ECE 385

FA 2020

Prof. Chushan Li

Final Project Proposal
FPGA-based 3D Graphics Renderer

Xie Tian 3180111631
Guan Zimu 3180111630

Date: Dec 5, 2020

Table of Contents

1 Idea and Overview

We propose to desgin and implement a FPGA-based 3D graphics renderer. Our renderer will be able to implement a basic 3D graphics pipeline to render an object, including model, view, projection transformation, rasterization and some further functionality if possible. Details will be introduced in next paragraph. We will implement all steps of MVP transformations and rasterization using SystemVerilog essential components such as the System Bus, SRAM, SDRAM, BRAM, Video Display and Keyboard. Our design will also include a NIOS II CPU for the purpose of interfacing with the USB keyboard so that we can change the view of the object. Additionaly, the NIOS II CPU may also process the .obj 3d model file and pass the triangles data into the hardware part to render. Our goal is to demonstrate a 3D object using VGA monitor, which can be adjusted by USB keyboard. Because we need a lot of decimal calculations, we will use Intel floating point IP core, or open source fixed point library if necessary. We will refer to some papers and projects in github to get some ideas, including but not limited to *FPGA-based 3D Graphics Pipeline and Advanced Rendering Effects* by Fotis Pegios and *FPGA-3D-Renderer-ECE385* by KevinPal.

Here is a breif introduction to 3D graphics pipeline. A 3D graphics pipeline is a process to display 3D objects on a 2D scene. In computer, an 3D object is usually stored as a bunch of triangles because their "discrete" property is easy for computer to handle with. To be specific, these data are usually 3D coordinates of vertices, sets of points that can form triangle sufaces, normal vectors of surfaces and so on. However, what we see on the display is just 2D images, so we need to project 3D objects onto a 2D scene, according to some rules (usually perspective relations). This step is called model, view, projection transformation, or MVP transformation. Obviously, it is not enough, because the resolution of a display is not infinity, it is composed of bunch of pixels. Therefore, we should convert the nearly continious object data into pixels, which is called rasterization. During the rasterization, we would have some algorithm, usually Bresenham algorithm, to draw lines. Besides, we probably need to compute and display the illumination of the object according to its properties. This is the so called shading. In real-time rendering, speed is significant, so engineers use texture images instead of real physical properties to give objects' color (The latter method is used in off-line rendering, usually done by ray tracing). After the calculation of one triangle is finished, the depth of it should be consider because what we would see is the nearest object. That's why we use a z-buffer to store the depth of the pixel. When we done all these step for all triangles and write colors into the frame buffer, an object can be displayed on the screen. Actually, there are numerous other techenologies in 3D graphics pipeline, for example, perspective correction, homogeneous space clipping, anti-aliasing, etc. Due to the limit time and complexity of hardware desgin, we are not going to implement these.

links:

- Intel floating point IP core:
  www.intel.com/content/www/us/en/programmable/documentation/eis1410764818924.html
- Open source fixed point library:
  github.com/freecores/verilog_fixed_point_math_library

- FPGA-based 3D Graphics Pipeline and Advanced Rendering Effects: dias.library.tuc.gr/view/manf/63332
- FPGA-3D-Renderer-ECE385: github.com/KevinPal/FPGA-3D-Renderer-ECE385

2 Block Diagram

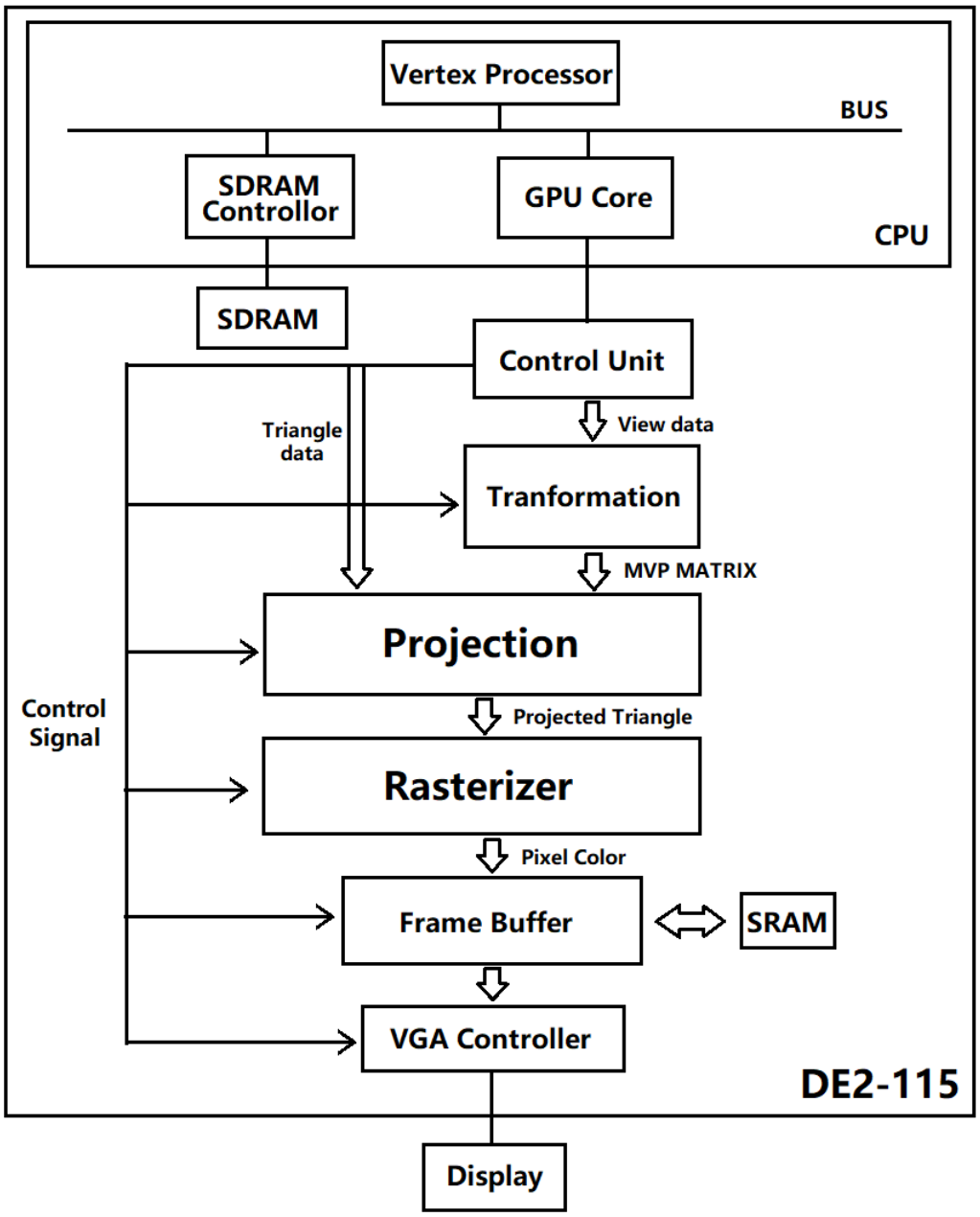

FIGURE 1. Block Diagram of Final Project

3 List of Features

3.1 Baseline set of features for the project to be considered working

- Able to use NIOS II as a CPU to give the information of the rendered object to the hardware design.
- Able to calculate the model, view, projection matrixes according to the given location and perspective of camera and apply matrix multiplication to get the final MVP transformation matrix in hardware design.
- Able to rasterize the triangle (basiclly draw lines) using line drawing algorithm (probably Bresenham algorithm) in hardware design.
- Able to use frame buffer and output the rendered image to the dispaly through VGA port in hardware design.
- Able to control complex data flow during rendering through multiple finite state machines in hardware design.

3.2 Additional features that may be implemented for extra difficulty

- Able to use USB keyboard to control the view of the camera in NIOS II.
- Able to read .obj file, analyse it and generate triangles data in NIOS II.
- Able to realize illumination in rendering, would need to analyse normal vectors of the object and interpolation. In hardware desgin.
- Able to realize texture mapping in rendering in hardware desgin (need to consider the u-v coordinates of texture).
- Able to apply some parallel calculation features in hardware desgin.

4 Expected Difficulty

10 for the Baseline set of features and 10++ for the Additional features.

First of all, the framework of the graphics pipeline is very different from what we learn to display games in the lectures. Our group members should spend a lot of time to learn many things about the pipeline. Besides, because there is no decimal numbers in hardware, we need to learn and realize computing floating point data (or fixed point data) in harware and apply these complex calculations while generating MVP transform matrixs (including matrix multiplications). We will also realize line drawing algorithm which need us to desgin the hardware component. The timing of the frambuffer and VGA output is also a difficult problem. Finally, to desgin a state machine to control all these data flow is also difficult enough.

For the Additional features, using USB key board to control the view of the camera need us to consider the different speed of CPU and hardware, and create an interface. Analysis of .obj file and generate triangle data require a comprehensive understand of .obj format. For illumination, we need to process the relation between normal

vectors and light sources, interpolate values inside the triangle, needing extra calculation. Also, a z-buffer is needed if we want to color the surface of triangles. So we also need to consider the speed of memories on the board so that z-buffer can work smoothly. For texture mapping, we should store texture in memory and find the color of corresonding pixel in texture. For parallel calculation, we need to understand and implement parallel alogrithm, which is extremely difficult.

5 Proposed Timeline

- WEEK 1: understand the graphics pipeline, desgin the framework of the project.
- WEEK 2: implement line drawing algorithm and MVP matrix generating separately.
- WEEK 3: Design the data flow between different module, complete the frame buffer desgin.
- WEEK 4: Apply additional features.
- Mid-checkpoint: we would be able to draw lines on the display according to the given triangle vertexes or calculate MVP matrix according to the given data of the camera and output the matrix to the console. (Because mid-checkerpoint is too close, only almost one week, and there are three exams near it, so we hold a negative attitude to what we could do in just 1 week.)