3D RENDERING IN FPGA

CSEE 4840
Spring 2014

Earvin, Gautham, Garvit and Annjana

Original proposal

- We had initially proposed to make a Ball Balancer Mario Party game
- Milestones were designed to implement the whole game
- A 3D plate with a ball on it, was to be balanced by the player using a PS3 controller

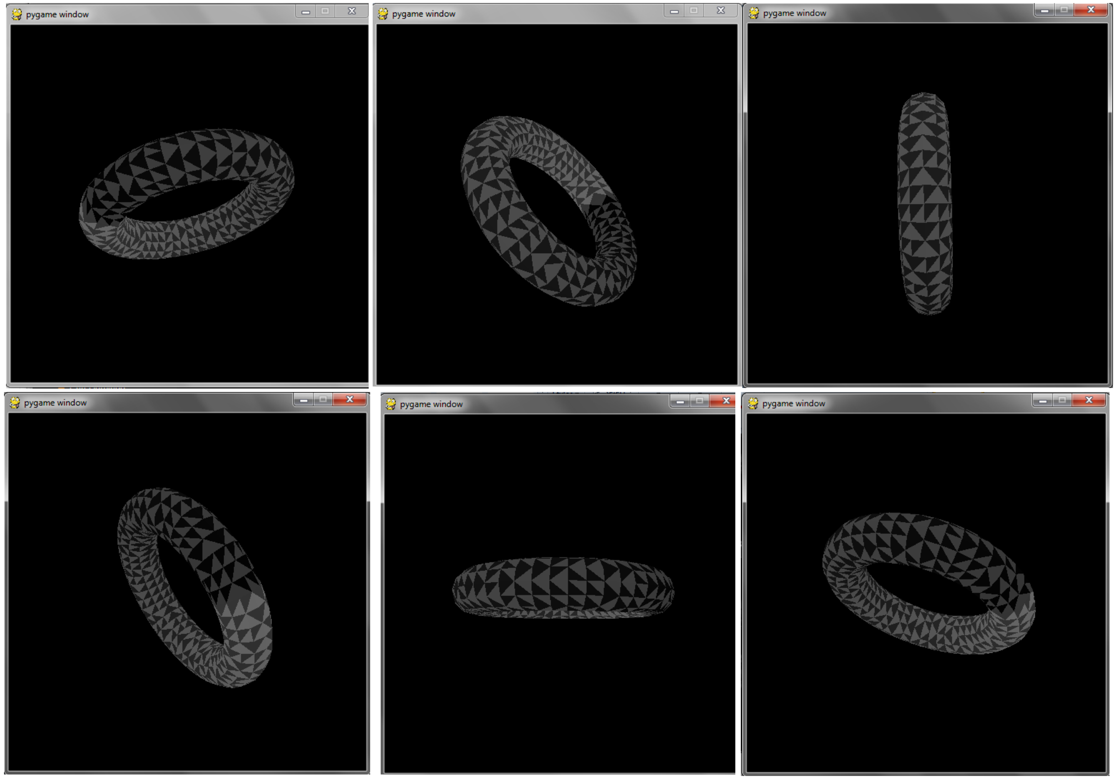
Actual Implementation

- …. Well, then we decided to concentrate on the most difficult part of the game, and make sure we got that right - which is the 3D Rendering part, which includes a Shader module and z-buffering
- Using the combination of both software and hardware, our project can render any object of your choice in 3-D, and it can dance/'blink' (you'll know why later) to the whims of the PS3 controller!

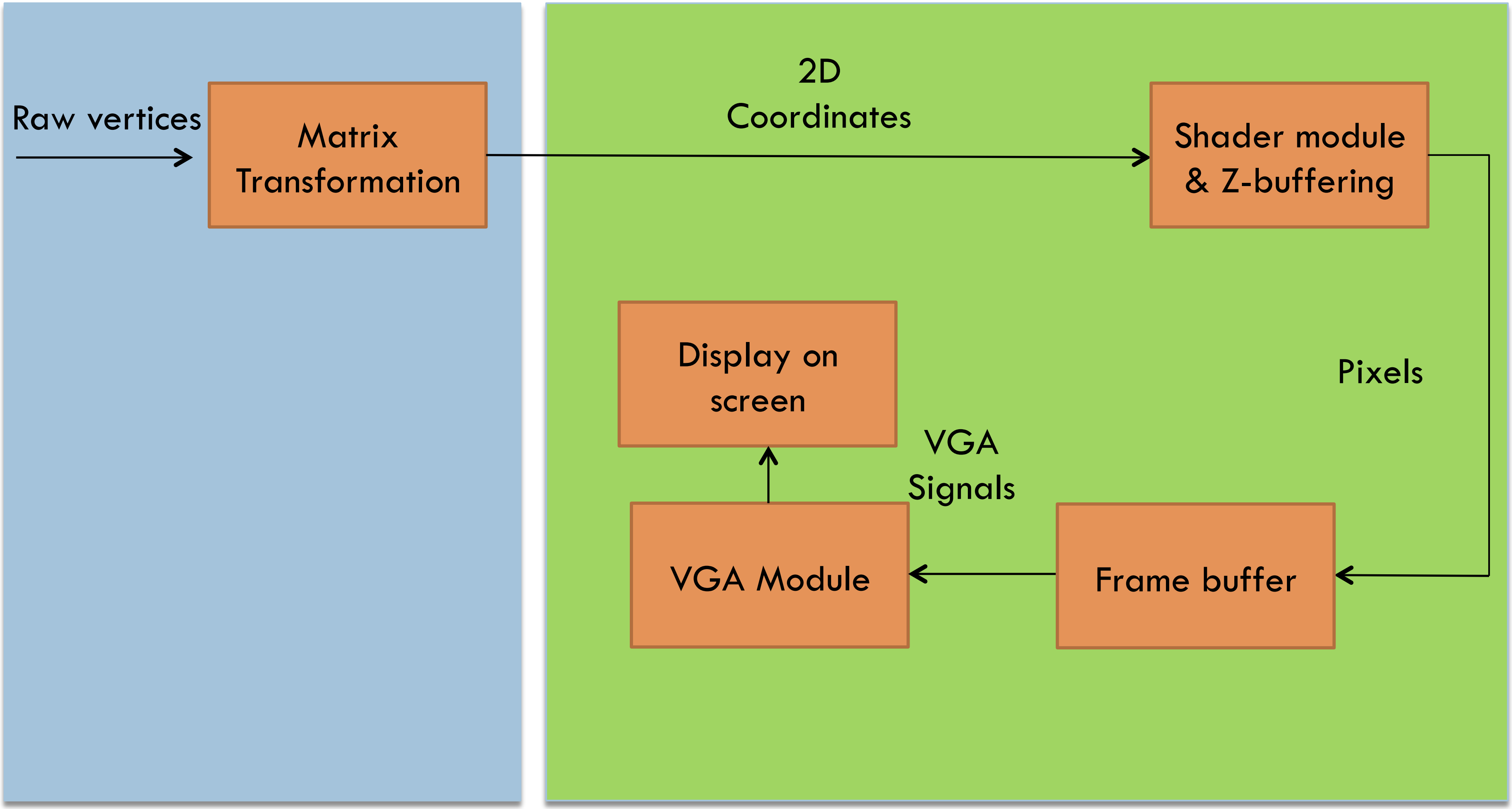
Verification of Design

- For proof of concept, we wrote the entire 3D Render module in Python
- The mathematics involved in the project was tested and verified
- The aim was to translate this functionality in an equivalent hardware-software interface on the Sockit board
- This was fundamental for building our project

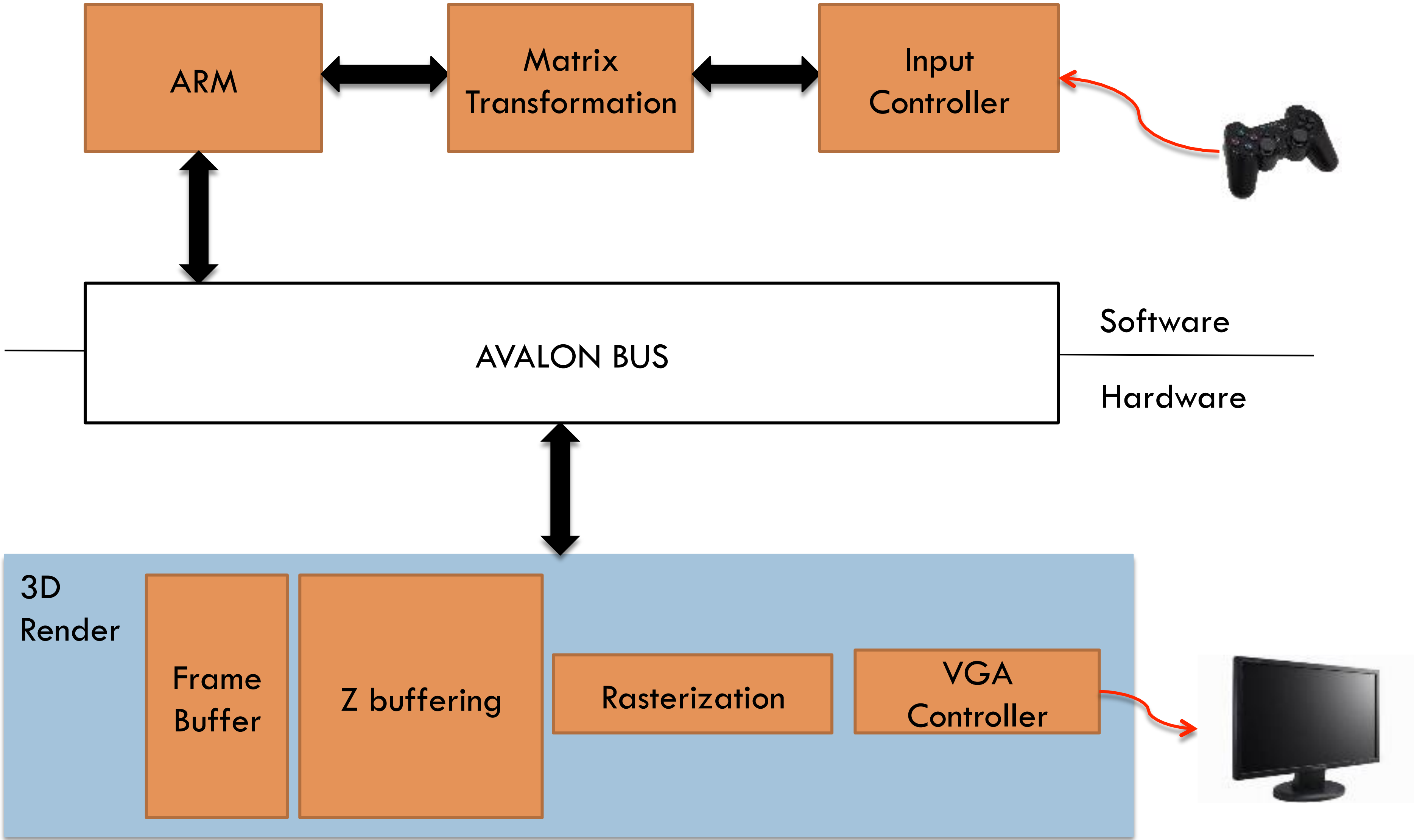
Blender module sample



3D Rendering Design Flow



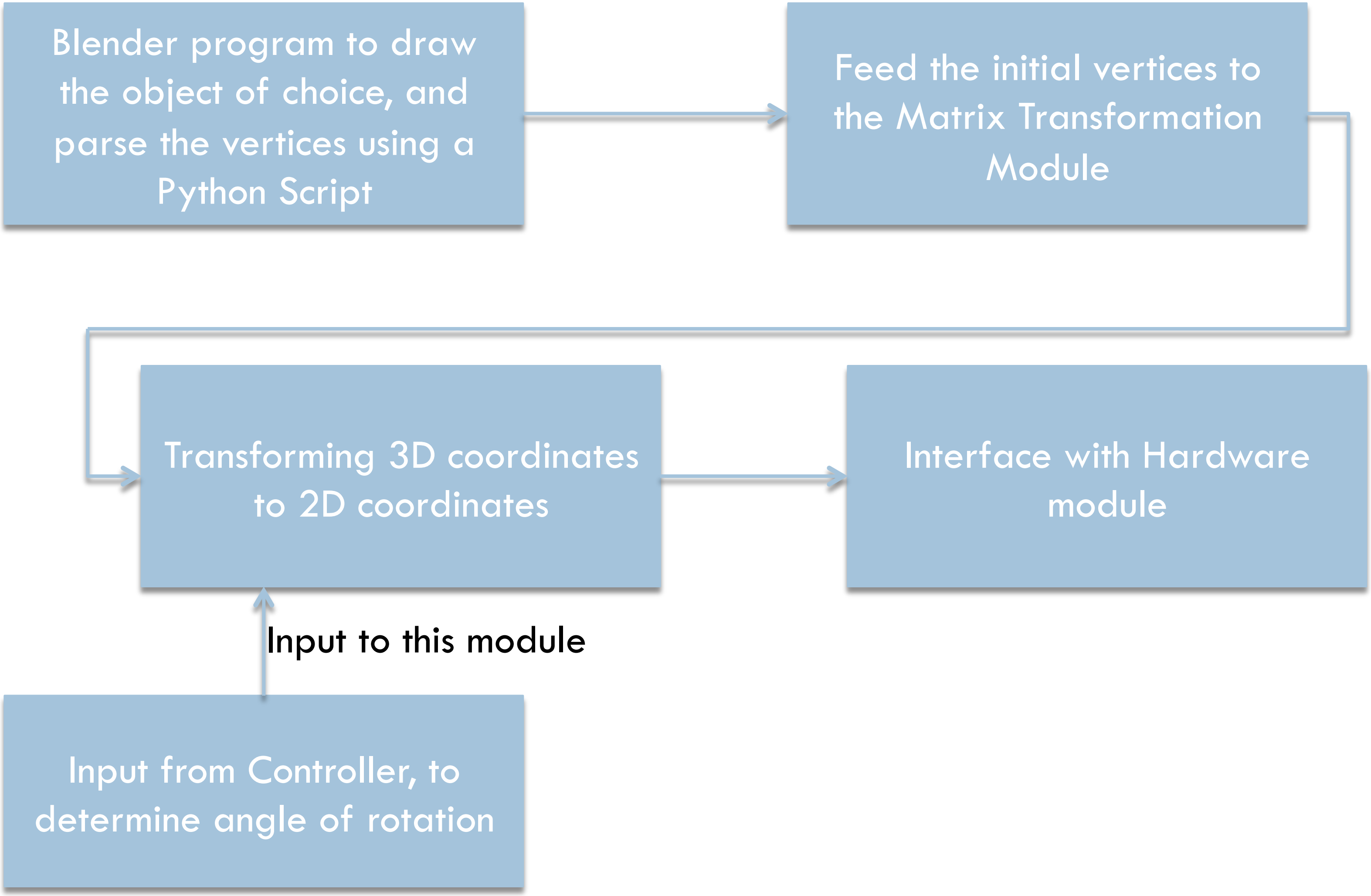
System Overview



Software Overview

- Blender program
  - 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
- Matrix Transformation
  - The mathematical calculation of the model that takes angles as inputs, and transforms them into vertices
- PS3 Controller
  - Interface the controller, and map the input from the controller to appropriate angles that are fed into the Matrix Transformation module
- Software interfacing
  - Software driver to communicate with the hardware

Flow of Software



Blender program to draw the object of choice, and parse the vertices using a Python Script

Feed the initial vertices to the Matrix Transformation Module

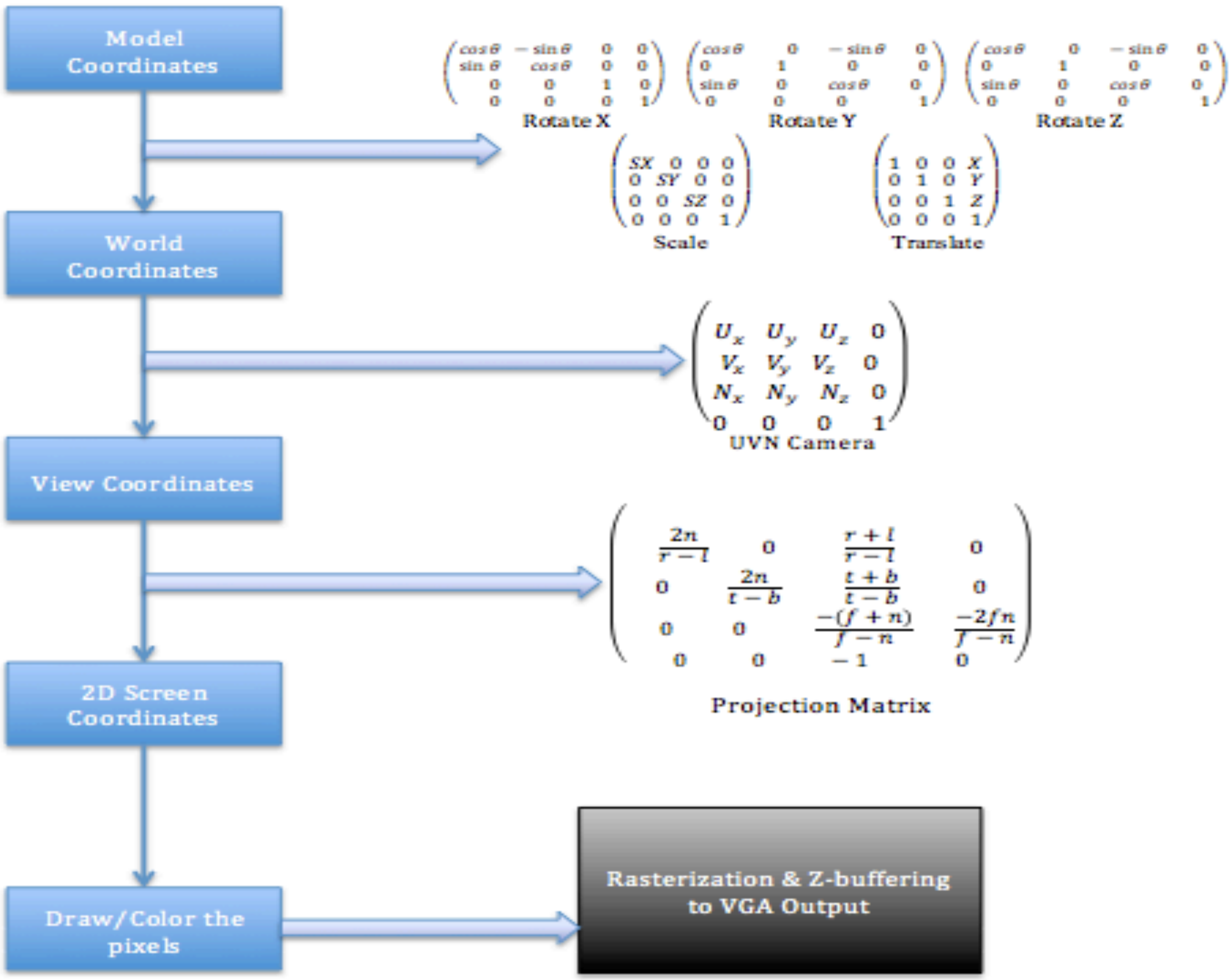
Transforming 3D coordinates to 2D coordinates

Interface with Hardware module

Input to this module

Input from Controller, to determine angle of rotation

Matrix Transformation



$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate X

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate Y

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate Z

$$\begin{pmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scale

$$\begin{pmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translate

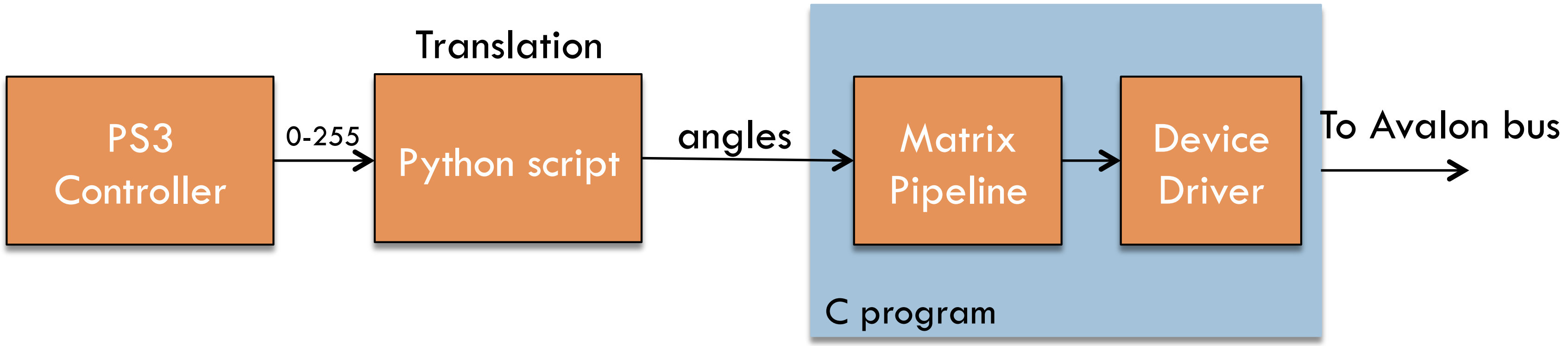
$$\begin{pmatrix} U_x & U_y & U_z & 0 \\ V_x & V_y & V_z & 0 \\ N_x & N_y & N_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

UVN Camera

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Projection Matrix

Controller module



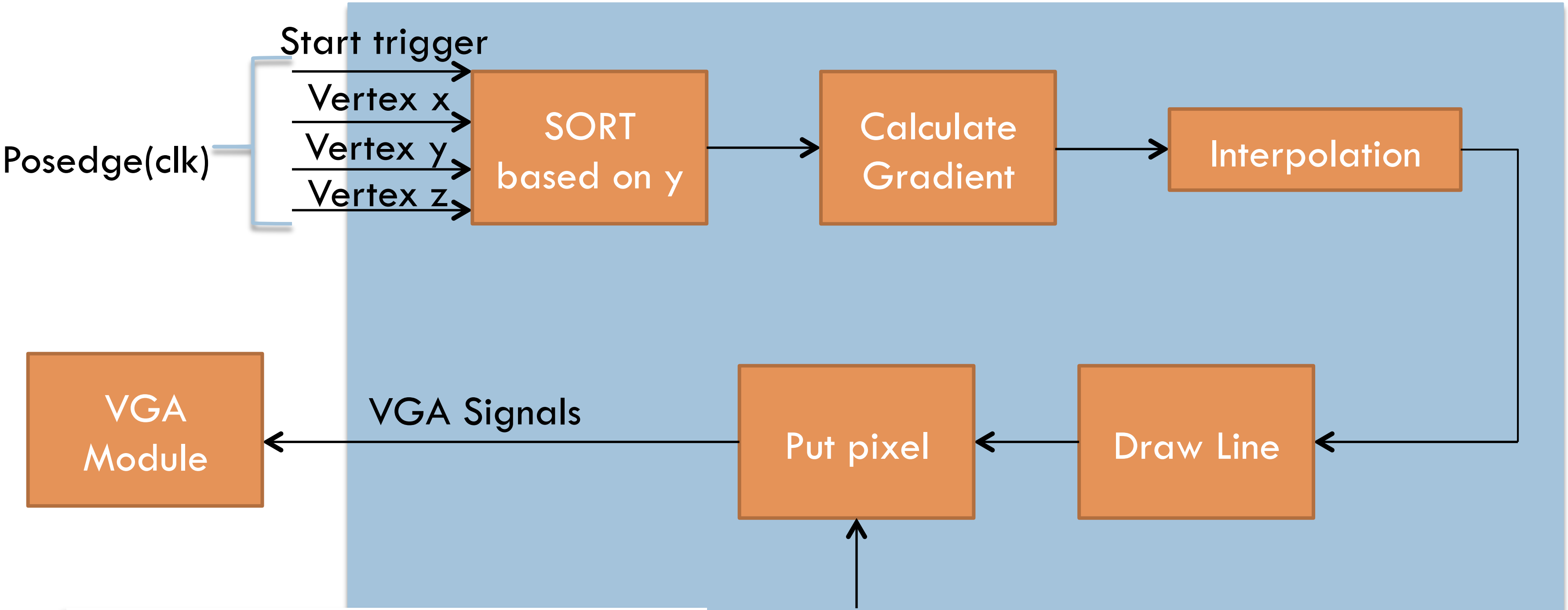
Hardware Overview

- 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 – Rasterization and display

Shader module & Z-buffering



Posedge(clk)

Start trigger

Vertex x

Vertex y

Vertex z

SORT based on y

Calculate Gradient

Interpolation

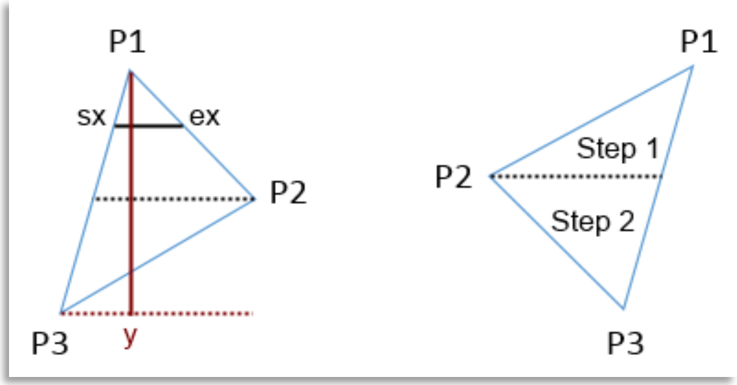
Draw Line

Put pixel

VGA Signals

VGA Module

Color picker (2 bit)



Problems 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