**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 Arguments Available for glBegin():**

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

Triangles with Adjacency

1 2 3
4 5 6
7 8 9
10 11

Triangle Strip with Adjacency

1 2 3 4 5 6
7 8 9 10

Figure 6.22: Triangle primitive using GL_TRIANGLES, SYMmetrical, ADJacency

Figure 6.23: Ordering of vertices for GL_TRIANGLES, LTRADJacency

If a Vertex Shader writes any variables, then the Geometry Shader will read them as:

- gl_Position
- gl_PointSize
- gl_Layer
- gl_InstanceID

In the Geometry Shader, the dimensions indicated by N are given by the variable `gl_InstanceID`. Although you can read already known data from the type of geometry you are inputting, these are already declared for you:

- GL_POINTS
- GL_LINES
- GL_TRIANGLES
- GL_QUADS
- GL_QUAD_STRIP
- GL_TRIANGLES_ADJACENCY

The Geometry Shader can assign these built-in out variables:

- `gl_Position`
- `gl_PointSize`
- `gl_Layer`
- `gl_InstanceID`

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

When the Geometry Shader calls `gl_Position`:

- The vertices that have been saved in the Primitive Assembly state are then assembled, indexed, etc.

When the Geometry Shader calls `gl_InstanceID`:

- The instance ID is inserted into the vertex data.

Note: there is no `gl_InstanceID` in the geometry shader.

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

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

Note: These are used to define the storage

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

Another Example: Sphere Subdivision

It's often useful to be able to parameterize a triangle into (x, t), like this:

Example: Sphere Subdivision
Demo normal offset

Demo Explosion

Demo 4-views

GS Quads (bi-linear interpolation)

GS Culling
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 hanging shapes (triangles—lines) or the spine shader (halfpipe—points).
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the spine 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.

**Use a Tessellation Shader when** you need more than 6 input vertices to define the surface being tessellated.