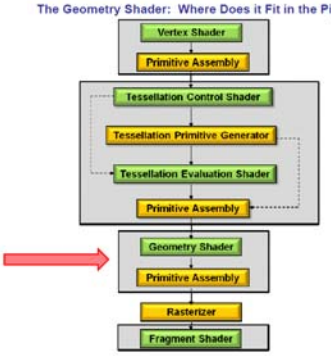
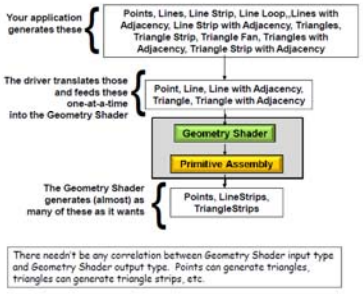
Geometry Shader

Thanks to Mike Bailey (OSU)

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



Geometry Shader: What Does it Do?



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.

Geometry Shaders

- Can cull geometry (do front/back/arbitrary culling
- Can amplify geometry (create geometry)
- Can emit different types than input
- Can generate multiple streams for single primative

Additional Arguments Available for glBegin( ):

GL_LINES_ADJACENCY

GL_LINE_STRIP_ADJACENCY

GL_TRIANGLES_ADJACENCY

GL_TRIANGLE_STRIP_ADJECENCY

Adjacency Primitives (and what they do by default)

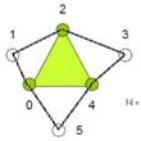
4N vertices are given.
(where N is the number of line segments to draw).
A line segment is drawn between #1 and #2.
Vertices #0 and #3 are there to provide adjacency information.

N+3 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.

Adjacency Primitives (and what they do by default)

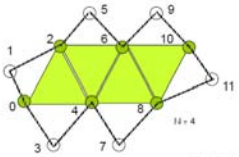
Triangles with Adjacency

6*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

4+2*N 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.



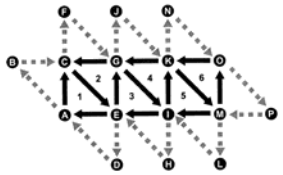


Figure 8.22: Triangles produced using GL_TRIANGLE_STRIP_ADJACENCY

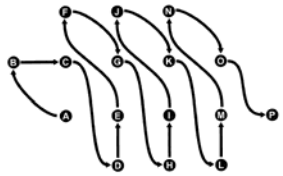


Figure 8.23: Ordering of vertices for GL_TRIANGLE_STRIP_ADJACENCY

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_PositionIn[ ]	gl_Position
gl_PointSize	gl_PointSizeIn[ ]	gl_PointSize
gl_Layer	gl_LayerIn[ ]	gl_Layer
"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

1 GL_POINTS
2 GL_LINES
4 GL_LINES_ADJACENCY
3 GL_TRIANGLES
6 GL_TRIANGLES_ADJACENCY

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

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

When the Geometry Shader calls ***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

Primitive Assembly

G

```
in vec4 gl_PositionIn[3];
in vec4 vColor[3];

out vec4 gl_Position;
out vec4 gColor;
gColor = vColor[ k ];
```

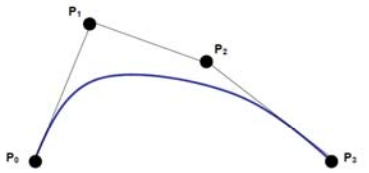
```
gl_Position = gl_PositionIn[0];
gColor = vColor[0];
EmitVertex( );
. . .
```

Primitive Assembly
Rasterizer

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

Lines Adjacency used for four points
(0, 0, 0)
(1, 1, 1)
(2, 1, 2)
(3,-1, 0)

beziercurve.vert

```
void main( )
{
	gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

beziercurve.frag

```
void main( )
{
	fFragColor = vec4( 0., 1., 0., 1. );
}
```

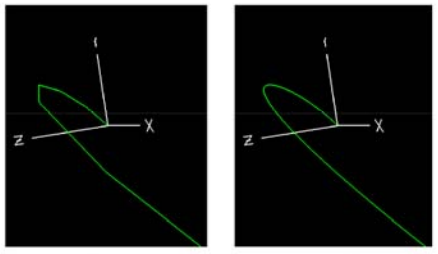
Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments

beziercurve.geom

```
#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: these are used to define the storage

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



Num = 5 Num = 25

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

beziercurve.vert

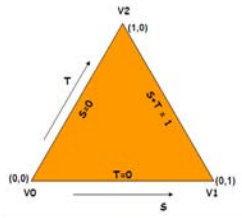
```
void main( )
{
	gl_Position = aVertex;
}
```

beziercurve.geom

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

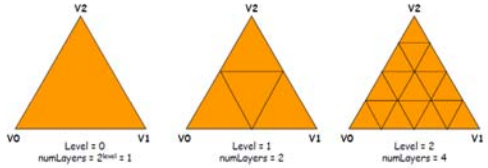
Another Example: Sphere Subdivision

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



V(s,t) = V0 + s*(V1-V0) + t*(V2-V0)

Example: Sphere Subdivision



Example: Sphere Subdivision

spheresubd.vert

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

Triangles [0. 0. 1.] [1. 0. 0.] [0. 1. 0.]
Triangles [1. 0. 0.] [0. 0. -1.] [0. 1. 0.]
Triangles [0. 0. -1.] [-1. 0. 0.] [0. 1. 0.]
Triangles [-1. 0. 0.] [0. 0. 1.] [0. 1. 0.]

Triangles [0. 0. 1.] [1. 0. 0.] [0. -1. 0.]
Triangles [1. 0. 0.] [0. 0. -1.] [0. -1. 0.]
Triangles [0. 0. -1.] [-1. 0. 0.] [0. -1. 0.]
Triangles [-1. 0. 0.] [0. 0. 1.] [0. -1. 0.]

spheresubd.frag

```
uniform vec4 uColor;
in float gLightIntensity;
out vec4 fFragColor;

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

mjb – January 3, 2012

Example: Sphere Subdivision

spheresubd.geom

```
#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 );  // the transformed normal

    vec4 ECposition = uModelViewMatrix * vec4( (uRadius*v), 1. );
    g LightIntensity  = dot( normalize(LIGHTPOS - ECposition.xyz), tnorm );
    g LightIntensity = abs( LightIntensity );

    gl_Position = uProjectionMatrix * ECposition;
    EmitVertex();
}
```

mjb – January 3, 2012

Example: Sphere Subdivision

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++ )
    {
        . . .
```

Triangles [0. 0. 1.] [1. 0. 0.] [0. 1. 0.]
Triangles [1. 0. 0.] [0. 0. -1.] [0. 1. 0.]
Triangles [0. 0. -1.] [-1. 0. 0.] [0. 1. 0.]
Triangles [-1. 0. 0.] [0. 0. 1.] [0. 1. 0.]

Triangles [0. 0. 1.] [1. 0. 0.] [0. -1. 0.]
Triangles [1. 0. 0.] [0. 0. -1.] [0. -1. 0.]
Triangles [0. 0. -1.] [-1. 0. 0.] [0. -1. 0.]
Triangles [-1. 0. 0.] [0. 0. 1.] [0. -1. 0.]

mjb – January 3, 2012

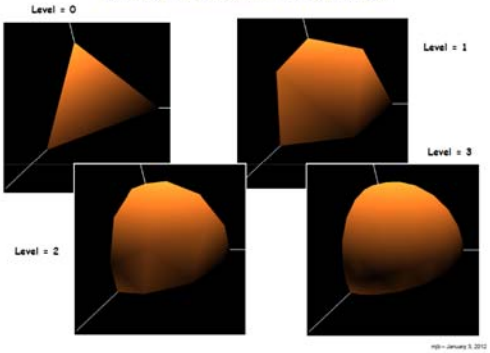
Example: Sphere Subdivision

spheresubd.geom

```
    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;
    }
}
```

mjb – January 3, 2012

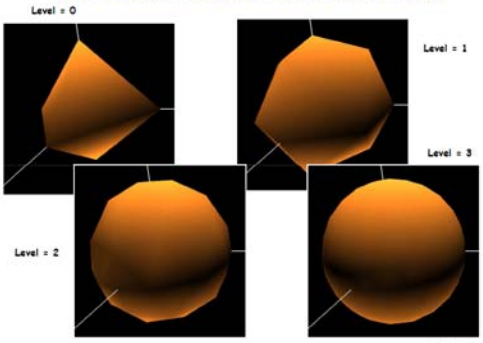
Example: Sphere Subdivision with One triangle



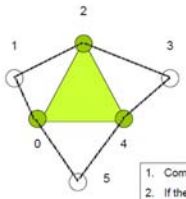
mjb – January 3, 2012

Stopped here

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



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

silh.vert

```
void main( )
{
	gl_Position = uModelViewMatrix * aVertex;
}
```

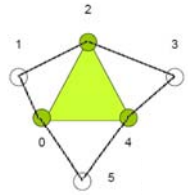
silh.frag

```
uniform vec4 uColor;
out vec4 fFragColor;

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

Example: Silhouettes

silh.geom

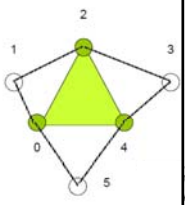
```
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles_adjacency ) in;
layout( line_strip, max_vertices=200 ) out;

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 );
	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. )
		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;
```



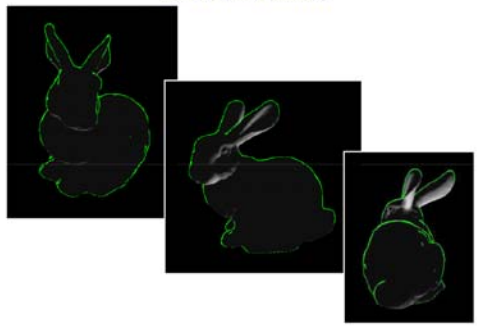
Example: Silhouettes

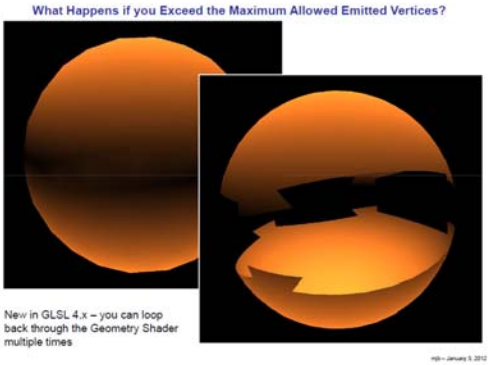
silh.geom

```
	if( N042.z * N021.z <= 0. )
	{
		gl_Position = uProjectionMatrix * vec4( V0, 1. );
		EmitVertex();
		gl_Position = uProjectionMatrix * vec4( V2, 1. );
		EmitVertex();
		EndPrimitive();
	}

	if( N042.z * N243.z <= 0. )
	{
		gl_Position = uProjectionMatrix * vec4( V2, 1. );
		EmitVertex();
		gl_Position = uProjectionMatrix * vec4( V4, 1. );
		EmitVertex();
		EndPrimitive();
	}

	if( N042.z * N405.z <= 0. )
	{
		gl_Position = uProjectionMatrix * vec4( V4, 1. );
		EmitVertex();
		gl_Position = uProjectionMatrix * vec4( V0, 1. );
		EmitVertex();
		EndPrimitive();
	}
}
```



Example: Bunny Silhouettes





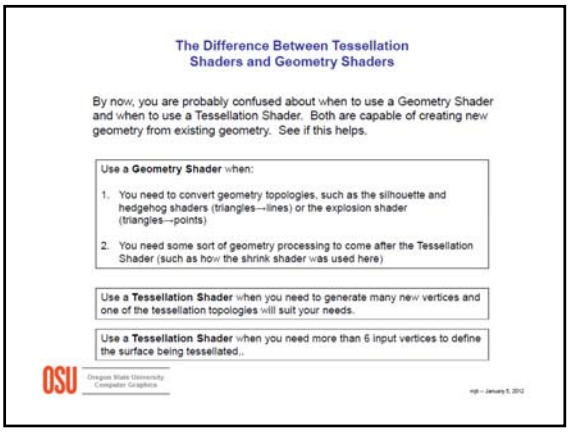
Demo normal offset

Demo Explosion

Demo 4-views

GS Quads (bi-linear interpolation)

GS Culling


The Difference Between Tessellation Shaders and Geometry Shaders
By now, you are probably confused about when to use a Geometry Shader and when to use a Tessellation Shader. Both are capable of creating new geometry from existing geometry. See if this helps.
Use a Geometry Shader when:
1. You need to convert geometry topologies, such as the silhouette and hedgehog shaders (triangles→lines) or the explosion shader (triangles→points)
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the shrink shader was used here)
Use a Tessellation Shader when you need to generate many new vertices and one of the tessellation topologies will suit your needs.
Use a Tessellation Shader when you need more than 6 input vertices to define the surface being tessellated..
OSU
Oregon State University
Computer Graphics