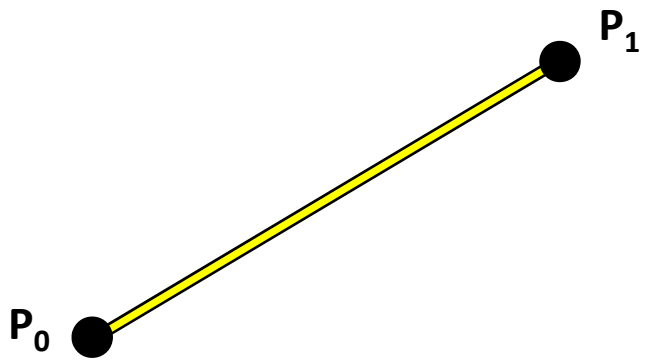
CS 457/557
Winter Quarter 2017

Project #7:

A Tessellated Parametric Triangular Bézier Patch

Due: March 15, 2017

A Parametric Line



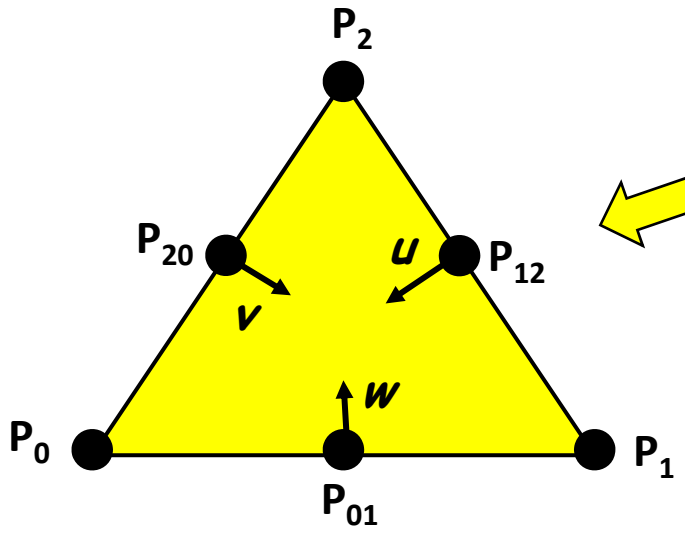
$$P = B_0 * P_0 + B_1 * P_1$$

$B_0 = (1-u)$
$B_1 = u$

} Blending Functions

A Parametric Triangular Bézier Patch

Warning: the directions of u, v, and w are different from what was in the Tessellation Shader notes. These match the paper I took this from. Sorry.



These are known as ***Barycentric Coordinates*** because u+v+w=1

$$P = B_0 * P_0 + B_1 * P_1 + B_2 * P_2 + B_{01} * P_{01} + B_{12} * P_{12} + B_{20} * P_{20}$$

$B_0 = u^2$

$B_1 = v^2$

$B_2 = w^2 = (1-u-v)^2$

$B_{01} = 2uv$

$B_{12} = 2vw = 2v(1-u-v)$

$B_{20} = 2wu = 2u(1-u-v)$

Let's get rid of the w's in these equations because w is not actually an independent coordinate. It's equal to 1.-u-v

Derivatives of the Blending Functions (we will use these to get the normal vectors)

$B_0 = u^2$

```
float db0du =  2.*u;
float db0dv =  0.;
```

$B_1 = v^2$

```
float db1du =  0.;
float db1dv =  2.*v;
```

$B_2 = (1-u-v)^2$

```
float db2du = -2.*(1.-u-v);
float db2dv = -2.*(1.-u-v);
```

$B_{01} = 2uv$

```
float db01du =  2.*v;
float db01dv =  2.*u;
```

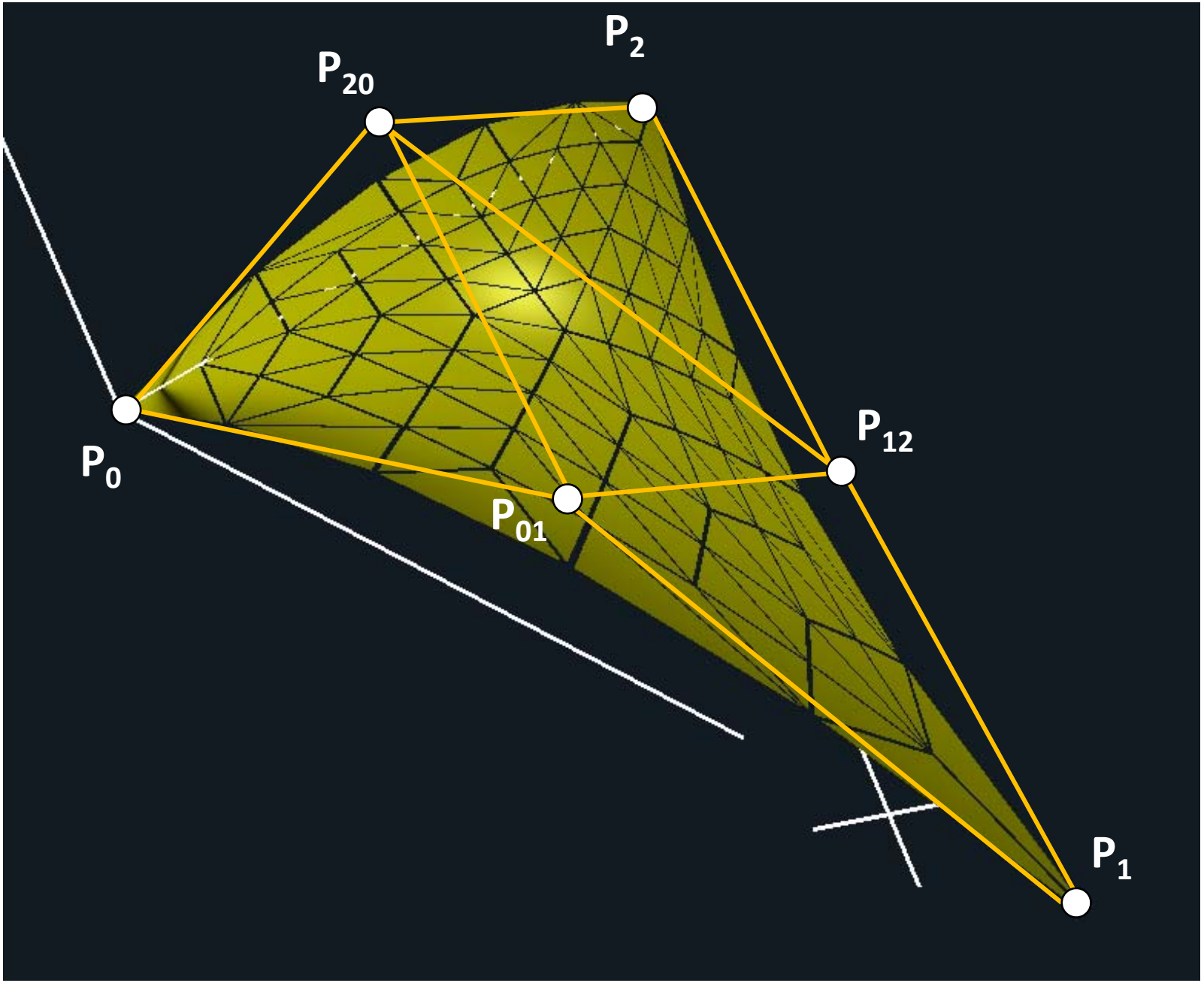
$B_{12} = 2v(1-u-v) = 2.*(v-uv-v^2)$

```
float db12du = -2.*v;
float db12dv =  2.*(1.-u-2.*v);
```

$B_{20} = 2u(1-u-v) = 2.*(u-u^2-uv)$

```
float db20du =  2.*(1.-2.*u-v);
float db20dv = -2.*u;
```

A Parametric Triangular Bézier Patch



Project Requirements

1. Draw a Bézier triangle surface patch, defined by 3 corner vertices.

2. Define the 3 control point vertices midway along the edges between corner vertices.

3. Have the 3 control point Z values, Z_{01}, Z_{12}, Z_{20} controlled by sliders. **[20]**

4. Tessellate the triangle with a tessellation shader.

5. Have the outer and inner densities controlled by sliders. **[20]**

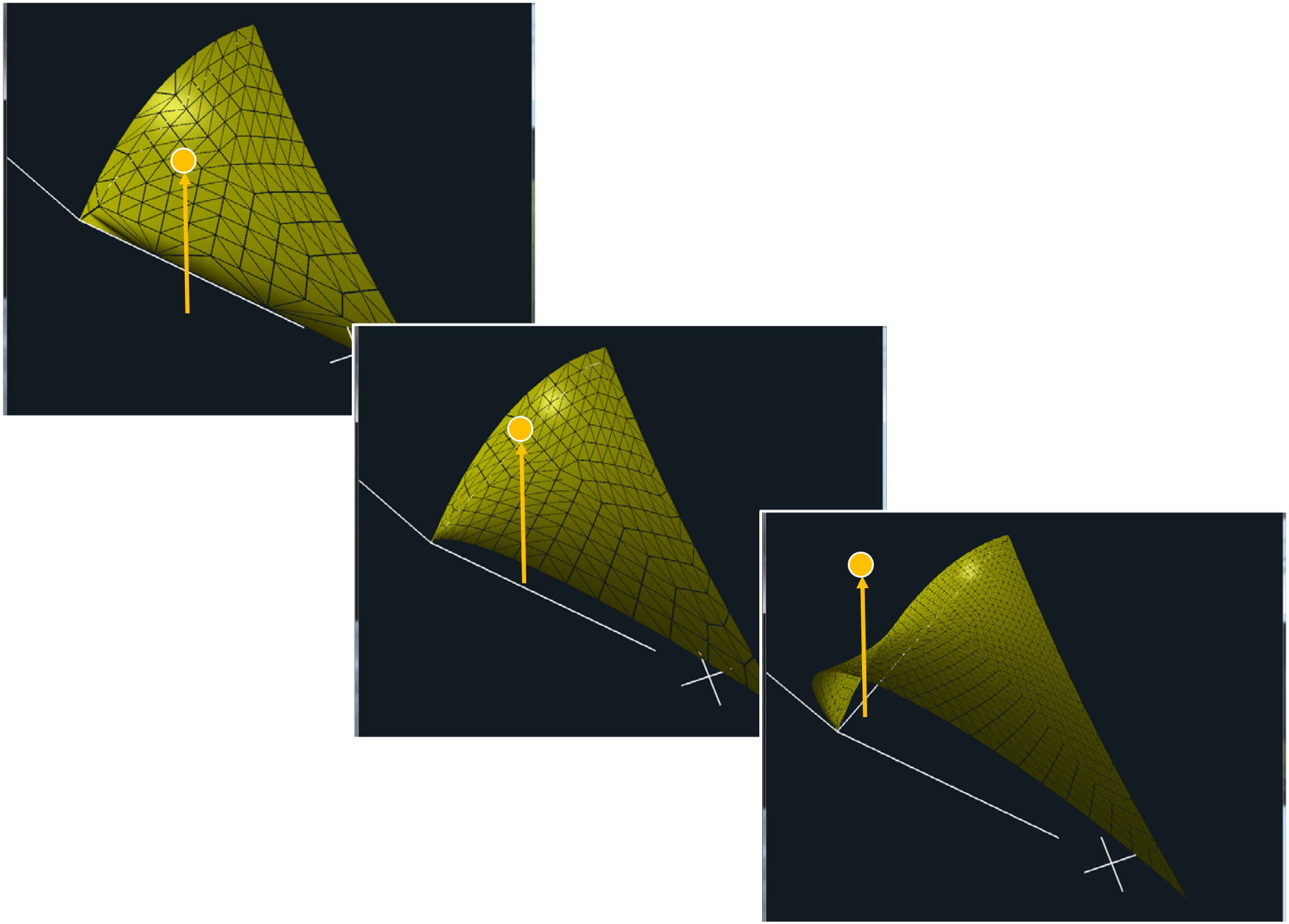
6. Under checkbox control, have the outer and inner densities adapt to changes in the geometry. How you choose to do this is up to you, but pick something that makes sense. **[30]**

7. Do some sort of lighting. This means you need to compute normals. **[20]**

8. Include the *shrink* geometry shader so you can show that your tessellation works. **[10]**

9. Your PDF report must show that all of these features work!

Adapting to Changes in Geometry



Glib File

```
##OpenGL GLIB
Perspective 70
LookAt 2 -1 1  0 2 -1   0 0 1

Vertex             mtriangle.vert
TessControl     mtriangle.tcs
TessEvaluation mtriangle.tes
Geometry         mtriangle.geom
Fragment         mtriangle.frag
Program MidTriangle                                                    \
                        uOuter01 <???>                                 \
                        uOuter12 <???>                                 \
                        uOuter20 <???>                                 \
                        uInner <???>                                   \
                        uZ01 <???.>                                    \
                        uZ12 <???.>                                    \
                        uZ20 <???>                                     \
                        uAdaptToZs <false>                             \
                        uShrink <0. 0.9 1.>                            \
                        uKa <0. 0.1 1.0>                               \
                        uKd <0. 0.7 1.0>                               \
                        uKs <0. 0.2 1.0>                               \
                        uShininess <3. 10. 1000.>                      \
                        uLightX <-10. 0. 10.>   uLightY <-10. 8. 10.>   uLightZ <-10. 8. 10.>

Color 1. .5 0.
NumPatchVertices 3
glBegin gl_patches
   glVertex 0. 0. 0.
   glVertex 2. 0. 0.
   glVertex 0. 2. 0.
glEnd
```

Uniform Variables

Vertex Shader


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

Tessellation Control Shader

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

uniform float       uZ01, uZ12, uZ20;
uniform int  uOuter01, uOuter12, uOuter20, uInner;
uniform bool        uAdaptToZs;

layout( vertices = 3 )  out;

void
main( )
{
        gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;

        if( uAdaptToZs )
        {
                gl_TessLevelOuter[0] = ???
                gl_TessLevelOuter[1] = ???
                gl_TessLevelOuter[2] = ???
                gl_TessLevelInner[0]  = gl_TessLevelInner[1]  = ???;
        }
        else
        {
                gl_TessLevelOuter[0] = float(uOuter12);
                gl_TessLevelOuter[1] = float(uOuter20);
                gl_TessLevelOuter[2] = float(uOuter01);
                gl_TessLevelInner[0]  = gl_TessLevelInner[1]  = float(uInner);
        }
}
```

Tessellation Evaluation Shader, I

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable
layout( triangles, equal_spacing, ccw)  in;

uniform float  uZ01, uZ12, uZ20;

out vec3  teNormal;
out vec3  teECposition;

void
main( )
{
        vec4 p0 = gl_in[0].gl_Position;
        vec4 p1 = gl_in[1].gl_Position;
        vec4 p2 = gl_in[2].gl_Position;
        vec4 p3 = gl_in[3].gl_Position;

        vec4 p01 = ???
        vec4 p12 = ???
        vec4 p20 = ???

        float u = gl_TessCoord.x;
        float v = gl_TessCoord.y;
        float w = gl_TessCoord.z;

        float b0 = ???;
        float b1 = ???;
        float b2 = ???;
        float b01 = ???;
        float b12 = ???;
        float b20 = ???;
```

Tessellation Evaluation Shader, II

```
	float db0du =  2.*u;
	float db0dv =  0.;

	float db1du =  0.;
	float db1dv =  2.*v;

	float db2du = -2.*(1.-u-v);
	float db2dv = -2.*(1.-u-v);

	float db01du =  2.*v;
	float db01dv =  2.*u;

	float db12du = -2.*v;
	float db12dv =  2.*(1.-u-2.*v);

	float db20du =  2.*(1.-2.*u-v);
	float db20dv = -2.*u;

	teECposition = ( gl_ModelViewMatrix * ( b0*p0 + b01*p01 + b1*p1 + b12*p12 + b2*p2 + b20*p20 ) ).xyz;
	gl_Position = vec4( teECposition, 1. );

	vec4 dpdu   = db0du*p0 + db01du*p01 + db1du*p1 + db12du*p12 + db2du*p2 + db20du*p20;
	vec4 dpdv   = db0dv*p0 + db01dv*p01 + db1dv*p1  + db12dv*p12 + db2dv*p2 + db20dv*p20;

	teNormal = gl_NormalMatrix * normalize( cross( dpdu.xyz, dpdv.xyz ) );
}
```

Geometry Shader, I


```
#version 400 compatibility
#extension GL_EXT_gpu_shader4: enable
#extension GL_EXT_geometry_shader4: enable

layout( triangles ) in;
layout( triangle_strip, max_vertices=32 ) out;

uniform float uShrink;
uniform float uLightX, uLightY, uLightZ;

in vec3		teECposition[3];
in vec3		teNormal[3];

out vec3 gNs;
out vec3 gLs;
out vec3 gEs;

vec3 LightPos = vec3( uLightX, uLightY, uLightZ );

vec3 V[3];
vec3 CG;
```

Geometry Shader, II


```
void
ProduceVertex( int v )
{
	gNs = teNormal[v];
	gLs = LightPos - teECposition[v];
	gEs = vec3(0.,0.,0.) - teECposition[v];

	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 );
}
```

Fragment Shader


```
#version 400 compatibility

in vec3 gNs;
in vec3 gLs;
in vec3 gEs;

uniform float uKa, uKd, uKs;
uniform float uShininess;

void
main( )
{
	???

	gl_FragColor = ???;
}
```