GLSL Geometry Shaders

Here's What We Know So Far

One Vertex In

One Vertex Out

Here's What We Have Next

One Vertex In

Array of Vertices Out

Arrays of Vertices Out, Possibly with a Change of Topology

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

If in use, it is always the last stop before the Rasterizer

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

Geometry Shader: What Does it Do?

Your application generates these

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

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

Points, Line, Line with Adjacency, Triangle, Triangle with Adjacency

Points, Line, Line with Adjacency, Triangle, Triangle with Adjacency

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

A line segment is drawn between #1 and #2.
Vertices #0 and #3 are there to provide adjacency information.

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 and #N+1.
Vertices #0 and #N+2 are there to provide adjacency information.

Triangles with Adjacency

Points 0, 2, and 4 define the triangle.
Points 1, 3, and 5 tell where adjacent triangles are.

N vertices are given.
(where N is the number of triangles to draw).

N = 3
N = 1

Adjacency Primitives (and what they do when using shaders)

In general, we will use the “with adjacency” primitives 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?

If a Vertex Shader Writes Variables to the Fragment Shader will

and will Write Them to

If a Vertex Shader Writes Variables to the Fragment Shader will

In the Geometry Shader, the dimensions indicated by [ ] are given by the variable gl_VertexInfo, although you will already know this by the type of geometry you are inputting

What Do the Outputs to a Geometry Shader Look Like?

When the Geometry Shader calls

the vertices that have been saved in the Primitive Assembly elements are then assembled, rasterized, etc.

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 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.

```glsl
out vec4 gl_Position;
out vec4 vColor;
vColor = gl_Color;

in vec4 vColor[3];
in vec4 gColor;
gColor = vColor[ k ];
```

These are already declared for you:

```glsl
in vec4 gl_PositionIn[3];
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 Bézier Curve with a Variable Number of Line Segments

**Vertex**

- `beziercurve.vert`

**Geometry**

- `beziercurve.geom`

**Fragment**

- `beziercurve.frag`

**Program**

- `BezierCurve` with `uNum < 2 4 50>`

**Layout Directives**

- `#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;`

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

Note: layout directives are a GLSL-ism and are used to define what the storage looks like.
Another Example: Shrinking Triangles

Example: Shrinking Triangles

\[ CG = \frac{P_0 + P_1 + P_2}{3}; \]

\[ P'_0 = CG + u_{\text{Shrink}} \times (P_0 - CG); \]

\[ P'_1 = CG + u_{\text{Shrink}} \times (P_1 - CG); \]

\[ P'_2 = CG + u_{\text{Shrink}} \times (P_2 - CG); \]

Example: Sphere Subdivision

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

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

Note! There is no place in this triangle where \( s = t = 1 \).
Example: Sphere Subdivision

spheresubd.geom

void main()
{
    gl_Position = gl_Vertex;
}

uniform vec4 uColor;
in float gLightIntensity;

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

spheresubd.vert

#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( );
}

spheresubd.geom

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 with One triangle

Level = 0

Level = 1

Level = 3

Level = 2

Example: Sphere Subdivision with the Whole Sphere (8 triangles)

Level = 0

Level = 1

Level = 3

Level = 2
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: 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'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

```
#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 * 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();
  }
}
```
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] );
    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;
    }
    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()

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