

**This is the work of a student with specific learning difficulties.**

**Please mark in accordance with the guidelines.**

**Name/Student No. (whichever to be used to identify this assignment):**

**Stuart Andrew Leslie Hayes - 20363714 / CE**

**………………………………………………………………………………………**

**Course:**

**BSc (Hons) Comp Games Dev FDE**

**……………………………………………………………………………………….**

**Subject (if Combined Honours)**

**…………………………………………………………………………………………**

# Overall process of using shaders in a graphical application

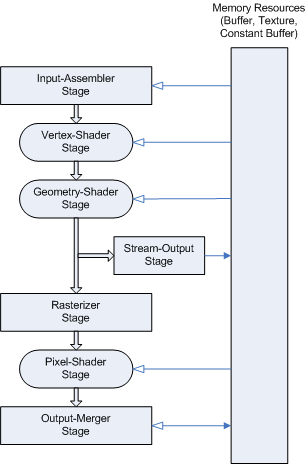
 To render anything in a graphical application, data will go throught the graphics pipeline.

Figure 1: DirectX 10 pipeline by Microsoft

## Input

This section collects all vertexes and indexes used in the geometry of all objects to be drawn.

## Vertex Shader

The vertex shader is the section where per-vertex operations take place, e.g. transformation. The data is used here to make 3D geometry to 2D viewport space. Each vertex must have one input and one output.

## Geometry Shader

This section is optional in the pipeline but is used to give more control. The geometry shader processes the entire primitives.

“Algorithms that can be implemented in the geometry shader include: Point Sprite Expansion, Dynamic Particle Systems, Fur/Fin Generation, Shadow Volume Generation, Single Pass Render-to-Cubemap, Per-Primitive Material Swapping, Per-Primitive Material Setup” (Microsoft, 2018).

## Rasterizer

This stage processes 2D polygons before rendering. This is the section culling happens. Once this is done the output of a pixel is its position in screen-space.

## Pixel Shader

The pixel shader is the section where per-pixel operations take place, e.g. lighting. This is the section where a pixel gets its colour. UV maps are placed onto polygons and lighting are combined to get its final colour.

## Output

The final stage of the graphics pipeline. This is the output of all processing. This contains blending of the final polygon colours with existing viewport pixels. Depth buffers values are tested and/or Witten in this section.

# Specific techniques

## Normal Mapping

Normal mapping required an additional texture which stores the direction of the normal of that pixel. This is then considered in the lighting stage to give the effect of bumpiness and give the object a 3D effect on a flat surface.

## Parallax Mapping

Parallax mapping tries to show depth as well as bumpy lighting that normal mapping produces. Parallax mapping requires a height map. Using the height map we attempt ot correct the texture with an offset from the height map so that when looking at the texture we don’t just look at the texture on the pixel but the one that would be there at the interception point instead. This method approximates the correct pixel and is not always the true pixel but gives a ‘good enough’ effect to trick the mind into the texture being bumpy. This is usually stored in the alpha channel of the normal map.

## Per Pixel Lighting

Light is calculated from its source to each pixel. This means we can add light to just a single pixel rather than the whole vertex which causes problems when some objects like a wall use very few vertices but over a large area. This means we can add light to one side of an object and not on the other side of the object with the same vertex. There are two types of lighting that I use, Diffuse light and Specular light. The diffuse light uses the diffuse texture and the light source and its distance to give the model colour and light to it. Specular light is uses light, a specular map and the camera as well to see if there is a reflection giving the object a reflection from the light onto it. These are then added up to get the final product.

## Blending

This adds a texture onto of anything else, this give the view of transparency. I used this for the lights in the scene since they come from a point but just light, this give a good effect to look like a light and all that is done is the diffuse map is just tinted with a given colour.

# Improvements

## Lighting structure passing to the gpu

I could have improved the lighting structure on the gpu side by adding a struct, which would be closely related to the structure on the cpu side, this would mean I would pass over a structure of a light instead of its individual values. This would mean I could loop through lights in the gpu.

## Shadows

I could have added shadows. Shadows would have been spotlights with a given direction, and to give a point light shadows I would have just used 6 spotlights or 5 if there is no roof to reduce unnecessary calculations.

## Added mirrors to the portal

Mirrors do not differ too much from portals and could be modified but just setting a camera to be looking through the portal at a set location behind the portal.

# Additions

## Controls

1 toggles Parallax

2 toggles moving textures

3 toggles model wiggle

8 decreases the power of the wiggle

9 resets wiggle power

0 increases the power of the wiggle

To move the camera; w, s, a, d, e, q, z, x

To move the portal camera; t, g, f, h, n, b, v, m

To move the cube and decal; i, k, j, l, u, o, ,. ,,

References

Microsoft, (2018). Graphics Pipeline (Windows). [online] Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/bb205123(v=vs.85).aspx.

OpenGL Wiki contributors, (2017). Rendering Pipeline Overview - OpenGL Wiki. [online] Available at: https://www.khronos.org/opengl/wiki/Rendering\_Pipeline\_Overview.