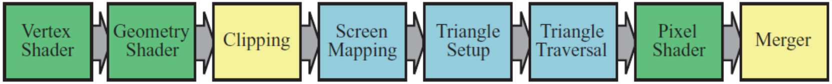
Geometry Shader Part 2

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

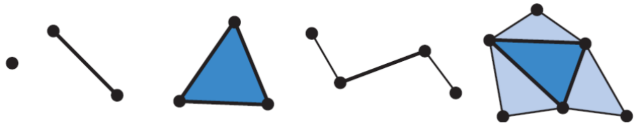
What is the Geometry Shader?

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:



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

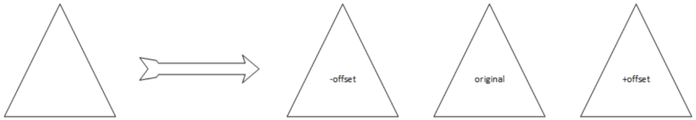
The output operates by having a few extra functions in GLSL:

- 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

Geometry Shader Output

- The first trick we will do with the geometry shader is to repeat the geometry input
- We will do this by providing an offset value to our geometry shader
- The general idea is to output the same geometry three times as illustrated below

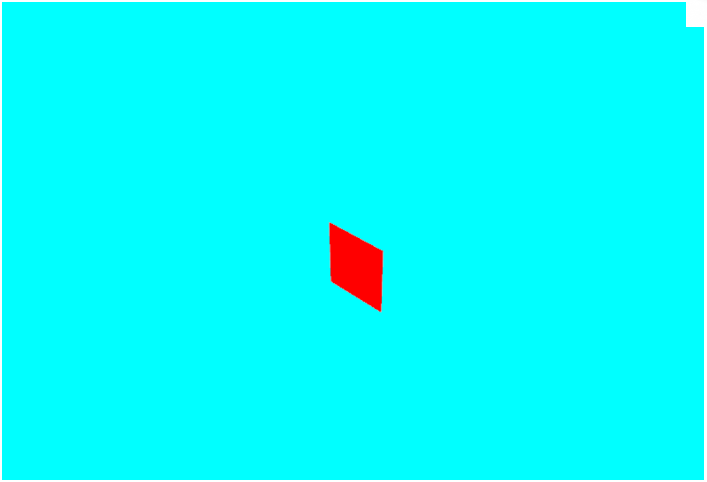


Outputting Geometry

- Remember that we output a single vertex using EmitVertex()
- Remember that we also state that we have output a primitive (point, line, triangle) using EndPrimitive()
- We are inputting triangles and outputting triangles
- OK, so how do we repeat geometry? Well let us just output the incoming geometry. This is as follows

```
for(int i=0;i<3;++i)
{
  glPosition=
  MVP * gl_in[i].glPosition;
  EmitVertex();
}
EndPrimitive();
```

Initial Scene



Geometry Shader Output

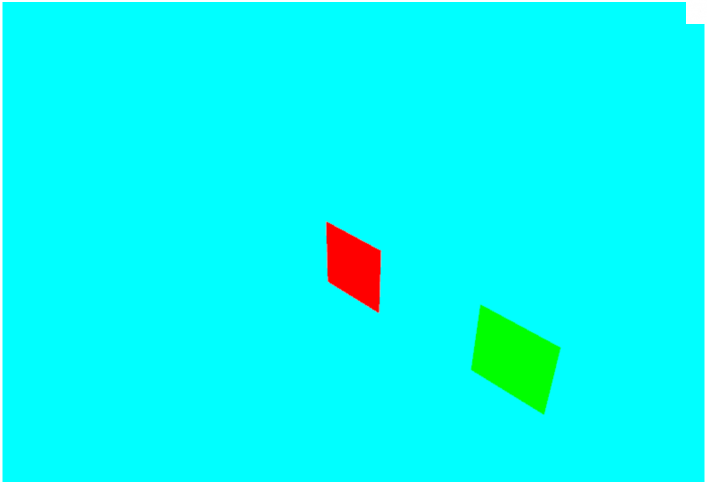
- The first trick we will do with the geometry shader is to repeat the geometry input
- We will do this by providing an offset value to our geometry shader
- The general idea is to output the same geometry three times as illustrated below

Repeating Once

- Now what about if we want to output the geometry again? Then we do just that!
- We add a second for loop to deal with the new output:

```
for(inti=0;i<3;++i)
{
  glPosition=
  MVP * gl_in[i].glPosition;
  EmitVertex();
}
EndPrimitive();
for(inti=0;i<3;++i)
{
  glPosition=
  MVP * (gl_in[i].glPosition+offset);
  EmitVertex();
}
```

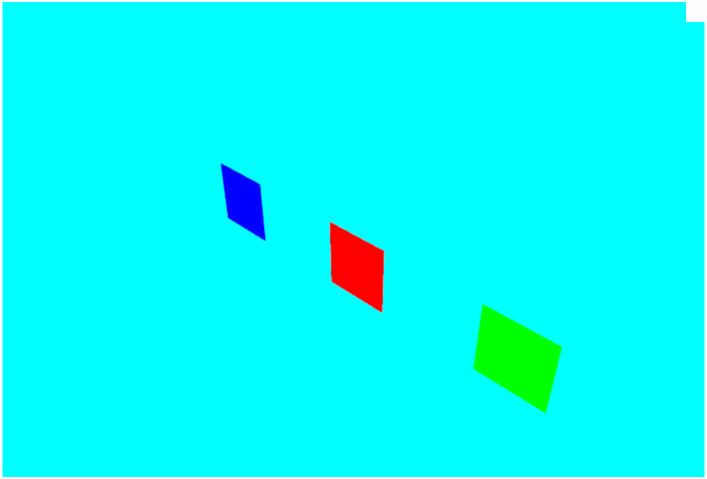
Repeated Once



Repeating Twice

- To output the geometry again we just add another for loop (this time subtracting the offset)
- Notice that we have been performing the screen mapping transformation in the geometry shader
- This is because we are offsetting the object in the world. We need to move the object prior to transformation to do this (or we would have to transfer back first)
- This technique can be used for any geometry type - it will execute for all triangles in the mesh

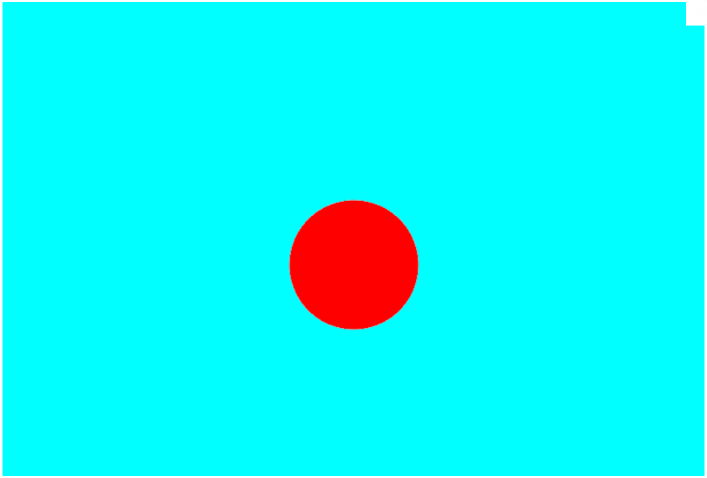
Repeated Twice



Exploding Shape

- Our next example explodes a shape
- By explode we mean that we will move a triangle by its surface normal by a factor
- We could call this a separate geometry shader
- Again we are inputting triangles and outputting triangles
- Let us look at a sphere

Initial Scene



Calculating Surface Normal

- We have to calculate the surface normal in the geometry shader
- Using the incoming normal with the geometry will not provide the correct effect - the shape will just expand but not separate
- Remember to calculate the surface normal we use the cross product:

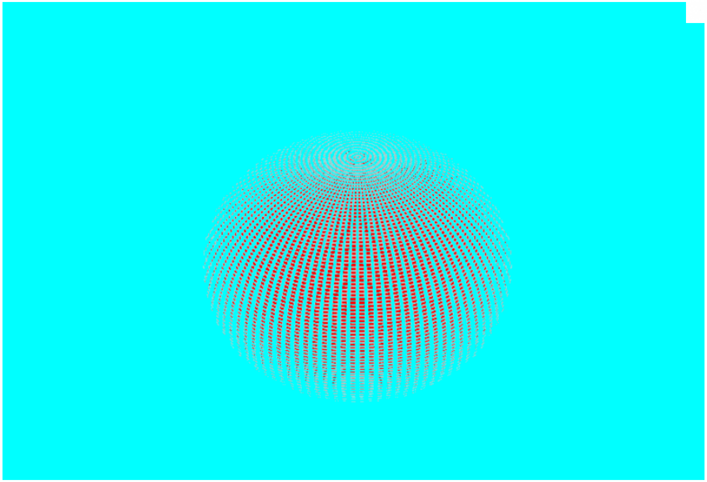
$$side_1 = v_1 - v_0$$
$$side_2 = v_2 - v_0$$
$$n = side_1 \times side_2$$

Explosion Shader

- Once the surface normal is calculated we can manipulate the shape
- We follow the basic premise as before - manipulate the incoming value and output
- The following gives the general idea

```
//Calculatesurfacenormal
for(inti=0;i<3;++i)
{
  glPosition=gl_in[i].glPosition +(normal * explodefactor);
  EmitVertex();
}
EndPrimitive();
```

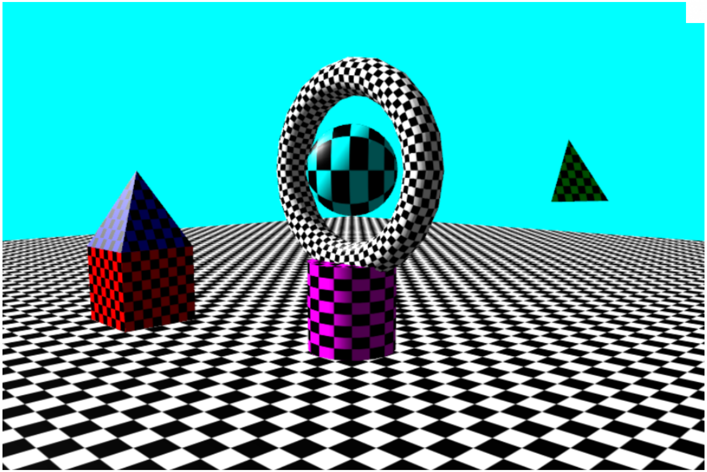
Sphere Exploded



Rendering Normals

- OK lets do something useful with a geometry shader
- One thing we can do is output the normals from our geometry
- Unlike last time we will use the incoming normal information
- Our starting scene is as follows

Initial Scene

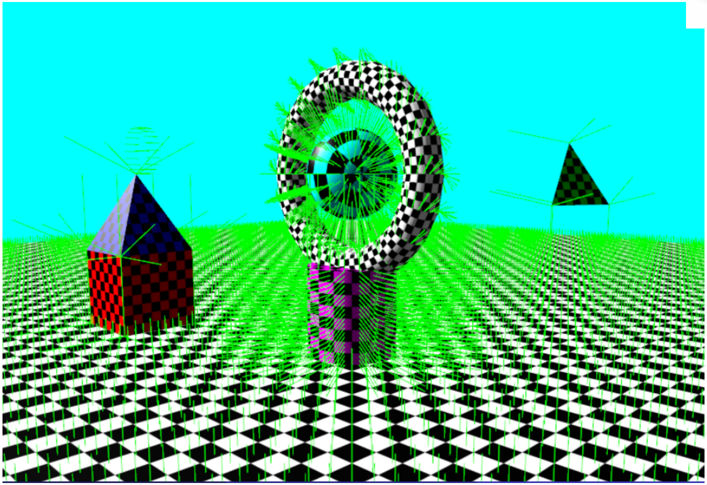


Outputting Normals

- This time we are going to output lines from our initial points (points in, lines out)
- Our point requires a start and an end
- The start point is the incoming vertex position
- The end point is the start point plus the normal - pretty easy
- The general idea is:

$$start = MVP \times position$$
$$end = MVP \times (position + normal)$$

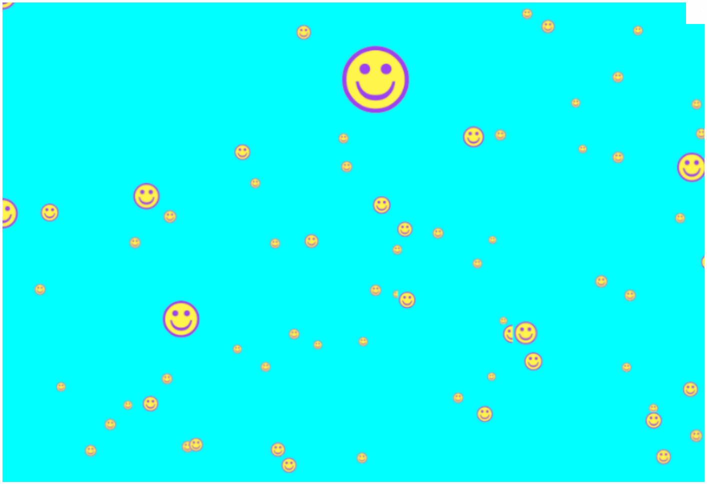
Normals Rendered



Billboarding

- Billboarding using the geometry shader takes in point data and outputs quads (triangle strip) for texturing
- The incoming points are already transformed into camera space (using the MV matrix)
- We calculate the corners of the quad, transformed into screen space (using the P matrix), with texture coordinates
- The idea is very simple. It allows us to have objects that face us.
- This technique is used for grass, smoke, fire, explosions

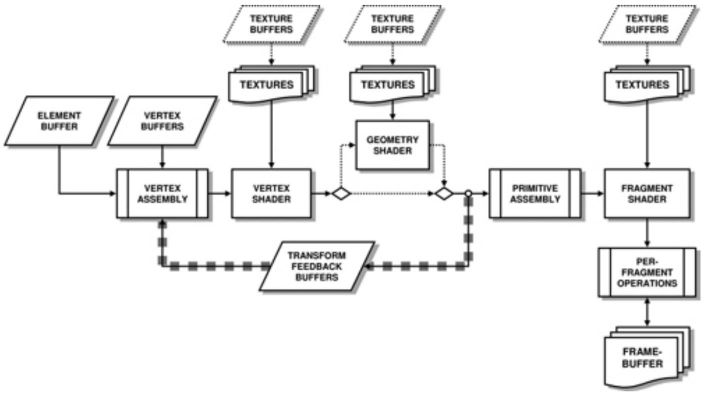
Billboarding Output



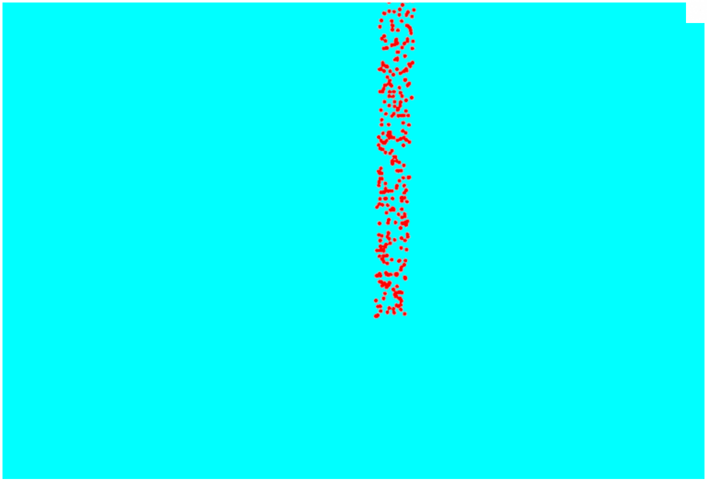
Stream Output

- 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

Transform Feedback



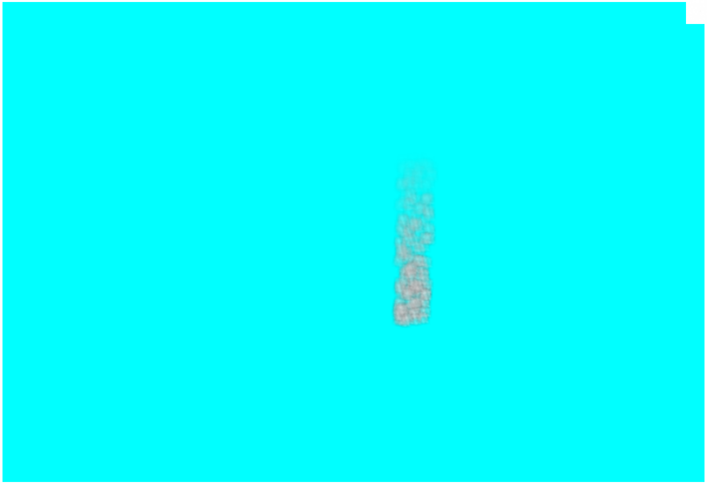
Particle Output



Smoke Effect

- We can combine billboarding and particle effects to create a smoke effect
- Points come in, we billboard and then texture with a heavily transparent texture
- The effect can be used beyond smoke to fire and explosions

Smoke Output



Summary

- The geometry shader lessons will go up this weekend (just tweaking the last little bit)
- These examples are really just the surface of what is possible with the geometry shader
- When working with physics (3rd year) you can combine ideas with the geometry shader for some good and fast effects