Geometry Shader

Thanks to Mike Bailey (OSU)

Geometry Shaders

- Can cull geometry (do front/back/arbitrary culling)
- Can amplify geometry (create geometry)
- Can emit different types than input
- Can generate multiple streams for single primitive

Additional Types Introduced for Geometry Shader

- GL_LINES_ADJACENCY
- GL_LINE_STRIP_ADJACENCY
- GL_TRIANGLES_ADJACENCY
- GL_TRIANGLE_STRIP_ADJACENCY
Adjacency Primitives (and what they do by default)

Lines with Adjacency

All vertices are given.

List the number of segments to draw.

If more than one, there will be a 0 before and 0 after.

Vertices 0 and 0 are usually removed.

Adjacent vertices are always there to provide adjacency information.

Line Strip with Adjacency

0 1 2 3

Lines with Adjacency

0 1 2 3 4 5

Lines strips with Adjacency

Adjacency Primitives (and what they do by default)

Triangles with Adjacency

All vertices are given.

Triangles are drawn.

Brush 0 to define the triangles.

Brush 1 to work with triangles.

Triangles are adjacent.

Triangle Strip with Adjacency

0 1 2 3 4 5

Adjacency Primitives (and what they do by default)

Figure 8.20: Lines produced using Lines with Adjacency Primitives

Figure 8.22: Triangles produced using GL_TRIANGLES_STRIP, ADJACENCY

Figure 8.23: Deleting of vertices for GL_TRIANGLES, ADJACENCY

If a Vertex Shader Writes Variables as:

\[ \text{gl\_Position} \]

Then the Geometry Shader will Read Them as:

\[ \text{gl\_Position} \]

and will Write Them to the Fragment Shader as:

\[ \text{gl\_Position} \]

The Geometry Shader can Assign These Built-in Variables:

\[ \text{gl\_Position} \]

\[ \text{gl\_PointSize} \]

\[ \text{gl\_Layer} \]

\[ \text{gl\_PrimitivedID} \]

Plus any of your own that you have declared to be:

\[ \text{EndVertex()} \]

When the Geometry Shader calls

\[ \text{BeginVertex()} \]

this set of variables is copied to a slot in the shader's Primitive Assembly state.

Also, the vertices that have been saved in the Primitive Assembly state are then assembled, materials, etc.

Note: there is no "EndVertex()" function. It is implied by (1) the end of the geometry shader, or (2) returning from the EndPrimitive() call

Note: there is no need to call EndPrimitive() at the end of the Geometry Shader - it is implied.
If you are using a Geometry Shader, then the GS must be used if you want to pass information from the Vertex Shader to the Fragment Shader.

Example: A Bezier Curve

\[ P(\mu) = (1-\mu)^3 P_0 + 3(1-\mu)^2 \mu P_1 + 3(1-\mu)\mu^2 P_2 + \mu^3 P_3 \]

Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments

Lines Adjacency used for four points:

\((0, 0, 0)\)
\((1, 1, 1)\)
\((2, 1, 2)\)
\((3, 1, 0)\)

```
// beziercurve.vert
void main() { 
  gl_Position = uModelView*projectionMatrix * vertex;
}
```

```
// beziercurve.frag
void main() {
  fragColor = vec4(0.1, 0.1, 1.0);
}
```

Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments

Note: It would have made no difference if the Matrix Transform had been done in the Geometry Shader instead.

```
// beziercurve.geom
void main() { 
  gl_Position = uModelView*projectionMatrix * vertex;
}
```
Geom Shader Billboards

- Goal: create a quad (just a tri-strip)
Geom Shader Billboards

Need the quad to face the camera
  • Consider camera (black) and billboard location (red) in world space
  • Create a vector from the billboard to the camera
  • Add an ‘up’ vector
  • Make a mutually orthogonal vector (billboard location and range)
  • Then the third mutually orthogonal vector

```
VS
#version 330
layout (location = 0) in vec3 Position;
void main()
{
  gl_Position = vec4(Position, 1.0);
}

FS
#version 330
uniform sampler2D gColorMap;
in vec2 TexCoord;
out vec4 FragColor;
void main()
{
  FragColor = texture2D(gColorMap, TexCoord);
  if (FragColor.r == 0.0 && FragColor.g == 0.0 && FragColor.b == 0.0)
  {
    discard;
  }
}
```
Geom Shader Billboards

- GS
  
  ```
  #version 330

  layout (points) in;
  layout (triangle_strip) out;
  layout (max_vertices = 4) out;

  uniform mat4 gVP;
  uniform vec3 gCameraPos;
  uniform float size;

  vec3 ul = Pos - (right + newUp) * size;
  gl_Position = gVP * vec4(ul, 1.0);
  TexCoord = vec2(0.0, 1.0);
  EmitVertex();

  vec3 lr = Pos + (right - newUp) * size;
  gl_Position = gVP * vec4(lr, 1.0);
  TexCoord = vec2(1.0, 0.0);
  EmitVertex();

  vec3 ll = Pos - (right - newUp) * size;
  gl_Position = gVP * vec4(ll, 1.0);
  TexCoord = vec2(0.0, 0.0);
  EmitVertex();
  
  vec3 ur = Pos + (right + newUp) * size;
  gl_Position = gVP * vec4(ur, 1.0);
  TexCoord = vec2(1.0, 1.0);
  EmitVertex();

  EndPrimitive();
  ```

Demo normal offset (from superBible)

Demo Explosion (from superBible)

Demo 4-views (from superBible)

GS Quads (bi-linear interpolation) (from superBible)

GS Culling (from superBible)
The Difference Between Tessellation Shaders and Geometry Shaders

By now, you are probably confused about when to use a Geometry Shader and when to use a Tessellation Shader. Both are capable of creating new geometry from existing geometry. See if this helps.

**Use a Geometry Shader when:**
1. You need to convert geometry topologies, such as the silhouette and hatching shapes (triangles—lines) or the explosion shader (multipe—points).
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the skin shader was used here).

**Use a Tessellation Shader when:**
- You need to generate many new vertices and one of the tessellation topologies will suit your needs.
- You use a Tessellation Shader when you need more than 6 input vertices to define the surface being tessellated.