GLSL Geometry Shaders

Here's What We Know So Far

One Vertex In

- Vertex Shader

One Vertex Out

Here's What We Want Next

One Vertex In

Array of Vertices Out

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

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

Your application generates these:

- Points, Lines, Line Strips, Line Loops, Lines with Adjacency
- Triangle Strips, Triangle Fan, Triangles with Adjacency

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

Points, Lines, Line Strips, Line Loops

Points, Lines, Line Strips

Points, Lines, Line Strips, Triangle Strips

Points, Lines, 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, points can generate points, etc.
Additional Topologies that Geometry Shaders made Available:

- GL_LINES_ADJACENCY
- GL_LINE_STRIP_ADJACENCY
- GL_TRIANGLES_ADJACENCY
- GL_TRIANGLE_STRIP_ADJACENCY

Adjacency Primitives (and what they do when not using shaders)

Lines with Adjacency

- \( N \) vertices are given, where \( N \) is the number of line segments to draw.
- \( N+3 \) vertices are given, with \( N \) defining the line segments.
- Vertices 0 and \( N+2 \) are there to provide adjacency information.

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.

In these examples, we will use the "with adjacency" primitives only as a way of importing some number of vertices into the geometry shader.

These are the most useful:

- GL_LINES_ADJACENCY 4 vertices
- GL_TRIANGLES_ADJACENCY 6 vertices

What Do the Inputs to a Geometry Shader Look Like?

- gl_Position
- gl_PointSize
- Plus, any of your own variables that you have declared as out

When the Geometry Shader calls EmitVertex(),

- The set of variables is copied into the Shader's Primitive Assembly step.
- The vertices that have been saved in the Primitive Assembly elements are then assembled, rasterized, etc.

What Do the Outputs from a Geometry Shader Look Like?

- gl_Position
- gl_PointSize
- Plus, any of your own variables that you have declared as out

Note: there is no "BeginPrimitive()" function. It is implied by (1) the start of the Geometry Shader, or (2) returning from the EndPrimitive() call. Also, there is no need to call EndPrimitive() at the end of the Geometry Shader – it's 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.

```glsl
evColor = gl_Color;

in vec4 vColor;
in vec4 gColor;

out vec4 gColor;
gColor = vColor[0];

gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
EmitVertex();
t += dt;
```

That is, this is a pipeline. You cannot pass variables directly from the vertex shader to the fragment shader anymore.

Example: A Bézier Curve

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

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

```glsl
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 = omt2 * omt;
        float t2 = t * t;
        float t3 = t2 * omt;
        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: layout directives are a GLSL-ism and are used to define what the storage looks like.
Another Example: Shrinking Triangles

Another Example: Sphere Subdivision

Example: Shrinking Triangles

Example: Sphere Subdivision
Example: Sphere Subdivision

spheresubd.geom

```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( gl_NormalMatrix * n );  // the transformed normal
    vec4 ECposition = gl_ModelViewMatrix * vec4( (uRadius*v), 1. );
    gLightIntensity = abs(   dot( normalize(LIGHTPOS - ECposition.xyz), tnorm )   );
    gl_Position = gl_ProjectionMatrix * ECposition;
    EmitVertex( );
}
```

Since you are multiplying by the ModelView and Projection matrices here, don't multiply by any of these matrices in the vertex shader!

Example: Sphere Subdivision

spheresubd.geom

```glsl
for int it = 0; it < numLayers; it++ )
{
    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 with One triangle

spheresubd.geom

```glsl
for int it = 0; it < numLayers; it++ )
{
    float s_bot = it * dt;
    float s_top = (it + 1.) * dt;
    
    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

spheresubd.geom

```glsl
for int it = 0; it < numLayers; it++ )
{
    ProduceVertex( s_bot, t_bot );
    ProduceVertex( s_top, t_top );
    s_top += ds_top;
    s_bot += ds_bot;
}
```

Example: Sphere Subdivision

spheresubd.geom

```glsl
for int it = 0; it < numLayers; it++ )
{
    
    int numa = 0;
    float da_bot = numa * dt;
    float da_max = numa * dt;
    
    float s_max = 0.;
    float s_bot = 0.;
    
    float sBot = numa * dt;
    float sTop = numa * dt;
    
    int numa = 0;
    float da_bot = numa * dt;
    float da_max = numa * dt;
    
    float s_max = 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 with One triangle

spheresubd.geom

```glsl
void main() {
    gl_Position = gl_Vertex;
}
```
Example: Sphere Subdivision with the Whole Sphere (8 triangles)

Level = 0

Level = 1

Level = 2

Level = 3

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

1. Compute the normal vectors of each of the four triangles (one in the center and three around the outside)
2. If there is a sign difference between the z component of the center triangle’s normal and the z component of an adjacent triangle’s normal, draw their common edge

I.e., you are looking for a crease.
Example: Silhouettes

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

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

```
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles_adjacency ) in;
layout( line_strip, max_vertices=200 ) out;
void main( ) {
  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 * N243.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
```
#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;
    gColor = vColor[0];
    EmitVertex( );
    v.xyz += uStep * n;
    v.y -= uDroop * float(i*i);
  }
  EndPrimitive( );
}

void main( )
{
  V0  =   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] );
  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;
  }
}

int gl_PrimitiveIDIn

• Tells the number of primitives processed since the last time glBegin( ) was called
• Calling a vertex buffer drawing 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.