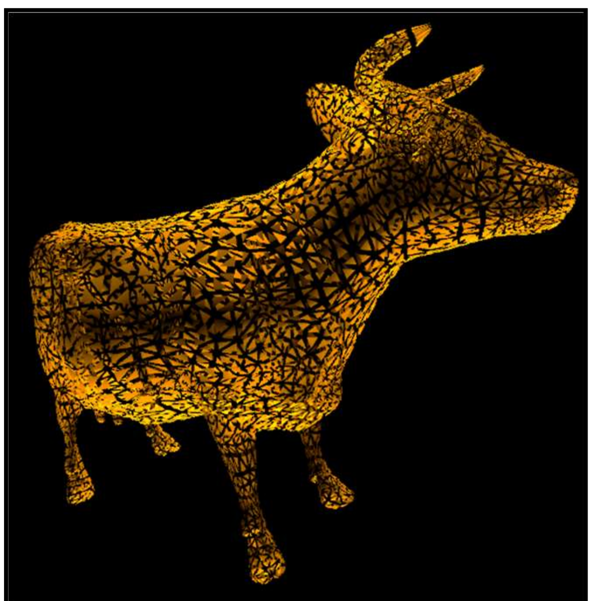
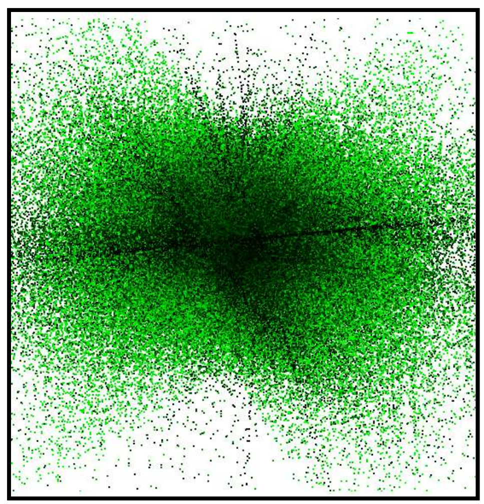
GLSL Geometry Shaders

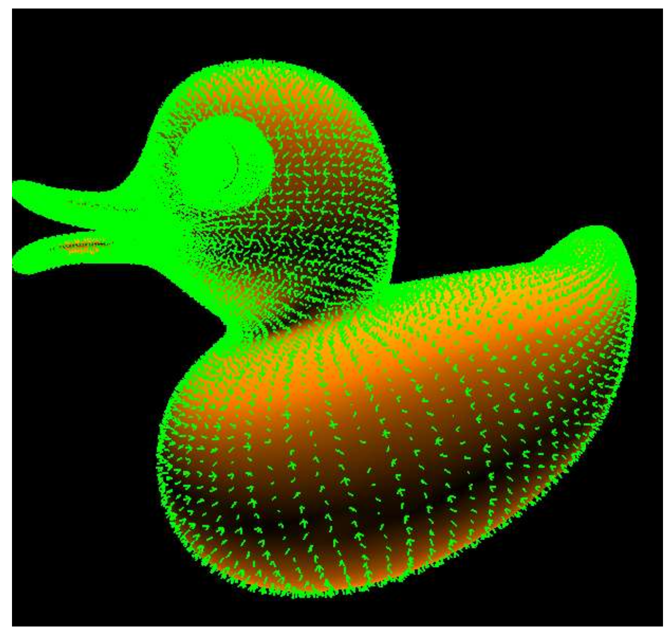
Oregon State University

Mike Bailey

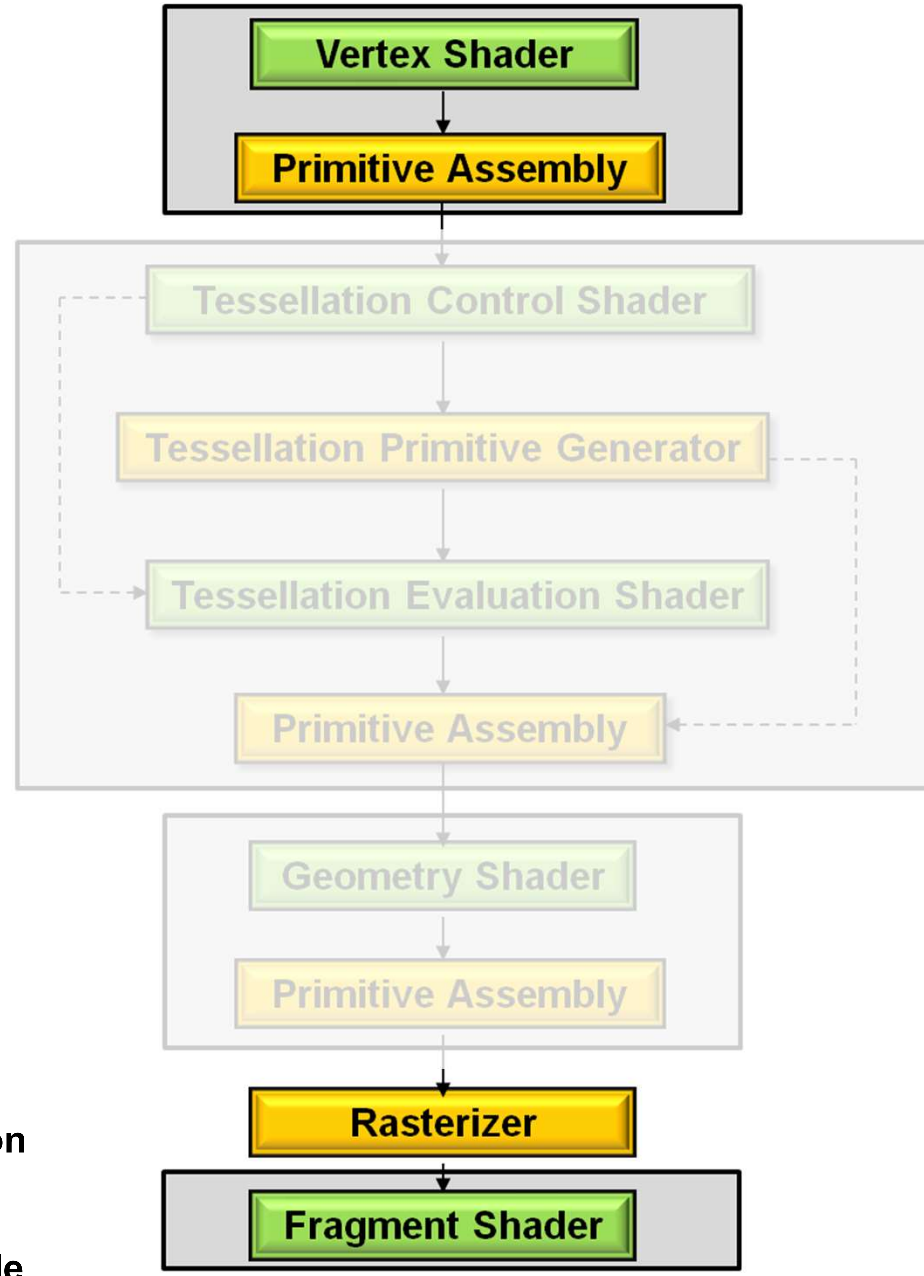
mjb@cs.oregonstate.edu







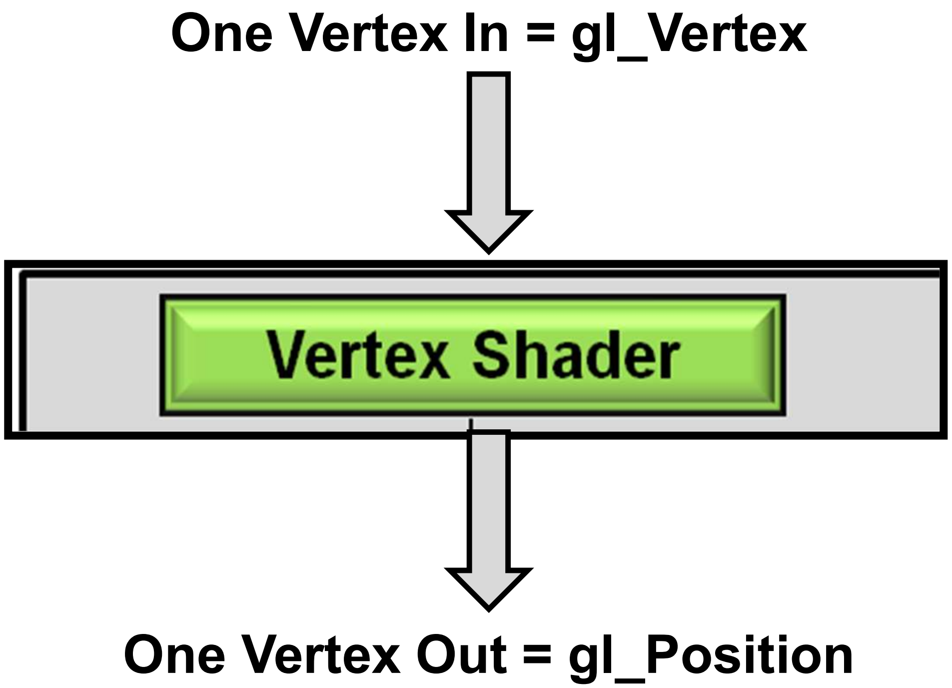
Here's What We Know So Far



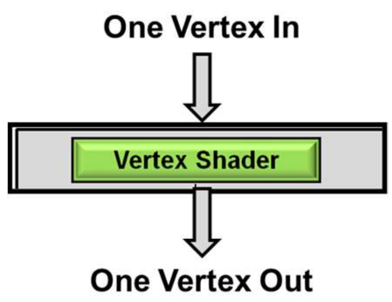
= Fixed Function

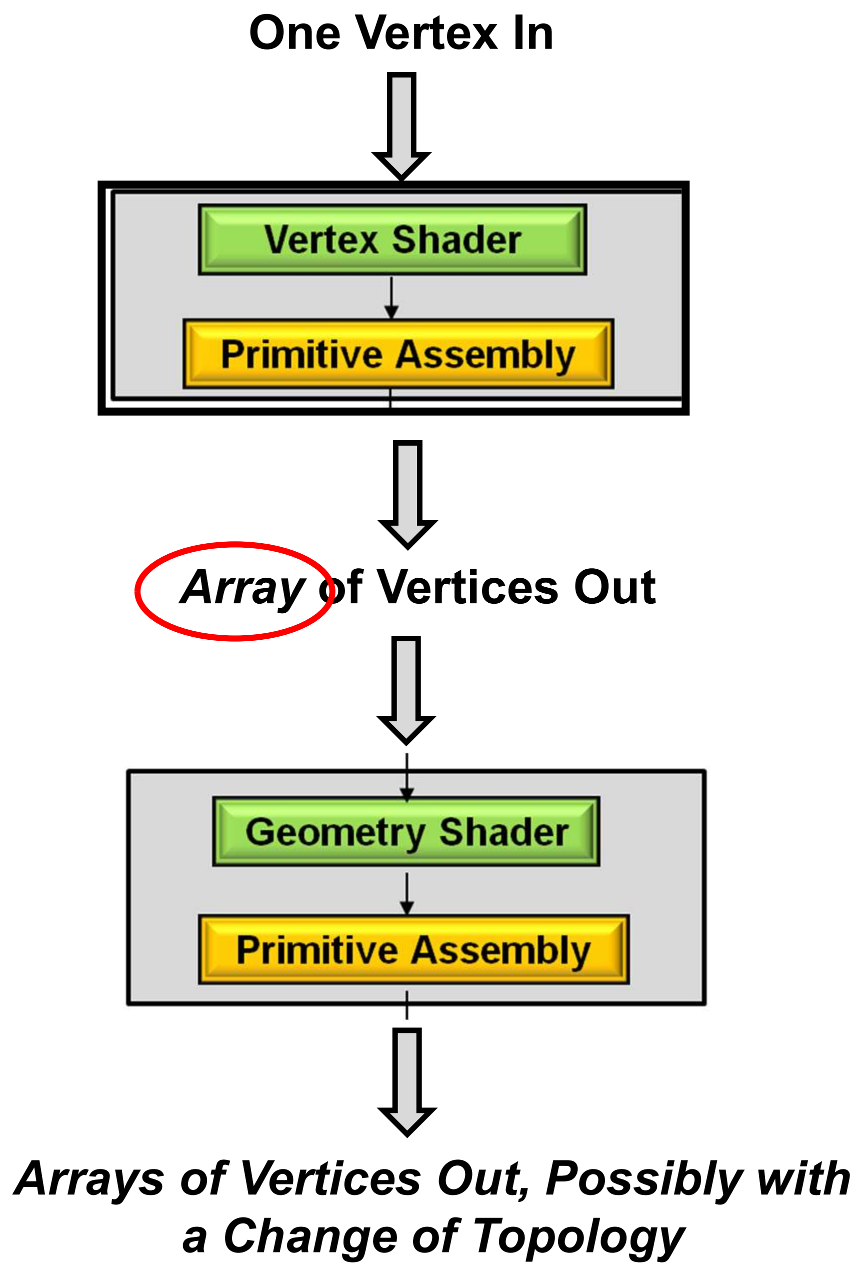
= Programmable

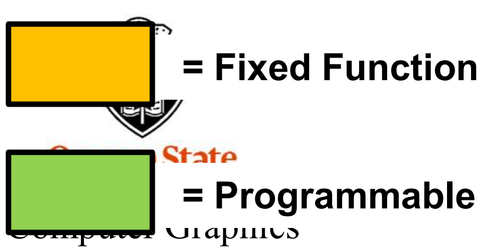
Here's What We Know So Far



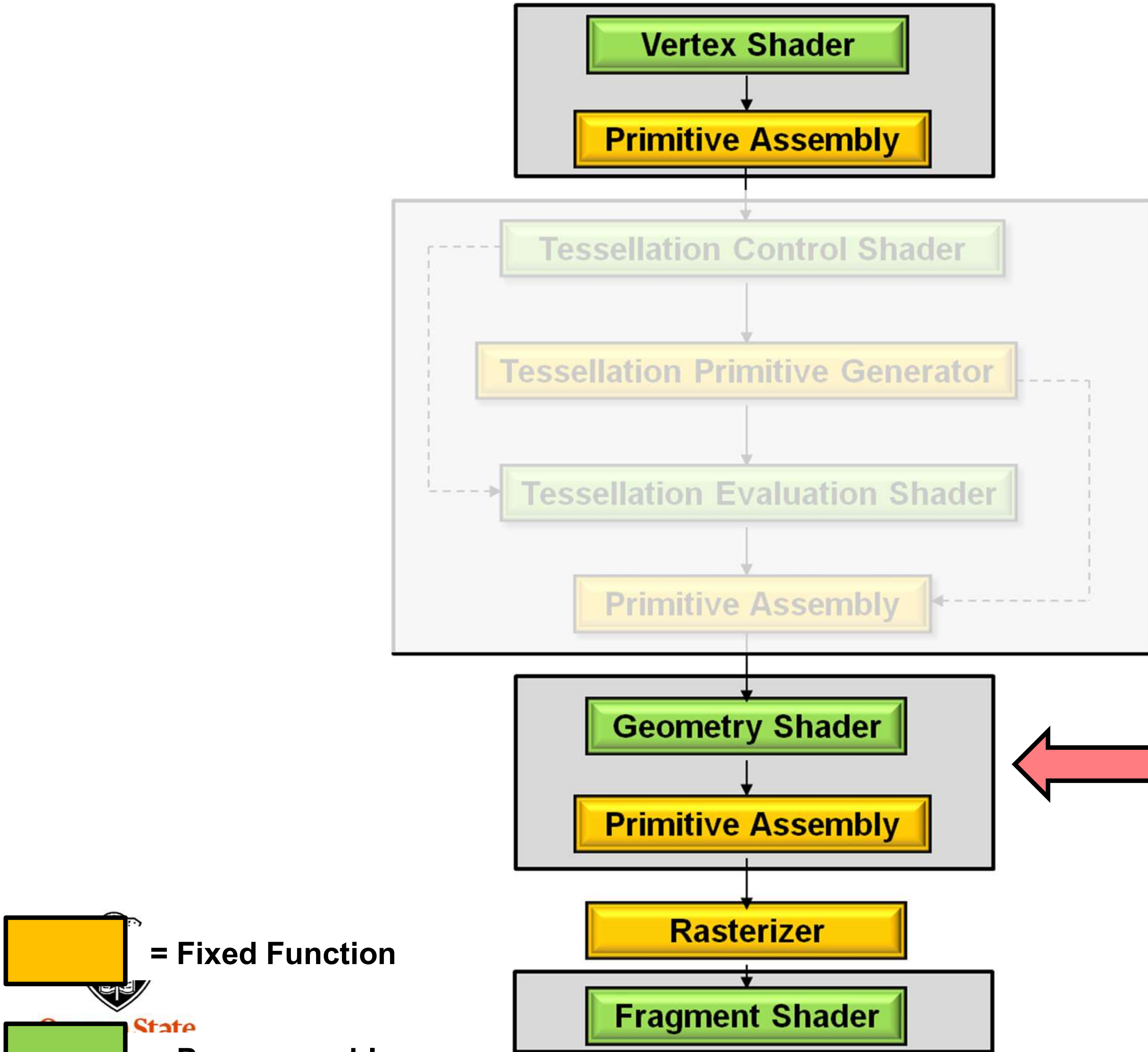
What is that *Primitive Assembly* Step For?





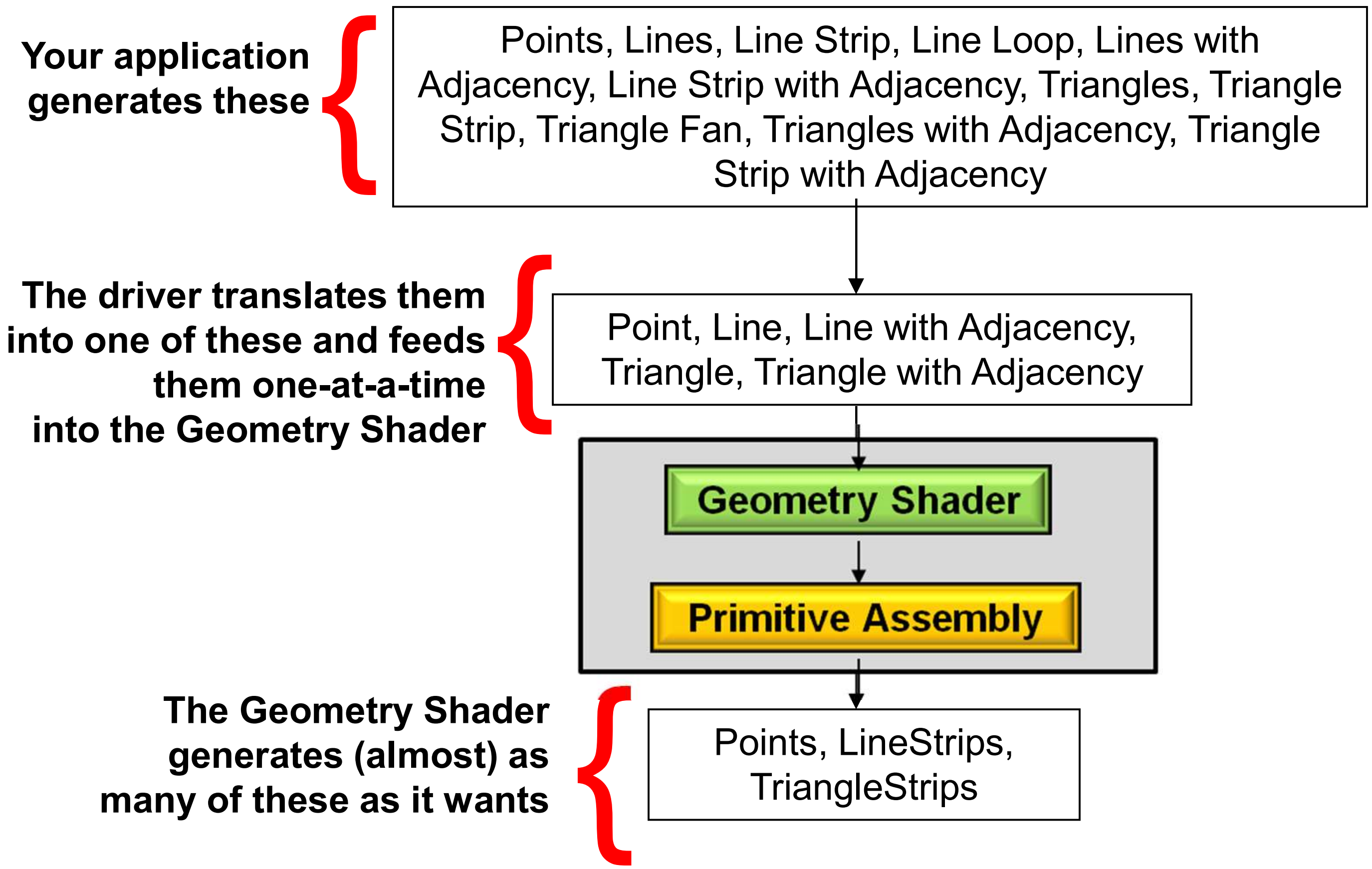


The Geometry Shader: Where Does it Fit in the Pipeline?



If a Geometry Shader is used, it is *always* the last stop before the Rasterizer

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, points can generate points, etc.

Additional Topologies were added with Geometry Shaders:

GL_LINES_ADJACENCY

GL_LINE_STRIP_ADJACENCY

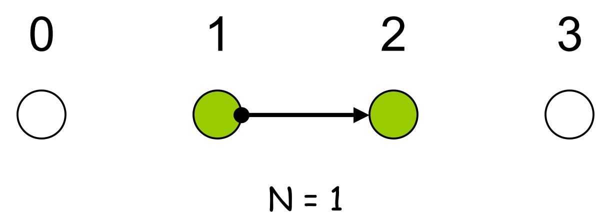
GL_TRIANGLES_ADJACENCY

GL_TRIANGLE_STRIP_ADJECENCY

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

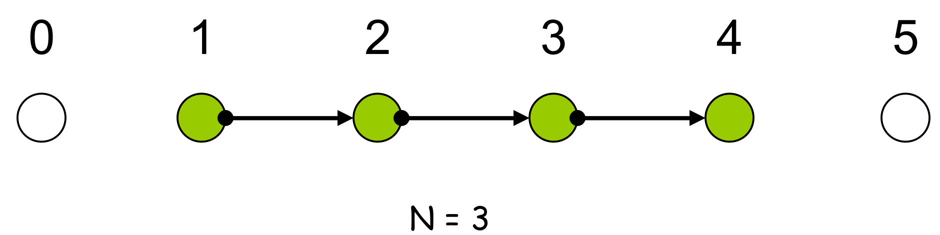
This is what Fixed-Function OpenGL expects these topologies to mean. In Shader World, they can mean whatever you want them to mean. In Shader World, it's just a way to get some number of vertices into a Geometry Shader.

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.

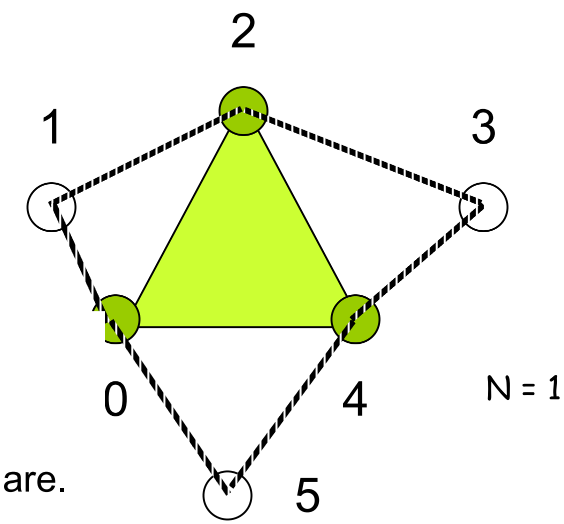
Line Strip with Adjacency



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.

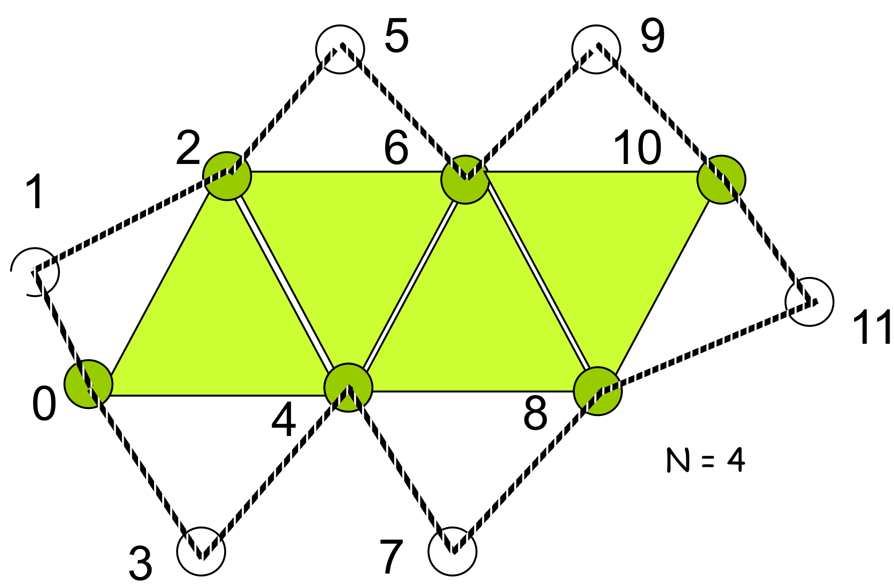
Adjacency Primitives (and what they do when not using shaders)

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.

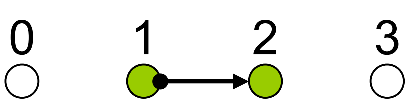
Adjacency Primitives (and what we really do with them)

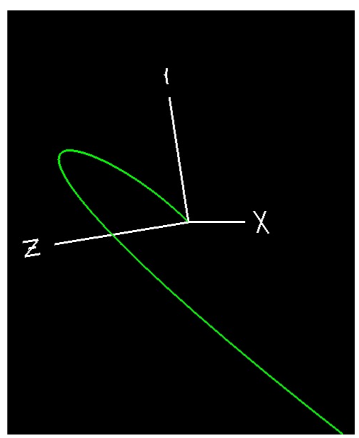
In these examples, we will use the "with adjacency" primitives only as a way of importing some number of vertices into the geometry shader.

These are the most useful:

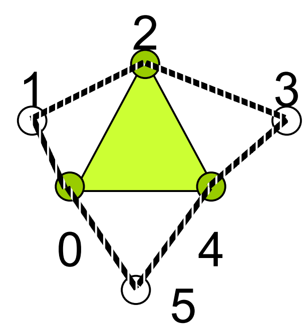
GL_LINES_ADJACENCY	4 vertices
GL_TRIANGLES_ADJACENCY	6 vertices

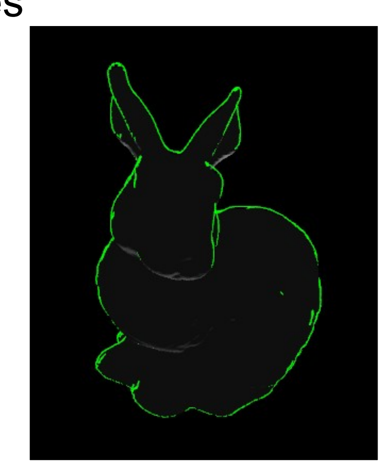
GL_LINES_ADJACENCY -- 4 vertices
Example is a Bezier curve:



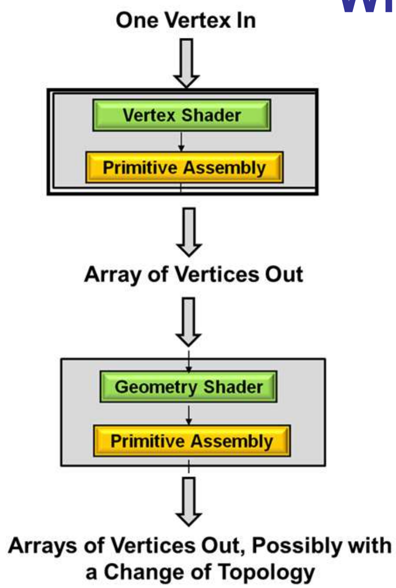


GL_TRIANGLES_ADJACENCY -- 6 vertices
Example is silhouette edges





What Do the Inputs to a Geometry Shader Look Like?

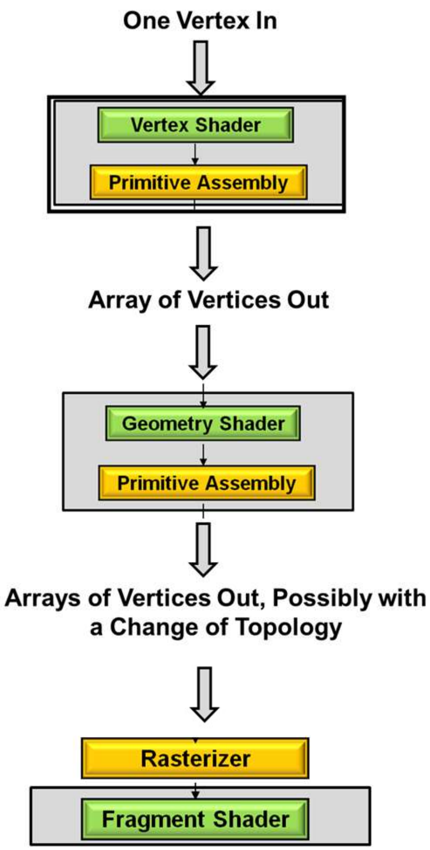


If a Vertex Shader Writes Variables as:	then the Geometry Shader will Read Them as:	and will Write Them to the Fragment Shader as:
gl_Position →	gl_PositionIn[ ] →	gl_Position
gl_PointSize →	gl_PointSizeIn[ ] →	gl_PointSize
"out"	*"in"*	*"out"*

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

What Do the Outputs from a Geometry Shader Look Like?



Basically, they look like what you already know:

- 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

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

V

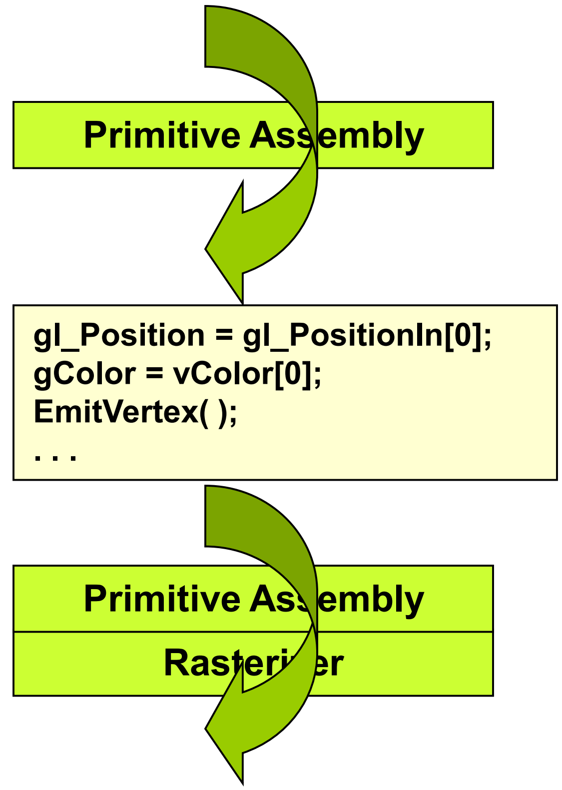
```
out vec4 vColor;
vColor = gl_Color;
```

G

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

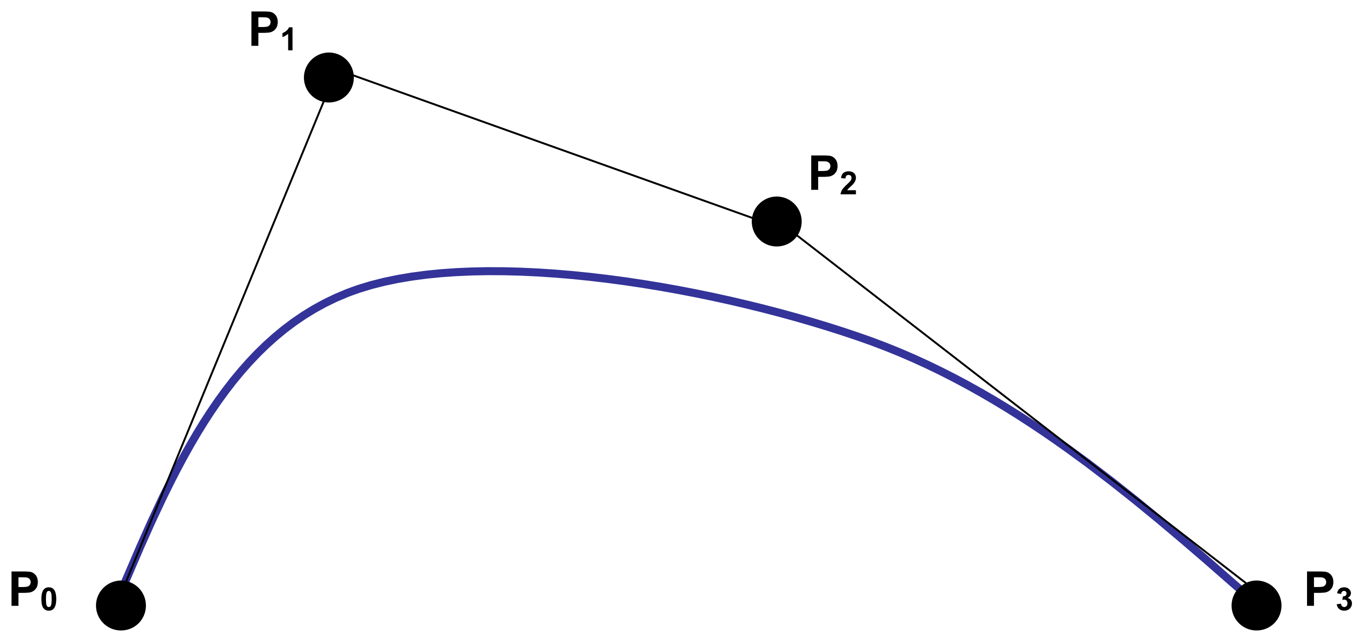
F

```
in vec4 gColor;
```



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

Example: A Bézier Curve



$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

$$0. \le t \le 1.$$

Need to pass **4** points in to define the curve. You need to pass **N** points out to draw the curve as a Line Strip.

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

beziercurve.glib

```
Vertex      beziercurve.vert
Geometry beziercurve.geom
Fragment beziercurve.frag
Program  BezierCurve  uNum <2 4 50>

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

beziercurve.vert

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

beziercurve.frag

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

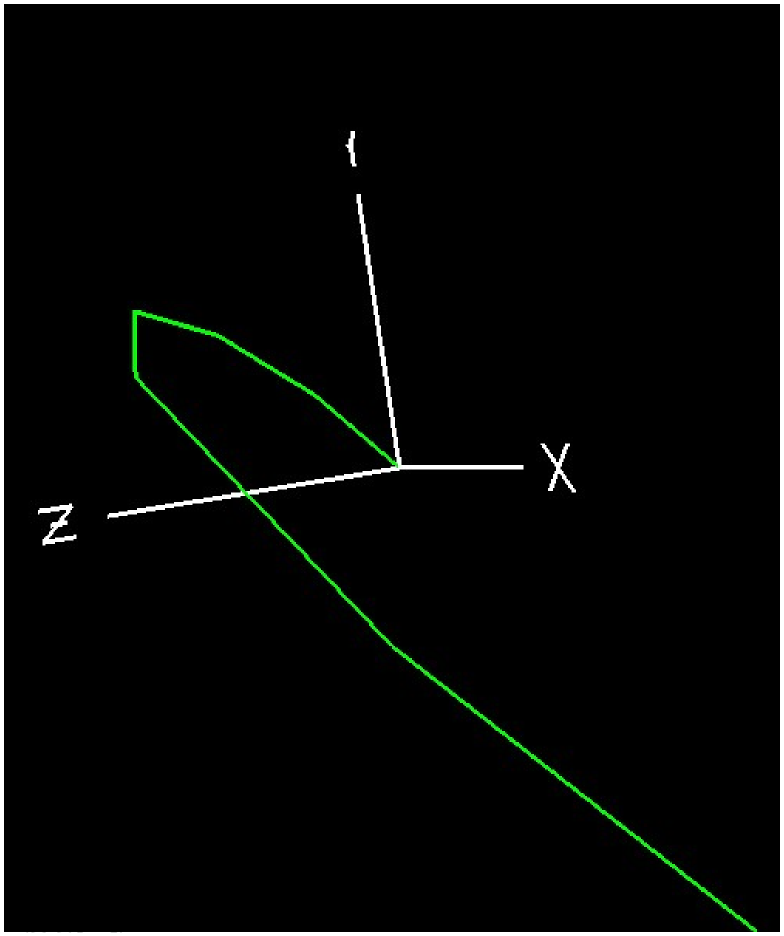
Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments

beziercurve.geom

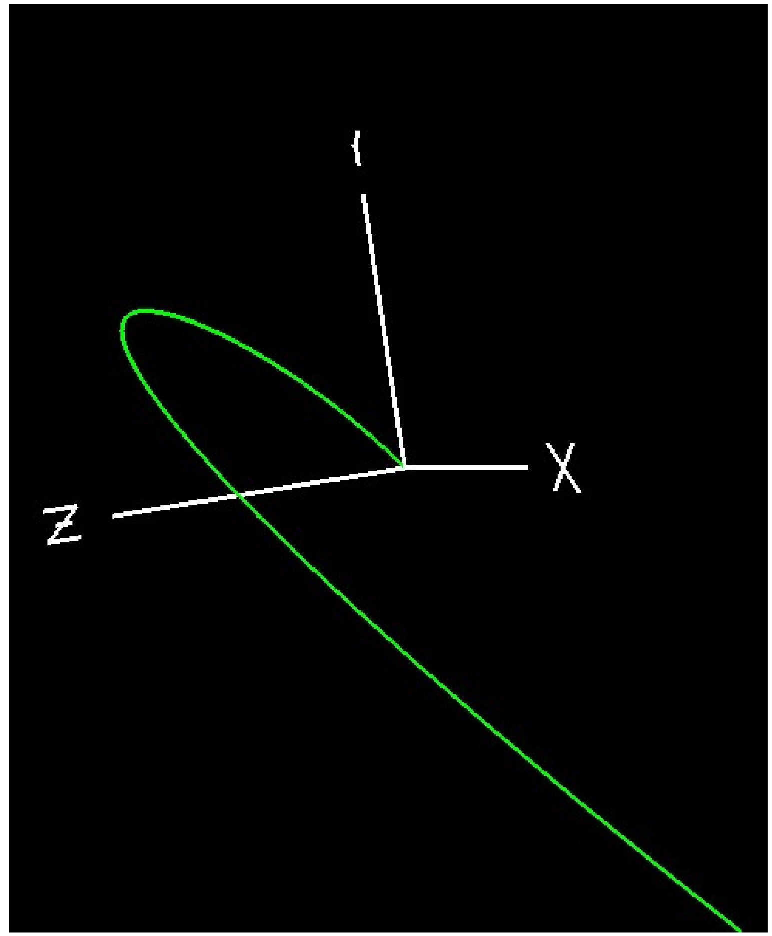
```
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( lines_adjacency )  in;
layout( line_strip, max_vertices=200 )  out;
uniform int uNum;
void
main( )
{
		float dt = 1. / float(uNum);
		float t = 0.;
		for( int i = 0; i <= uNum; i++ )
		{
			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 = gl_ProjectionMatrix * xyzw;
			EmitVertex( )
			t += dt;
		}
}
```

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

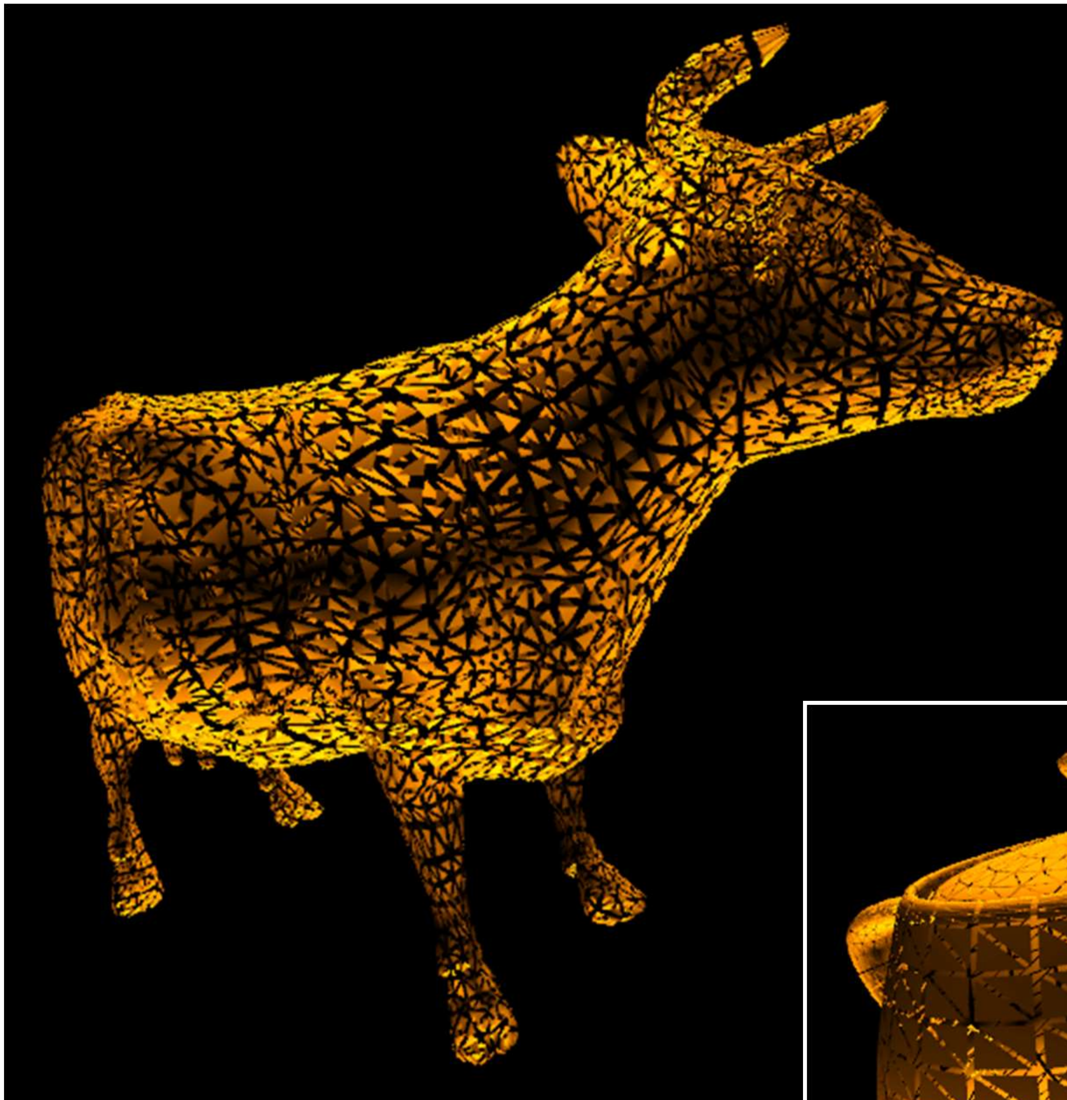


uNum = 5

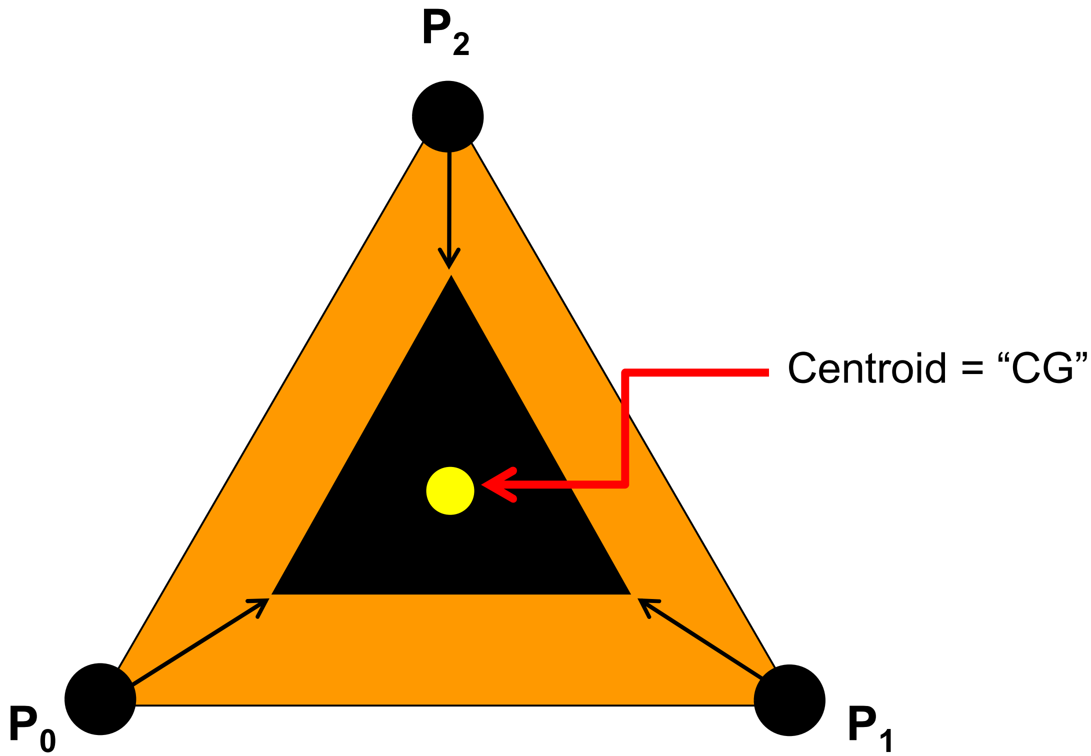


uNum = 25

Another Example: Shrinking Triangles



Example: Shrinking Triangles



$$CG = (P_0 + P_1 + P_2) / 3.;$$

$$P_0' = CG + uShrink * (P_0 - CG)$$
$$P_1' = CG + uShrink * (P_1 - CG)$$
$$P_2' = CG + uShrink * (P_2 - CG)$$

shrink.vert

```
#version 400 compatibility

out vec3 vNormal;

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

Not a typo – we are going to save the Projection Matrix multiply for the Geometry Shader!

shrink.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 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;
	V[2]  =  gl_PositionIn[2].xyz;
	CG = (  V[0] + V[1] + V[2]  ) / 3.;
	ProduceVertex( 0 );
	ProduceVertex( 1 );
	ProduceVertex( 2 );
}
```

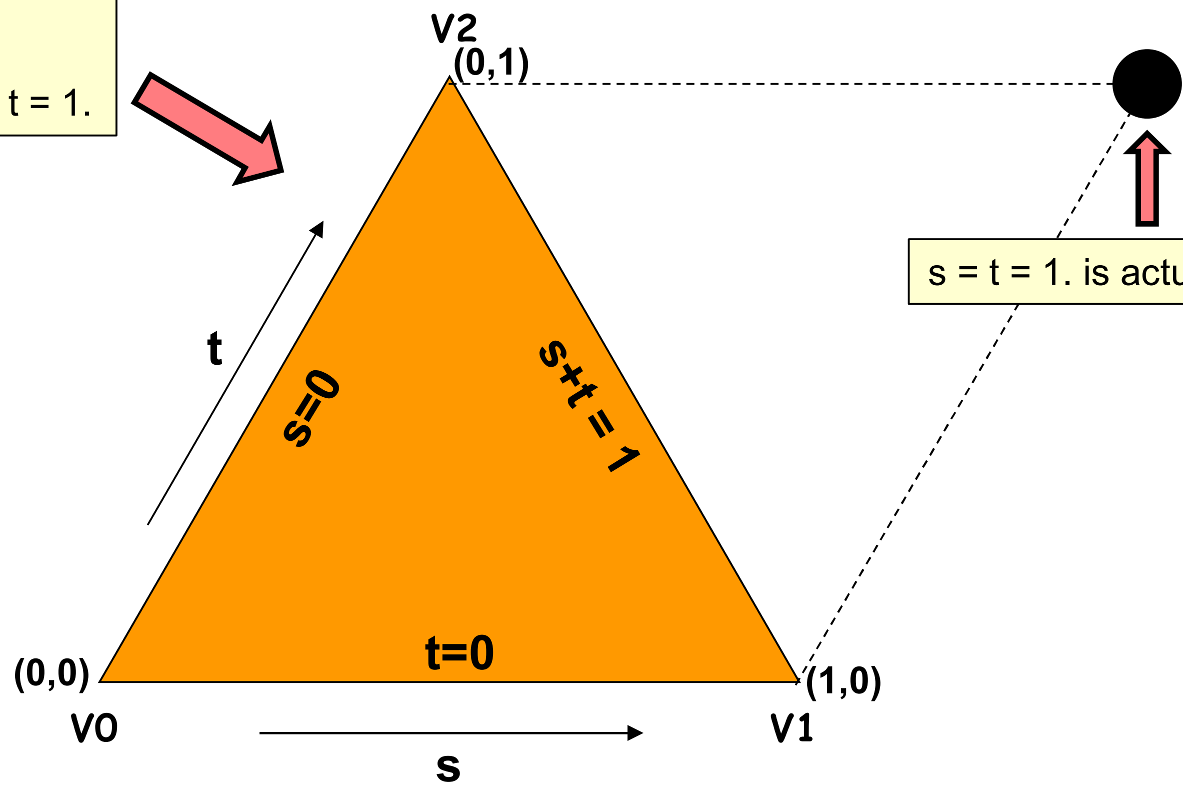
Since you are multiplying by the Projection matrix here, don't multiply by it in the vertex shader!

$CG = (P_0 + P_1 + P_2) / 3.;$
$P_0' = CG + uShrink * (P_0 - CG)$
$P_1' = CG + uShrink * (P_1 - CG)$
$P_2' = CG + uShrink * (P_2 - CG)$

Another Example: Sphere Subdivision

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

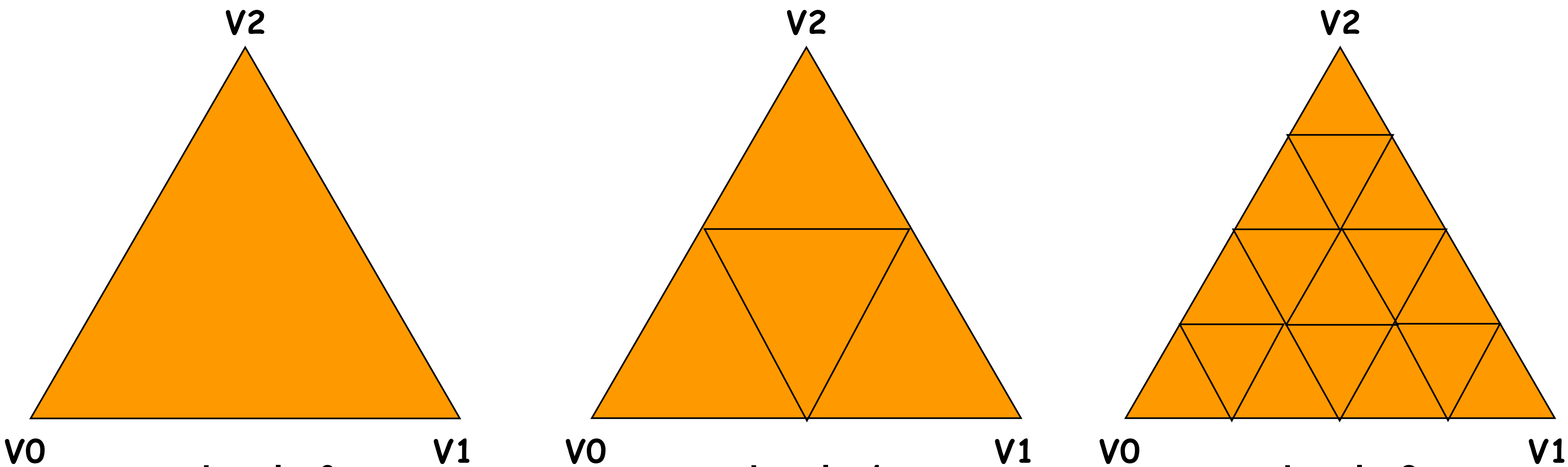
Note! There is no place *inside* this triangle where s = t = 1.



s = t = 1. is actually up here!

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

Example: Sphere Subdivision



uLevel = 0
numLayers = 2^{level} = 1

uLevel = 1
numLayers = 2

uLevel = 2
numLayers = 4

Example: Sphere Subdivision

spheresubd.glib

```
Vertex      spheresubd.vert
Geometry spheresubd.geom
Fragment spheresubd.frag
Program  SphereSubd  uLevel <0 0 10>  uRadius <.5 1. 5.>  uColor { 1. .5 .15 1. }


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_ModelViewMatrix * gl_Vertex;
}
```

spheresubd.frag

```
uniform vec4 uColor;
in float		gLightIntensity;

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

Example: Sphere Subdivision

spheresubd.geom, I

```
#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( );
}
```

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

Example: Sphere Subdivision

spheresubd.geom, II

```
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++ )
	{
		. . .
```

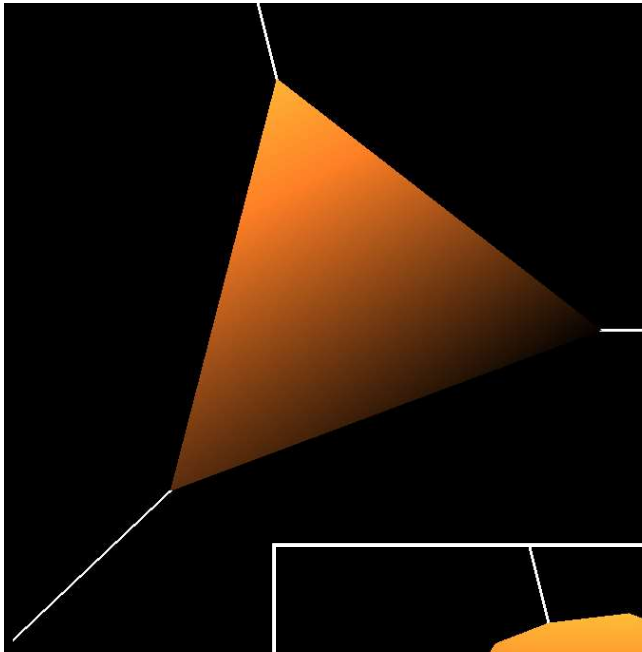
spheresubd.geom, III

Example: Sphere Subdivision

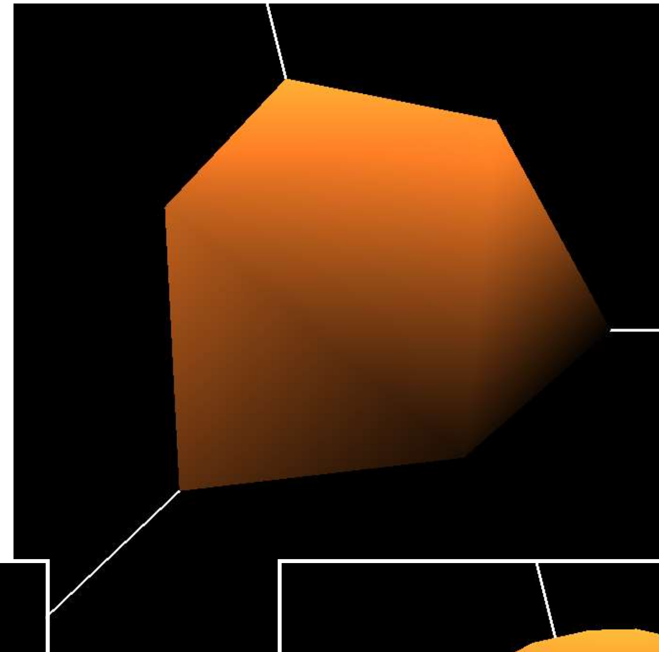
```
	for( int it = 0; it < numLayers; it++ )
	{
		float t_bot = t_top - dt;
		float smax_top = 1. - t_top;
		float smax_bot = 1. - t_bot;

		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

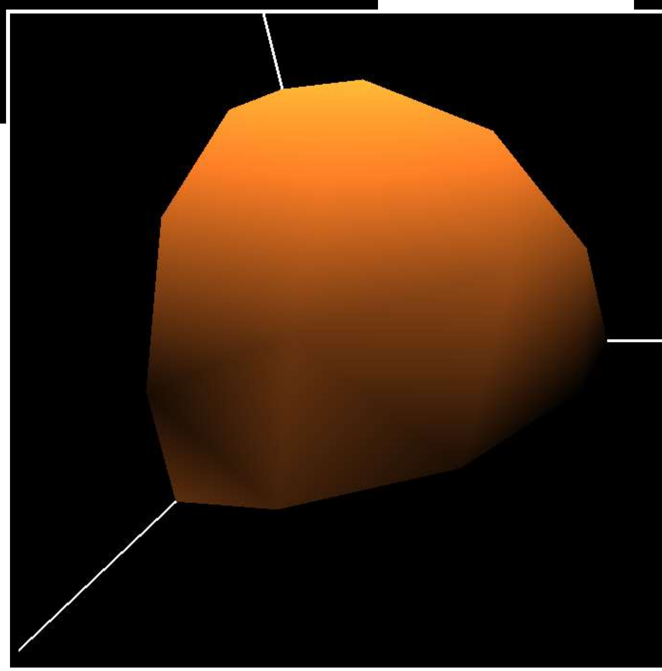
Level = 0



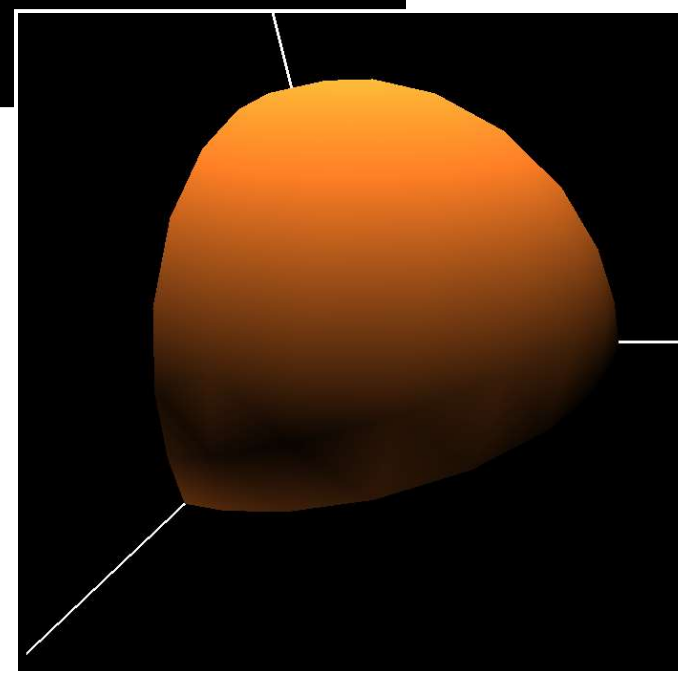
Level = 1



Level = 2

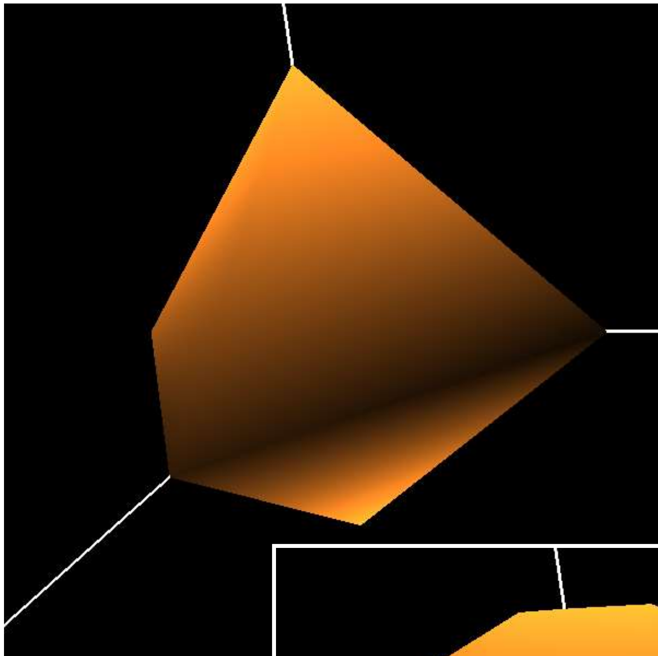


Level = 3

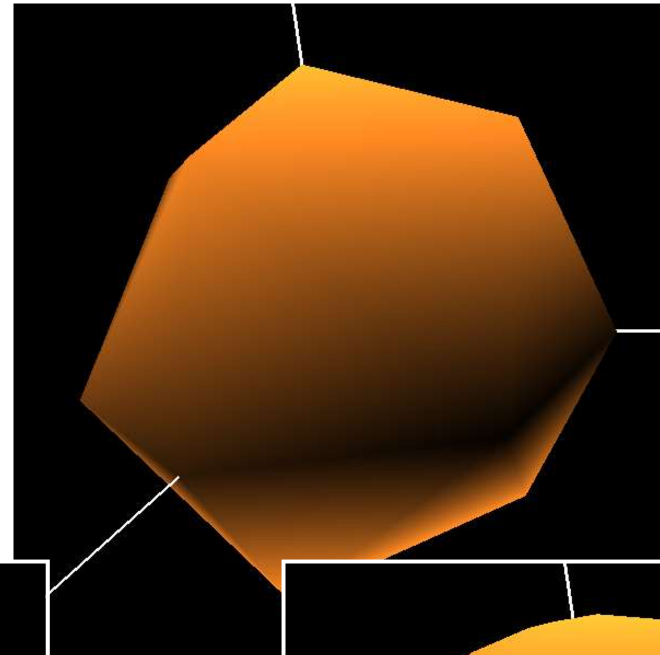


Example: Sphere Subdivision with the Whole Sphere (8 triangles)

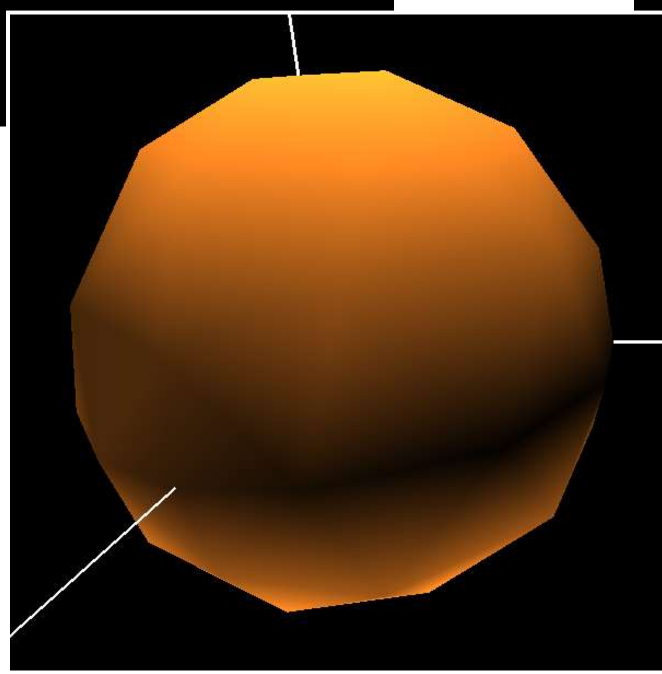
Level = 0



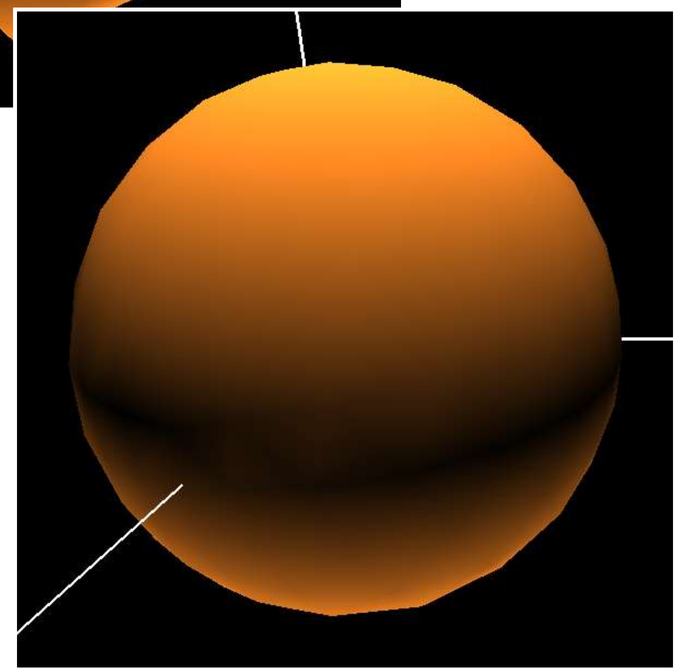
Level = 1



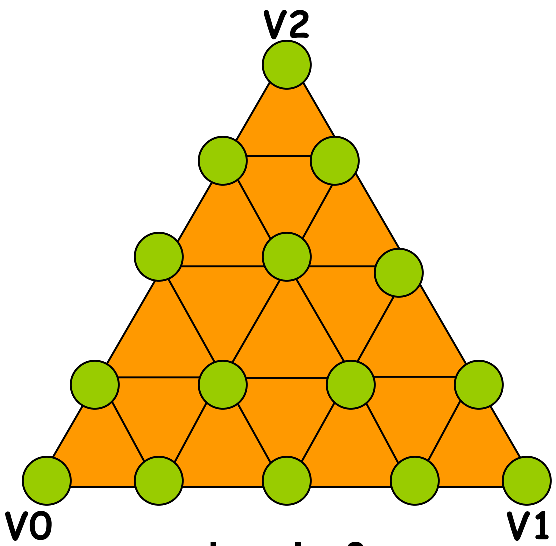
Level = 2



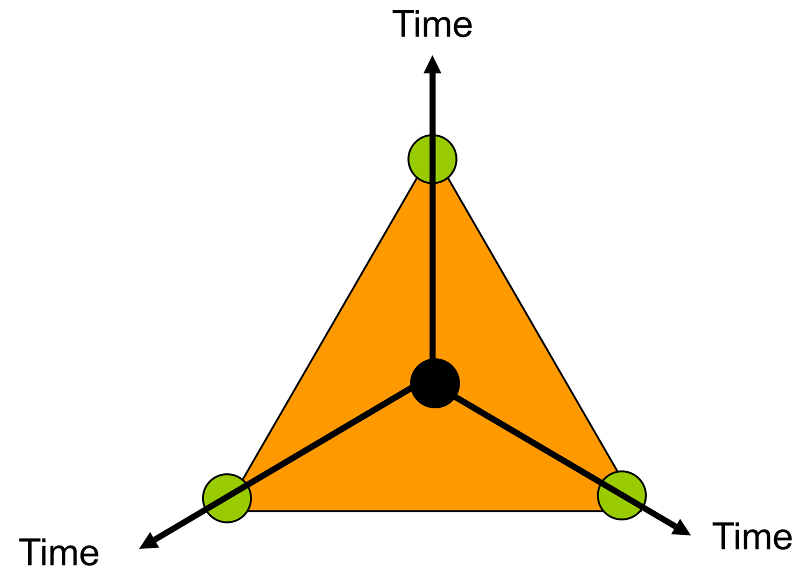
Level = 3



Another Example: Explosion



Level = 2
numLayers = 4



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

explode.vert

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

explode.geom, I

```
#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( );
}
```

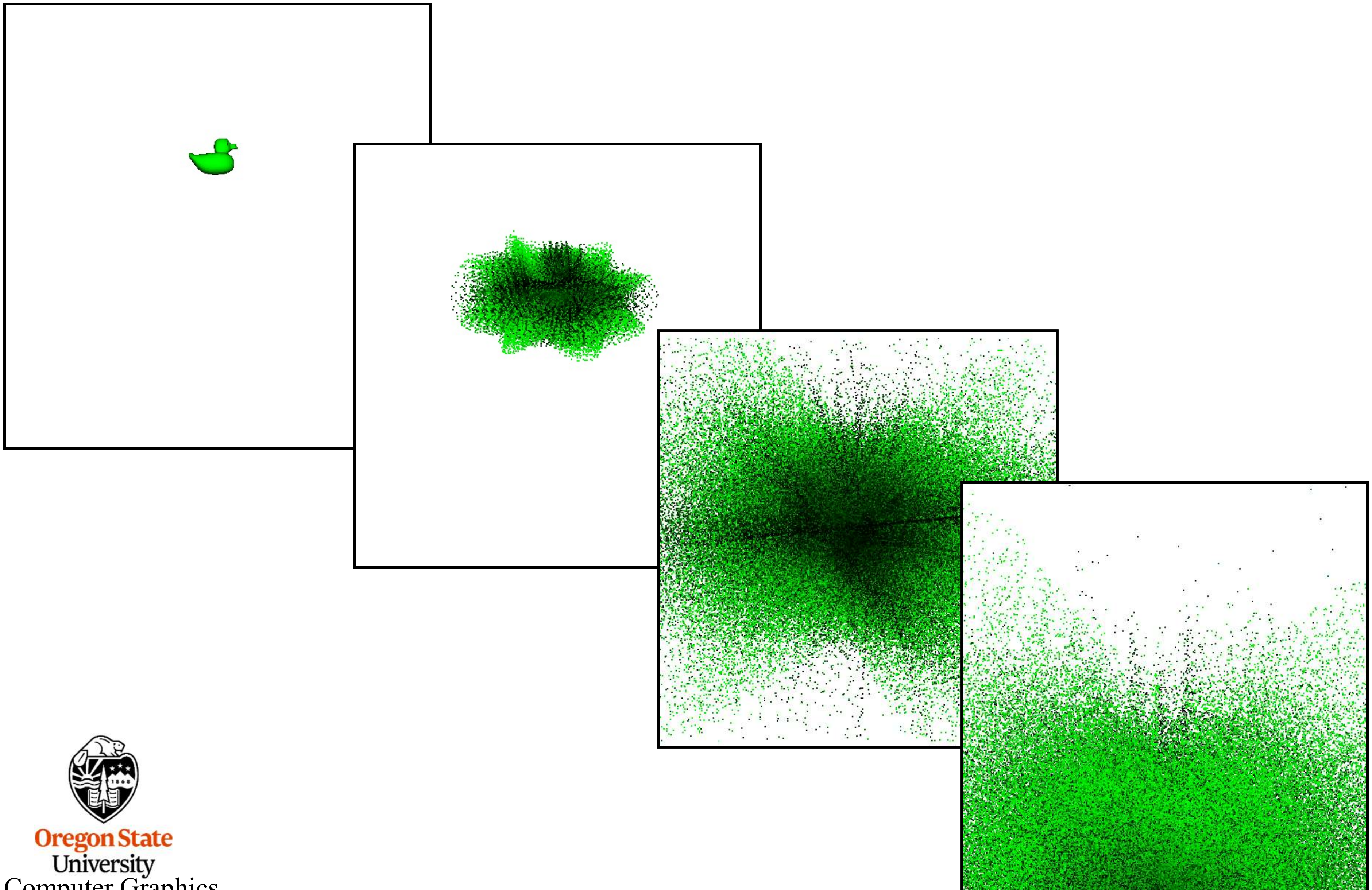
Since you are multiplying by the Projection matrix in the geometry shader, don't also multiply by it in the vertex shader!

explode.geom, II

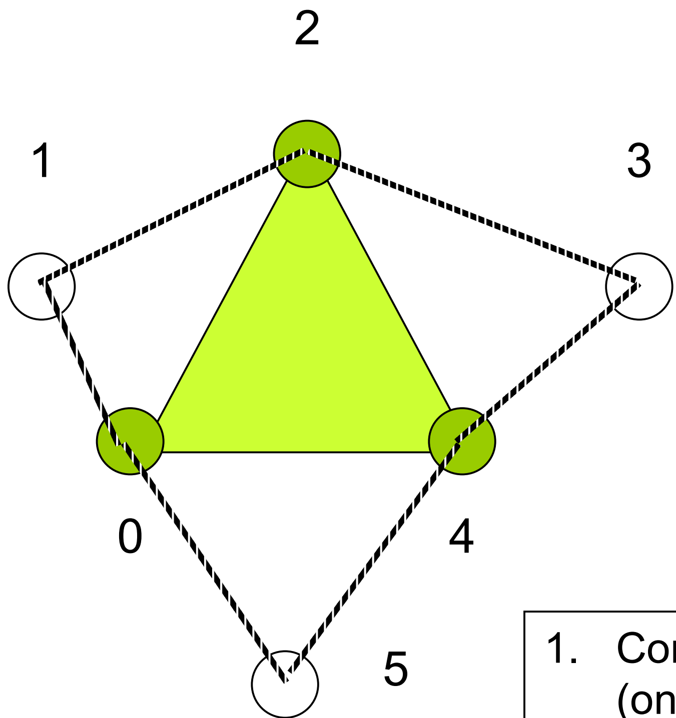
Example: Explosion

```
void
main( )
{
	V01 = ( gl_PositionIn[1] - gl_PositionIn[0] ).xyz;
	V02 = ( gl_PositionIn[2] - gl_PositionIn[0] ).xyz;
	V0  =   gl_PositionIn[0].xyz;
	CG = ( gl_PositionIn[0].xyz + gl_PositionIn[1].xyz + gl_PositionIn[2].xyz ) / 3.;

	int numLayers = 1 << uLevel;

	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++ )
		{
			ProduceVertex( s, t );
			s += ds;
		}

		t -= dt;
	}
}
```

Example: Explosion



Another 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*.

Example: Silhouettes

silh.glib

```
Obj bunny.obj

Vertex      silh.vert
Geometry silh.geom
Fragment  silh.frag
Program   Silhouette  uColor { 0. 1. 0. 1. }

ObjAdj   bunny.obj
```

Example: Silhouettes

silh.vert

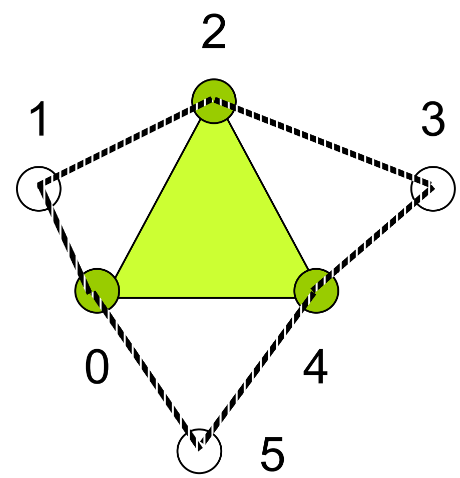
```
void
main( )
{
		gl_Position = gl_ModelViewMatrix * gl_Vertex;
}
```

silh.frag

```
uniform vec4 uColor;

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

silh.geom, I

Example: Silhouettes

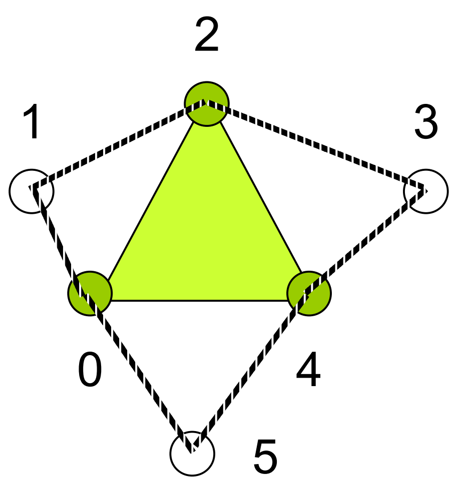



```
#version 330 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable
layout( triangles_adjacency ) in;
layout( line_strip, max_vertices=200 ) out;
void main( )
{
        vec3 V0 = gl_PositionIn[0].xyz;
        vec3 V1 = gl_PositionIn[1].xyz;
        vec3 V2 = 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
                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;
```

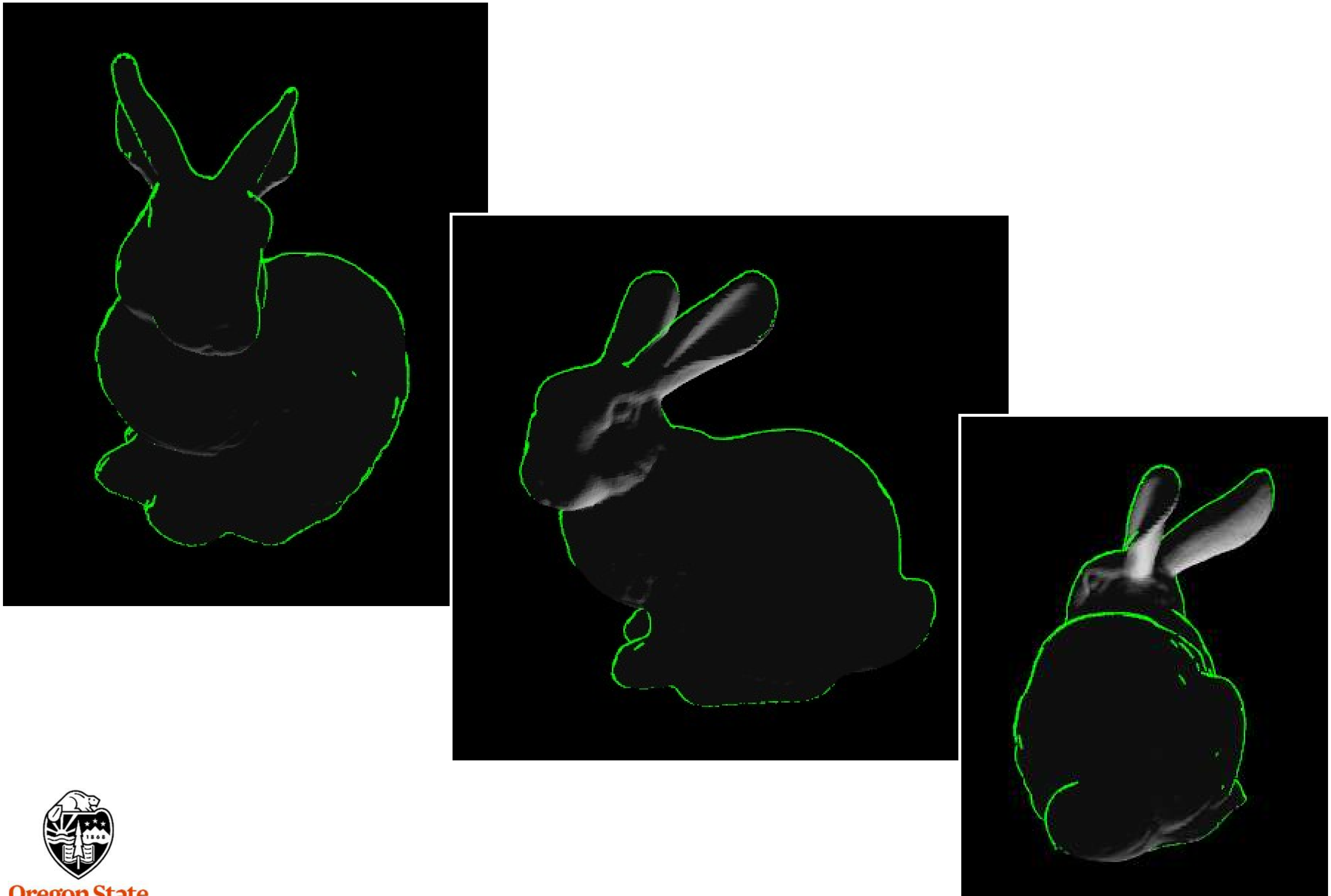
silh.geom, II

Example: Silhouettes

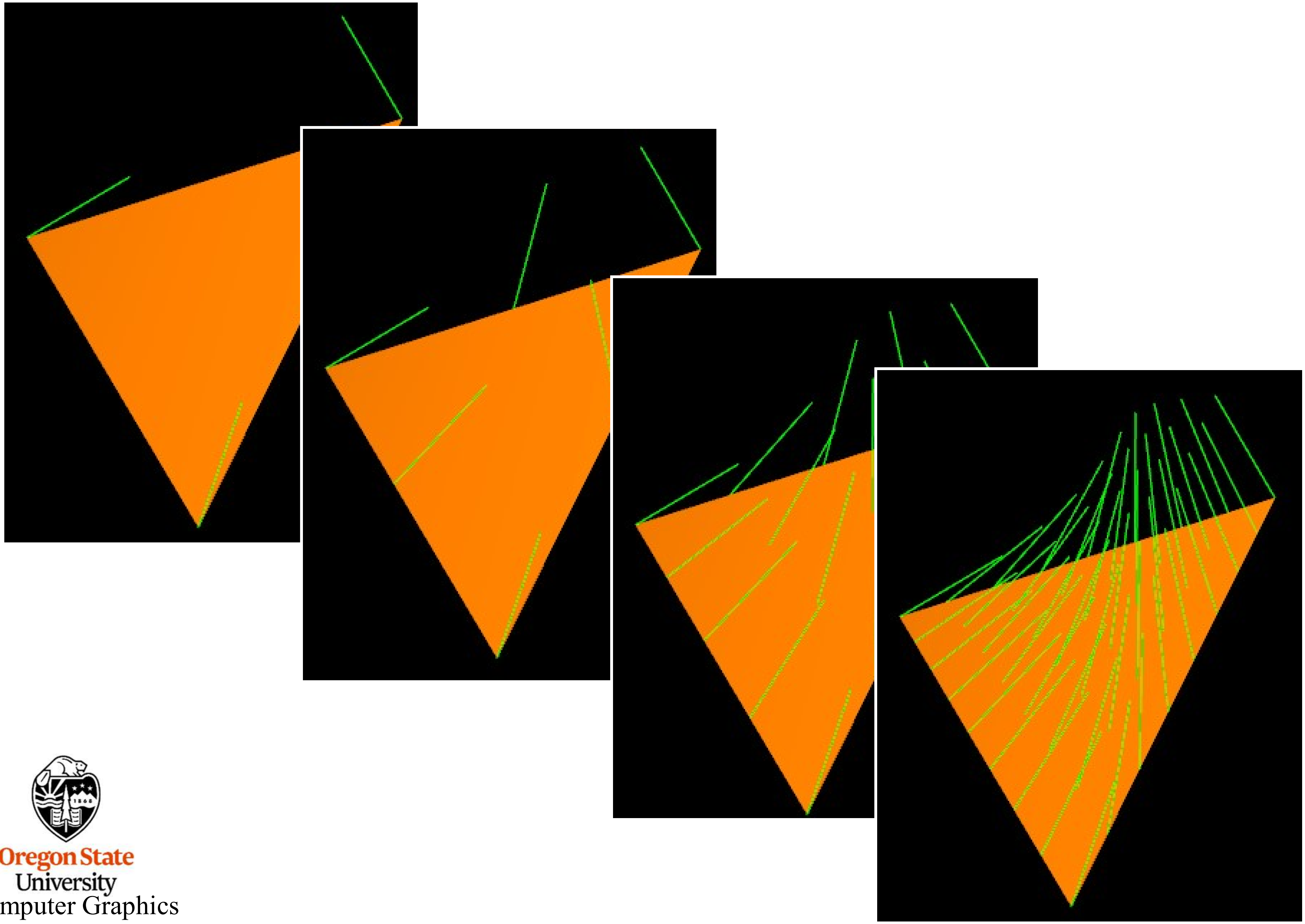
```
        if( N042.z * N021.z <= 0. )
        {
                gl_Position = gl_ProjectionMatrix * vec4( V0, 1. );
                EmitVertex( );
                gl_Position = gl_ProjectionMatrix * vec4( V2, 1. );
                EmitVertex( );
                EndPrimitive( );
        }

        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



Another Example: Hedgehog Plots



Oregon State University
Computer Graphics

hedgehog.geom, I

```
#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( );
}
```

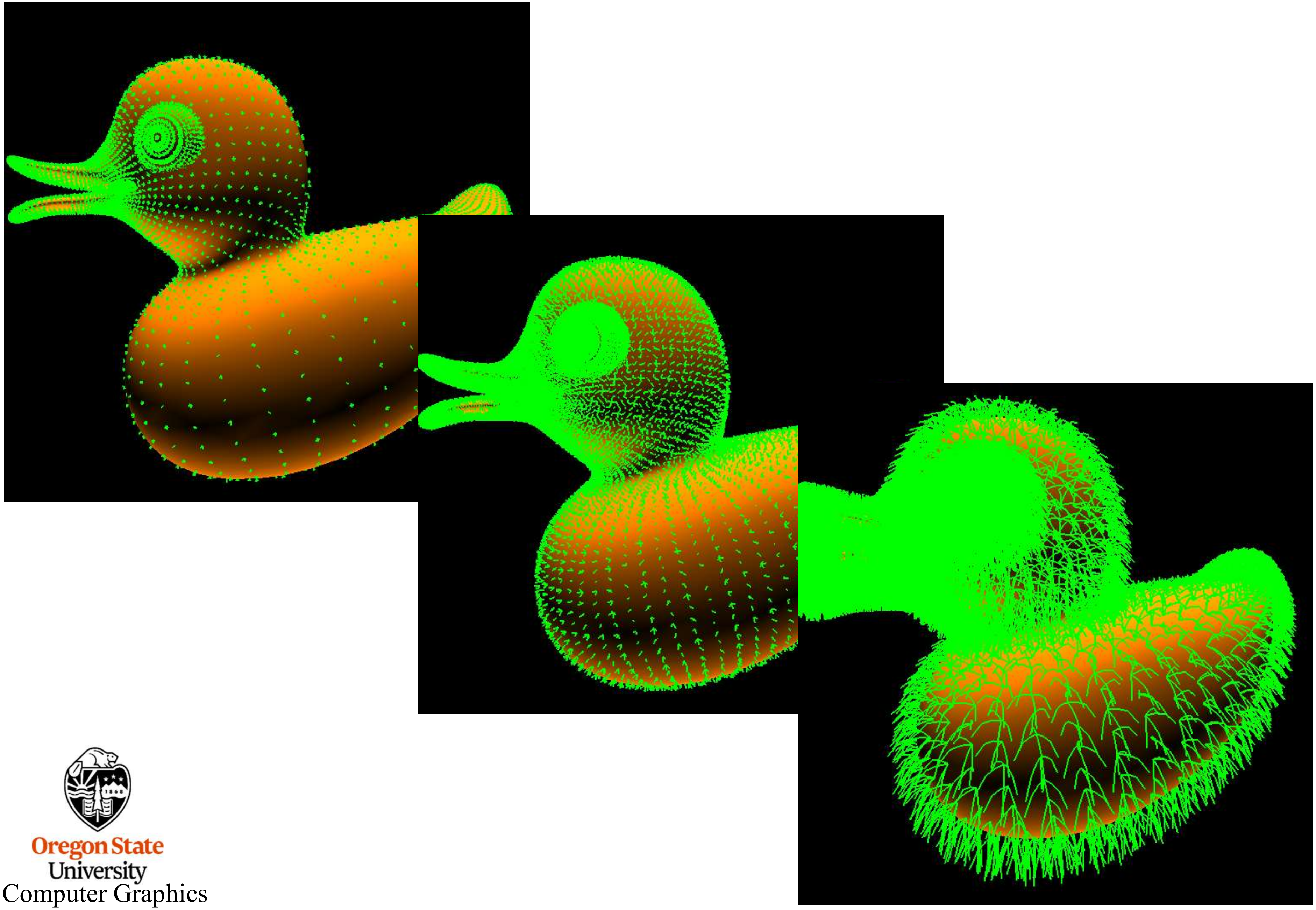
hedgehog.geom, II

```
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;
```

hedgehog.geom, III

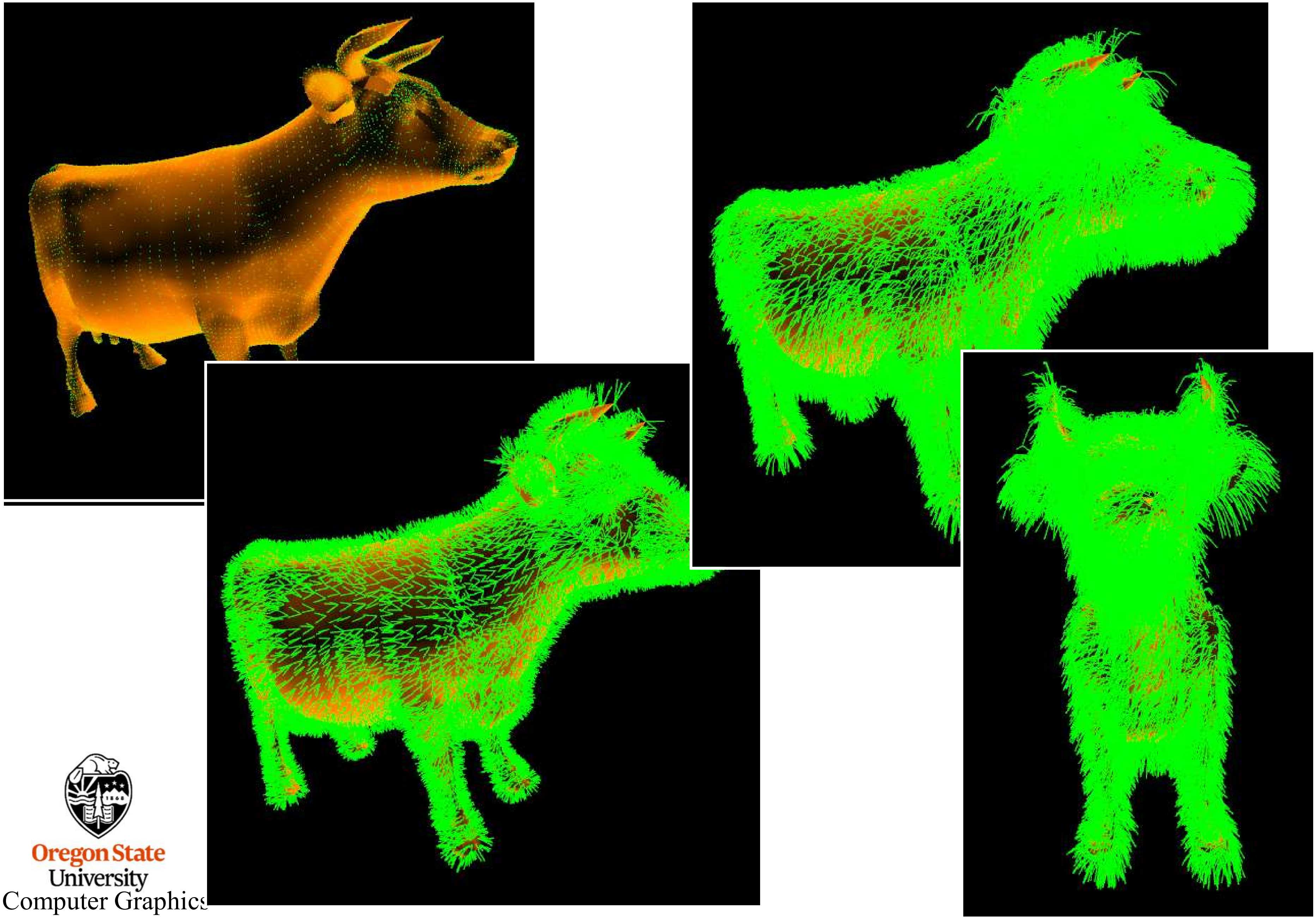
```
	float dt = 1. / float( numLayers );
	float t = 1.;
	for( int it = 0; it <= numLayers; it++ )
	{
		float smax = 1. - t;

		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;
	}
}
```

Ducky Hedgehog Plot



Oregon State
University
Computer Graphics

Hedgehog Plots Gone Wild



A 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 buffer drawing function counts as an implied glBegin( )**
- **gl_PrimitiveIDIn is 0 for the first primitive after the glBegin( )**

Geometry shaders can use 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?