The Graphics Pipeline

Computer Graphics - SET08116

EDINBURGH NAPIER UNIVERSITY

Outline

1. Background
2. Working with Data
3. Graphics Pipeline
4. Pipeline Stages
5. Summary

What is a Pipeline?

A pipeline is just a series of processes that occur in order:

Read File ⇒ Process Data ⇒ Print Results

A pipeline can have any number of processes within it.

The point is that data from one process is fed into the next process.

- Think of this as an assembly line.

Parallelising a Pipeline

One of the main advantages of a pipeline approach is that they are easy to parallelise.

- Pipelining
  - Each process works independently.
  - Whenever one process has data ready for the next, it forwards it on.
  - The sending process can still be processing data
- Internal Parallelisation
  - Each process can itself work on different parts of the incoming data
  - Allows the process to speed up based on the number of internal processes

Throughput of Data

Data throughput in a pipeline is fast in comparison to traditional processing

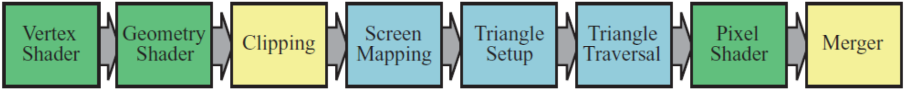
There are two main reasons for this:

- Data parallelism
- Streamlining

A key concept to understand is that throughput is determined by the slowest process in the system.

- Some processes may be very fast
- In graphics processing, pixel shading is slow in comparison to vertex manipulation

Graphics Pipeline



- Vertex Shader - manipulates individual vertexes.
- Geometry Shader - manipulates primitives.
- Clipping - removes unseen geometry, and fixes partially seen.
- Screen Mapping - mapping triangles to the screen
- Triangle Setup and Traversal - discover which triangles affect which pixels
- Pixel Shader - manipulates pixel colour
- Merger - determines final colour of a pixel

Vertex Data

Initially we are going to take a step back and think about how data is presented to the graphics card. This is important to understand as we will shortly be working with buffers and shaders.

Data sent to the graphics card is called vertex data:

- Position of the vertex
- Colour of the vertex
- Texture coordinates
- Normals

These items are used to define a geometric object.

```
// Vertex 1
glVertex3f(1.0f,0.0f,0.0f);
glColor3f(1.0f,0.0f,0.0f);

// Vertex 2
glVertex3f(1.0f,1.0f,0.0f);
glColor3f(1.0f,0.0f,1.0f);

// etc.
```

Vertex Buffers

To stream vertex information efficiently, we use vertex buffers.

Vertex buffers can be thought of as arrays of vertex data.

Buffers can exist for vertex data, normal data, colour data, or mixed (along with some others).

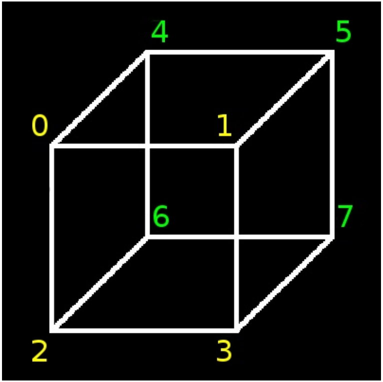
```
GLfloat v[] =
{
1.0f,0.0f,0.0f,
1.0f,1.0f,0.0f,
0.0f,1.0f,0.0f
};

glEnableClientState
    (GL_VERTEX_ARRAY);
glVertexPointer
    (3, GL_FLOAT, 0, v);
glDrawArrays
    (GL_TRIANGLES, 0, 3);
```

Index Data

Working with buffers allows us to exploit index data to further improve efficiency.

Indices means that we reuse vertex data from a buffer at different points.



Index Buffers

Index data is also stored in a buffer.

Buffer contains index numbers relating to positions of vertexes in the vertex buffer.

Therefore, the rendering reuses vertexes from the buffer.

```
// Create 8 vertexes
GLfloat v[] = { .. };
glVertexPointer
    (3, GL_FLOAT, 0, v);
// Create indices
GLubyte i[] =
{
    0, 3, 1,
    1, 3, 2,
    // etc.
};
glDrawElements
    (GL_TRIANGLES, 36,
     GL_UNSIGNED_BYTE, i);
```

History Recap

The first work on 3D rendering occurred in the 1960s.

Most of the common 3D rendering techniques evolved in the 1960s-1970s.

The 1980s saw the introduction of graphics hardware

- Commodore Amiga first consumer computer to have a chip to support graphical elements

The 1990s saw the introduction of 3D graphics hardware and supporting APIs

- OpenGL
- DirectX

Fixed-Function Pipeline

Initially, graphics cards were not programmable.

- Fixed function pipeline

Programmers could turn off and on states, and provide some values to these states.

- This is all you have done so far with OpenGL.
- Nintendo Wii still has a fixed function pipeline.

Although the fixed function pipeline advanced the capabilities of consumer level graphics rendering, there was little flexibility.

Programmable Pipeline

To add flexibility, GPUs were given programmable stages.

- The programs that run on the GPU are referred to as shaders.

First shader programs were used in the 1980s.

- Pixar's RenderMan

2001 Nvidia introduced the first GPU with programmable shader support.

- Originally only vertex shading was supported.

Today, nearly all GPUs support some form of programmable shading.

- Vertex shader
- Geometry shader
- Tessellation shader (new - DirectX 11 and OpenGL 4)
- Pixel / fragment shader

Application $\Rightarrow$ Geometry $\Rightarrow$ Rasterizer

The three main stages of the graphics pipeline are:

- Application
  - OpenGL / DirectX stage
  - Application stage feeds geometric data (vertex data) to the geometry stage.
- Geometry
  - GPU stage
  - Works on the vertex data and generated polygons to lead to screen mapping.
- Rasterizer
  - GPU stage
  - Sets pixel colours for the final rendered image (e.g. the screen)

Application Stage

The application stage is completely controlled by the application developer.

Programmers use a graphics API to send data to the GPU.

This is where we have been working at the moment. We have not done much (beyond setting matrices) that involves direct manipulation of the GPU.

A key task we may do here is culling, where we determine what to draw. We can greatly improve performance by only attempting to draw what is visible.

Geometry Stage

In is the responsibility of the GPU to transform input Application Stage data into screen mapped data.

The Geometry stage has six internal tasks to perform:

- Model and View Transform
- Vertex Shader
- Geometry Shader (we will discuss in a later lecture)
- Projection
- Clipping
- Screen Mapping

Model and View Transform

A model initially exists within its own model coordinate space.

- This allows movement, rotation, etc.

Model transformation is almost exclusively done using matrix transforms on the GPU.

- We will refresh our knowledge of matrix math soon.

View transformation involves moving the model so it exists within a camera's coordinate space.

- The camera is considered to be on the origin.
- The geometry is moved to reflect this
- Generally, we combine the model and view matrices together, therefore requiring only one calculation.

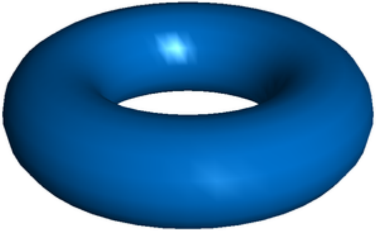
Vertex Shading

The vertex shader is the first programmable stage in the pipeline.

The vertex shader works on the incoming data that describes an individual vertex.

The vertex data is then used to perform certain calculations, the results of which can be passed onto the next stages of the pipeline.

For example, we can perform per-vertex shading using the Gouraud shading technique. Here you can see the triangles.



Projection

The 3D data can now be projected onto the screen using the projection matrix.

The view (which can be represented by a frustum) is transformed into a unit cube.

- (-1,-1,-1) to (1,1,1) or similar

There are two common projection methods:

- Orthographic (essentially flat)
    - Parallel lines remain parallel no matter their orientation.
- Perspective
    - Distance from the camera has an effect.

Again, it is likely that the projection matrix will be combined with the model and the view matrix to create one matrix. This matrix is commonly referred to as the model-view-projection or world-view-projection matrix.

Clipping

In rendering, we only ever try and render what we can actually see.

- Culling at the application stage should hopefully have removed much of the unseen geometry for us already at a coarse level.

After model, view and projection transform, only some of the primitives that we attempted to draw will still be visible.

Primitives no longer visible are removed. Primitives only partially visible are modified so that only a fully visible primitive is worked on. This involves introducing new vertexes.

The removal and modification of primitives is referred to as clipping.

Screen Mapping

Finally, we are ready to map individual primitives to the screen to proceed with the next stage of rendering.

Screen mapping is the final process in the geometry stage of the render pass.

Only visible primitives need to be mapped to the screen, hence the previous use of clipping.

At this point, screen mapping is relatively simple - the transformed (x, y) coordinates are used as screen coordinates.

Rasterizer Stage

The goal of the rasterizer stage is to start assigning colours to individual pixels on the screen.

There are four tasks to perform at this stage in the pipeline:

- Triangle Setup
- Triangle Traversal
- Pixel Shading
- Merging

Triangle Setup and Traversal

Triangle Setup and Triangle Traversal are fixed stages in the graphics pipeline.

The goal here is to determine which triangles cover which pixels on the final rendered image (e.g. the screen).

Another key aspect here is also utilising the depth of the triangle pixel so that the colour of the triangle pixel nearest the camera is used (unless it has some transparency).

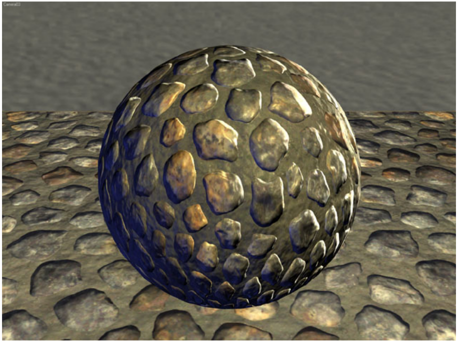
The data gathered here is used later in the merging stage.

Pixel Shading

The Pixel Shader (or Fragment Shader in OpenGL) is the last programmable stage in the pipeline.

The Pixel Shader allows manipulation of the individual pixel colours produced in a render, utilising information from previous stages, and Application Stage set data.

This can involve lighting (which appears smoother), texturing, and effects such as normal mapping (see left).



Merging

Finally, the GPU merges all the pixel colours together to determine the final pixel colour to use. This is done at the merging stage.

Data from the pixel shader, colour buffers, depth buffers, etc. are used to determine the final colour of a screen pixel.

Other operations can be performed here as well, based on the pipeline state.

High Level Stages

For our purposes, we can think of the graphics pipeline as two stages:

- Programmable Shader Stage
  - Vertex shader to perform some lighting calculations, geometry transformation, etc.
  - Geometry shader - more on this later.
- Rasterizer Stage
  - Colouring of individual pixels
  - Pixel shader can be used to perform per-pixel lighting, texturing, etc.

Understanding the two stage process in the GPU is fundamental, but also be aware of the internal operations.

Summary

From this lecture, you should now know about:

- What the graphics pipeline is, and how it improves performance.
- How data is sent to the GPU.
- The three main stages of the graphics pipeline:
  - Application ⇒ Geometry ⇒ Rasterizer
- The tasks performed at each of these three stages.

Recommended Reading

Real-Time Rendering, chapters 1-3.

Interactive Computer Graphics: A Top-Down Approach, chapters 1-2.