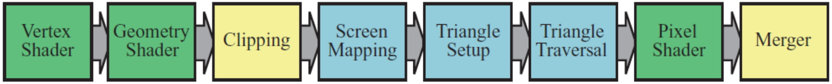
Geometry Shader Part 1

Computer Graphics - SET08116

EDINBURGH NAPIER UNIVERSITY

Outline

Review - 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

Review - Programmable Shaders

As already mentioned, there are three stages in the graphics pipeline that are programmable:

- Vertex Shader - manipulates individual vertexes
- Geometry Shader - works with primitives (triangles, lines, etc.)
- Pixel Shader - determines colour of individual pixels

There is also the tessellation shader and compute shaders, but these are beyond the scope of the module.

Having the ability to write programs for individual stages of the pipeline is advantageous, and allows us to both optimise, and produce more elaborate effects than the fixed function pipeline was capable of.

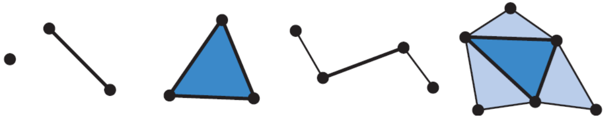
What is the Geometry Shader?

The geometry shader is the stage in the pipeline that occurs directly after the vertex shader stage.

The geometry shader operates with output from the vertex shader in the form of geometric primitives:

- Points
- Lines
- Triangles
- Quads

Can also operate using adjacency data:



What Can the Geometry Shader Do?

The job of the geometry shader is to work on (and therefore manipulate) a piece of geometry

- No longer working at the level of manipulating a single vertex

The geometry shader can also add geometry to a scene

- No longer single vertex in, single vertex out

Combining these two features together allows manipulation of the geometry in new manners

- Allowing us to create some neat effects

Example

Geometry Shader Input

The input into a geometry shader comes in the form of geometric primitives:

- Points, lines, triangles, quads

The geometry shader may also take in an adjacency block

- Vertices adjacent to the piece of geometry in question

The piece of geometry is basically an array of vertex positions - and possibly other data related to them

Geometry Shader Output

The geometry shader outputs a strip of the defined geometry type

- e.g. Triangles in, triangle strip out

The output operates by having a few extra functions in GLSL:

- EmitVertex() - emits a vertex to the next stage of the pipeline
- EndPrimitive() - ends the vertex creation for the particular primitive

It is possible to restrict the number of output vertices

Stream Output

It is also possible to output data from the geometry shader stage of the pipeline

Output from the geometry shader can be used for further rendering if required

- Useful for letting the GPU do point physics calculations
- Each vertex has a position, velocity, acceleration, mass, etc.
- Output is the new position, velocity

GLSL Geometry Shader Example

```
#version 330

layout (triangles) in;
layout (triangle_strip, max_vertices = 6) out;

uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 modelMatrix;

in vec4 normal[];
out vec4 vertex_colour;
```

GLSL Geometry Shader Example

```
void main()
{
  mat4 modelViewMatrix =
    viewMatrix * modelMatrix;
  mat4 viewProjectionMatrix =
    projectionMatrix * viewMatrix;
  int i;
```

GLSL Geometry Shader Example

```
// First triangle - identical to input
for (i = 0; i < 3; ++i)
{
  vec4 view_pos =
    modelViewMatrix * gl_in[i].gl_Position;
  gl_Position =
    projectionMatrix * view_pos;
  vertex_colour = vec4(0.0, 1.0, 0.0, 1.0);
  EmitVertex();
}
EndPrimitive();
```

GLSL Geometry Shader Example

```
  // Second triangle - translated version
  for (i = 0; i < 3; ++i)
  {
    vec4 N = normal[i];
    vec4 world_pos = modelMatrix
      * (gl_in[i].gl_Position + normalize(N) * 10.0);
    gl_Position =
      viewProjectionMatrix * world_pos;
    vertex_colour = vec4(1.0, 1.0, 0.0, 1.0);
    EmitVertex();
  }
  EndPrimitive();
}
```

Input Values

Notice that we define the input value types using the layout specificiation

- Layout is also used in the vertex shader to tell the GPU where to send particular pieces of data

The input layout is important for using the geometry shader correctly

- It needs to know the type of primitive that it is dealing with

We can also declare incoming adjacency vertices

Output Values

Notice that we also set the outgoing data format for the geometry shader

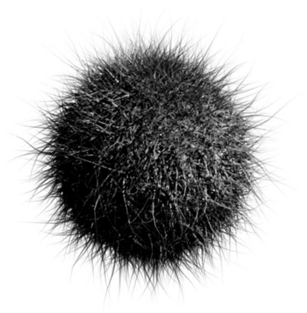
- A triangle strip in this instance

We can also set the output data type, and the number of expected vertices.

This can be useful when dealing with potentially large pieces of geometry.

Fur

YouTube Video



Point Sprite Expansion

YouTube Video

Dynamic Particles

YouTube Video

Morphing / Shape Manipulation

YouTube Video

Tesselation

YouTube Video

YouTube Video

Summary

This has been a very brief overview of what the geometry shader does, and what it is capable of.

What you should be aware of after this lecture is:

- How the geometry shader operates
- The type of data the gometery shader works with
- The idea that the geometry shader can emit new geometry, allowing some very unique effects
- Some of the effects that the geometry shader is capable of

As mentioned, we won’t be using the geometry shader much, but if you are interested there is a wealth of information available.

Recommended Reading

The geometry shader is a relatively new concept.

- Real-Time Rendering mentions it in passing
- Interactive Computer Graphics has no real discussion

You should do some further research online to find out some of the possibilities available via the geometry shader.