Habib University

Implementing the Graphics Pipeline

STRP 2022

Final Research Report

Muhammad Meesum Ali Qazalbash

ID: 06861

Supervisor: *Dr Waqar Saleem*

August 21, 2022

Contents

1 Problem Statement

To bring the graphics pipeline into software. The pipeline is tasked with rendering the scene on the screen. The pipeline will serve as a template for students in following CG courses.

2 Rationale

Graphical Pipeline is often implemented as hardware on GPUs. We will create the pipeline in software to better understand it. Later on, we'd like students to construct their own pipeline, stage by stage.

3 Literature Review

3.1 Introduction

This was my first introduction to computer graphics. Dr. Waqar Saleem recomended a book *Interactive Computer Graphics: A Top-Down Approach with WebGL*[1], 7th edition, for my foundational knowledge. The fundamentals of computer graphics and the elements necessary to comprehend the graphical pipeline were both addressed in this book.

3.2 Related Work

I identified a handful repositories on GitHub. They were decent, but virtually all of them either employed any framework or library that ran the pipeline on the GPU, such as WebGL, OpenGL, or DirectX, or their code was very specific to the purpose. Because we want to emulate the pipeline on the CPU, this was not a suitable assistance for our project.

3.3 WebGL for experiments

I used WebGL - a framework of JavaScript for computer graphics - to get some hands on experience of how the shader works and likewise how the pipeline works. WebGL was very well developed and was very easy to use. I used basic structure of the WebGL to design my own pipeline.

4 Methodologies

The methodology for the project was as follows:

1. Literature Review
2. Designing the Pipeline
3. Testing the Pipeline

The design for the pipeline is discussed in the next section. But for the testing we have render simple scene like random points, lines and triangles.

5 Design

The pipeline is inspired from the working of WebGL. There are three major stages in WebGL.

1. Application Program
2. GL Context
3. Graphical Pipeline

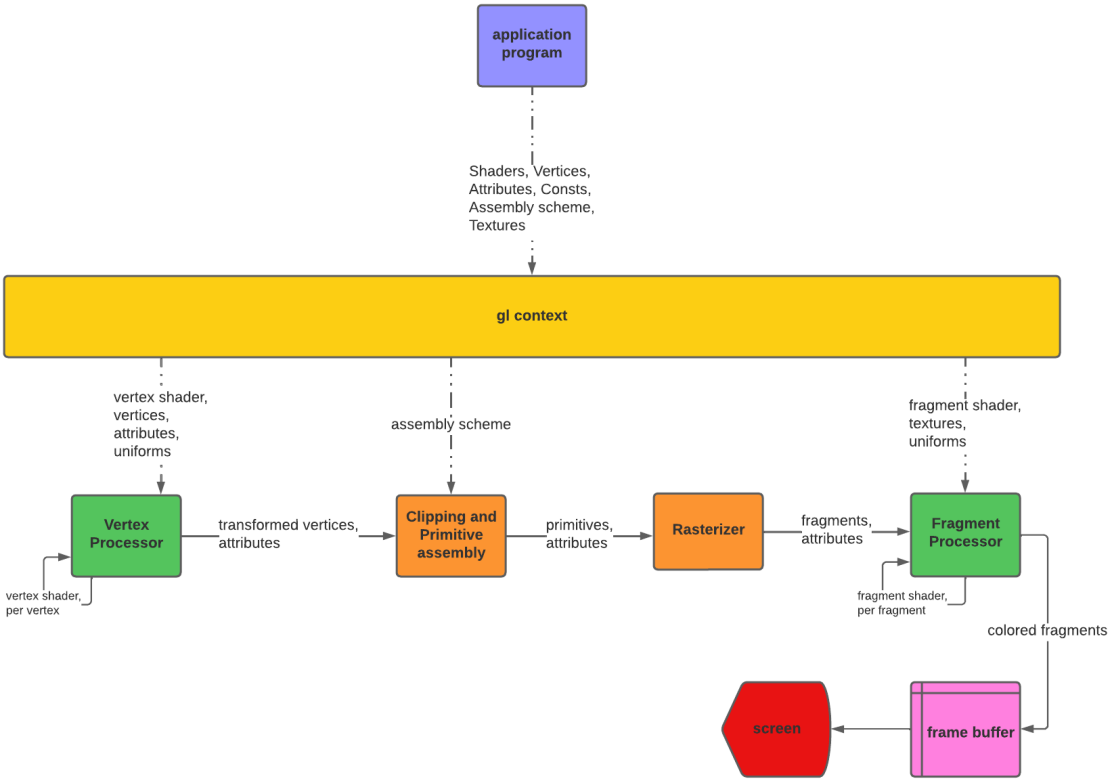

Figure 1: Design of the pipeline

5.1 Application Program

The application program is provided with all the essential pieces of the pipeline.

5.2 GL Context

The gl-context is the object that is used to store all the information about the pipeline. This object can be accessed by any stage in the pipeline.

5.3 Graphical Pipeline

Then there are four stages in the pipeline. In the 1, the stages are depicted. Two of them are green and other two are orange. Green stages are programmable, which means that the program that would run on in these stages is provided by the user. Orange stages are fixed, which means that their logic is fixed.

5.3.1 Vertex Processor

The only task of the vertex processor is to run the vertex shader. The shader will be provided by the user. But there is one constraint on it is that the shader can only have two parameter; first one is the dictionary with all attributes of the vertex and second one is the dictionary with all uniforms, and the shader should return the transformed vertex as a 4-dimensional vector.

5.3.2 Clipping & Primitive Assembly

The clipping stage would clip and assemble vertices into the primitives. The assembling scheme is provided by the user. There are seven different types of assembling schemes that we have considered.

1. Point
2. Line
3. Line-strip
4. Line-loop
5. Triangle
6. Triangle-strip
7. Triangle-fan

This stage will return the vertices that are inside the viewport as primitives.

5.3.3 Rasterizer

The rasterizer's task is the most simple of all, it has to raster the primitives coming from the clipping stage. It will return the fragments that are raster by the primitives.

5.3.4 Fragment Processor

The fragments coming from the rasterizer are colored in the fragment processor. The fragment processor will run the fragment shader - provided by the user - to color each fragment. There are same restrictions on fragment shader as of vertex shader. The fragment shader instead of returning colored fragments to frame buffer, it would render the scene as a form of image.

6 Research Results

The result is the pipeline soley built on python3 that is running on CPU, but It is a little but faster because of the vecotrization in the Clipping & Primitive Assembly Stage and Rasterizer Stage. The pipeline is in the GitHub repository. One of the output from the pipeline is the rendered image.

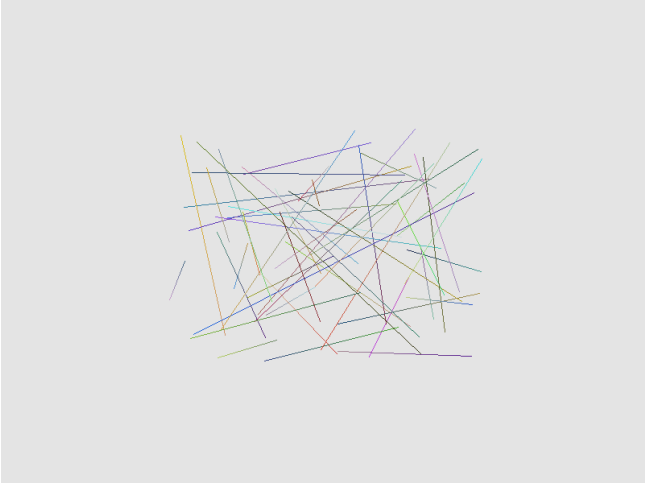

Figure 2: Some radome lines generated by the pipeline

7 Conclusion

We have successfully emulated the graphical pipeline in software, before that it was only existing in the hardware on GPUs.

8 Reflections on Research Journey

8.1 Challenges

This was first experience in research (under a supervisor) and also first introduction to computer graphics. Initially things were pretty fast because I had to understand almost the 3 to 4 weeks material of computer graphics by my self in just 2 weeks, but later Dr Waqar understand the situation and helped me alot.

8.2 Future Work

In the Clipping & Primitive Assembly stage and the Rasterization stage I did vectorization[1]. I wish to devise some way to vectorize the vertex shader and fragment shader.

9 Acknowledgements

References

Angel, E. and Shreiner, D., 2016. *Interactive Computer Graphics: A Top-Down Approach with WebGL*. 7th ed. Boston [etc.]: Pearson.

[1] Running a process all at once for every object