GLSL Geometry Shaders

Here's What We Know So Far

One Vertex In

Vertex Shader

One Vertex Out

The Geometry Shader: Where Does it Fit in the Pipeline?

Geometry Shader:
What Does it Do?

Point, Line, Line Strip, Line Loop, Lines with Adjacency, Line Strip with Adjacency, Triangles, Triangle Strip, Triangle Fan, Triangles with Adjacency, Triangle Strip with Adjacency

The driver translates these and feeds these one-at-a-time into the Geometry Shader

Your application generates these

The Geometry Shader generates (almost) as many of these as it wants

Points, Lines, Line Strip, Line Loop, Lines with Adjacency, Line Strip with Adjacency, Triangles, Triangle Strip, Triangle Fan, Triangles with Adjacency, Triangle Strip with Adjacency

Your application generates these

There needn't be any correlation between Geometry Shader input type and Geometry Shader output type. Points can generate triangles, triangles can generate triangle strips, etc.

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)

Lines with Adjacency

N+3 vertices are given (where N is the number of line segments to draw).
A line segment is drawn between #1 and #2. Vertices #0 and #3 are there to provide adjacency information.

Line Strip with Adjacency

N+2 vertices are given (where N is the number of line segments to draw).
A line segment is drawn between #1 and #2, #2 and #3, …, #N and #N+1. Vertices #0 and #N+2 are there to provide adjacency information.

Triangles with Adjacency

N vertices are given (where N is the number of triangles to draw).
Points 0, 2, and 4 define the triangle. Points 1, 3, and 5 tell where adjacent triangles are.

Triangle Strip with Adjacency

N+2 vertices are given (where N is the number of triangles to draw).
Points 0, 2, 4, 6, 8, 10, … define the triangles. Points 1, 3, 5, 7, 9, 11, … tell where adjacent triangles are.

Adjacency Primitives (and what they do by default)

If a Vertex Shader Writes Variables as: then the Geometry Shader will Read Them as:

- gl_Position: gl_PositionIn[
- gl_PointSize: gl_PointSizeIn[
- gl_Layer: gl_LayerIn[

"out" "in" "out"

In the Geometry Shader, the dimensions indicated by ‘\[’ are given by the variable gl_VerticesIn, although you will already know this by the type of geometry you are inputting.

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.

The Geometry Shader Can Assign These Built-In out Variables:

- gl_Position
- gl_PointSize
- gl_Layer
- gl_PrimitiveID

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

Example: A Bézier 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

```glsl
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( lines_adjacency ) in;
layout( line_strip, max_vertices=200 ) out;
uniform int uNum;
void
main( )
{
    float dt = 1. / float(uNum);
    float t = 0.;
    for( int i = 0; i <= uNum; i++ )
    {
        float omt = 1. - t;
        float omt2 = omt * omt;
        float omt3 = omt * omt2;
        float t2 = t * t;
        float t3 = t * t2;
        vec4 xyzw = omt3 * gl_PositionIn[0].xyzw +
                     3. * t * omt2 * gl_PositionIn[1].xyzw +
                     3. * t2 * omt * gl_PositionIn[2].xyzw +
                     t3 * gl_PositionIn[3].xyzw;
        gl_Position = xyzw;
        EmitVertex( )
        t += dt;
    }
}
```

Note: It would have made no Difference if the Matrix Transform had been done in the Geometry Shader Instead

```glsl
beziercurve.vert

void main( )
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Note: these are used to define the storage

Example: Shrinking Triangles

```glsl
vec4 xyzw = omt3 * gl_PositionIn[0].xyzw +
            3. * t * omt2 * gl_PositionIn[1].xyzw +
            3. * t2 * omt * gl_PositionIn[2].xyzw +
            t3 * gl_PositionIn[3].xyzw;

CG = (P0 + P1 + P2) / 3.;
P0' = CG + uShrink * ( P0 - CG )
P1' = CG + uShrink * ( P1 - CG )
P2' = CG + uShrink * ( P2 - CG )
```

Another Example: Shrinking Triangles

```glsl
CG = (P0 + P1 * P2) / 3.;
P0' = CG + uShrink * ( P0 - CG )
P1' = CG + uShrink * ( P1 - CG )
P2' = CG + uShrink * ( P2 - CG )
```
Example: Sphere Subdivision

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

\[
V(s,t) = V_0 + s(V_1 - V_0) + t(V_2 - V_0)
\]

Example: Sphere Subdivision

Example: Sphere Subdivision

Example: Sphere Subdivision

Example: Sphere Subdivision
```c
void main( )
{
  V01 = ( gl_PositionIn[1] - gl_PositionIn[0] ).xyz;
  V02 = ( gl_PositionIn[2] - gl_PositionIn[0] ).xyz;
  V0 = gl_PositionIn[0].xyz;
  int numLayers = 1 << uLevel;
  float dt = 1. / float( numLayers );
  float t_top = 1.;
  for( int it = 0; it < numLayers; it++ )
  {
    ...

    float t_bot = t_top - dt;
    float smax_top = 1. - t_top;
    float smax_bot = 1. - t_bot;
    int nums = it + 1;
    float ds_top = smax_top / float( nums - 1 );
    float ds_bot = smax_bot / float( nums );
    float s_top = 0.;
    float s_bot = 0.;
    for( int is = 0; is < nums; is++ )
    {
      ProduceVertex( s_bot, t_bot );
      ProduceVertex( s_top, t_top );
      s_top += ds_top;
      s_bot += ds_bot;
    }
    ProduceVertex( s_bot, t_bot );
    EndPrimitive( );
    t_top = t_bot;
    t_bot -= dt;
  }
}
```

**Example: Sphere Subdivision**

1. Break the triangles into points
2. Treat each point’s distance from the triangle’s CG as an initial velocity
3. Follow the laws of projectile motion:

\[
\begin{align*}
X &= X_0 + V_0 t \\
Y &= Y_0 + V_y t + \frac{1}{2} a t^2
\end{align*}
\]

---

**Another Example: Explosion**

1. Break the triangles into points
2. Treat each point’s distance from the triangle’s CG as an initial velocity
3. Follow the laws of projectile motion:

---

**Example: Explosion**

```c
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles ) in;
layout( points, max_vertices=200 ) out;
uniform int uLevel;
uniform float uGravity;
uniform float uTime;
uniform float uVelScale;
vec3 V0, V01, V02;
vec3 CG;
void ProduceVertex( float s, float t )
{
  vec3 v = V0 + s*V01 + t*V02;
  vec3 vel = uVelScale * ( v - CG );
  v = CG + vel*uTime + 0.5*vec3(0.,uGravity,0.)*uTime*uTime;
  gl_Position = gl_ProjectionMatrix * vec4( v, 1. );
  EmitVertex( );
}
```
```c
void main()
{
    float smax = 1. - t;
    int nums = it + 1;
    float ds = smax / float( nums - 1 );
    float s = 0.0;
    for( int is = 0; is < nums; is++ )
    {
        ProduceVertex( s, t );
        s += ds;
    }
    t -= dt;
}
```

Another Example: Silhouettes

1. Compute the normals of each of the four triangles
2. If there is a sign difference between the z component of the center triangle and the z component of an adjacent triangle, draw their common edge

```c
uniform vec4 uColor;
void main()
{
    gl_FragColor = vec4( uColor.rgb, 1.0 );
}
```

Example: Silhouettes

```c
void main()
{
    gl_Position = gl_ModelViewMatrix * gl_Vertex;
}
```

uniform vec4 uColor;
void main()
{
    gl_FragColor = vec4( uColor.rgb, 1.0 );
}

Example: Silhouettes

Obj bunny.obj
Vertex silh.vert
Geometry silh.geom
Fragment silh.frag
Program Silhouette uColor { 0. 1. 0. 1. }

ObjAdj bunny.obj

Example: Silhouettes

```c
1. Compute the normals of each of the four triangles
2. If there is a sign difference between the z component of the center triangle and the z component of an adjacent triangle, draw their common edge

```c
vec3 V0 = gl_PositionIn[0].xyz;
vec3 V1 = gl_PositionIn[1].xyz;
vec3 V2 = gl_PositionIn[2].xyz;
vec3 V3 = gl_PositionIn[3].xyz;
vec3 V4 = gl_PositionIn[4].xyz;
vec3 V5 = gl_PositionIn[5].xyz;
vec3 N042 = cross( V4-V0, V2-V0 ); // the center triangle's normal
vec3 N021 = cross( V2-V0, V1-V0 );
vec3 N243 = cross( V4-V2, V3-V2 );
vec3 N405 = cross( V0-V4, V5-V4 );
if( dot( N042, N021 ) < 0. ) // make sure each outer triangle's normal is in the same general direction
    N021 = vec3(0.,0.,0.) - N021;
if( dot( N042, N243 ) < 0. )
    N243 = vec3(0.,0.,0.) - N243;
if( dot( N042, N405 ) < 0. )
    N405 = vec3(0.,0.,0.) - N405;
```
if( N042.z * N021.z <= 0. )
{
    gl_Position = gl_ProjectionMatrix * vec4( V0, 1. );
    EmitVertex( );
    gl_Position = gl_ProjectionMatrix * vec4( V2, 1. );
    EmitVertex( );
    EndPrimitive( );
}
if( N042.z * N243.z <= 0. )
{
    gl_Position = gl_ProjectionMatrix * vec4( V2, 1. );
    EmitVertex( );
    gl_Position = gl_ProjectionMatrix * vec4( V4, 1. );
    EmitVertex( );
    EndPrimitive( );
}
if( N042.z * N405.z <= 0. )
{
    gl_Position = gl_ProjectionMatrix * vec4( V4, 1. );
    EmitVertex( );
    gl_Position = gl_ProjectionMatrix * vec4( V0, 1. );
    EmitVertex( );
    EndPrimitive( );
}

Example: Silhouettes

Another Example: Hedgehog Plots

Example: Bunny Silhouettes

hedgehog.geom, I

float dt = 1. / float( numLayers );
float t = 1.;
for( int il = 0; il <= numLayers; il++ )
{
    float smax = 1. - t;
    int numa = il + 1;
    float da = smax / float( numa - 1 );
    for( int is = 0; is < numa; is++ )
    {
        ProduceVertices( s, t );
        s += da;
    }
    t -= dt;
}
A GLSL Built-in Variable for the Geometry Shaders

```glsl
int gl_PrimitiveIDIn
```

- Tells the number of primitives processed since the last time `glBegin()` was called
- Calling a vertex array function counts as an implied `glBegin()`
- `gl_PrimitiveIDIn` is 0 for the first primitive after the `glBegin()`

Geometry shaders can set the built-in variable `gl_PrimitiveID` to send a primitive number to the fragment shader.

What Happens if you Exceed the Maximum Allowed Emitted Vertices?

New in GLSL 4.x – you can loop back through the Geometry Shader multiple times.