GLSL Geometry Shaders

Mike Bailey
mjb@cs.oregonstate.edu

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?

Your application generates these:
- 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

The driver translates these and feeds them into the Geometry Shader:
- Point, Line, Line with Adjacency
- Triangle, Triangle with Adjacency
- Geometry Shader generates (almost) as many of these as it wants
- Points, Line Strips, Triangle Strips

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

4N 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.

N = 3
N = 1

Line Strip 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, #2 and #3, ..., #N-1 and #N.
Vertices #0 and #N-2 are there to provide adjacency information.

Adjacency Primitives (and what they do by default)

Triangles with Adjacency

6N 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.

N = 4

Triangle Strip with Adjacency

4+2N 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.
If a Vertex Shader Writes Variables as:  
then the Geometry Shader will Read Them as:  
and will Write Them to the Fragment Shader as:

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

"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 (GL_POINTS, GL_LINES, GL_TRIANGLES, etc.).

---

The Geometry Shader Can Assign These Built-in `out` Variables:

- `gl_Position`
- `gl_PointSize`
- `gl_Layer`
- `gl_PrimitiveID`

When the Geometry Shader calls

- `EmitVertex()`
  
  this set of variables is copied to a slot in the shader’s Primitive Assembly step

- `EndPrimitive()`
  
  the vertices that have been saved in the Primitive Assembly step are then assembled, rasterized, etc.

Note: there is no "BeginPrimitive()" routine. It is implied by (1) the start 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.

- **V**
  
  out vec4 gl_Position;
  
  out vec4 vColor;
  
  vColor = gl_Color;
  
  These are already declared for you

- **G**
  
  in vec4 gl_PositionIn[i];
  
  in vec4 vColor[i];

- **F**
  
  in vec4 gColor;

---

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

### beziercurve.glib

- **Vertex** `bezercurve.vert`
- **Geometry** `bezercurve.geom`
- **Fragment** `bezercurve.frag`
- **Program** `BezierCurve` `uNum <2 4 50>`

**LineWidth**: `3.0`  
**LinesAdjacency**: `[0.0.0.] [1.1.1.] [2.1.2.] [3.1.0.]`

### bezier.curve.vert

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

### bezier.curve.frag

```c
void main()
{
    gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

### Note:

- These are used to define the storage.
Another Example: Shrinking Triangles

Example: Shrinking Triangles

\[ CG = \frac{(P_0 + P_1 + P_2)}{3}; \]
\[ P_0' = CG + u_{\text{Shrink}} \times (P_0 - CG) \]
\[ P_1' = CG + u_{\text{Shrink}} \times (P_1 - CG) \]
\[ P_2' = CG + u_{\text{Shrink}} \times (P_2 - CG) \]

\[ CG = (P_0 + P_1 + P_2) / 3; \]
\[ P_0' = CG + u_{\text{Shrink}} \times (P_0 - CG) \]
\[ P_1' = CG + u_{\text{Shrink}} \times (P_1 - CG) \]
\[ P_2' = CG + u_{\text{Shrink}} \times (P_2 - CG) \]

Another Example: Sphere Subdivision

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

\[ V(s, t) = V_0 + s \times (V_1 - V_0) + t \times (V_2 - V_0) \]
Example: Sphere Subdivision

V0 V1

V0 V1

V0 V1

uLevel = 0
numLayers = 2
uLevel = 1
numLayers = 2
uLevel = 2
numLayers = 4

Example: Sphere Subdivision

spheresubd.glsl

```glsl
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles ) in;
layout( triangle_strip, max_vertices=200 ) out;

uniform int uLevel;
uniform float uRadius;
out float gLightIntensity;
const vec3 LIGHTPOS = vec3( 0., 10., 0. );

vec3 V0, V01, V02;

void ProduceVertex( float s, float t ) {
    vec3 v = V0 + s*V01 + t*V02;
    v = normalize(v);
    vec3 n = v;
    vec3 tnorm = normalize( uNormalMatrix * n );
    vec4 ECposition = gl_ModelViewMatrix * vec4( (uRadius*v), 1. );
    gLightIntensity = abs( dot( normalize(LIGHTPOS - ECposition.xyz), tnorm ) );
    gl_Position = gl_ProjectionMatrix * ECposition;
    EmitVertex( );
}
```

Example: Sphere Subdivision

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

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```

Example: Sphere Subdivision

```
void main( ) {
    gl_FragColor = vec4( gLightIntensity*uColor.rgb, 1. );
}
```
```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++) {
            ProducVertex(s_bot, t_bot);
            ProducVertex(s_top, t_top);
            s_bot += ds_bot;
            s_top += ds_top;
        }
        ProducVertex(s_bot, t_bot);
        EndPrimitive();
        t_top = t_bot;
        t_bot -= dt;
    }
}
```
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:

\[ x = x_0 + v_x t \]
\[ y = y_0 + v_y t + \frac{1}{2} a_y t^2 \]

Example: Explosion

```glsl
#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( );
}
```

Example: Explosion

```glsl
void main( )
{
    V01 = ( gl_PositionIn[1] - gl_PositionIn[0] ).xyz;
    V02 = ( gl_PositionIn[2] - gl_PositionIn[0] ).xyz;
    V0 = gl_PositionIn[0].xyz;
    CG = ( gl_PositionIn[0].xyz + gl_PositionIn[1].xyz + gl_PositionIn[2].xyz ) / 3. ;

    int numLayers = 1 << uLevel;
    float dt = 1. / float( numLayers );
    float t = 1. ;
    for( int it = 0; it <= numLayers; it++ )
    {
        float smax = 1. - t;
        int nums = it + 1;
        float ds = smax / float( nums - 1 );
        float s = 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

Example: Silhouettes

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

uniform vec4 uColor;
void main() {
    gl_FragColor = vec4( uColor.rgb, 1. );
}
Example: Silhouettes

```cpp
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: Bunny Silhouettes

Another Example: Hedgehog Plots

```cpp
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles ) in;
layout( line_strip, max_vertices=200 ) out;
uniform int uDetail;
uniform float uDroop;
uniform int uLength;
uniform float uStep;
in vec3 vTnorm[3];
in vec4 vColor[3];
out vec4 gColor;
int ILength;
vec3 Norm[3];
vec3 N0, N01, N02;
vec4 V0, V01, V02;
void ProduceVertices( float s, float t ) {
    vec4 v = V0 + s*V01 + t*V02;
    vec3 n = normalize( N0 + s*N01 + t*N02 );
    for( int i = 0; i <= uLength; i++ ) {
        gl_Position = gl_ProjectionMatrix * v;
        EmitVertex( );
        v.xyz += uStep * n;
        v.y -= uDroop * float(i*i);
    }
    EndPrimitive( );
}
```
void main()
{
    VS = gl_PositionIn[0];
    V01 = ( gl_PositionIn[1] - gl_PositionIn[0] );
    V02 = ( gl_PositionIn[2] - gl_PositionIn[0] );
    Norm[0] = vTnorm[0];
    Norm[1] = vTnorm[1];
    Norm[2] = vTnorm[2];
    if( dot( Norm[0], Norm[1] ) < 0. )
        Norm[1] = -Norm[1];
    if( dot( Norm[0], Norm[2] ) < 0. )
        Norm[2] = -Norm[2];
    N0   = normalize( Norm[0] );
    N01 = normalize( Norm[1] - Norm[0] );
    N02 = normalize( Norm[2] - Norm[0] );
    int numLayers = 1 << uDetail;
    float dt = 1. / float( numLayers );
    float t = 1.;
    for( int it = 0; it <= numLayers; it++ )
    {
        float smax = 1. - t;
        int nums = it + 1;
        float ds = smax / float( nums - 1 );
        float s = 0.;
        for( int is = 0; is < nums; is++ )
        {
            ProduceVertices( s, t );
            s += ds;
        }
        t -= dt;
    }
}

Ducky Hedgehog Plot

Hedgehog Plots Gone Wild
**A GLSL Built-in Variable for the Geometry Shaders**

```c
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