Mert Furkan Ergüden 170709015

PART 1: FIXED FUNCTION PIPELINE VS PROGRAMMABLE SHADERS

Introduction

In this paper I examined evolution of Shaders;fixed Function Pipeline to Programmable Shaders.The following information,I have created in my computer Graphics Lesson and as a result of some research on the internet.

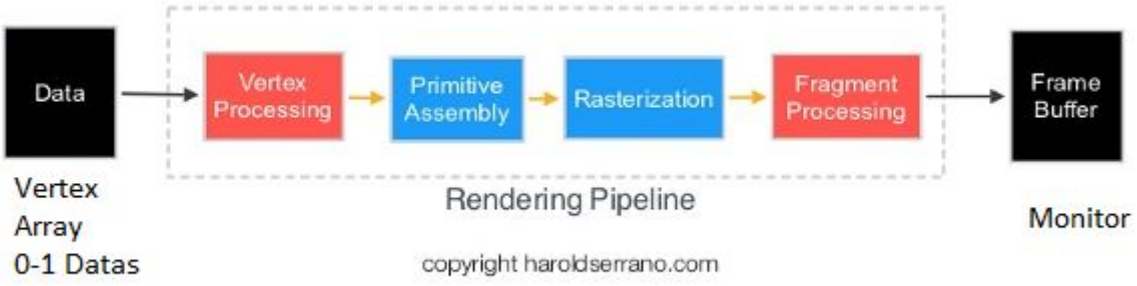
A shader is a piece of code that is executed on the **Graphics Processing Unit (GPU)**, usually found on a graphics card, to manipulate an image before it is drawn to the screen. Shaders allow for various kinds of rendering effect, ranging from adding an X-Ray view to adding cartoony outlines to rendering output.

Fixed function pipeline GPUs have exactly one illumination mode: A Lambertian illumination model, implemented using Gourad or Phong shading. There were a few tricks to slightly alter the illumination model, for example to be anisotropic filtering, but you had to somehow outsmart (or outdumb to be hones) the GPU for this. With a programmable pipeline you simply do what you wanted to do in the first place.

However ,In a programmable pipeline however the GPU is clean slate. It's completely up to the programmer how the various stages of the rendering process (vertex transformation, tesselation, fragment processing) are carried out.
And we can use which formula is useful fit for the task.

Fixed-Function Pipeline

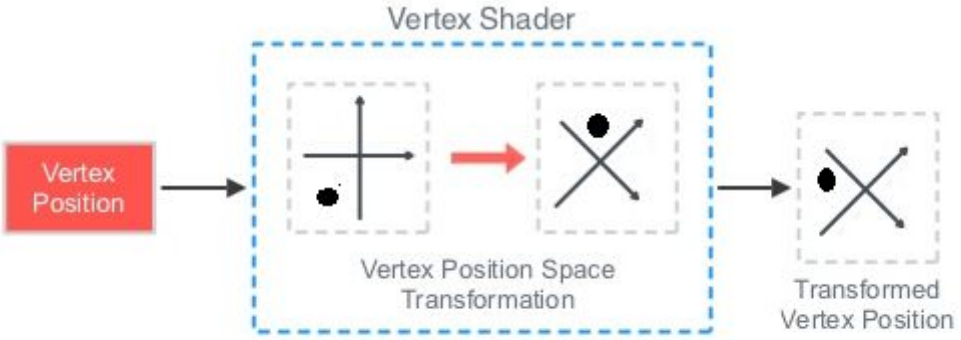
The fixed-function pipeline is functionality fixed.I want to example, if the pipeline contains a list of methods to rasterize geometry and shade pixels, that is pretty much it.
You cannot add any more methods.Although these pipelines were good at rendering scenes, they enshrine certain deficiencies. In broad terms, you can perform linear transformations and then rasterize by texturing, interpolate a color across a face by combinations and permutations of those things.



After that we are going to process that data into **Rendering Pipeline** and **Rendering Pipeline** is the stages your device goes through before the test appears. No CPU is involved in the processing line.
The GPU is responsible for this process.Here image I want to explain Rendering Pipeline As shown in the picture, fixed-function pipeline consists of 4 steps.

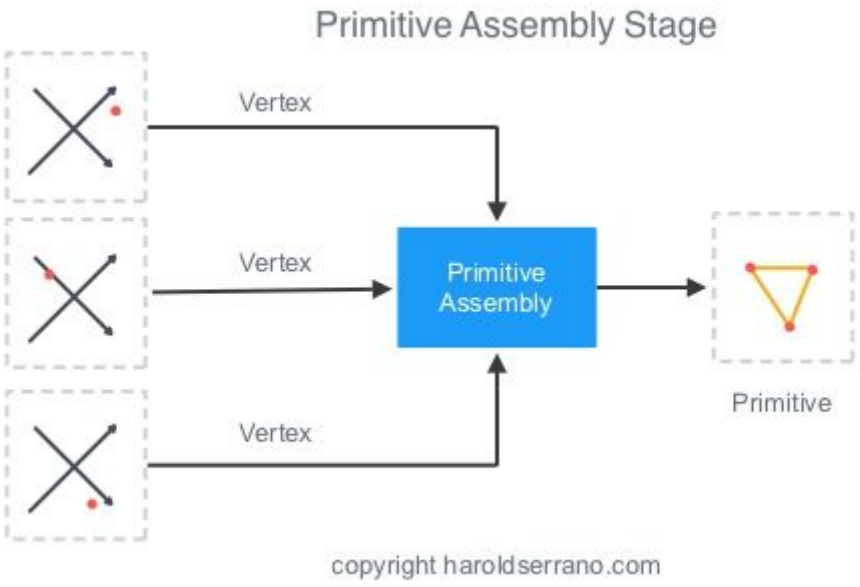
a) Vertex Processing

The Vertex Processing stage processes incoming geometrical data, vertices. In this step, each vertex coordinate system is transformed by a space matrix, thus changing the vertex original coordinate system to a new coordinate system.



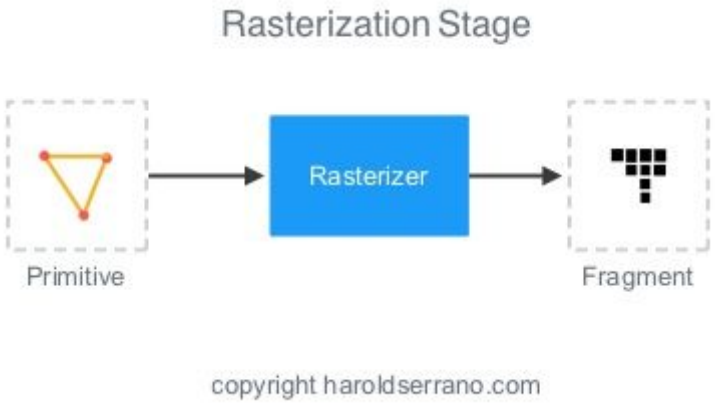
b) Primitive Assembly

The purpose of the primitive assembly step is to convert a vertex stream into a sequence of base primitives. For example, using 3 processed vertices from the Vertex Processing stage.

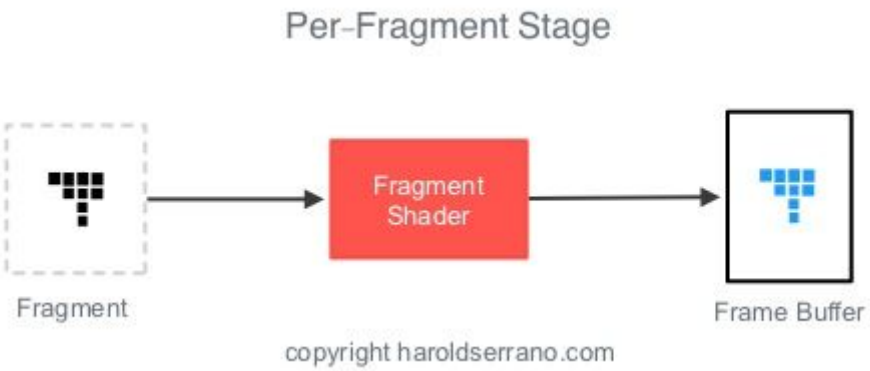


c) Rasterization

Many primitives come to this step and are then rasterized in the order in which they are given. The result of a principled rasterization is a Fragment sequence produced.

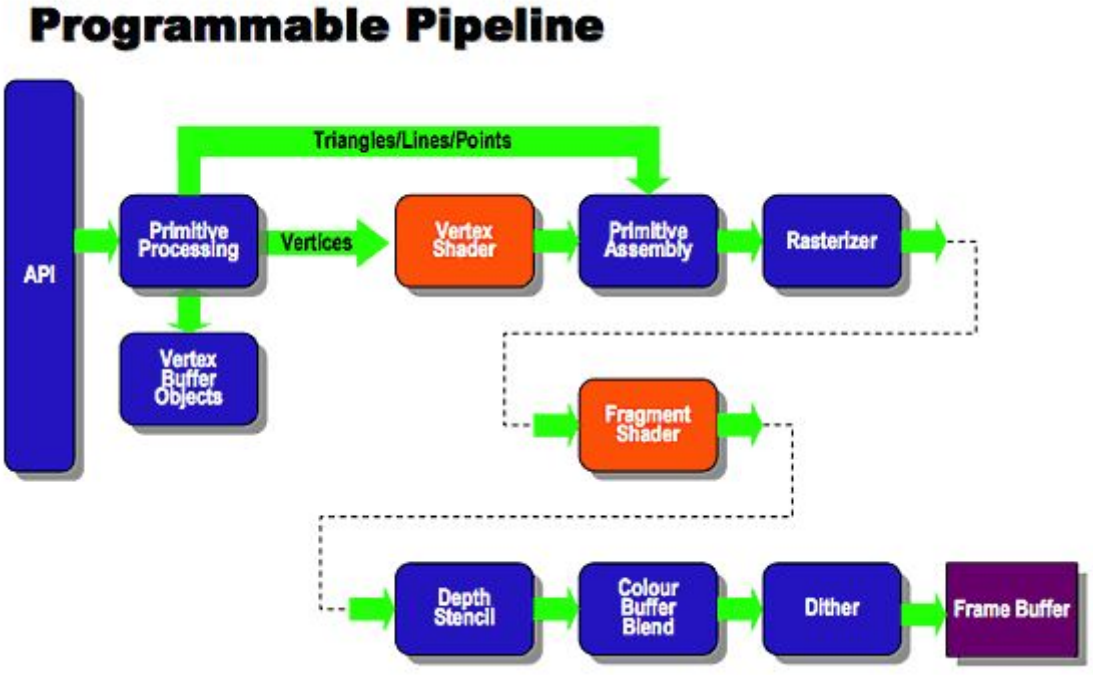


Fragment Processing,
At this stage, we are approaching the Final
The responsibility of the Fragment Processing stage it to apply color or texture to the fragment.Thus, the particle becomes more visible.For example;

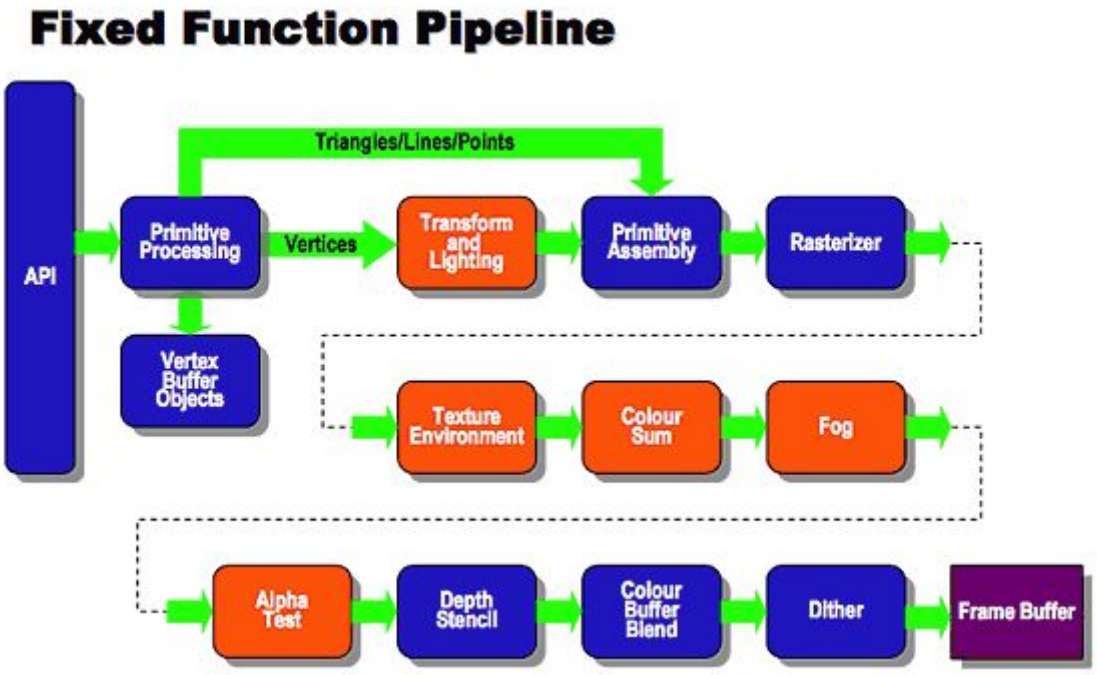


Developing computer graphics was a pain in the 1980s; each hardware needed its own special software. For this purpose, **OpenGL** and **DirectX** were introduced in 1992, respectively. **OpenGL** and **DirectX** provide an easy-to-use graphics API to provide (hardware) abstraction. In 1995, the first consumer video card was introduced.

At this time the graphics APIs and video hardware only offered the fixed-function pipeline (FFP) as the one and only graphics processing pipeline. The FFP allowed for various customization of the rendering process. However the possibilities are predefined and therefore limited. The FFP did not allow for custom algorithms. Shader a-like effects are possible using the FFP, but only within the predefined constraints.



VS



In Conclusion

The fixed function pipeline doesn't involve many changes to shaders (there may be only a single "fixed function" shader in the driver's fixed-function emulation path) which are one of the more expensive things to change.

In the past, the Rendering Pipeline does not affect the output of any stage, and this problem was fixed by developers. **Fixed-Functions** are replaced with general phases in which you can modify the data however you want; The PP allows you to do whatever you can program. You're in total control of every vertex and pixel on the screen and by now also geometry. However, with more power comes more responsibility. The PP required a lot more work as well. Shader languages, which we'll discuss in a later section, help to address this problem.

Today, you control the outputs of two stages in the pipeline; the Vertex Processing and Fragment Processing stages. We control these stages through GPU programs called "Shaders".The GPU program that controls the Vertex Processing stage is known as a Vertex Shader. However, the program that controls the Fragment Processing stage is known as a Fragment Shader. There are many other variables here, too, of course. we don't list your shaders directly.Perhaps you have some kind of very bad performance piece of code you've added to a shader that the default ones don't have. Maybe you're using vertex buffers inefficiently. Maybe you're doing one of dozen of things.

References

*https://www.khronos.org/opengl/wiki/Fixed_Function_Pipeline

*https://www.khronos.org/opengl/wiki/Fragment_Shader

*https://www.tutorialspoint.com/cuda/cuda_fixed_functioning_graphics_pipelines.html