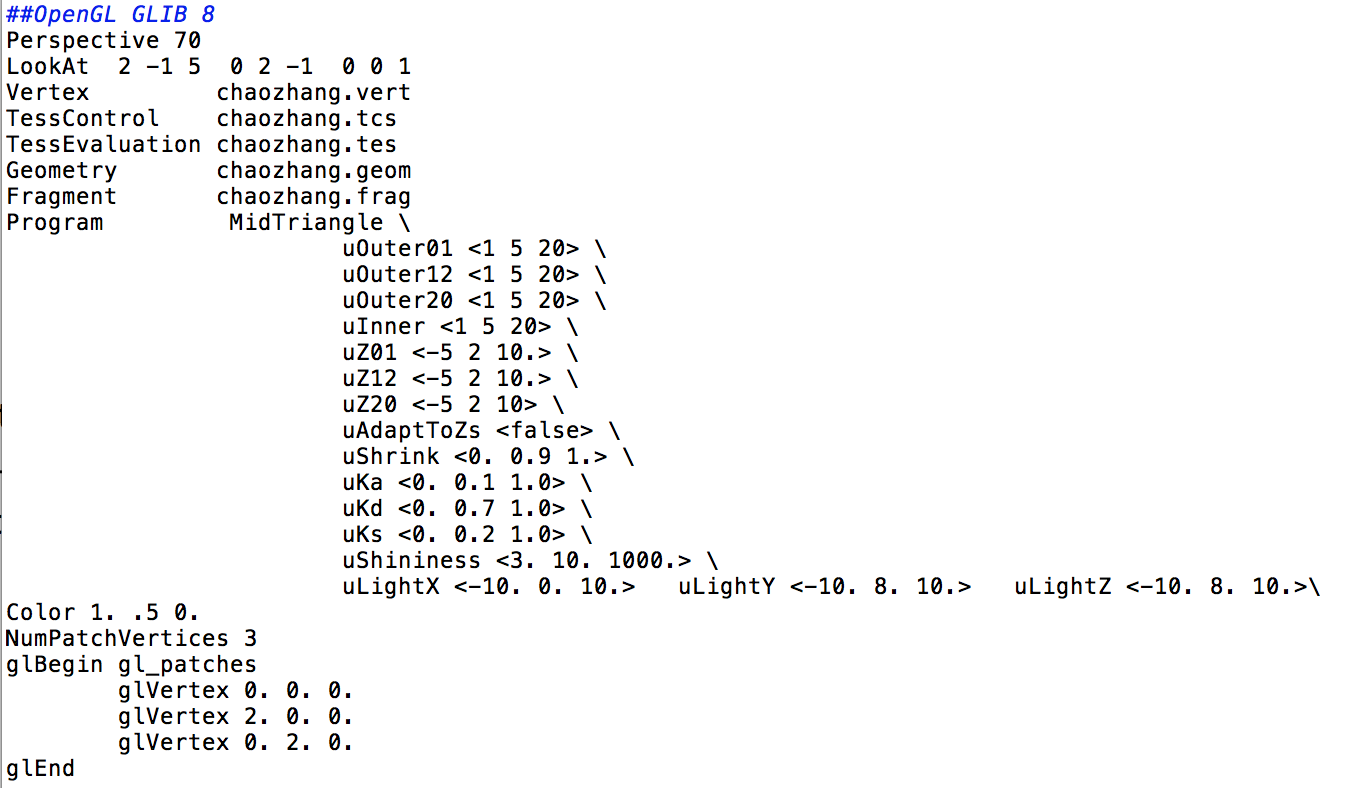
***RenderMan and OpenGL Shaders***

CS557

Project # 7

Chao Zhang

1. Source listings



chaozhang.glib

This file includes all the definition I need to use. The uOuter01, uOuter12 and uOuter12uOuter20 will change the number of the edges in the side, and the default is 5. With increase of those numbers, the side will look more smooth. The inner is the control value of the inside edges. uZ01, uZ12 and uZ20 are the control point’s value on the Z coordinate of each side, so it can be negative. The shrink will change the size of the tessellation.

glBegin gl\_patches

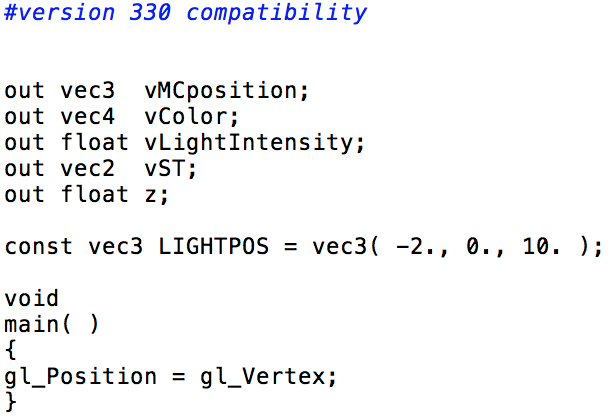
glVertex 0. 0. 0.

glVertex 2. 0. 0.

glVertex 0. 2. 0.

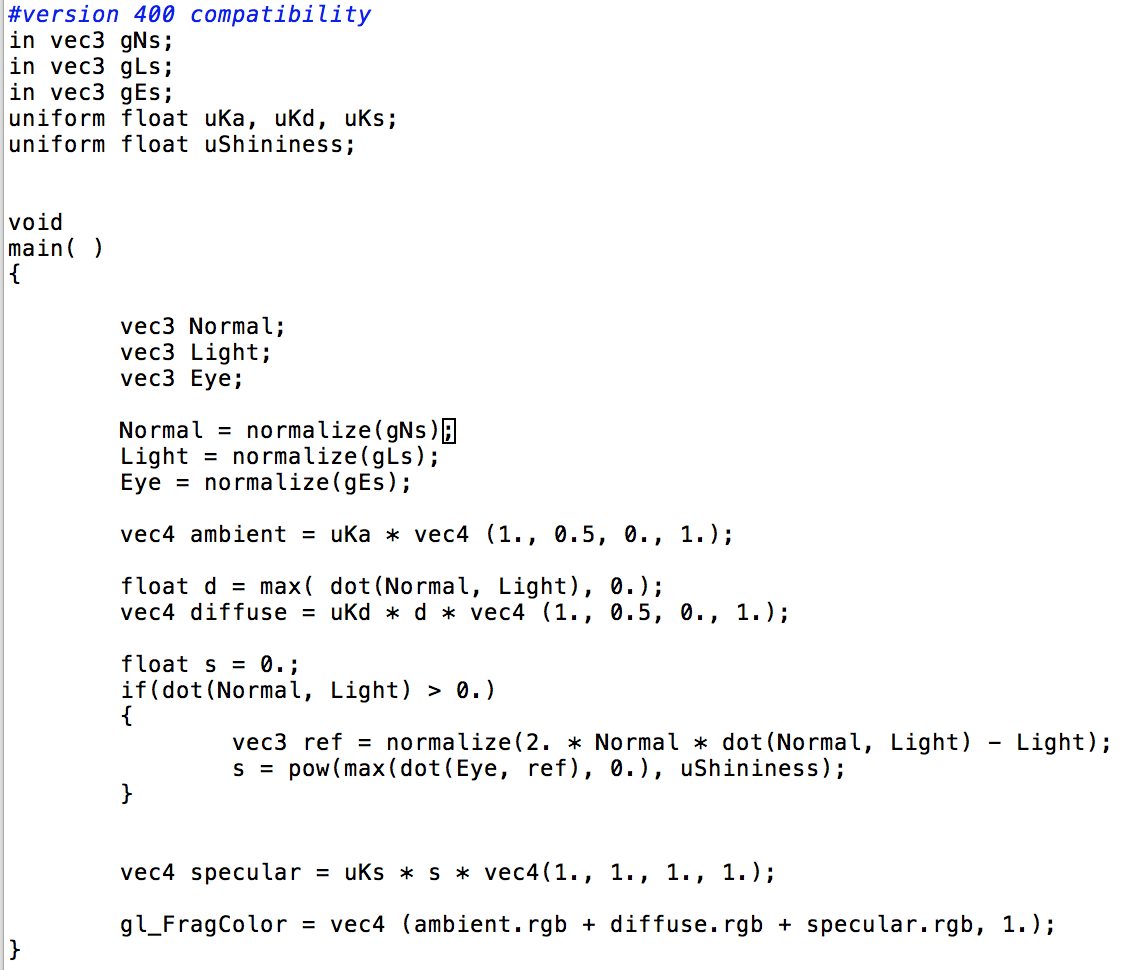
glEnd

those are the three corner vertices.



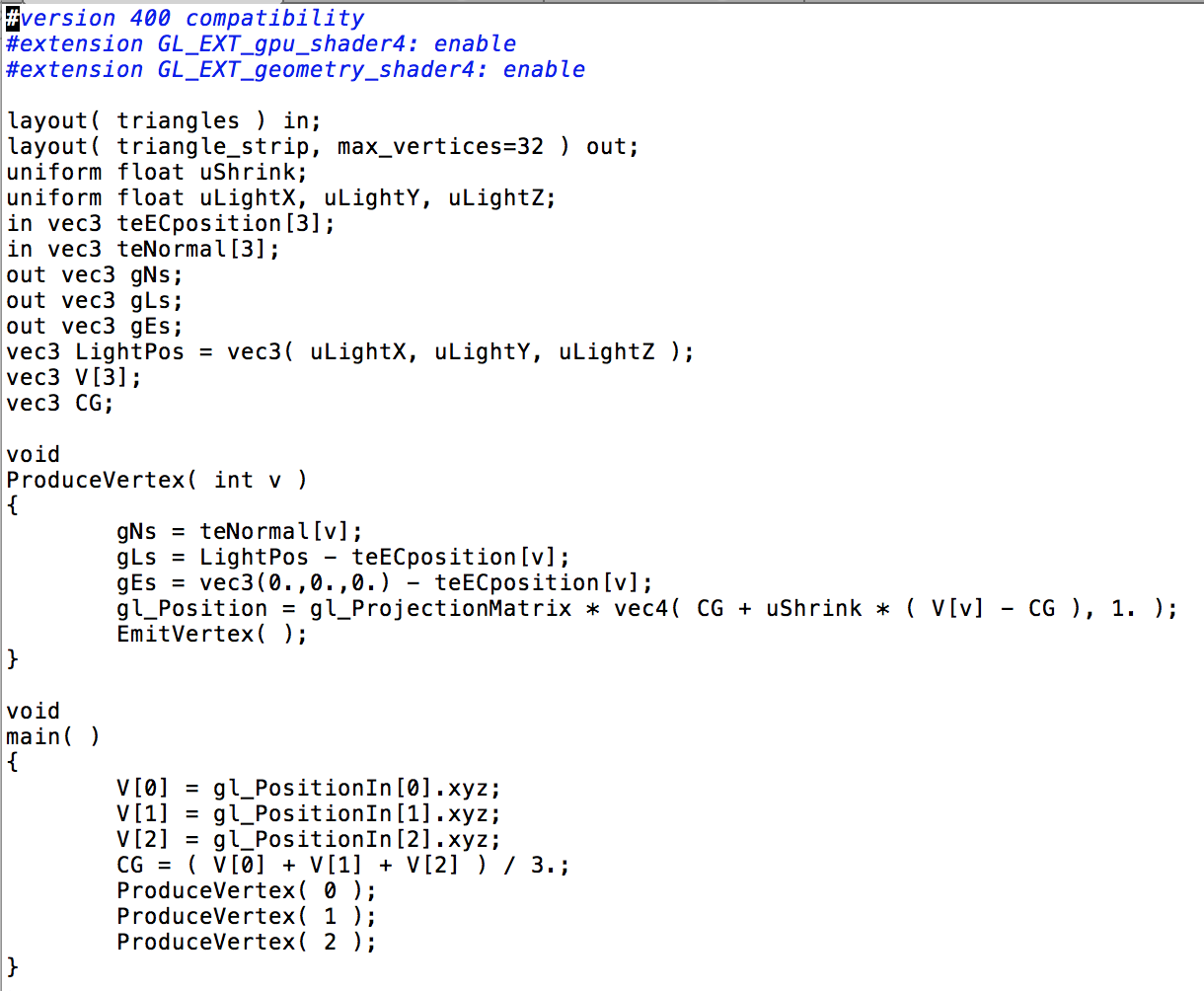
chaozhang.vert

only the gl\_position included.



chaozhang.freq

This is the .freq file. This fragment shader is used to compute the normal and use the lighting to show it is right.



chaozhang.goem

This is a new shader I used first time. This shader will control the geometry.

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( );

}

Those lines will produce the vertices, that is how the ushrink works. The geometry shader is kind of more powerful vertex shader.

V[0] = gl\_PositionIn[0].xyz;

V[1] = gl\_PositionIn[1].xyz;

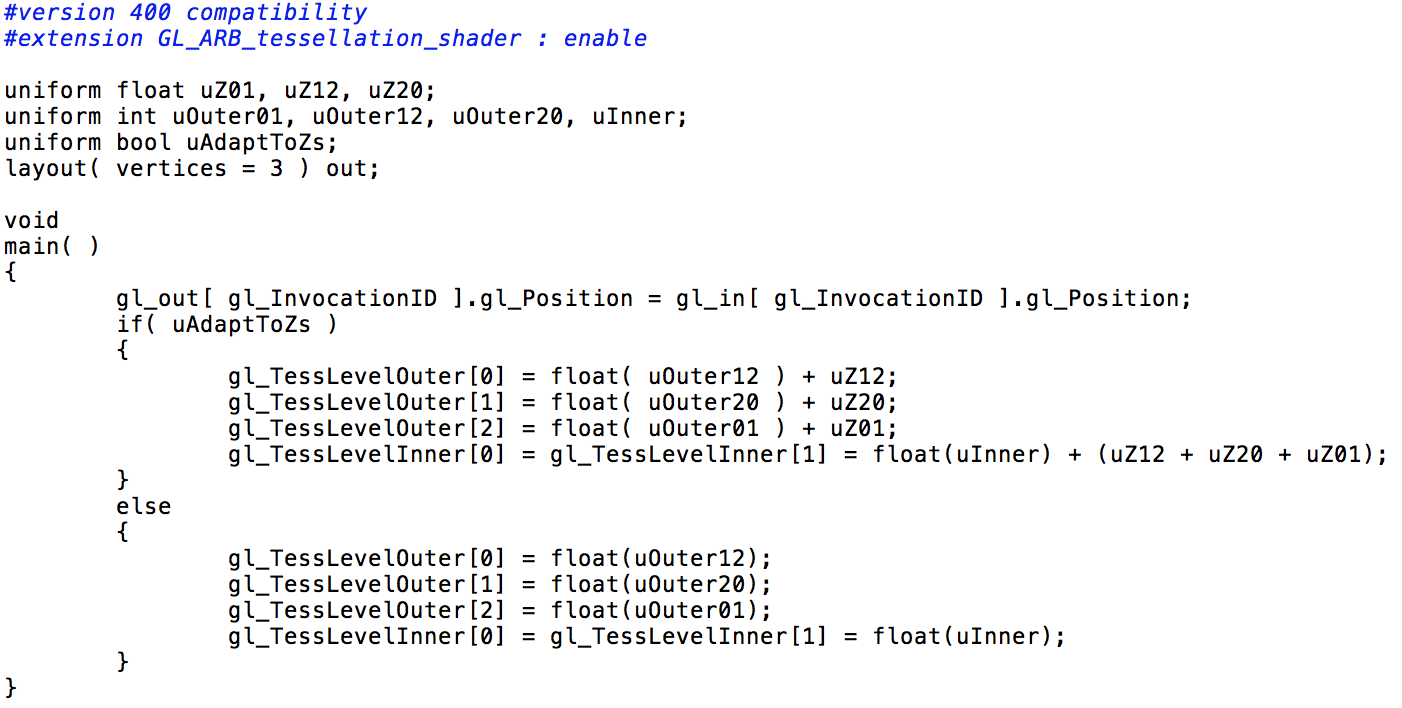
V[2] = gl\_PositionIn[2].xyz;

CG = ( V[0] + V[1] + V[2] ) / 3.;

ProduceVertex( 0 );

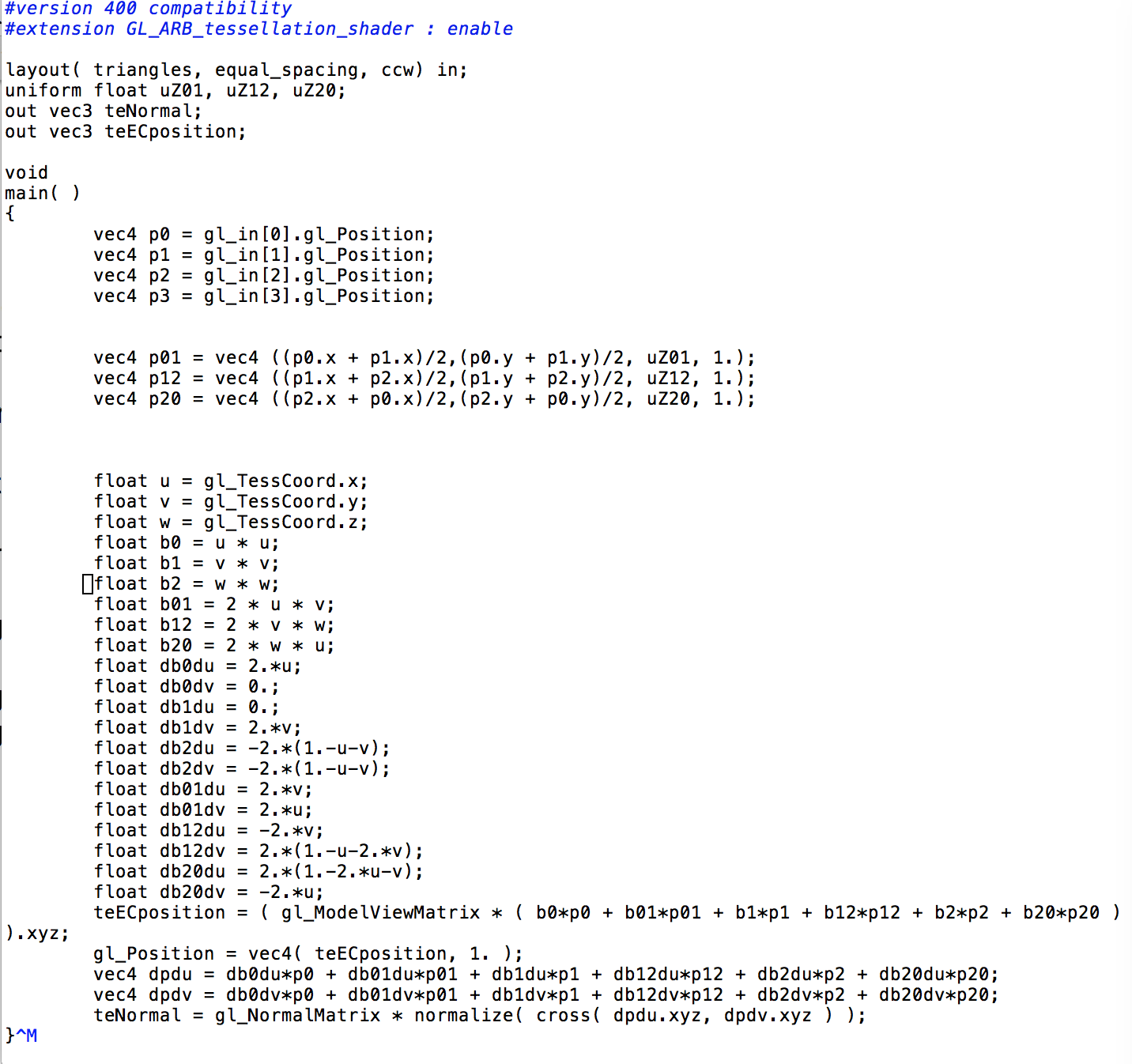
ProduceVertex( 1 );

ProduceVertex( 2 ); Those will finish the produce.



chaozhang.tcs

This is the tessellation control shader. In this shader I can define the control point to the tessellation. If the uAdaptToZs is true, the outer will add the uZ value and the same to the inner. Else, the outer will be the value of the uOuter.



chaozhang.tes

This is the tessellation evaluation shader. In this shader, I will computing all the vertices position.

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; Those are the gl\_positions.

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 ) );

}

Those are use the position are computed in the previous equation to get the teECposition and teNormal. With those the whole project will work correctly.

1. Results

