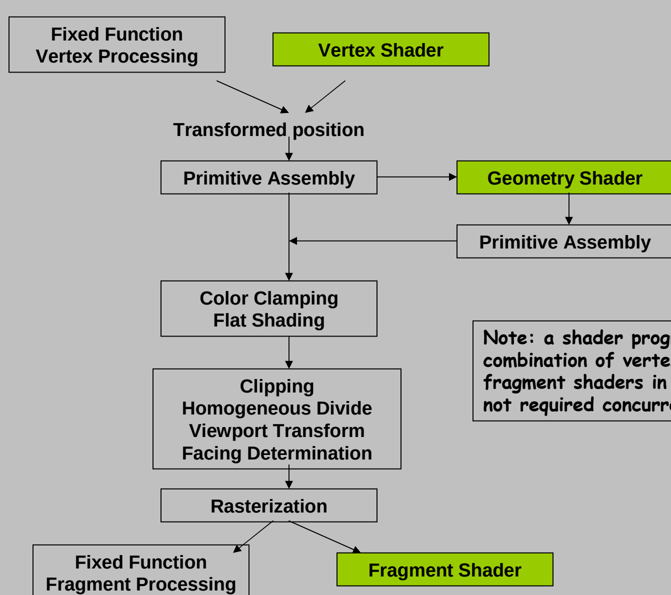
GLSL Geometry Shaders

Mike Bailey

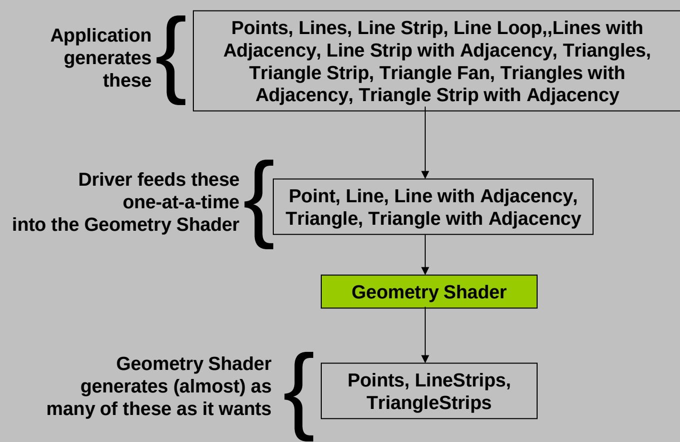
Oregon State University

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



Note: a shader program can have any combination of vertex, geometry, and fragment shaders in it. All three are not required concurrently.

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.

Additional Arguments to glBegin( ):

GL_LINES_ADJACENCY_EXT

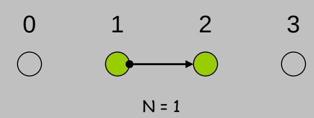
GL_LINE_STRIP_ADJACENCY_EXT

GL_TRIANGLES_ADJACENCY_EXT

GL_TRIANGLE_STRIP_ADJECENCY_EXT

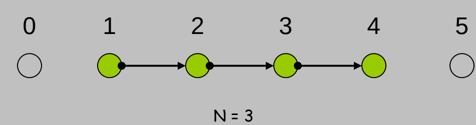
New Adjacency Primitives

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.

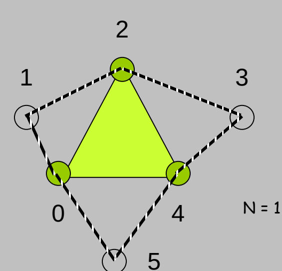
Line Strip with Adjacency



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.

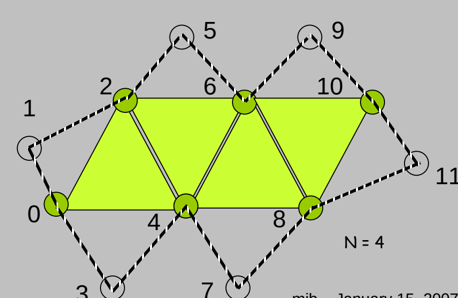
New Adjacency Primitives

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.

glProgramParameter Must Be Called Before the Shaders are Linked

glProgramParameteriEXT(progname, GL_GEOMETRY_VERTICES_OUT_EXT, int value)

Maximum number of vertices this Geometry Shader will be emitting

mjb – January 15, 2007

glProgramParameter Must Be Called Before the Shaders are Linked

glProgramParameteriEXT(progname, GL_GEOMETRY_INPUT_TYPE_EXT, int value)

The primitive type that this Geometry Shader will be receiving

GL_POINTS
GL_LINES
GL_LINES_ADJACENCY_EXT
GL_TRIANGLES
GL_TRIANGLES_ADJACENCY_EXT

Could actually come from GL_LINES, GL_LINE_STRIP, or GL_LINE_LOOP

Could actually come from GL_LINES_ADJACENCY_EXT or GL_LINE_STRIP_ADJACENCY_EXT

Could actually come from GL_TRIANGLES, GL_TRIANGLE_STRIP, or GL_TRIANGLE_FAN

Could actually come from GL_TRIANGLES_ADJACENCY_EXT or GL_TRIANGLE_STRIP_ADJACENCY_EXT

mjb – January 15, 2007

glProgramParameter Must Be Called Before the Shaders are Linked

```
glProgramParameteriEXT( progname, GL_GEOMETRY_OUTPUT_TYPE_EXT, int value )
```

The primitive type that this Geometry Shader will be sending on to the rest of the pipeline

GL_POINTS
GL_LINE_STRIP
GL_TRIANGLE_STRIP

Warning: glProgramParameteriEXT( ) calls can go into a Display List, deferring their execution until it is too late! (Bad idea…)

```
GLuint dl = glGenLists( 1 );
glNewList( dl, GL_COMPILE );
. . .
program = glCreateProgram();
. . .
glProgramParameteriEXT( program, GL_GEOMETRY_INPUT_TYPE_EXT,  inputGeometryType );

glProgramParameteriEXT( program, GL_GEOMETRY_OUTPUT_TYPE_EXT, outputGeometryType );

glProgramParameteriEXT( program, GL_GEOMETRY_VERTICES_OUT_EXT, 101 );

glLinkProgram( program );

glUseProgram( program );
. . .
glEndList( );
```

This gets executed *now.*

This gets executed *now*, probably with the wrong Program Parameter settings, generating an unexpected Link Error!

These get executed later, whenever the display list is glCallList'ed.

Moral: If you are creating a display list from a stream of input data, defer both the setting of Program Parameters and the Linking of the Program until after the Display List is complete. There is rarely a good reason to have calls to glProgramParameteriEXT( ) in a display list.

If a Vertex Shader Writes Variables as:	then the Geometry Shader will Read Them as:	and will Write Them as:
gl_Position →	gl_PositionIn[■] →	gl_Position
gl_Normal →	gl_NormalIn[■] →	gl_Normal
gl_TexCoord[ ] →	gl_TexCoordIn[ ] [■] →	gl_TexCoord[ ]
gl_FrontColor →	gl_FrontColorIn[■] →	gl_FrontColor
gl_BackColor →	gl_BackColorIn[■] →	gl_BackColor
gl_PointSize →	gl_PointSizeIn[■] →	gl_PointSize
gl_Layer →	gl_LayerIn[■] →	gl_Layer
gl_PrimitiveID	gl_PrimitiveIDIn[■]	gl_PrimitiveID

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_EXT
3	GL_TRIANGLES
6	GL_TRIANGLES_ADJACENCY_EXT

mjb – January 15, 2007

The Geometry Shader Can Assign These Variables:

- gl_Position
- gl_TexCoord[ ]
- gl_FrontColor
- gl_BackColor
- 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, and then is "reset"

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.

mjb – January 15, 2007

Notes

• In a Vertex Shader, varying variables become (1) inputs to the rasterizer if there is no Geometry Shader, or (2) inputs to the Geometry Shader if there is one.

• If there is a Geometry Shader, varying variables from the Vertex Shader are collected by the primitive assembly step and passed to the Geometry Shader once enough vertices have been collected for the current geometry input type.

• If there is a Geometry Shader, then there *must* also be a Vertex Shader.

• Geometry Shaders can access uniform variables just like Vertex and Fragment shaders can.

• Geometry Shaders can access all of the standard OpenGL-defined variables such as the transformation matrices. Thus, you can transform the original vertices in the Vertex Shader, or transform them as they are being emitted from the Geometry Shader, whichever is more convenient.

• In a Geometry Shader, the user-defined input varying variables, coming from the Vertex Shader, are declared as *varying in*. The Geometry Shader's output varying variables, headed to the rasterizer, are declared as *varying out*.

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

bezier.glib

```
GeometryInput     gl_lines_adjacency
GeometryOutput  gl_line_strip
Vertex bezier.vert
Geometry bezier.geom
Fragment bezier.frag
Program Bezier  FpNum <2. 10. 50.>

LineWidth 3.
LinesAdjacency [0. 0. 0.]  [1. 1. 1.]  [2. 1. 2.]  [3. -1. 0.]
```

bezier.vert

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

bezier.frag

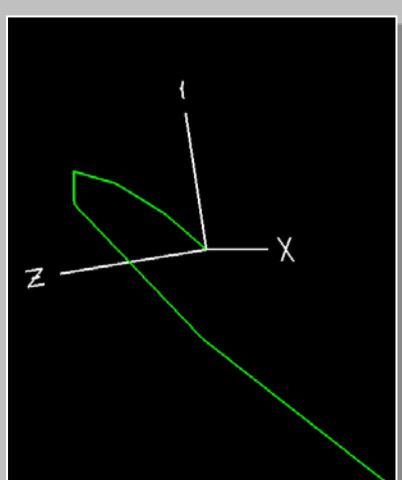
```
void main()
{
   gl_FragColor = vec4( 0., 1., 0., 1. );
}
```

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

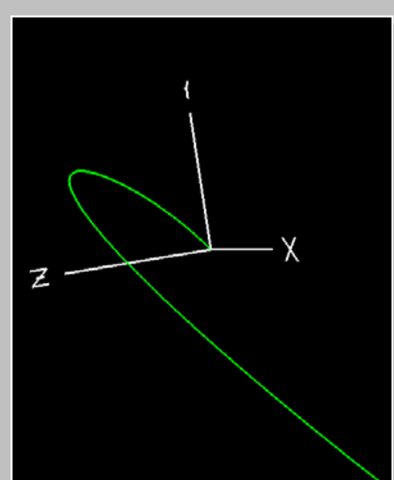
bezier.geom

```
#version 120
#extension GL_EXT_geometry_shader4: enable
uniform float FpNum;
void main()
{
        int num = int( FpNum + 0.99 );
        float dt = 1. / float(num);
        float t = 0.;
        for( int i = 0; i <= num; 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;
        }
}
```


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



FpNum = 5



FpNum = 25

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

bezier.vert

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

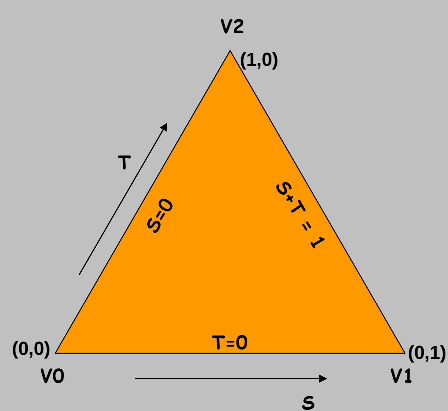
bezier.geom

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

mjb – January 15, 2007

Example: Sphere Subdivision

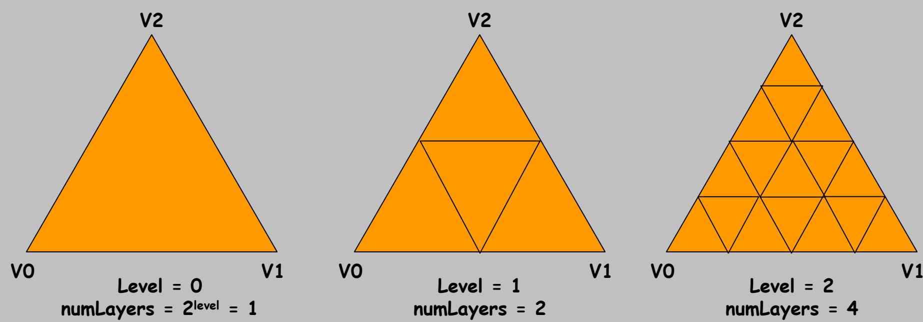
It's sometimes handy to parameterize a triangle into (S,T):



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

mjb – January 15, 2007

Example: Sphere Subdivision



Example: Sphere Subdivision

spheresubd.glib

```
GeometryInput    gl_triangles
GeometryOutput gl_triangle_strip
Vertex   spheresubd.vert
Geometry spheresubd.geom
Fragment spheresubd.frag
Program SphereSubd  FpLevel <0. 0. 10.>  Radius <.5 1. 5.>  Color { 1. .5 .15 }


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.]
```

Example: Sphere Subdivision

spheresubd.vert

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

spheresubd.frag

```
varying float LightIntensity;

uniform vec4 Color;


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


Example: Sphere Subdivision

spheresubd.geom

```
#version 120
#extension GL_EXT_geometry_shader4: enable

uniform float FpLevel;
uniform float Radius;
varying float LightIntensity;
vec3 V0, V01, V02;

void
ProduceVertex( float s, float t )
{
	const vec3 lightPos = vec3( 0., 10., 0. );
	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( (Radius*v), 1. );
	LightIntensity  = dot( normalize(lightPos - ECposition.xyz), tnorm );
	LightIntensity = abs( LightIntensity );
	LightIntensity *= 1.5;

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

spheresubd.geom

Example: Sphere Subdivision

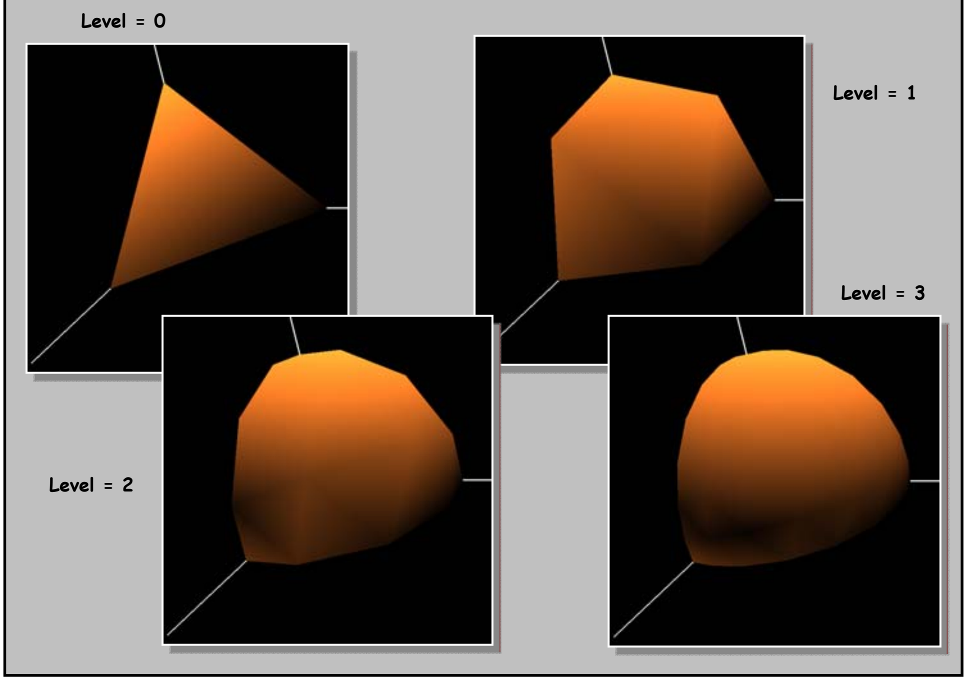
```
void
main()
{
	V01 = ( gl_PositionIn[1] - gl_PositionIn[0] ).xyz;
	V02 = ( gl_PositionIn[2] - gl_PositionIn[0] ).xyz;
	V0  =   gl_PositionIn[0].xyz;

	int level = int( FpLevel );
	int numLayers = 1 << level;

	float dt = 1. / float( numLayers );

	float t_top = 1.;

	for( int it = 0; it < numLayers; it++ )
	{
		. . .
```


spheresubd.geom

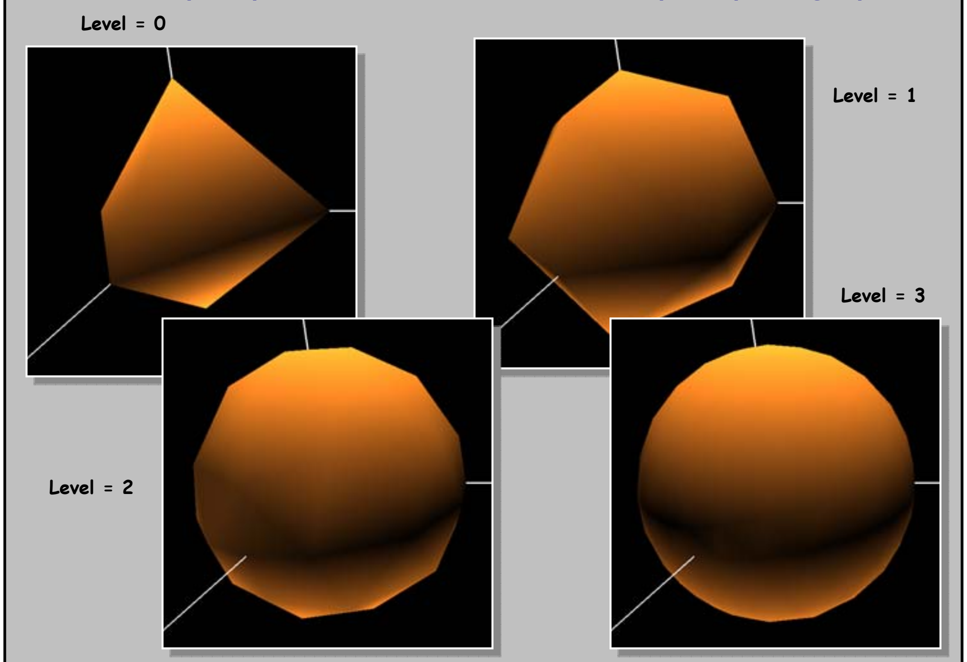
Example: Sphere Subdivision

```
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;
	}
}
```

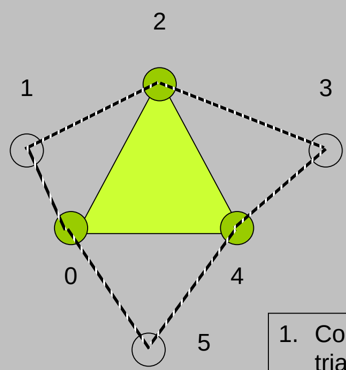
Example: Sphere Subdivision with One triangle



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



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

mjb – January 15, 2007

Example: Silhouettes

silh.glib

```
Obj bunny.obj

GeometryInput      gl_triangles_adjacency
GeometryOutput   gl_line_strip
Vertex   silh.vert
Geometry silh.geom
Fragment silh.frag
Program Silhouette  Color { 0. 1. 0. }

ObjAdj bunny.obj
```

mjb – January 15, 2007

Example: Silhouettes

silh.vert

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

silh.frag

```
uniform vec4 Color;

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

mjb – January 15, 2007

Example: Silhouettes

silh.geom

```
#version 120
#extension GL_EXT_geometry_shader4: enable

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;
```

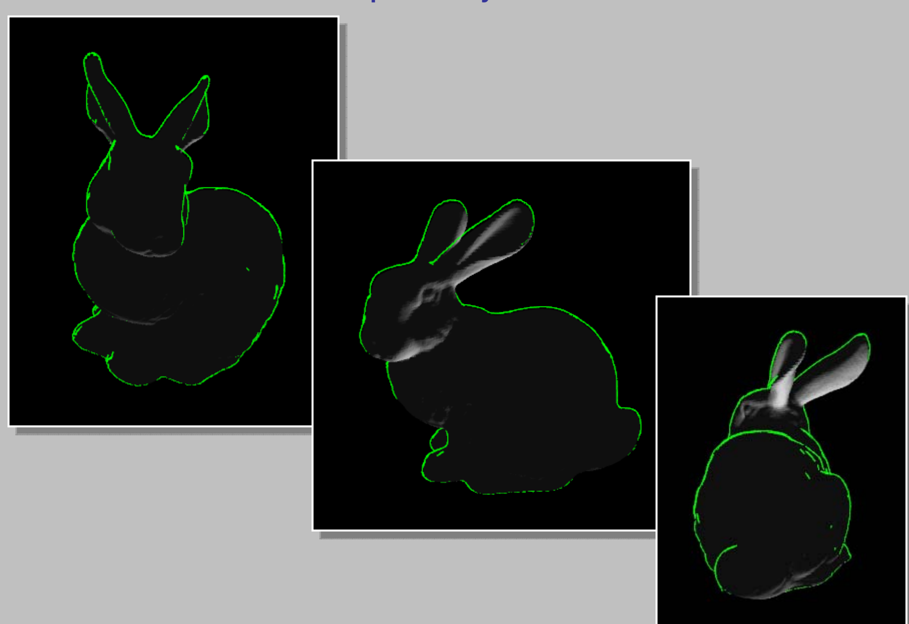
mjb – January 15, 2007

silh.geom

Example: Silhouettes

```
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



A New GLSL Built-in Variable for the Geometry Shaders

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

mjb – January 15, 2007

GLSL Geometry Shaders

Mike Bailey

Oregon State University