Here's What We Know So Far

One Vertex In = \texttt{gl\_Vertex}

One Vertex Out = \texttt{gl\_Position}

What is that Primitive Assembly Step For?

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?

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

Your application generates these

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

Geometry Shader: What Does it Do?

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

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

Points, Points, Points, TriangleStrip

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 were added with Geometry Shaders:

- 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**

- 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

**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 and #N+1.
  - 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.

**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.

Adjacency Primitives (and what we really do with them)

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

**Example is a Bezier curve:**

**Example is silhouette edges**

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(), this set of variables is copied to an entry in the shader’s Primitive Assembly step.

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

When the Geometry Shader calls EndPrimitive(), 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’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
out vec4 vColor;
vColor = gl_Color;
in vec4 vColor[3];
out vec4 gColor;
gColor = vColor[k];
```

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

Example: A Bézier Curve

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

\( 0 \leq t \leq 1 \).

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

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 = gl_ProjectionMatrix * xyzw;
        EmitVertex();
        t += dt;
    }
}
```

Another Example: Shrinking Triangles
Example: Shrinking Triangles

Centroid = "CG"

\[
CG = \left( \frac{P_0 + P_1 + P_2}{3} \right)
\]

\[
P_0' = CG + uShrink \times (P_0 - CG)
\]

\[
P_1' = CG + uShrink \times (P_1 - CG)
\]

\[
P_2' = CG + uShrink \times (P_2 - CG)
\]

shrink.vert

```glsl
#version 400 compatibility

out vec3 vNormal;

void main( )
{
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
    gl_Position = gl_ModelViewMatrix * gl_Vertex;
}
```

shrink.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 float uShrink;

in vec3 vNormal[3];

out float gLightIntensity;

const vec3 LIGHTPOS = vec3( 0., 10., 0. );

vec3 V[3];

vec3 CG;

void ProduceVertex( int v )
{
    gLightIntensity = dot( normalize(LIGHTPOS- V[v]), vNormal[v] );
    gLightIntensity = abs( gLightIntensity );
    gl_Position = gl_ProjectionMatrix * vec4( CG + uShrink * ( V[v] - CG ), 1. );
    EmitVertex( );
}

void main( )
{
    V[0] = gl_PositionIn[0].xyz;
    V[1] = gl_PositionIn[1].xyz;
    ProduceVertex( 0 );
    ProduceVertex( 1 );
    ProduceVertex( 2 );
}
```

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

```glsl
// Vertex shader
in vec3 vPosition;

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

```glsl
// Geometry shader
out vec3 vNormal;

void main()
{
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
    gl_Position = gl_ModelViewMatrix * gl_Vertex;
}
```

```glsl
// Fragment shader
out vec4 fColor;

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

Example: Sphere Subdivision

```glsl
// vertex shader
in vec3 vPosition;

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

```glsl
// geometry shader
out vec3 vNormal;

void main()
{
    vNormal = normalize( gl_NormalMatrix * gl_Normal );
    gl_Position = gl_ModelViewMatrix * gl_Vertex;
}
```

```glsl
// fragment shader
out vec4 fColor;

void main()
{
    fColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
```
Example: Sphere Subdivision

spheresubd.vert

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

spheresubd.frag

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

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 = vec4( uRadius*v, 1. );
    gLightIntensity = abs(   dot( normalize(LIGHTPOS - ECposition.xyz), tnorm )   );
    gl_Position = gl_ProjectionMatrix * ECposition;
    EmitVertex( );
}
```

Example: Sphere Subdivision with One triangle

```glsl
int numLayers = 1 << uLevel;
float dt = 1. / float( numLayers );
float t_top = 1.;
for( int it = 0; it < numLayers; it++ )
{
    . . .
}
```

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

```glsl
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;
}
```
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:

\[ \begin{align*}
X &= X_0 + V_x t \\
y &= Y_0 + V_y t + \frac{1}{2} a_y t^2
\end{align*} \]

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

Since you are multiplying by the Projection matrix in the geometry shader, don't also multiply by it in the vertex shader!

Example: Silhouettes

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.

`silh.glib`

```glsl
Obj bunny.obj
Vertex silh.vert
Geometry silh.geom
Fragment silh.frag
Program Silhouette uColor ( 0. 1. 0. 1. )
```

Obj `bunny.obj`
Example: Silhouettes

**silh.vert**

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

**silh.frag**

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

Example: Silhouettes

**silh.geom**

```glsl
#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
        N021 = vec3(0.,0.,0.) - N021;  // normal is in the same general direction
    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: Bunny Silhouettes

**hedgehog.geom**

```glsl
#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;
in[ ]    N0, N01, N02;
in[ ]    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( );
}
```

Another Example: Hedgehog Plots
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]) < 0.0)
        Norm[1] = -Norm[1];
    if (dot(Norm[0], Norm[2]) < 0.0)
        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 = 1.0 / float(numLayers);
    float t = 1.0;
    for (int it = 0; it <= numLayers; it++)
    {
        float smax = 1.0 - t;
        int nums = it + 1;
        float ds = smax / float(nums - 1);
        float s = 0.0;
        for (int is = 0; is < nums; is++)
        {
            ProduceVertices(s, t);
            s += ds;
        }
        t -= dt;
    }
}

A GLSL Built-in Variable for the Geometry Shaders

Int gl_PrimitiveID

• 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_PrimitiveID is 0 for the first primitive after the glBegin()