‘w’ coordinate makes these Homogeneous coordinates, which are used to simplify the operations, involved for 3D rendering. The ‘w’ can be used for other rendering purposes such as shadows, which will be out of the scope of this project. Here we describe each of the coordinate blocks in the pipeline and the matrix operations that transform the vertices from one domain to another.

i. The Model Coordinates

Each surface in the 3D model to be displayed is considered to be a collection of triangles. The triangles are defined by a set of vertices. To start with, the \(x, y, z\) coordinates are defined relative to the center of the object which is a vertex at \((0,0,0)\). We want to be able to rotate, translate and scale our model. Each of the above transformations requires a matrix to be multiplied with the vertices. The model matrix is a product of the following matrices; the 3 rotation matrices (for \(x\), \(y\) and \(z\) axes) scale and translate matrices.

ii. The World Coordinates

After model matrices are applied to the vertices, the model went from being in Model Space (all vertices defined relative to the center of the model) to World Space (all vertices defined relative to the center of the world).

To understand the view matrix, consider a camera moving around your world space. If the camera looking at the entire world rotates 3 units to the right, it is equivalent to your world space rotating 3 units to the left. The view matrix defines which point in world space is being viewed. In other words, it describes the direction the camera is pointed in. In our project, we plan on keeping the camera/view fixed while moving the object/world.

The method we will use to generate the view matrix is the “UVN Camera”. The details of this method can be found in [1] and [2].

The \(U\), \(V\) and \(N\) vectors that make up the view matrix are described.

\[ \text{N} - \text{The vector from the camera to its target. Also known as the 'look at' vector in some 3D literature. This vector corresponds to the Z-axis.} \]

\[ \text{V} - \text{When standing upright this is the vector from your head to the sky. If you are writing a flight simulator and the plane is reversed that vector may very well point to the ground. This vector corresponds to the Y-axis.} \]

\[ \text{U} - \text{This vector points from the camera to its "right" side". It corresponds to the X-axis.} \]

For our project, we plan to place the plate object along the \(z\)-axis so that the movement of the camera will be minimal. The world coordinates are transformed into the view coordinates (all vertices defined relative to the camera) after multiplying the vertices by the view matrix.
iii. The View Coordinates

Once we have the view coordinates, we need to convert the 3D Coordinates into 2D Coordinates that represent the 3D object being projected onto a 2D screen. This requires another matrix transformation. The matrix generated at this step is called the Projection matrix. It defines the window that the 3D coordinates will be projected onto. There are 6 main parameters: near, far, left, right, top and bottom. Near and far are used to specify the camera’s field of view, that is, how near and far the camera is able to see. Left, right, top and bottom represent the boundaries of the display in their respective directions. Note that the left and right boundaries are dependent on the angle of view of the camera, which can be specified in the software. The top and bottom boundaries are dependent on the aspect ratio of the display. These parameters are listed in the equations below and the dependencies are reflected in these equations.

\[ n = \text{near} \]
\[ \text{size} = n \times \tan \left( \frac{\text{angle of view}}{2.0} \right) \]
\[ \text{left} = -\text{size} = l \]
\[ \text{right} = \text{size} = r \]
\[ \text{bottom} = -\text{size}/\text{aspect ratio} = b \]
\[ \text{top} = \text{size} / \text{aspect ratio} = t \]

See references [3] and [4] for details on how the projection matrix is derived.

iv. The 2D Coordinates:

After the projection transformation, we have the 2D space vertices, which can be used for displaying the model on screen. To fill in the model’s surfaces, we use a basic rasterization algorithm that includes z-buffering. A set of three vertices represents a triangle, which is the primitive shape that makes up the surface of an object. Figure 4 shows a sample triangle defined by vertices P1, P2 and P3.

![Figure Triangle defined by 3 vertices](image)

The three vertices allow us to calculate the gradient of lines P1-P2 and P1-P3. Using these gradients, we can determine sx and ex, which represent the start location and end location of the triangle, for each value y. Similarly, we can calculate a sz and ez which represents the start and end z-coordinates that correspond to the sx and ex values. Given these values, we can determine the boundaries of the triangle that needs to be rasterized.
The addition of the z coordinate can tell us if particular pixels should reside in front of other pixels, which maintain the 3D look of the object when projected on the screen.

**Driver for interfacing with Hardware**

For every clock cycle, we are writing a set of x, y and z coordinate values to the ioctl. Our driver is written as an FSM that is using states to trigger what has to be done when. The refresh rate of the clock for the pixels is 50 MHz.

We tested the software-hardware interfacing by performing the following steps, considering a simple cube as our object to be rendered:

- We hard coded one set of vertices (obtained from the Python prototype). When this was passed to the hardware, a single triangle was displayed on screen.
- When we tried two triangles, there was no issue either.
- The interesting thing was, when we tried displaying three or more triangles, only the first and last triangles were displayed.
- We had to add a sleep cycle of 500ms, as you will see in our program, to let all the triangles be displayed on screen.

Once a cube was rendered correctly, we tried our luck with the plate, and it worked! We had a plate on screen. But there were a few color issues of the triangles. This is when we decided to implement Z-buffering on hardware, and make the plate look clean and smooth.

Next, when we tried including the controller into this picture, we were facing another problem with respective to synchronization. The pixels were being transferred faster than the time taken for the VGA block to read it. To solve this issue, we had to add additional sleep cycles on software.

**Pixel Values**

After the rotation values are provided by the Python wrapper and consumed by the matrix pipeline, transformation matrices are generated and are applied to every vertex as listed in vga_led.h.

Each face points to three vertices, and each vertex now has three pixel values after transformation. Thus we create a single dimensional array (‘pixellist’) of size (NUM_OF_FACES)*9 and store every pixel value, so that it becomes easier to access and send down to the hardware.

Since the hardware uses fixed-point arithmetic, we scale each pixel by 32, so that a good amount of precision is preserved as an integer value and then will be shifted appropriately.
Control Logic

For each face to be printed on the screen, we dump 9 indices from ‘pixellist’. These 9 indices correspond to 3 vertices as described about. However, we added 3 additional control signals, taking the total number of values sent to the Avalon Bus to 12. The three control signal are the following:

1. Colour: We need to print a different colour for each face so that we get a perception of depth, which is important for a 3D representation of an object. 2 bits represent this signal, to represent 4 different colours. We have 3 different shades of blue, from light to dark and black, to clear screen.

2. Show/Hide Screen: We want to wait till all the polygons have been printed to the framebuffer before displaying it. This is largely for aesthetic reasons, as watching a render face by face can be jarring. A single bit represents this signal.

3. Clear frame: Since we have implemented only one framebuffer, we need to clear it before we can write a new face. So we essentially resend the same render but paint everything black, so that it gives the same effect as a cleared screen. Unfortunately, this causes our renders to flicker. A single bit represents this signal.
CONCLUSION AND RESULTS

Challenges Faced

- Screen refresh
- Fixed-point, signed arithmetic in FPGA
- Z-buffer implementation due to resolution
- Limited memory resources, difficult to get DDR3 working
- Coloring of the triangle in the 3D model
- A race against the clock

Lessons Learnt

- Plan well in advance
- We ran into quite a few issues with the external memory. On-chip registers are much easier to implement, but difficult to optimize. Thank god for MegaFunction
- Compiling on FPGA is time consuming. We’ve never appreciated ModelSim more for making our lives so much easier
- Software prototyping was invaluable
- Priorities change as project progresses
- Get help from other groups! We tried doing everything on our own, but could have benefited from others’ work
- Strategize properly; make the wise choice when you have options! We underestimated a lot of our choices, for example with memory, using rasterization Vs. ray casting
- We stopped work on a lot of modules, as and when we realized they were infeasible

Role of each individual

- Earvin Caceres – Software prototype of the whole project in Python, 3D Rendering on Hardware (Shader module, VGA Module, Rasterization and Z-buffering)
- Gautham Vunnam – Kernel modification, Controller interfacing, Software translation, Matrix Transformation Pipeline
- Garvit Singh – Memory configuration, testing, and optimization
- Annjana Ramesh – Software interfacing and testing of the hardware module
REFERENCES

APPENDIX

SOFTWARE FILES
**********************************************

/*
 * main.c
 * Userspace program that communicates with the vga device driver
 * primarily through ioctl's
 *
 * Source Code:
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * Modified by:
 *
 * Gautham Vunnam
 * Columbia University
 */

#include <stdio.h>
#include <unistd.h>
#include "vga_led.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include "mat_operations.h"
#include "data.h"

int vga_led_fd; // vga_led file descriptor

int main(int argc, char *argv[]) {

    vga_led_arg_t vla;
    int i, j=0, facenumber = 0,k;
    char color = 1;
    static const char filename[] = "/dev/vga_led";
    unsigned int coords[12];
    unsigned int z;

    // matrix objects
    float xw, yw, zw;

    float adjusted_vertices[62][4];
    float adjusted_vertex[4][1];
    float vertex[4][1];
    float rotation[4][4];
    float transform[4][4];
    float temp[4][4];
    float world[4][4];
float temp41[4][1];
float pixels[62][3];
int pixellist[1080];

getRotationMatrix(-atoi(argv[1]), -atoi(argv[2]), 180, rotation);
dotProduct44(TRANSLATE, rotation, temp);
dotProduct44(temp, SCALE, world);
dotProduct44(PROJECTION, VIEW, temp);
dotProduct44(temp, world, transform);

for(i = 0; i < 62; i++)
{
  for(j = 0; j < 4; j++)
  {
    vertex[j][0] = vertices[i][j];
  }

dotProduct44(transform, vertex, adjusted_vertex);
scaleMat41(adjusted_vertex, 1/adjusted_vertex[3][0], temp41);

  xw = (((640 / 2.0) * temp41[0][0]) + (0 + (640 / 2.0)));
  yw = (((480 / 2.0) * temp41[1][0]) + (0 + (480 / 2.0)));
  zw = (((9) / 2.0) * temp41[2][0] + ((11) / 2.0));

  pixels[i][0] = xw;
  pixels[i][1] = yw;
  pixels[i][2] = zw;
}

for(i = 0; i < 120; i++)
{
  pixellist[i*9] = (int) pixels[faces[i][0][0]*32;
  pixellist[i*9 + 1] = (int) pixels[faces[i][0][1]*32;
  pixellist[i*9 + 2] = (int) pixels[faces[i][0][2]*32;
  pixellist[i*9 + 3] = (int) pixels[faces[i][1][0]*32;
  pixellist[i*9 + 4] = (int) pixels[faces[i][1][1]*32;
  pixellist[i*9 + 5] = (int) pixels[faces[i][1][2]*32;
  pixellist[i*9 + 6] = (int) pixels[faces[i][2][0]*32;
  pixellist[i*9 + 7] = (int) pixels[faces[i][2][1]*32;
  pixellist[i*9 + 8] = (int) pixels[faces[i][2][2]*32;
}

/* just open vga driver like it's a file
   using the file descriptor specified earlier */
if ( (vga_led_fd = open(filename, O_RDWR)) == -1) {
  fprintf(stderr, "could not open %s\n", filename);
  return -1;
}

if((atoi(argv[3]) != 1) && (atoi(argv[4]) == 1))
{
  for (i = 0; i<1080; i += 9) {
    coords[0] = pixellist[i];
    coords[1] = pixellist[i+1];
coords[2] = pixellist[i+2];
coords[3] = pixellist[i+3];
coords[4] = pixellist[i+4];
coords[5] = pixellist[i+5];
coords[6] = pixellist[i+6];
coords[7] = pixellist[i+7];
coords[8] = pixellist[i+8];

if (facenumber < 40) {
    color = 3;
}
else if (facenumber >= 40 && facenumber < 79) {
    color = 2;
}
else {
    color = 1;
}

coords[10] = 0;

// first time around send a color and write screen
coords[9] = 0; // send plate color
coords[11] = 1; // clear the screen

if (facenumber == 119) {
    coords[10] = 0; // show screen
}
else {
    coords[10] = 1; // hide screen
}

write_coordinates(coords);

facenumber++;
usleep(1000);

if(atoi(argv[4]) != 1)
{
    for (i = 0; i<1080; i += 9) {
        coords[0] = pixellist[i];
        coords[1] = pixellist[i+1];
        coords[2] = pixellist[i+2];
        coords[3] = pixellist[i+3];
        coords[4] = pixellist[i+4];
        coords[5] = pixellist[i+5];
        coords[6] = pixellist[i+6];
        coords[7] = pixellist[i+7];
        coords[8] = pixellist[i+8];

        if (facenumber < 40) {
            color = 3;
        }
        else if (facenumber >= 40 && facenumber < 79) {
            color = 2;
        }
    }
}
else {
    color = 1;
}

coords[10] = 0;

if (facenumber == 119) {
    coords[10] = 0; // show screen
} else {
    coords[10] = 1; // hide screen
}

// first time around send a color and write screen
coords[9] = color; // send plate color
coords[11] = 0; // don't clear the screen

write_coordinates(coords);

facenumber++;
usleep(1000);
}
}

usleep(2000); // wait 10 sec for the screen to clear

if(atoi(argv[3]) != 1) {
    for (i = 0; i<1080; i += 9) {
        coords[0] = pixellist[i];
        coords[1] = pixellist[i+1];
        coords[2] = pixellist[i+2];
        coords[3] = pixellist[i+3];
        coords[4] = pixellist[i+4];
        coords[5] = pixellist[i+5];
        coords[6] = pixellist[i+6];
        coords[7] = pixellist[i+7];
        coords[8] = pixellist[i+8];

        if (facenumber < 40) {
            color = 3;
        }

        else if (facenumber >= 40 && facenumber < 79) {
            color = 2;
        }

        else {
            color = 1;
        }

        coords[10] = 0;

        // first time around send a color and write screen
        coords[9] = 0; // send plate color
        coords[11] = 1; // don't clear the screen

        if (facenumber == 119) {
            coords[10] = 0; // show screen
        }
```c
} else {
    coords[10] = 1; // hide screen
}

write_coordinates(coords);
facenumber++;
usleep(1000);
}

return 0;
}

/*==================================================================
/* mat_operations.c
 * Matrix transformation pipeline
 *
 */
#include <math.h>
#include "mat_operations.h"

void dotProduct44(float a[4][4], float b[4][4], float c[4][4]) {
    int i, j, k;
    float sum = 0;

    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            sum = 0;
            for (k = 0; k < 4; k++) {
                sum = sum + a[i][k] * b[k][j];
            }
            c[i][j] = sum;
        }
    }
}

void scaleMat41(float a[4][1], float scale, float b[4][1]) {
    int i, j, k;
    float sum = 0;

    for (i = 0; i < 4; i++) {
        b[i][0] = a[i][0]*scale;
    }
}

void dotProduct41(float a[4][4], float b[4][1], float c[4][1]) {
    int i, j, k;
    float sum = 0;

    for (i = 0; i < 4; i++) {
        // code
    }
```
for (j = 0; j < 1; j++) {
    sum = 0;
    for (k = 0; k < 4; k++) {
        sum = sum + a[i][k] * b[k][j];
    }
    c[i][j] = sum;
}
}

void printMatrix(int row, int col, float m[row][col]) {
    int i, j;
    printf("------------------------\n");
    for (i = 0; i < row; i++) {
        for (j = 0; j < col; j++) {
            printf(" %f ", m[i][j]);
        }
        printf("\n");
    }
    printf("------------------------\n");
}

void getRotationMatrix(int angle_x, int angle_y, int angle_z, float m[4][4]) {
    float rad_x = angle_x * .01743;
    float rad_y = angle_y * .01743;
    float rad_z = angle_z * .01743;

    float temp[4][4];
    float rotate_x[4][4] = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}};
    float rotate_y[4][4] = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}};
    float rotate_z[4][4] = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}};

    rotate_x[1][1] = cos(rad_x);
    rotate_x[1][2] = sin(rad_x);
    rotate_x[2][1] = -sin(rad_x);
    rotate_x[2][2] = cos(rad_x);

    rotate_y[0][0] = cos(rad_y);
    rotate_y[0][2] = -sin(rad_y);
    rotate_y[2][0] = sin(rad_y);
    rotate_y[2][2] = cos(rad_y);

    rotate_z[0][0] = cos(rad_z);
    rotate_z[0][1] = -sin(rad_z);
    rotate_z[1][0] = sin(rad_z);
    rotate_z[1][1] = cos(rad_z);

    dotProduct44(rotate_y, rotate_x, temp);
    dotProduct44(rotate_z, temp, m);
}
/* vga_led.c */
/* Device driver for the VGA LED Emulator */
/* */
/* A Platform device implemented using the misc subsystem */
/* */
/* Stephen A. Edwards */
/* Columbia University */
/* */
/* References: */
/* Linux source: Documentation/driver-model/platform.txt */
/* drivers/misc/arm-charlcd.c */
/* http://www.linuxforu.com/tag/linux-device-drivers/ */
/* http://free-electrons.com/docs/ */
/* */
/* "make" to build */
/* insmod vga_led.ko */
/* */
/* Check code style with */
/* checkpatch.pl --file --no-tree vga_led.c */
/* */

/* header files used only those from kernel sources */
/* typically need these headers for a device driver */
#include <linux/module.h> // define module_init() and module_exit() functions
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_led.h"

#define DRIVER_NAME "vga_led"

/* */
/* Information about our device */
/* */
struct vga_led_dev {
    resource_size_t start; /* Address of start of registers */
    resource_size_t size;
    void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;

static void write_coor(int coor, unsigned int val)
{
    iowrite16(val, dev.virtbase + coor*2); // mult by 2 since we write 16-bits
    (2bytes) at a time
}

/* */
/* Handle ioctl() calls from userspace: */
/* Read or write the segments on single digits. */
/* Note extensive error checking of arguments */
/* 
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg) 
{
  vga_led_arg_t vla;
  switch (cmd) {
    case VGA_LED_WRITE_DIGIT:
      if (copy_from_user(&vla, (vga_led_arg_t *) arg, sizeof(vga_led_arg_t)))
        return -EACCES;
      if (vla.coor > 12)
        return -EINVAL;
      write_coor(vla.coor, vla.val);
      break;
    default:
      return -EINVAL;
  }
  pr_info("in the ioctl function");
  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_led_fops = {
  .owner = THIS_MODULE,
  .unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
  .minor = MISC_DYNAMIC_MINOR,
  .name = DRIVER_NAME,
  .fops = &vga_led_fops,
};

/* Initialization code: get resources (registers) and display 
a a welcome message */
static int __init vga_led_probe(struct platform_device *pdev)
{
  int ret;
  struct resource res;

  /* Register ourselves as a misc device: creates /dev/vga_led */
  ret = misc_register(&vga_led_misc_device);

  /* Get the address of our registers from the device tree */
  ret = of_address_to_resource(pdev->dev.of_node, 0, &res);
  if (ret) {
    ret = -ENOENT;
    goto out_deregister;
  }

  /* Make sure we can use these registers */
  if (request_mem_region(res.start, resource_size(&res),
    DRIVER_NAME) == NULL) {
    ret = -EBUSY;
    goto out_deregister;
  }

  return 0;
}
dev.start = res.start;
dev.size = resource_size(&res);

/* Arrange access to these registers */
dev.virtbase = of_iomap(pdev->dev.of_node, 0);
if (dev.virtbase == NULL) {
    ret = -ENOMEM;
    goto out_release_mem_region;
}
return 0;

out_release_mem_region:
    release_mem_region(res.start, resource_size(&res));
out_deregister:
    misc_deregister(&vga_led_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
iounmap(dev.virtbase);
    release_mem_region(dev.start, dev.size);
    misc_deregister(&vga_led_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
  { .compatible = "altr,vga_led" },
},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_led_of_match),
    },
    .remove = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
    pr_info(DRIVER_NAME": init\n");
    return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
    platform_driver_unregister(&vga_led_driver);
    pr_info(DRIVER_NAME": exit\n");
}
module_init(vga_led_init); // correspond to user space command insmod
module_exit(vga_led_exit); // correspond to user space command rmmod

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA 7-segment LED Emulator");

// mat_operations.h
#ifndef MAT_OPERATIONS_H
#define MAT_OPERATIONS_H
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

// matrix dot products
void dotProduct44(float a[4][4], float b[4][4], float c[4][4]);
void dotProduct41(float a[4][4], float b[4][1], float c[4][1]);
void getRotationMatrix(int angle_x, int angle_y, int angle_z, float m[4][4]);
void printMatrix(int row, int col, float m[row][col]);
void scaleMat41(float a[4][1], float scale, float b[4][1]);
#endif

// vga_led.h
#ifndef _VGA_LED_H
#define _VGA_LED_H
#include <linux/ioctl.h>

#define NUM_COORDS 12
#define NUM_PIXELS 108

/* used to specify the location of the center of the ball */
typedef struct {
    int coor; // specifies x(0) or y(1)
    unsigned int val; // coordinate of ball center
} vga_led_arg_t;

// 120 faces with corresponding vertex indexes
int faces[120][3] = {
    {0, 1, 2},
    {0, 3, 1},
    {0, 4, 3},
    {0, 5, 4},
    {0, 6, 5},
    {0, 7, 6},
    {0, 8, 7},
    {0, 9, 8},
    {0, 10, 9},
    {0, 11, 10},
    {0, 12, 11},
    {0, 13, 12},
    {0, 14, 13},
};
{0, 15, 14},
{0, 16, 15},
{0, 17, 16},
{0, 18, 17},
{0, 19, 18},
{0, 20, 19},
{0, 2, 20},
{21, 22, 23},
{21, 23, 24},
{21, 24, 25},
{21, 25, 26},
{21, 26, 27},
{21, 27, 28},
{21, 28, 29},
{21, 29, 30},
{21, 30, 31},
{21, 31, 32},
{21, 32, 33},
{21, 33, 34},
{21, 34, 35},
{21, 35, 36},
{21, 36, 37},
{21, 37, 38},
{21, 38, 39},
{21, 39, 40},
{21, 40, 41},
{21, 41, 42},
{18, 19, 42},
{8, 43, 44},
{17, 18, 45},
{1, 46, 47},
{7, 44, 48},
{6, 48, 49},
{17, 50, 51},
{5, 49, 52},
{15, 16, 51},
{4, 52, 53},
{14, 15, 54},
{3, 53, 46},
{14, 55, 56},
{13, 56, 57},
{12, 57, 58},
{11, 58, 59},
{2, 47, 60},
{10, 59, 61},
{20, 60, 42},
{8, 9, 61},
{22, 47, 46},
{24, 23, 46},
{24, 53, 52},
{25, 52, 49},
{27, 26, 49},
{28, 27, 48},
{28, 44, 43},
{29, 43, 61},
{30, 61, 59},
{31, 59, 58},
{32, 58, 57},
{33, 57, 56},
{34, 56, 55},
{35, 55, 54},

39
\{37, 36, 54\},
\{37, 51, 50\},
\{39, 38, 50\},
\{40, 39, 45\},
\{40, 42, 60\},
\{41, 60, 47\},
\{45, 18, 42\},
\{7, 8, 44\},
\{50, 17, 45\},
\{2, 1, 47\},
\{6, 7, 48\},
\{5, 6, 49\},
\{16, 17, 51\},
\{4, 5, 52\},
\{54, 15, 51\},
\{3, 4, 53\},
\{55, 14, 54\},
\{1, 3, 46\},
\{13, 14, 56\},
\{12, 13, 57\},
\{11, 12, 58\},
\{10, 11, 59\},
\{20, 2, 60\},
\{9, 10, 61\},
\{19, 20, 42\},
\{43, 8, 61\},
\{23, 22, 46\},
\{53, 24, 46\},
\{25, 24, 52\},
\{26, 25, 49\},
\{48, 27, 49\},
\{44, 28, 48\},
\{29, 28, 43\},
\{30, 29, 61\},
\{31, 30, 59\},
\{32, 31, 58\},
\{33, 32, 57\},
\{34, 33, 56\},
\{35, 34, 55\},
\{36, 35, 54\},
\{51, 37, 54\},
\{38, 37, 50\},
\{45, 39, 50\},
\{42, 40, 45\},
\{41, 40, 60\},
\{22, 41, 47\}

};

// Vertices for all 120 faces of the plate
float vertices[62][4] = {
\{0., 0., 0., 1.\},
\{-0.309, 0., 0.9511, 1.\},
\{0., 0., 1., 1.\},
\{-0.5878, 0., 0.809, 1.\},
\{-0.809, 0., 0.5878, 1.\},
\{-0.9511, 0., 0.309, 1.\},
\{-1., 0., -0., 1.\},
\{-0.9511, 0., -0.309, 1.\},
\{-0.809, 0., -0.5878, 1.\},
\{-0.5878, 0., -0.809, 1.\},
\{-0.309, 0., -0.9511, 1.\},
{0.0, 0., -1., 1.},
{0.309, 0., -0.9511, 1.},
{0.5878, 0., -0.809, 1.},
{0.809, 0., -0.5878, 1.},
{0.9511, 0., -0.309, 1.},
{1., 0., 0., 1.},
{0.9511, 0., 0.309, 1.},
{0.809, 0., 0.5878, 1.},
{0.5878, 0., 0.809, 1.},
{0.309, 0., 0.9511, 1.},
{-0., 0.1903, 0., 1.},
{-0., 0.1903, 0.4789, 1.},
{-0.148, 0.1903, 0.4554, 1.},
{-0.2815, 0.1903, 0.3874, 1.},
{-0.3874, 0.1903, 0.2815, 1.},
{-0.4554, 0.1903, 0.148, 1.},
{-0.4789, 0.1903, 0., 1.},
{-0.4554, 0.1903, -0.148, 1.},
{-0.3874, 0.1903, -0.2815, 1.},
{-0.2815, 0.1903, -0.3874, 1.},
{-0.148, 0.1903, -0.4554, 1.},
{0., 0.1903, -0.4789, 1.},
{0.148, 0.1903, -0.4554, 1.},
{0.2815, 0.1903, -0.3874, 1.},
{0.3874, 0.1903, -0.2815, 1.},
{0.4554, 0.1903, -0.148, 1.},
{0.4789, 0.1903, 0., 1.},
{0.4554, 0.1903, 0.148, 1.},
{0.3874, 0.1903, 0.2815, 1.},
{0.2815, 0.1903, 0.3874, 1.},
{0.148, 0.1903, 0.4554, 1.},
{0.4743, 0.1903, 0.6528, 1.},
{-0.6528, 0.1903, -0.4743, 1.},
{-0.7674, 0.1903, -0.2493, 1.},
{0.6528, 0.1903, 0.4743, 1.},
{-0.2493, 0.1903, 0.7674, 1.},
{-0., 0.1903, 0.8068, 1.},
{-0.8068, 0.1903, 0., 1.},
{-0.7674, 0.1903, 0.2493, 1.},
{0.7674, 0.1903, 0.2493, 1.},
{0.8068, 0.1903, 0., 1.},
{-0.6528, 0.1903, 0.4743, 1.},
{-0.4743, 0.1903, 0.6528, 1.},
{0.7674, 0.1903, -0.2493, 1.},
{0.6528, 0.1903, -0.4743, 1.},
{0.4743, 0.1903, -0.6528, 1.},
{0.2493, 0.1903, -0.7674, 1.},
{0., 0.1903, -0.8068, 1.},
{-0.2493, 0.1903, -0.7674, 1.},
{0.2493, 0.1903, 0.7674, 1.},
{-0.4743, 0.1903, -0.6528, 1.}

// Default world view matrix
float VIEW[4][4] = {
    {1, 0, 0, 0},
    {0, 1, 0, 0},
    {0, 0, 1, -5},
    {0, 0, 0, 1}
};
```
// Default world projection matrix
float PROJECTION[4][4] = {
    {2.4178, 0, 0, 0},
    {0, 2.4178, 0, 0},
    {0, 0, -2, -3},
    {0, 0, -1, 0}
};

float TRANSLATE[4][4] = {
    {1, 0, 0, 0},
    {0, 1, 0, 0},
    {0, 0, 1, 0},
    {0, 0, 0, 1}
};

float SCALE[4][4] = {
    {1, 0, 0, 0},
    {0, 1, 0, 0},
    {0, 0, 1, 0},
    {0, 0, 0, 1}
};

/* ioctl's and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW( 0x08, 1, vga_led_arg_t *)
#define VGA_LED_READ_DIGIT _IOR(0x08, 2, vga_led_arg_t *)
#endif

#ps3.py
#controller parse
import usb.core
import usb.util
import sys
import os

# Scanning the tree for our DualShock 3 controller
controller = usb.core.find(idVendor=0x054c, idProduct=0x0268)

# In case the resource is busy
reattach = False
if controller.is_kernel_driver_active(0):
    reattach = True
    controller.detach_kernel_driver(0)

# Error message for device if not found
if controller is None:
    raise ValueError('Device not found')

# Set the active configuration. With no arguments, the first configuration will be the active one
controller.set_configuration()

# Initialising values
xaxis = 0
yaxis = 0
temp_xaxis = 0
temp_yaxis = 0
prev_stop = 0
```
lock = 0
stop = 0

while True:
    # Our “SetupPacket”
    ret = dev.ctrl_transfer(0xa1, 0x1, 0x0101, 0x0, 0x31)
    # Check for anti-clockwise rotation on the x-axis
    if ret[41] == 2:
        if ret[42] <= 110 and ret[42] >= 0:
            xaxis = ret[42]*90/110
            # Limit maximum to 90 degrees
            if xaxis > 90:
                xaxis = 90
    # Check for clockwise rotation on the x-axis
    if ret[41] == 1:
        if ret[42] <= 255 and ret[42] >= 142:
            xaxis = (ret[42] - 255)*90/113
            # Limit maximum to 90 degrees
            if xaxis < -90:
                xaxis = -90
    # Check for anti-clockwise rotation on the y-axis
    if ret[43] == 2:
        if ret[44] <= 110 and ret[44] >= 0:
            yaxis = ret[44]*90/110
            # Limit maximum to 90 degrees
            if yaxis > 90:
                yaxis = 90
    # Check for clockwise rotation on the y-axis
    if ret[43] == 1:
        if ret[42] <= 255 and ret[42] >= 142:
            yaxis = (ret[44] - 255)*90/115
            # Limit maximum to 90 degrees
            if yaxis < -90:
                yaxis = -90
    # Check if 'X' has been pressed, set the stop flag if so
    if ret[3] == 64:
        stop = 1
    # Clear the stop flag if 'X' has been released
    if ret[3] == 0:
        stop = 0
    # This logic sends the pervious rotation values again, so that the framebuffer can clear the previous render before
    # printing the new one
    if stop == 0:
        lock = 0
    if prev_stop == 1:
        os.system("./hello " + str(temp_yaxis) + " " + str(temp_xaxis) + " " + str(stop) + " " + str(prev_stop))
    else:
```
    os.system("./hello " + str(yaxis) + " " + str(xaxis) + " " + str(stop) + " " + str(prev_stop))

    # Storing the previous rotation values
    temp_yaxis = yaxis
    temp_xaxis = xaxis
    prev_stop = 0

    # If the 'X' button has been pressed, send the previous rotation values and stop
    # sending any new values to prevent the current from being overwritten in the framebuffer
    if stop == 1:
        if lock == 0:
            os.system("./hello " + str(temp_yaxis) + " " + str(temp_xaxis) + " " + str(stop) + " " + str(prev_stop))
            lock = 1
            prev_stop = 1

//Makefile
ifeq (${KERNELRELEASE},)
    # KERNELRELEASE defined: we are being compiled as part of the Kernel
    obj-m := vga_led.o
else
    # We are being compiled as a module: use the Kernel build system
        KERNEL_SOURCE := /usr/src/linux # specify where kernel src is
        PWD := $(shell pwd)

default: module main
main:
    gcc -o main main.c mat_operations.c -lm
module:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules
clean:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} cclean
    ${RM} hello

socfpga.dtb : socfpga.dtb
dtc -O dtb -o socfpga.dtb socfpga.dts
endif
```

HARDWARE FILES

/*mem_IO.sv
 * Avalon memory-mapped peripheral */

module mem_IO(input logic clk,
               input logic reset,
               input logic [15:0] writedata,
               input logic write,
               input logic chipselect,
               input logic [7:0] address, // two bits needed for x,y coordinates
               output logic [7:0] VGA_R, VGA_G, VGA_B,
               output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
               output logic VGA_SYNC_n);

typedef enum logic [11:0] {S0, S1} state_t;
state_t state;

logic start, done;
logic [15:0] v1x, v1y, v1z;
logic [15:0] v2x, v2y, v2z;
logic [15:0] v3x, v3y, v3z;
logic [15:0] pixel_color;
logic clear_screen;
logic hide_screen;

shader shader_inst(.*);

always_ff @(posedge clk)
  if (reset) begin
    state <= S0;
    start <= 1;
  end
  else if (chipselect && write) begin
    case (address)
      // vertices
      8'd0 : v1x <= writedata;
      8'd1 : v1y <= writedata;
      8'd2 : v1z <= writedata;
      8'd3 : v2x <= writedata;
      8'd4 : v2y <= writedata;
      8'd5 : v2z <= writedata;
      8'd6 : v3x <= writedata;
      8'd7 : v3y <= writedata;
      8'd8 : v3z <= writedata;
      /* ***/ Total of 12 addresses to write to
      // vertices
      0-8: vertices of face to draw
      9 : 2-bits representing 3 colors to draw. 0 represents black
      10 : hide screen signal to hide the display
      11 : clear screen signal to clear the screen ***/
    endcase
  end
endmodule
/**
color
8'd9 : pixel_color <= writedata;

// hide and clear screen signals
8'd10: begin
if (writedata == 1) begin
    hide_screen <= 1;
end
else begin
    hide_screen <= 0;
end
end

8'd11: begin
if (writedata == 1) begin
    clear_screen <= 1;
end
else begin
    clear_screen <= 0;
end

// after the last data item is sent start the shader
operation
state <= S0;
start <= 1;
endcase
end

else begin
    case (state)
    S0: begin
        if (start) begin
            if (done) begin
            state <= S1;
        end
        end

    S1: begin
        start <= 0;
        state <= S0;
    end
    endcase
end
endmodule
input logic start,
input logic [15:0] v1x, v1y, v1z,
input logic [15:0] v2x, v2y, v2z,
input logic [15:0] v3x, v3y, v3z,
input logic [15:0] pixel_color,
input logic clear_screen,
input logic hide_screen,

output logic [7:0] VGA_R, VGA_G, VGA_B,
output logic VGA_CLK, VGA_HS, VGA_VS,
VGA_BLANK_n, VGA_SYNC_n,

output logic done
);

/////////////////////////////////
//
//  for sorting input vertices
//
/////////////////////////////////

logic [15:0] p1x, p1y, p1z;
logic [15:0] p2x, p2y, p2z;
logic [15:0] p3x, p3y, p3z;

/////////////////////////////////
//
//  divider signals and modules
//
/////////////////////////////////

logic [15:0] dp1p2, dp1p2num, dp1p2den;
logic [15:0] dp1p3, dp1p3num, dp1p3den;

// divider signals
logic start_div;  // start divide
logic done_div1;  // done dividing
logic done_div2;

// divider modules i need for this section
divider inv_slope1 (clk, reset, start_div, dp1p2num, dp1p2den, dp1p2, done_div1);
divider inv_slope2 (clk, reset, start_div, dp1p3num, dp1p3den, dp1p3, done_div2);

/////////////////////////////////
//
//  draw signals and modules
//
/////////////////////////////////

// draw line signals
logic rst_draw, start_draw, done_draw;
logic [15:0] sx1, ex1;
logic [15:0] pax, pay, paz;
logic [15:0] pbx, pby, pbz;
logic [15:0] pcx, pcy, pcz;
logic [15:0] pdx, pdy, pdz;
logic [15:0] y_coord;
logic [15:0] start_y, end_y;
logic [15:0] sx, ex;
logic draw_line_done;

// wires
logic line_done;
logic bresenham_start;
logic bresenham_done;
logic pixel_write;
logic [10:0] pixel_x, pixel_y;
logic [15:0] z_wire1, z_wire2;
logic [10:0] z_buff_wire;

draw_line draw(
    .clk(clk), .reset(reset),
    .start(start_draw),
    .y_coord(y_coord),
    .pbx(pbx), .pby(pby), .pbz(pbz),
    .pdx(pdx), .pdy(pdy), .pdz(pdz),
    .draw(bresenham_start), .start_x(sx), .end_x(ex),
    .done(draw_line_done)
);

put_pixel put_pixel_inst(
    .clk(clk),
    .reset(reset), .start(bresenham_start),
    .x0(sx), .y0(y_coord), .x1(ex), .y1(y_coord),
    .plot(pixel_write), .x(pixel_x), .y(pixel_y), .done(bresenham_done),
    .z1(z_wire1), .z2(z_wire2), .z_out(z_buff_wire)
);

VGA_framebuffer screen(
    .clk50(clk),
    .reset(reset), .x(pixel_x), .y(pixel_y), .z(z_buff_wire),
    .pixel_color(pixel_color), .pixel_write(pixel_write),
    .clear_screen(clear_screen),
    .VGA_R(VGA_R),
    .VGA_G(VGA_G), .VGA_B(VGA_B), .hide_screen(hide_screen),
    .VGA_HS(VGA_HS), .VGA_VS(VGA_VS), .VGA_BLANK_n(VGA_BLANK_n),
    .VGA_SYNC_n(VGA_SYNC_n)
);

/////////////////////////////////
typedef enum logic [6:0] {S0, S1, S2, S3, S4, S5, S6} state_t;
state_t state;

always_ff @(posedge clk) begin
  if (reset) begin
    state <= S0;
    done <= 0;
  end
  else begin
    case (state)
      S0: begin // sorting state
        done <= 0;
        if (start) begin
          if ((v1y < v2y) & (v1y < v3y)) begin // if V1 is the smallest vertex
            p1x = v1x; // first vertex is V1
            p1y = v1y;
            p1z = v1z;
            if (v2y > v3y) begin // if V2 is bigger than V3,
              p2x = v3x; // second vertex is V3
              p2y = v3y;
              p2z = v3z;
              p3x = v2x; // third vertex is V2
              p3y = v2y;
              p3z = v2z;
            end
            else begin // if V3 is bigger than V2,
              p2x = v2x; // second vertex is
              p2y = v2y;
              p2z = v2z;
              p3x = v3x; // third vertex is V3
              p3y = v3y;
              p3z = v3z;
            end
          end
        end
    endcase
  end
end
if V2 is the largest vertex

else if ((v2y < v1y) & (v2y < v3y)) begin //

    p1x = v2x;  // first vertex is V2
    p1y = v2y;
    p1z = v2z;

    if (v1y > v3y) begin // if V1 is

        p2x = v3x;  // second vertex is
        p2y = v3y;
        p2z = v3z;

        p3x = v1x;  // third vertex is V3
        p3y = v1y;
        p3z = v1z;

    end

else begin // if V2 is bigger than V1, 

    p2x = v1x;  // second vertex is 
    p2y = v1y;
    p2z = v1z;

    p3x = v3x;  // third vertex is V2
    p3y = v3y;
    p3z = v3z;

end

else begin // if V is bigger than V1,

    plx = v3x;  // first vertex is V2
    ply = v3y;
    plz = v3z;

    if (vly > v2y) begin // if V1 is

        p2x = v2x;  // second vertex is
        p2y = v2y;
        p2z = v2z;

        p3x = vlx;  // third vertex is V3
        p3y = vly;
        p3z = v1z;

    end

else begin // if V2 is bigger than V1,

    p2x = vlx;  // second vertex is
    p2y = vly;
    p2z = vlz;

    p3x = v3x;  // third vertex is V2
    p3y = v3y;
    p3z = v3z;

end

end

state <= S1;
begin // calculate numerators and denominators for inverse slopes
if (p2y > p1y) begin
  dp1p2num = p2x - p1x;
  dp1p2den = p2y - p1y;
end
else begin // forces dp1p2 to 0 if slope is negative
  dp1p2num = 0;
  dp1p2den = 1;
end
if (p3y > p1y) begin
  dp1p3num = p3x - p1x;
  dp1p3den = p3y - p1y;
end
else begin // forces dp1p3 to 0 if slope is negative
  dp1p3num = 0;
  dp1p3den = 1;
end
state <= S2;
end

S2: begin
  start_div <= 1; // divide the inverse slopes
  if (done_div1 & done_div2) begin // wait until the divides are done
    state <= S3;
    start_div <= 0;
  end
end

S3: begin // at this point inverse slopes dp1p2 and dp1p3 are ready
  
  start_y = p1y >> 5;
  end_y = (p3y >> 5) + 1;
  y_coord = start_y;
  state <= S4;
end

S4: begin // sort vertices to be drawn for given y_coord. this determines if we're in Step1 or Step2 of the triangle
  if (dp1p2num * dp1p3den > dp1p3num * dp1p2den) begin
    if ( (y_coord << 5) < p2y) begin
      pax = p1x;
      pay = p1y;
      paz = p1z;
      pbx = p3x;
      pby = p3y;
    end
  end
```plaintext
pbz = p3z;
pbx = p3x;
pby = p3y;
pbz = p3z;

pcx = p1x;
pcy = p1y;
pcz = p1z;
pdx = p2x;
pdy = p2y;
pdz = p2z;

end
else begin
  pax = p1x;
pay = p1y;
paz = p1z;

  pbx = p3x;
pby = p3y;
pbz = p3z;

  pcx = p2x;
  pcy = p2y;
  pcz = p2z;

  pdx = p3x;
pdy = p3y;
pdz = p3z;

end
else begin
  if ( (y_coord << 5) < p2y) begin

    pax = p1x;
pay = p1y;
paz = p1z;

    pbx = p2x;
pby = p2y;
pbz = p2z;

    pcx = p1x;
    pcy = p1y;
    pcz = p1z;

    pdx = p3x;
pdy = p3y;
pdz = p3z;

  end
else begin
  pax = p2x;
pay = p2y;
paz = p2z;

  pbx = p3x;
pby = p3y;
pbz = p3z;

  pcx = p1x;
  pcy = p1y;
  pcz = p1z;

end
```
pdx = p3x;
pdy = p3y;
pdz = p3z;
end

state <= S5;
end

S5: begin
  start_draw <= 1; // draw current line
  if (draw_line_done) begin // wait until
    y_coord = y_coord + 1; // increment
    start_draw <= 0; // make sure to set start_draw back to 0 when we're done
    state <= S6;
  end
end

S6: begin
  if (y_coord < end_y) begin // jump back
    state <= S4;
    end
  else begin // here we've finished shading the
    done <= 1;
    state <= S0; // go back to
    end
  end // end state S5
endcase
end

// DETERMINES START X AND END X, FOR EVERY LINE IN TRIANGLE
module draw_line(input logic clk,
  input logic reset,
  input logic start,
  input logic bresenham_done,
  input logic [15:0] y_coord,
  input logic [15:0] pax, pay, paz,
  input logic [15:0] pbx, pby, pbz,
  input logic [15:0] pcx, pcy, pcz,
  input logic [15:0] pdx, pdy, pdz,

  output logic done,
  output logic draw,
  output logic [15:0] zout1, zout2,
  output logic [15:0] start_x, end_x
);
typedef enum logic [8:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8} state_t;
state_t state;

// internal signals
logic [15:0] gradient1num, gradient1den;
logic [15:0] gradient2num, gradient2den;
logic [15:0] gradientznum, gradientzden;
logic [15:0] gradient1;
logic [15:0] gradient2;
logic [15:0] gradientz;
logic [15:0] temp_y;
logic [15:0] sx, ex, z1, z2, z;
logic [15:0] temp_x;

// signals required for the dividers
logic rst_div; // reset signal for divider
logic start_div; // start divides
logic done_div1; // done dividing
logic done_div2;
logic done_div3;

// dividers required
divider14 div1(clk, reset, start_div, gradient1num, gradient1den,
gradient1, done_div1);
divider14 div2(clk, reset, start_div, gradient2num, gradient2den,
gradient2, done_div2);
divider14 div3(clk, reset, start_div, gradientznum, gradientzden,
gradientz, done_div3);

// signals for interpolation modules
logic start_int;
logic done_int_sx;
logic done_int_ex;
logic done_int_z1;
logic done_int_z2;
logic done_int_z;

// interpolation module initialization
interpolate interpolate_sx( .clk(clk), .start(start_int), .reset(reset),
.min_val(pax),
.max_val(pbx), .gradient(gradient1),
.done(done_int_sx), .val(sx) );

interpolate interpolate_ex( .clk(clk), .start(start_int), .reset(reset),
.min_val(pcx),
.max_val(pdx), .gradient(gradient2),
.done(done_int_ex), .val(ex) );

interpolate interpolate_z1( .clk(clk), .start(start_int), .reset(reset),
.min_val(paz),
.max_val(pbz), .gradient(gradient1),
.done(done_int_z1), .val(z1) );

interpolate interpolate_z2( .clk(clk), .start(start_int), .reset(reset),
.min_val(paz),
.max_val(pbz), .gradient(gradient1),
.done(done_int_z2), .val(z1) );
always_ff @(posedge clk) begin
  temp_y = y_coord << 5;
  if (reset) begin
    state <= S0;
    end
  else begin
    case (state)
      S0: begin
        if (start) begin
          if (pay != pby) begin
            gradient1num = temp_y - pay;
            gradient1den = pby - pay;
          end
          else begin
            gradient1num = 1;
            gradient1den = 1;
          end
          if (pcy != pdy) begin
            gradient2num = (temp_y - pcy);
            gradient2den = (pdy - pcy);
          end
          else begin
            gradient2num = 1;
            gradient2den = 1;
          end
          rst_div <= 1; // reset the dividers to get
          them ready
          state <= S1;
        end
      end
  S1: begin
    end
end
rst_div <= 0;  // bring reset back to 0
start_div <= 1;  // start the dividers
if (done_div1 & done_div2) begin
    start_div <= 0;
    state <= S2;
end
S2: begin
    start_int <= 1;  // start all interpolations
    if (done_int_z1 & done_int_z2 & done_int_ex & done_int_sx ) begin  // wait until the interpolations are done
        start_int <= 0;  // deassert the interpolations
        // output z1 and z2 for z-buffer calculations
        zout1 = z1;
        zout2 = z2;
        if (sx > ex) begin
            temp_x = sx;
            start_x = ex >> 5;
            end_x = temp_x >> 5;
        end
        else begin
            start_x = sx >> 5;
            end_x = ex >> 5;
        end
        draw <= 1;
        state <= S3;
    end
S3: begin  // figure out temp_product for gradient1
calculate gradients for z
    // at this point we have sx, ex, z1 and z2. need to
draw <= 0;
    if (bresenham_done) begin  // when bresenham is done
done <= 1;
end
    if (~start) begin  // wait until start is deasserted
done <= 0;
state <= S0;
end;
endcase
end
//CALCULATES Z VALUES, AND SENDS TO FRAME BUFFER
module put_pixel(input logic clk, reset,
                   input logic start,
                   input logic [15:0] x0, y0, x1, y1,
                   input logic [15:0] z1, z2,
                   output logic plot,
                   output logic [10:0] x, y,
                   output logic [10:0] z_out,
                   output logic done);

typedef enum logic [9:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8} state_t;
state_t state;
logic [15:0] x_coord;
logic [15:0] y_coord;
logic [15:0] z_coord;

// divider stuff
logic [15:0] gradientnum, gradientden, gradient;
logic start_div; // start divides
logic done_div;

// dividers required
divider14 div1(clk, reset, start_div, gradientnum, gradientden, gradient,
done_div);

// signals for interpolation modules
logic start_int;
logic done_int;

// higher resolution interpolate block
interpolate7 interpolate_inst( .clk(clk), .start(start_int),
                              .reset(reset),
                              .min_val(z1 << 2),
                              .max_val(z2 << 2), .gradient(gradient), // z1 and z2 need to be promoted to Q7
                              for a higher resolution interpolation
                              .done(done_int),
                              .val(z_coord) );

always_ff @(posedge clk) begin
  if (reset) begin
    done <= 0;
    state <= S0;
  end
  else begin
    case (state)
      S0: begin // get initial signals to start with
        if (start) begin
          x_coord = x0;
          y_coord = y0;
          x = x_coord;
          y = y0;
          state <= S1;
        end
      end
    endcase
  end
end
S1: begin
    if (x1 == x0) begin // if start x and end x are the
        same -> force gradient to 0
        gradientnum = 0;
        gradientden = 1;
    end
    else begin
        gradientnum = (x_coord << 5) - (x0 << 5);
        gradientden = (x1 << 5) - (x0 << 5);
    end
    start_div = 1;
    if(done_div) begin
        start_div <= 0;
        start_int <= 1; // since gradients are
        available, start interpolation for z-value
        state <= S2;
    end
end
S2: begin
    if(done_int) begin  // when interpolation for z is
        done done
        z_out <= (z_coord);
        start_int <= 0;
        plot <= 1;
        state <= S3;
    end
end
S3: begin
    plot = 0; // de-assert plot signal
    x_coord = x_coord + 1; // increment x_coord to place
    next pixel in the line
    state <= S4;
end
S4: begin
    if (x_coord > x1) begin // check if we've exceeded
        ex. If so, we're done with the line and we can finish
        done <= 1;
        state <= S5;
    end
    else begin // otherwise keep going to the next x-
        value
        x = x_coord;
        state <= S6;
    end
end
S5: begin  // clean up the signals
    done <= 0;
    state <= S0;
end
module VGA_framebuffer(
    input logic clk50, reset,
    input logic [10:0] x, // pixel x_coordinate
    input logic [10:0] y, // pixel y_coordinate
    input logic [10:0] z, // pixel z_coordinate
    input logic [15:0] pixel_color,
    input logic pixel_write,
    input logic clear_screen,
    input logic hide_screen,
    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 */

parameter HACTIVE = 11'd 1280,
    HFRONT_PORCH = 11'd 32,
    HSYNC = 11'd 192,
    HBACK_PORCH = 11'd 96,
    HTOTAL = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH;

//1600

parameter VACTIVE = 10'd 480,
    VFRONT_PORCH = 10'd 10,
    VSYNC = 10'd 2,
    VBACK_PORCH = 10'd 33,
    VTOTAL = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525

// Horizontal counter
logic [10:0] hcount;
logic endOfLine;
always_ff @(posedge clk50 or posedge reset)
    if (reset) hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else hcount <= hcount + 11'd 1;
assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0] vcount;
logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset) vcount <= 0;
  else if (endOfLine)
    if (endOfField) vcount <= 0;
    else              vcount <= vcount + 10'd 1;
assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x57F
// 101 0010 0000 to 101 0111 1111
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

logic blank;
// Horizontal active: 0 to 1279  Vertical active: 0 to 479
// 101 0000 0000  1280  01 1110 0000  480
// 110 0011 1111  1599 10 0000 1100  524
                   ( vcount[9] | (vcount[8:5] == 4'b1111) );

logic [1:0] framebuffer [262143:0]; // 640 x 480
logic [18:0] read_address, write_address;

assign write_address = x + (y << 9) + (y << 7);
assign read_address = (hcount >> 1) + (vcount << 9) + (vcount << 7);

/**Pixel write code using z-buffering**/

logic zbuffer_clear, zbuffer_write;
logic [10:0] zbuffer_rdata;
logic [10:0] zbuffer_wdata;
logic [18:0] temp_addr;
logic write_fb;
// initialize zbuffer RAM block
zbuffer zbuffer_shader (
  .aclr(zbuffer_clear),
  .address(write_address[17:0]),
  .clock(clk50),
  .data(zbuffer_wdata),
  .wren(zbuffer_write),
  .q(zbuffer_rdata));

typedef enum logic [2:0] {S0, S1, S2} state_t;
state_t state;

always_ff @ (posedge clk50) begin
  if (reset) begin
    write_fb <= 0;
    zbuffer_clear <= 0;
    zbuffer_write <= 0;
    state <= S0;
  end
  else begin
    case (state)
      S0: begin
        if (pixel_write) begin
          if (clear_screen) begin
            zbuffer_wdata <= 11'b0000000000;
          end
          else begin
            zbuffer_wdata <= z;
          end
          state <= S1;
        end
      end
      S1: begin
        if (clear_screen) begin
          write_fb <= 1;
          zbuffer_write <= 1;
        end
        else if (zbuffer_rdata == 0) begin  // first time
          write_fb <= 1;  // so just draw the pixels
          zbuffer_write <= 1;
        end
        else if (zbuffer_rdata >= z) begin  // otherwise
          write_fb <= 1;
          zbuffer_write <= 1;
        end
        state <= S2;
      end
      S2: begin
        // cleanup signals
        write_fb <= 0;
        zbuffer_write <= 0;
        zbuffer_clear <= 0;
        state <= S0; // go to S0 and wait for next pixel
        write endcase
      end
    endcase
  end
end
logic [1:0] pixel_read;

always_ff @(posedge clk50) begin
    if (write_fb) framebuffer[write_address] <= pixel_color[1:0];
    if (hcount[0]) begin
        pixel_read <= framebuffer[read_address];
        VGA_BLANK_n <= ~blank;  // sync blank with read pixel data
    end
end

assign VGA_CLK = hcount[0];  // 25MHz clock

always_ff @(posedge clk50) begin
    if (hide_screen) begin
        {VGA_R, VGA_G, VGA_B} = 24'h0;  // hide the screen from any updates
    end
    else if (pixel_read == 2'b01) begin
        {VGA_R, VGA_G, VGA_B} = {8'hca, 8'he1, 8'hff};  // light blue
    end
    else if (pixel_read == 2'b10) begin
        {VGA_R, VGA_G, VGA_B} = {8'h48, 8'h76, 8'hff};  // medium blue
    end
    else if (pixel_read == 2'b11) begin
        {VGA_R, VGA_G, VGA_B} = {8'h27, 8'h40, 8'hab};  // dark blue
    end
    else begin
        {VGA_R, VGA_G, VGA_B} = 24'h0;
    end
end
endmodule
module zbuffer (aclr, address, clock, data, wren, q);
  input aclr;
  input [17:0] address;
  input clock;
  input [10:0] data;
  input wren;
  output [10:0] q;

  tri0 aclr;
  tri1 clock;

  wire [10:0] sub_wire0;
  wire [10:0] q = sub_wire0[10:0];

  altsyncram_altsyncram_component (  .aclr0 (aclr),
    .address_a (address),
    .clock0 (clock),
    .data_a (data),
    .wren_a (wren),
    .q_a (sub_wire0),
    .aclr1 (1'b0),
    .address_b (1'b1),
    .addressstall_a (1'b0),
    .addressstall_b (1'b0),
    .byteena_a (1'b1),
    .byteena_b (1'b1),
    .clock1 (1'b1),
    .clocken0 (1'b1),
    .clocken1 (1'b1),
    .clocken2 (1'b1),
    .clocken3 (1'b1),
    .cl...
.data_b (1'b1),
.eccstatus (),
.q_b (),
.rden_a (1'b1),
.rden_b (1'b1),
.wren_b (1'b0));
defparam
altsyncram_component.clock_enable_input_a = "BYPASS",
altsyncram_component.clock_enable_output_a = "BYPASS",
altsyncram_component.intended_device_family = "Cyclone V",
altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
altsyncram_component.lpm_type = "altsyncram",
altsyncram_component.numwords_a = 262144,
altsyncram_component.operation_mode = "SINGLE_PORT",
altsyncram_component.outdata_aclr_a = "CLEAR0",
altsyncram_component.outdata_reg_a = "CLOCK0",
altsyncram_component.power_up_uninitialized = "FALSE",
altsyncram_component.read_during_write_mode_port_a = "NEW_DATA_NO_NBE_READ",
altsyncram_component.widthad_a = 18,
altsyncram_component.width_a = 11,
altsyncram_component.width_byteena_a = 1;
endmodule

// //==========================================
// // CNX file retrieval info
// //======================================================================================
// // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// // Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// // Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// // Retrieval info: PRIVATE: AclrData NUMERIC "0"
// // Retrieval info: PRIVATE: AclrOutput NUMERIC "1"
// // Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// // Retrieval info: PRIVATE: Clken NUMERIC "0"
// // Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// // Retrieval info: PRIVATE: MIFfilename STRING ""
// // Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "262144"
// // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
// // Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// // Retrieval info: PRIVATE: RegData NUMERIC "1"
// // Retrieval info: PRIVATE: RegOutput NUMERIC "1"
// // Retrieval info: PRIVATE: SYNTH.WRAPPER_GEN_POSTFIX STRING "0"
// // Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// // Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// // Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// // Retrieval info: PRIVATE: WidthAddr NUMERIC "18"
// // Retrieval info: PRIVATE: WidthData NUMERIC "11"
/** INTERPOLATE MODULE */
module interpolate ( input logic clk, 
    input logic start, 
    input logic reset, 
    input logic [15:0] min_val, 
    input logic [15:0] max_val, 
    input logic [15:0] gradient, 
    output logic done, 
    output logic [15:0] val );

// // internal signals
//

logic [31:0] temp_product; // Q.19 format
logic [31:0] product; // Q.19 format

typedef enum logic [8:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8} state_t;
state_t state;
always_ff @(posedge clk) begin
  if (reset) begin
    done <= 0;
    state <= S0;
  end
  else begin
    case (state)
      S0: begin // calculate the product term
        done <= 0;
        if (start) begin
          temp_product = (max_val - min_val) * gradient;
          state <= S1;
        end
      end
      S1: begin // get magnitude of product term
        if (temp_product[31]) begin
          product = ~(temp_product - 1);
        end
        else begin
          product = temp_product;
        end
        state <= S2;
      end
      S2: begin // shift to get Q.5
        if (temp_product[31]) begin
          product = ~(product >> 14) + 1;
        end
        else begin
          product = product >> 14;
        end
        state <= S3;
      end
      S3: begin
        if (gradient[15]) begin // gradient < 0
          val = min_val; // gradient = 0;
        end
        else if ( (gradient >> 14) >= 1) begin
          val = min_val + (max_val - min_val); //
        end
        else begin
          val = min_val + product;
        end
        done <= 1; // we're done!
      end
    end_case
  end
end

66
module interpolate7 (input logic clk,
input logic start,
input logic reset,
input logic [15:0] min_val, // Q.7 format
input logic [15:0] max_val, // Q.7 format
input logic [15:0] gradient, // Q.14 format
output logic done,
output logic [15:0] val );

// internal signals

logic [31:0] temp_product; // Q.21 format
logic [31:0] product; // Q.21 format

typedef enum logic [8:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8} state_t;
state_t state;

always_ff @(posedge clk) begin
  if (reset) begin
    done <= 0;
    state <= S0;
  end
  else begin
    case (state)
      S0: begin // calculate the product term
        done <= 0;
        if (start) begin
          temp_product = (max_val - min_val) * gradient;
          state <= S1;
        end
      end

      S1: begin // get magnitude of product term
        if (temp_product[31]) begin
          product = ~(temp_product - 1);
        end
      end
    endcase
  end
end
else begin
    product = temp_product;
end
state <= S2;
end

S2: begin  //= shift to get Q.11
    if (temp_product[31]) begin
        product = ~(product >> 14) + 1;
    end
    else begin
        product = product >> 14;
    end
    state <= S3;
end

S3: begin
// clamps gradient between 0 and 1

    if (gradient[15]) begin  //= gradient < 0
        val = (min_val);
    end
    else if ((gradient >> 14) >= 1) begin
        val = (min_val + (max_val - min_val)); //=
        gradient = 1;
    end
    else begin
        val = (min_val + product);
    end
    done <= 1;  //= we're done!
end
if (~start) begin
    state <= S0;
end
endcase
end
endmodule

/*====================================================================*/
module divider(input clk,
               input reset,
               input start,
               input logic [15:0] num,
               input logic [15:0] den,
               output logic [15:0] result,
               output logic done);
/*====================================================================*/
typedef enum logic [3:0] {S0, S1, S2, S3} state_t;
state_t state;

logic [15:0] temp_num;
logic [15:0] temp_den;
logic [15:0] temp_result;

always_ff @(posedge clk) begin
  done <= 0;
  if (reset) begin
    state <= S0;
  end
  else begin
    case (state)
      S0: begin
        if (start) begin
          if (num[15] & den[15]) begin  // if both the
            // divide only the magnitudes
            temp_num = ~(num - 1);  // get
            temp_den = ~(den - 1);
            state <= S1;
          end
          // following two conditions make sure the
          // result is negative
          else if (num[15] & ~den[15]) begin  // if only
            temp_num = ~(num - 1);  // get
            temp_den = den;
            state <= S2;
          end
          else if (~num[15] & den[15]) begin // if only
            temp_num = num;
            temp_den = ~(den - 1);
            state <= S2;
          end
        end
      end
      S1: begin  // result is positive
        result = (temp_num << 10) / (temp_den << 5);
        done <= 1;
        if (~start) begin
          done <= 0;
          state <= S0;
        end
      end
    endcase
  end
end
S2: begin  // result is negative
    temp_result = (temp_num << 10) / (temp_den << 5);
    result = ~(temp_result) + 1; // maintains Qx.5 format
    done <= 1;
    if (~start) begin
        done <= 0;
        state <= S0;
    end
endcase
end
endmodule

/* HIGH RESOLUTION DIVIDER WITH 14 BITS FOR THE FRACTION NUMBER, 1 BIT FOR
INTEGER AND SIGNED BIT*/
module divider14(input clk,
    input reset,
    input start,
    input logic [15:0] num,
    input logic [15:0] den,
    output logic [15:0] res
    output logic done);

typedef enum logic [3:0] {S0, S1, S2, S3} state_t;
state_t state;
logic [15:0] temp_num;
logic [15:0] temp_den;
logic [31:0] temp_result;
always_ff @(posedge clk) begin
    done <= 0;
    if (reset) begin
        state <= S0;
    end
    else begin
        case (state)
            S0: begin
                if (start) begin
                    if (num[15] & den[15]) begin  // if both the
                        // num and den are negative
                            // divide only the magnitudes
                            temp_num = ~(num - 1);
                            temp_den = ~(den - 1);
                            state <= S1;
                        end
                        // following two conditions make sure the
                        result is negative
                        else if (num[15] & ~den[15]) begin  // if only
num is negative
magnitude of the numerator
    temp_num = ~(num - 1); // get
    temp_den = den;
    state <= S2;
end

else if (~num[15] & den[15]) begin // if only
    den is negative
    temp_num = num;
    temp_den = ~(den - 1);
    state <= S2;
end

else begin // both numbers are positive
    temp_num = num;
    temp_den = den;
    state <= S1;
end
end

S1: begin // result is positive
    temp_result = (temp_num << 16) / (temp_den << 2);
    result = temp_result[15:0];
    done <= 1;
    if (~start) begin
        done <= 0;
        state <= S0;
    end
end

S2: begin // result is negative
    temp_result = (temp_num << 16) / (temp_den << 2);
    result = ~(temp_result[15:0]) + 1; // maintains Qx.14
    done <= 1;
    if (~start) begin
        done <= 0;
        state <= S0;
    end
end
dcase
end

endmodule
SOFTWARE PROTOTYPE FILES IN PYTHON

####pygpu – RENDER MODULE in python
import numpy as np
import pygame
from math import *
from constants import *

# Class to describe a Camera object
class Camera(object):
    def __init__(self, position, target):
        self.position = position  # camera position in the 3d world
        self.target = target      # where the camera is looking (location)

# Class to describe a Mesh object
class Mesh(object):
    def __init__(self, vertices, faces, name, position, rotation):
        self.name = name          # mesh name
        self.vertices = vertices  # collection of (x,y,z) vertices to define mesh
        self.faces = faces        # collections of faces to define the mesh
        self.position = position  # its position in the 3d world
        self.rotation = rotation  # rotation state of the mesh

# Class to define the face of a mesh
# Face objects simply index the vertices index to get the
# three vertices of a face
class Face(object):
    def __init__(self, a, b, c):
        self.a = a  # faces are triangles so have a set of 3 indexes
        self.b = b
        self.c = c

# Class to define the rendering device
class Device(object):
    # constructor initializes frame buffer
    def __init__(self, pixel_width, pixel_height):
        self.pixel_width = pixel_width
        self.pixel_height = pixel_height

        pygame.init()  # init pygame
        self.main_clock = pygame.time.Clock()
        self.display = pygame.display.set_mode((self.pixel_width,
                                                 self.pixel_height), 0, 32)  # create display

        self.depth_buffer = np.full((self.pixel_width, self.pixel_height), MAX_VAL)

        # clear screen to one solid color
        def clear(self, color):
            self.display.fill(color)
            self.depth_buffer[:] = MAX_VAL  # clear depth buffer

        # set pixel
```python
def put_pixel(self, x, y, z, color):
    #print "z buffer shows: " + str(self.depth_buffer[x][y])
    #print "current z is: " + str(z)

    if self.depth_buffer[x][y] < z:
        #print "not drawing this"
        return  # ignore put pixel if depth buffer is less than z

    self.depth_buffer[x][y] = z
    self.display.set_at((x,y), color)

    # update output
    def update_display(self):
        pygame.display.update()

    # transform 3d coords to window coords using the transformation matrix
    def project_to_window(self, vertex, transform_matrix):
        # get clipped coordinates (to window)
        clip_coords = np.dot(transform_matrix, vertex)

        # get normalized device coordinates
        n_coords = clip_coords / float(clip_coords[3])

        # now translate into window coordinates
        viewport_x = 0  # specify x coordinate of bottom left-most point
        viewport_y = 0  # specify y coordinate of bottom left-most point
        viewport_w = self.pixel_width
        viewport_h = self.pixel_height

        xw = ((viewport_w / 2.0) * n_coords[0]) + (viewport_x + (viewport_w / 2.0))
        yw = ((viewport_h / 2.0) * n_coords[1]) + (viewport_y + (viewport_h / 2.0))
        zw = ((FAR - NEAR) / 2.0) * n_coords[2] + ((FAR + NEAR) / 2.0)

        return (xw, yw, zw)

    # does clipping to make sure only pixels on screen will be drawn
    def draw_point(self, point, color):
        x = point[0]
        y = point[1]
        z = point[2]

        self.put_pixel(int(x), int(y), z, color)

    # draw line b/t two points using Bresenham's algorithm
    def draw_line(self, point0, point1):
        x0 = int(point0[0])
        y0 = int(point0[1])
        x1 = int(point1[0])
        y1 = int(point1[1])

        dx = abs(x1 - x0)
        dy = abs(y1 - y0)

        if x0 < x1:
            sx = 1
```

73
else:
    sx = -1

if y0 < y1:
    sy = 1
else:
    sy = -1

err = dx - dy

while True:
    point_to_draw = (x0, y0)
    self.draw_point(point_to_draw)

    # check if we're done
    if ((x0 == x1) and (y0 == y1)):
        break

    # otherwise update our point
    e2 = 2 * err
    if e2 > -dy:
        err -= dy
        x0 += sx
    if e2 < dx:
        err += dx
        y0 += sy

    # clamp value between 0 and 1
    def clamp(self, value, min_val = 0, max_val = 1):
        return max( float(min_val), min(float(value), float(max_val)) )

    # interpolate the value between 2 vertices
    # min is the starting point, max is the ending
    def interpolate(self, min_val, max_val, gradient):
        val = float(min_val) + ( float(max_val) - float(min_val) ) * self.clamp(float(gradient))
        return val

    # draw line between two points left to right
    # papb -> pcpd
    def process_scan_line(self, y, pa, pb, pc, pd, color):
        # gradient between vertex a and vertex b
        if pa[Y] != pb[Y]:
            gradient1 = (y - pa[Y]) / (pb[Y] - pa[Y])
        else:
            gradient1 = 1

        # gradient between vertex c and vertex d
        if pc[Y] != pd[Y]:
            gradient2 = (y - pc[Y]) / (pd[Y] - pc[Y])
        else:
            gradient2 = 1

        # compute sx and ex
        sx = int(self.interpolate(pa[X], pb[X], gradient1))
        ex = int(self.interpolate(pc[X], pd[X], gradient2))
# starting z and ending z
z1 = self.interpolate(pa[Z], pb[Z], gradient1)
z2 = self.interpolate(pc[Z], pd[Z], gradient2)

for x in range(sx, ex):  # draw a point from startx to finish x
    gradient = (float(x) - float(sx)) / (float(ex) - float(sx))
    z = self.interpolate(z1, z2, gradient)
    self.draw_point((x, y, z), color)

# function to draw the triangle given 2 points and the color
def draw_triangle(self, p1, p2, p3, color):
    # first sort p1, p2 and p3 so that p1 is at the top
    # followed by p2 and then p3
    if p1[Y] > p2[Y]:
        temp = p2
        p2 = p1
        p1 = temp
    if p2[Y] > p3[Y]:
        temp = p2
        p2 = p3
        p3 = temp
    if p1[Y] > p2[Y]:
        temp = p2
        p2 = p1
        p1 = temp

    # now calculate inverse slopes
    if p2[Y] - p1[Y] > 0:
        dp1p2 = (float(p2[X]) - float(p1[X])) / (float(p2[Y]) - float(p1[Y]))
    else:
        dp1p2 = 0.0
    if p3[Y] - p1[Y] > 0:
        dp1p3 = (float(p3[X]) - float(p1[X])) / (float(p3[Y]) - float(p1[Y]))
    else:
        dp1p3 = 0.0

    # now draw scan line for the triangles
    start_y = int(p1[Y])
    end_y = int(p3[Y]) + 1
    if dp1p2 > dp1p3:
        for y in range(start_y, end_y):
            if y < p2[Y]:
                self.process_scan_line(y, p1, p3, p1, p2, color)
            else:
                self.process_scan_line(y, p1, p3, p2, p3, color)
    else:
        for y in range(start_y, end_y):
            if y < p2[Y]:
                self.process_scan_line(y, p1, p3, p1, p2, color)
            else:
                self.process_scan_line(y, p2, p3, p1, p3, color)
# main render function that re-computes each vertex projection each frame

def render(self, camera, meshes):
    view = view_matrix(camera, UP_VECTOR)

    # generate projection matrix
    aspect_ratio = float(self.pixel_height) / float(self.pixel_width)
    projection = perspective_projection(NEAR, FAR, ANGLE_OF_VIEW,
                                        aspect_ratio)

    for mesh in meshes:
        # generate world matrix for the mesh
        rotation_angles = (mesh.rotation[0], mesh.rotation[1],
                           mesh.rotation[2])
        translations = (mesh.position[0], mesh.position[1],
                         mesh.position[2])
        world = world_matrix(SCALING, rotation_angles, translations)

        # generate final transform matrix
        transform_matrix = np.dot(view, world)
        transform_matrix = np.dot(projection, transform_matrix)

        faceindex = 0
        for face in mesh.faces:
            # now render each face on the screen
            vertex_a = mesh.vertices[face.a]
            vertex_b = mesh.vertices[face.b]
            vertex_c = mesh.vertices[face.c]

            # convert the 3-d vertices to 2-d pixels that can be drawn
            pixel_a = self.project_to_window(vertex_a, transform_matrix)
            pixel_b = self.project_to_window(vertex_b, transform_matrix)
            pixel_c = self.project_to_window(vertex_c, transform_matrix)

            # color triangles
            color = 2 + (faceindex % len(mesh.faces)) * 100 / len(mesh.faces)

            if faceindex == 0 or faceindex == 1:
                color_vec = (255, 255, 0, 255)
            else:
                color_vec = (color, color, color, 255)
            self.draw_triangle(pixel_a, pixel_b, pixel_c, color_vec)
            faceindex += 1

    # draw a line between each pixel
    self.draw_line(pixel_a, pixel_b)
    self.draw_line(pixel_b, pixel_c)
    self.draw_line(pixel_c, pixel_a)

###############################
#
# Matrix transform functions
#
###############################
# create scale matrix

def scale_matrix(cx, cy, cz):
    scale = np.identity(4)  # start with 4x4 identity matrix
    scale[0, 0] = cx
    scale[1, 1] = cy
    scale[2, 2] = cz

    return scale

def translate_matrix(dx, dy, dz):
    trans = np.identity(4)
    trans[0, 3] = dx
    trans[1, 3] = dy
    trans[2, 3] = dz

    return trans

def rotate_matrix_x(angle):
    rad = angle * 0.01743  # pi/180 = 0.01743

    rotate = np.identity(4)
    rotate[1, 1] = cos(rad)
    rotate[1, 2] = sin(rad)
    rotate[2, 1] = -sin(rad)
    rotate[2, 2] = cos(rad)

    return rotate

def rotate_matrix_y(angle):
    rad = angle * 0.01743  # pi/180 = 0.01743

    rotate = np.identity(4)
    rotate[0, 0] = cos(rad)
    rotate[0, 2] = -sin(rad)
    rotate[2, 0] = sin(rad)
    rotate[2, 2] = cos(rad)

    return rotate

def rotate_matrix_z(angle):
    rad = angle * 0.01743  # pi/180 = 0.01743

    rotate = np.identity(4)
    rotate[0, 0] = cos(rad)
    rotate[0, 1] = -sin(rad)
    rotate[1, 0] = sin(rad)
    rotate[1, 1] = cos(rad)

    return rotate

def orthographic_projection(near, far, left, right, bottom, top):
    ortho = np.zeros((4, 4))

    # first column changes
    ortho[0, 0] = 2.0 / (right - left)

    # second column changes
    ortho[1, 1] = 2.0 / (top - bottom)
# third column changes
ortho[2, 2] = -2.0 / (far - near)

# fourth column changes
ortho[0, 3] = -(right + left) / (right - left)
ortho[1, 3] = -(top + bottom) / (top - bottom)
ortho[2, 3] = -(far + near) / (far - near)
ortho[3, 3] = 1

return ortho

def perspective_projection(near, far, angle_of_view, aspect_ratio):
def perspective_projection(near, far, angle_of_view, aspect_ratio):
    # start with some maths needed to get the matrix
    rad_view = angle_of_view * 0.01743  # change angle of view to rads
    size = near * tan(rad_view / 2.0)
    left = -size
    right = size
    bottom = -size / aspect_ratio
    top = size / aspect_ratio

    persp = np.zeros((4, 4))
    # first column changes
    persp[0, 0] = 2.0 * near / (right - left)
    # second column changes
    persp[1, 1] = 2.0 * near / (top - bottom)
    # third column changes
    persp[0, 2] = (right + left) / (right - left)
persp[0, 2] = (right + left) / (right - left)
persp[1, 2] = (top + bottom) / (top - bottom)
persp[2, 2] = -(far + near) / (far - near)
persp[3, 2] = 1.0

    # fourth column changes
    persp[2, 3] = -(2 * far * near) / (far - near)
    return persp

# create the view matrix where you pass the camera object
def view_matrix(camera, up):
def view_matrix(camera, up):
    # first specify the translation matrix based on camera position
    translate = translate_matrix(camera.position[0], camera.position[1],
    translate = translate_matrix(camera.position[0], camera.position[1],
camera.position[2])
    translate = np.linalg.inv(translate)  # invert translate matrix

    # create target and up vectors
    target_vector = np.array([camera.target[0], camera.target[1],
camera.target[2]])
    up_vector = np.array([up[0], up[1], up[2]])

    # now get camera rotation matrix based on tutorial
    http://ogldev.atspace.co.uk/www/tutorial13/tutorial13.html
    N = target_vector
    N = N / np.linalg.norm(N)  # need to normalize N array

    U = up_vector
    U = U / np.linalg.norm(U)  # need to normalize U array
    U = np.cross(U, target_vector)

    V = np.cross(N, U)
# generate matrix based on U,V,N vectors
rotate = np.identity(4)  # start with identity matrix
rotate[0, 0] = U[0]
rotate[0, 1] = U[1]
rotate[0, 2] = U[2]
rotate[1, 0] = V[0]
rotate[1, 1] = V[1]
rotate[1, 2] = V[2]
rotate[2, 0] = N[0]
rotate[2, 1] = N[1]

view = np.dot(rotate, translate)
return view

def world_matrix(scale_factors, rotation_angles, translations):
    # scale the object
    scale = scale_matrix(scale_factors[0], scale_factors[1], scale_factors[2])
    # rotate the object
    rotate_x = rotate_matrix_x(rotation_angles[0])
    rotate_y = rotate_matrix_y(rotation_angles[1])
    rotate_z = rotate_matrix_z(rotation_angles[2])

    # translate the object
    translate = translate_matrix(translations[0], translations[1], translations[2])

    world = np.dot(rotate_x, scale)
    world = np.dot(rotate_y, world)
    world = np.dot(rotate_z, world)
    world = np.dot(translate, world)
    return world

# application that uses the above pygpu render module
from pygpu import *
import numpy as np
import pygame
import json

def load_mesh_from_json(filename):
    meshes = []  # list of meshes we will return

    json_data = open(filename)  # open json file to read
data = json.load(json_data)  # load the data for parsing

    num_meshes = len(data['meshes'])  # holds number of meshes in model

    for mesh_index in range(num_meshes):
        # first get vertices and indices array
        vertices_array = data['meshes'][mesh_index]['vertices']
        indices_array = data['meshes'][mesh_index]['indices']
# get uvCount (i have no idea what this is???)
> uv_count = int(data['meshes'][mesh_index]['uvCount'])

vertices_step = 1

# determines how many of the vertices we
# want to use from the json file. we don't
# need all of them
if uv_count == 0:
    vertices_step = 6
elif uv_count == 1:
    vertices_step = 8
elif uv_count == 2:
    vertices_step = 10

# get number of vertices we care about
vertices_count = len(vertices_array) / vertices_step

# get number of faces we care about which is size of arr / 3 (a,b,c)
faces_count = len(indices_array) / 3

# get the vertices array
vertices = []  # init empty vertices list
for index in range(vertices_count):
    x = float(vertices_array[index * vertices_step])
    y = float(vertices_array[index * vertices_step + 1])
    z = float(vertices_array[index * vertices_step + 2])
    w = 1

    vertices.append(np.matrix([[x], [y], [z], [w]]))

# get the faces array
faces = []  # init empty faces list
for index in range(faces_count):
    a = int(indices_array[index * 3])
    b = int(indices_array[index * 3 + 1])
    c = int(indices_array[index * 3 + 2])

    faces.append(Face(a, b, c))

# get the position specified in blender
px = float(data['meshes'][mesh_index]['position'][0])
py = float(data['meshes'][mesh_index]['position'][1])
pz = float(data['meshes'][mesh_index]['position'][2])

position = (px, py, pz)

# get mesh name
name = data['meshes'][mesh_index]['name']

# set default rotation values
rotation = (0, 0, 180)

# now we have all of the mesh data so we can create he mesh object
new_mesh = Mesh(vertices, faces, name, position, rotation)

meshes.append(new_mesh)  # add new mesh to our meshes array

return meshes
my_device = Device(500, 500)  # screen 100 x 100

camera_position = (0, 0, 5)  # move 5 units along the z axis
camera_target = (0.0, 0.0, 1.0)  # pointed straight at z axis
my_camera = Camera(camera_position, camera_target)

meshes = load_mesh_from_json("monkey.babylon")

done = False

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:  # wait for user to close window
            done = True
            break

    if done:
        break

    my_device.clear((0, 0, 0, 255))  # clear screen
    # rotate cube slightly during each frame rendered
    for mesh in meshes:
        mesh.rotation = (mesh.rotation[0], mesh.rotation[1] + 40,
                         mesh.rotation[2])  # move mesh position slightly
        my_device.render(my_camera, meshes)  # render meshes
        my_device.update_display()  # update display

# CONSTANTS.PY–HOLDS the constants of the programs–easy to modify

# important constants
NEAR = 1  # how close to you you can see
FAR = 10  # how far out you can see
ANGLE_OF_VIEW = 45.0  # angle of view in front of you
UP_VECTOR = (0, 1, 0)  # direction of camera UP. leave at (0, 1, 0) for now
SCALING = (1.0, 1.0, 1.0)  # object scaling
MAX_VAL = 9999

# indicies
X = 0
Y = 1
Z = 2
W = 4