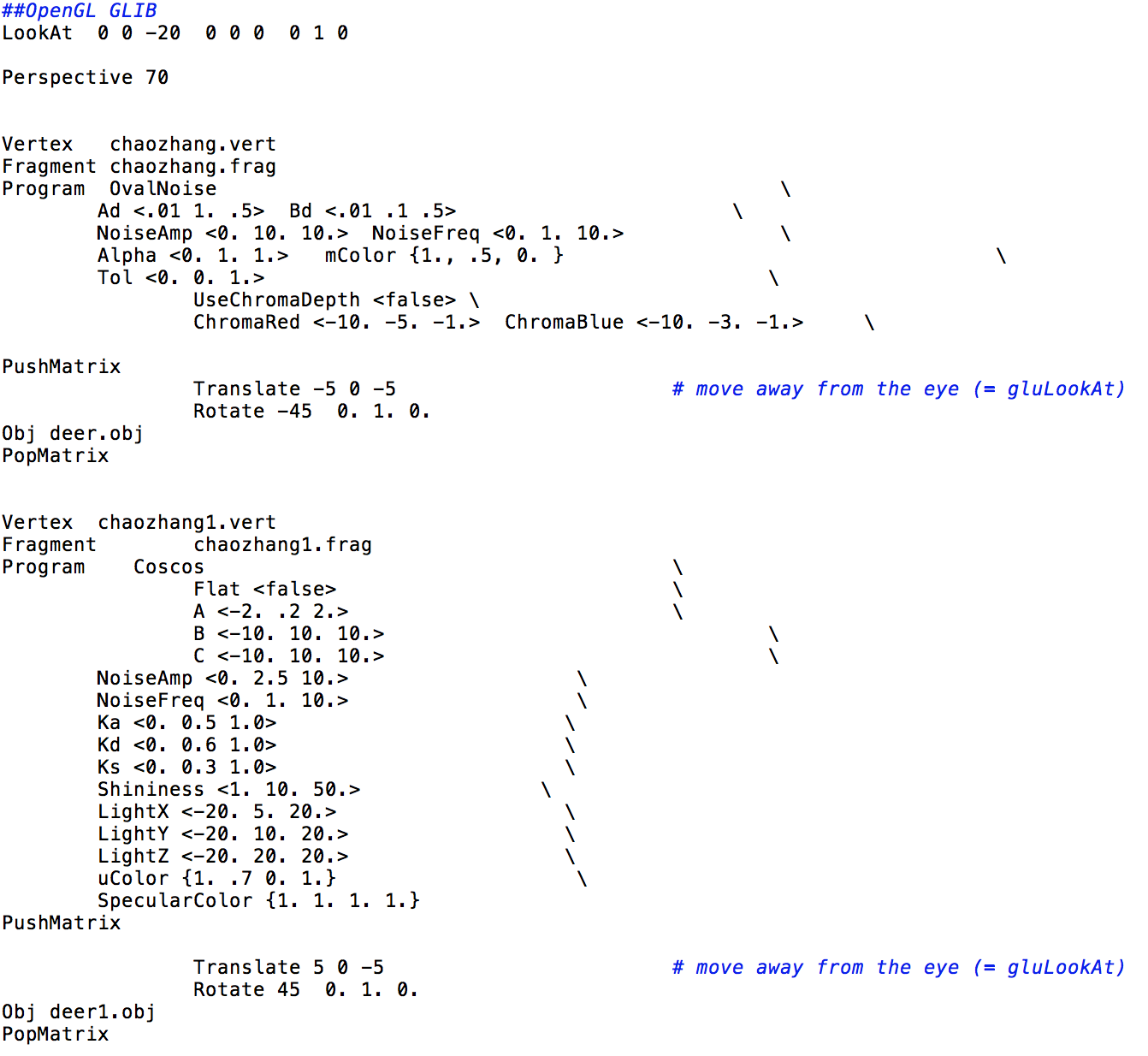
***RenderMan and OpenGL Shaders***

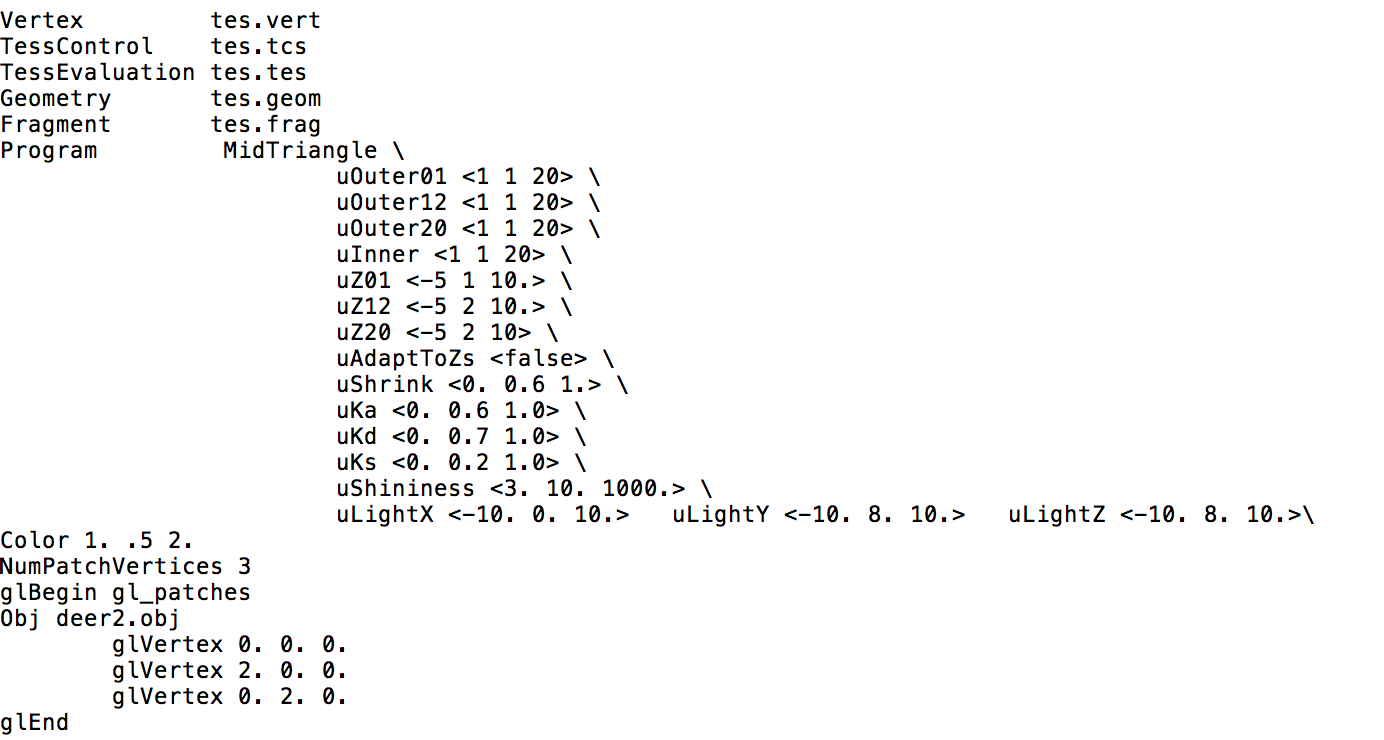
CS557

Final Project

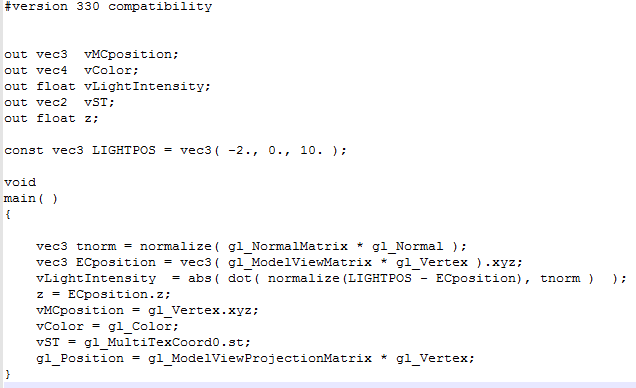
Chao Zhang

1. Source listings



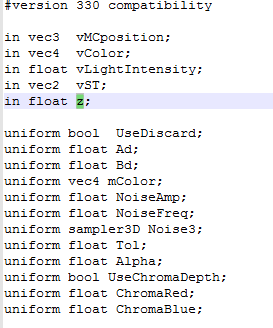


chaozhang.glib

This file includes all the definition I need to use. The ChromaRed and ChromaBlue are both in the range of -10 to -1, and the default is -5. mColor is still the beaver orange. I had three individual objects in this file, each of them implement different effect, the Tessellation, cos bumping mapping and the noise with ChromaDepth. All those will implement on the object deer.

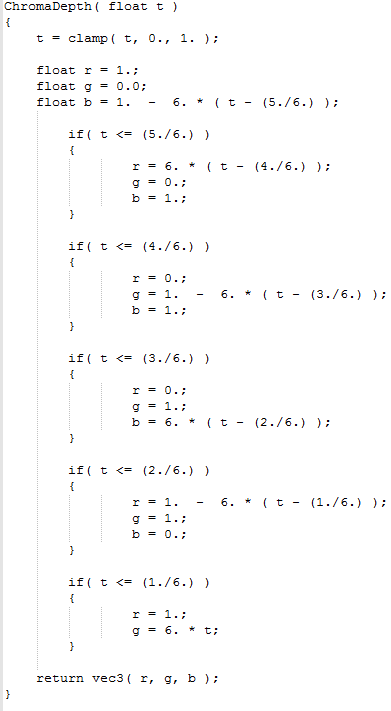
chaozhang.vert

Z= ECposition.z is the vertex position in eye coordinates. The vLightIntensity is the computed ligthingin the vertex shader. tnormfor storing the transformed normal. The vMCposition, vcolor, cLightIntensity, VST and z will be trans to the .freq file.



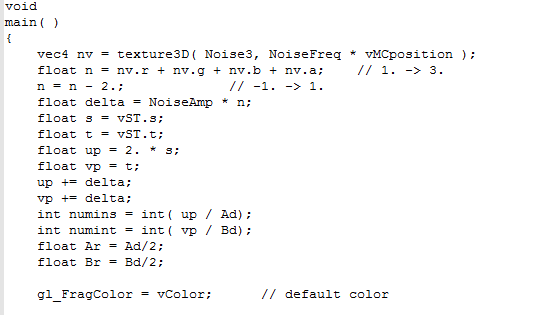
Part1 chaozhang.freq

This is part of the .freq file. This part will get the data from the vertex shader and define the variable.



Part2 chanzhang.freq

This part is to implement the different color for the ChromaDepth. Use different color for different range of t. This can change the sphere into rainbow sphere.



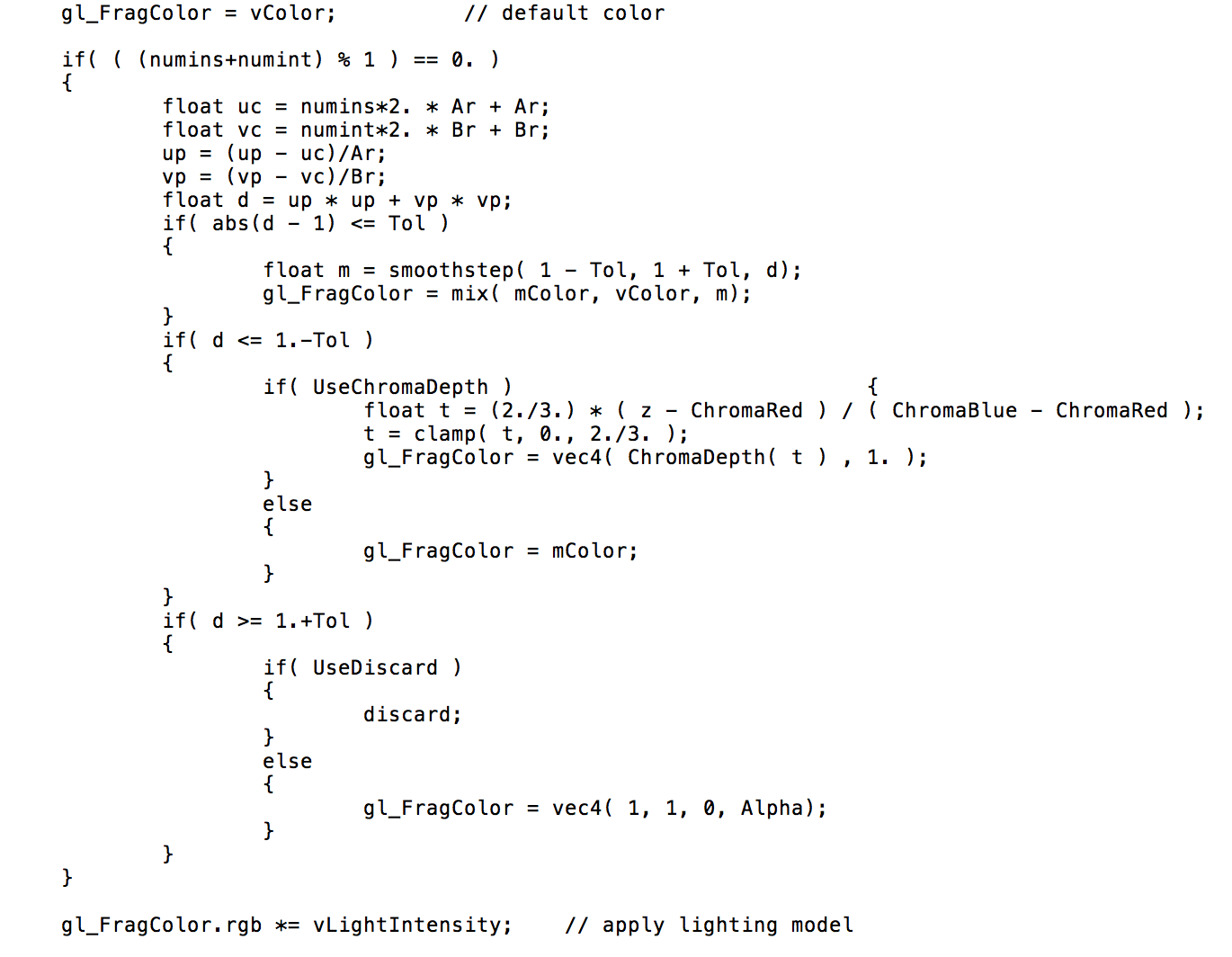
Part3 chaozhang.freq

This part use the Noise3 to create the moise. There are four values in this, the r,g,b and a. The “noise vector” texture nv is a vec4 whose components have separate meanings.

float n = nv.r + nv.g + nv.b + nv.a;

n = n – 2;

Those two lines made the range of the four-octave function from 0 to 1. The rest of the code is like the project1 and 2. It is for the eclipse.



part4 chaozhang.freq

This part is the most important part of this project.

if( abs(d - 1) <= Tol )

{

float m = smoothstep( 1 - Tol, 1 + Tol, d);

gl\_FragColor = mix( mColor, vColor, m);

}

Those lines for the Tol function. With the increase of tol, the edge of the eclipse smooth. I mix the color to achieve the smooth goal.

if( d <= 1.-Tol )

{

if( UseChromaDepth )

{

float t = (2./3.) \* ( z - ChromaRed ) / ( ChromaBlue - ChromaRed );

t = clamp( t, 0., 2./3. );

gl\_FragColor = vec4( ChromaDepth( t ) , 1. );

}

else

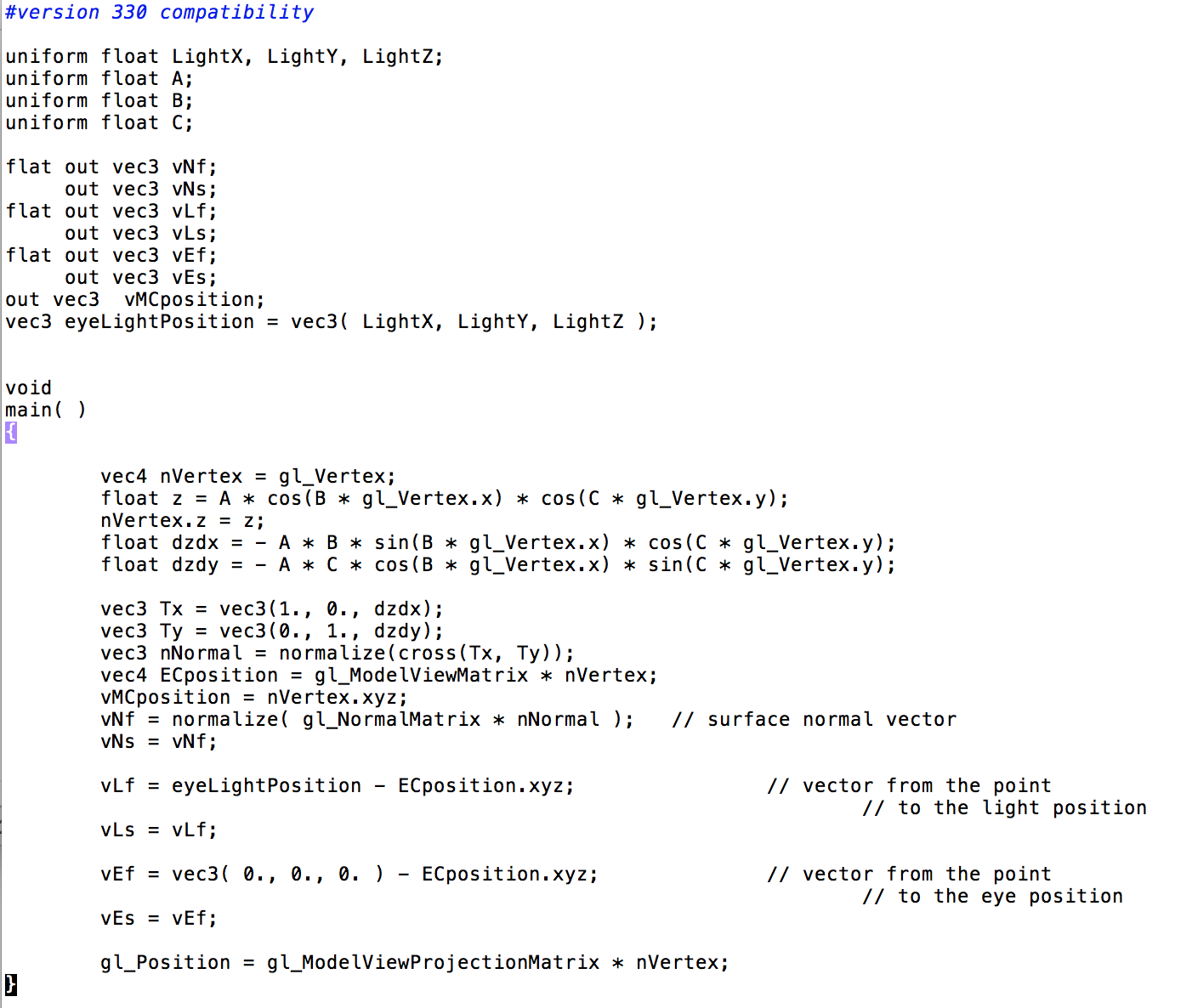
{

gl\_FragColor = mColor;

}

}

the inside loop is for the chromadepth. The reason it is a inside loop is because I only turn the eclipse color to rainbow not include the background color. The range of t is from 0 to 2/3 is because we want the red and blue.



chaozhang.vert

This is the vertex shader file. This file has all the vertex shader information. The compute of the normal, position of the lights. The nVertex mean the new vertex position, it is equal to gl\_vertex in the beginning. Then I use the float z = A \* cos(B \* gl\_Vertex.x) \* cos(C \* gl\_Vertex.y); to compute the position in the z axle of the gl\_vertex and give this new z value to the nVertex. This make the change of A works.

float dzdx = - A \* B \* sin(B \* gl\_Vertex.x) \* cos(C \* gl\_Vertex.y);

float dzdy = - A \* C \* cos(B \* gl\_Vertex.x) \* sin(C \* gl\_Vertex.y);

This two line can get the tangent and then I use

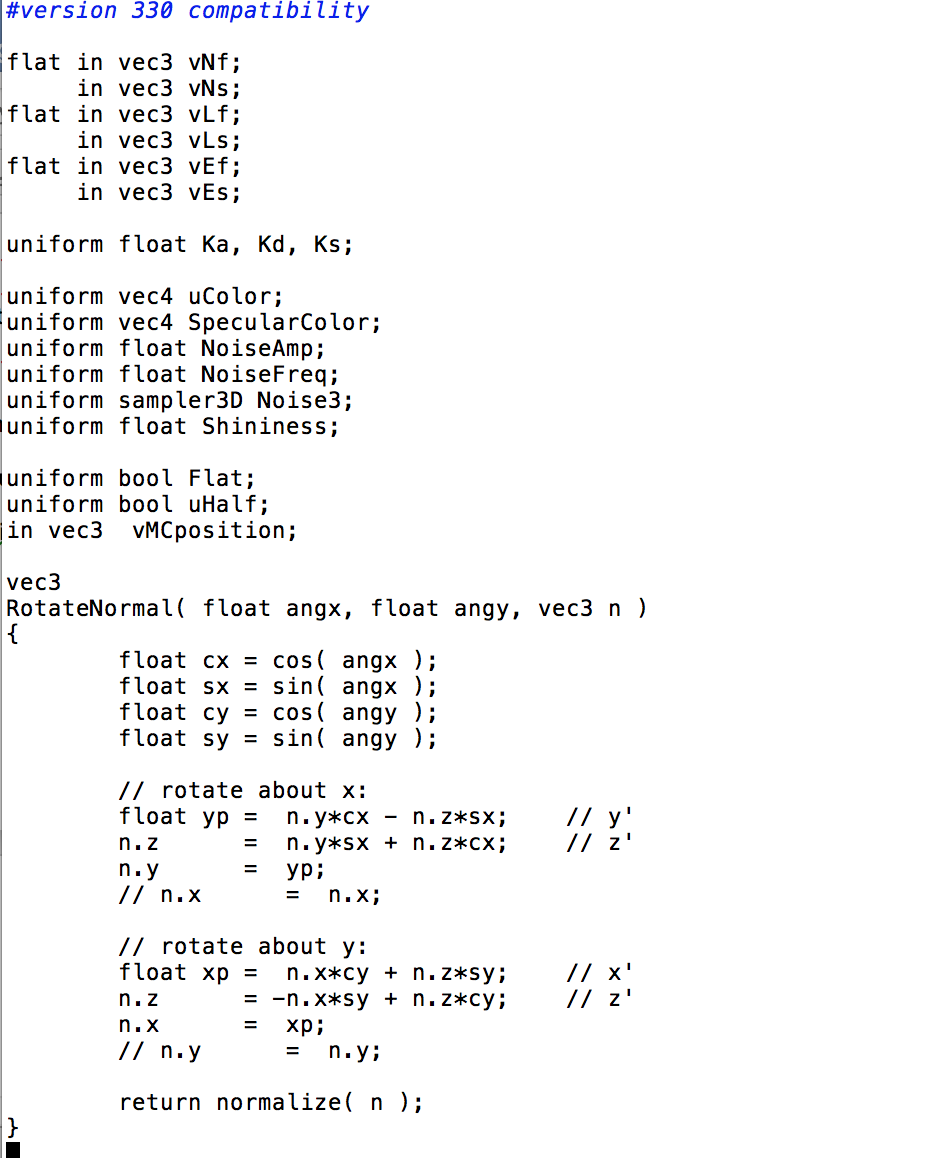
vec3 Tx = vec3(1., 0., dzdx);

vec3 Ty = vec3(0., 1., dzdy); to get the tangent vectors. The Tx and Ty can help us to get the new normal. vec3 nNormal = normalize(cross(Tx, Ty));

Because of the change of the A, I need to change the light position.

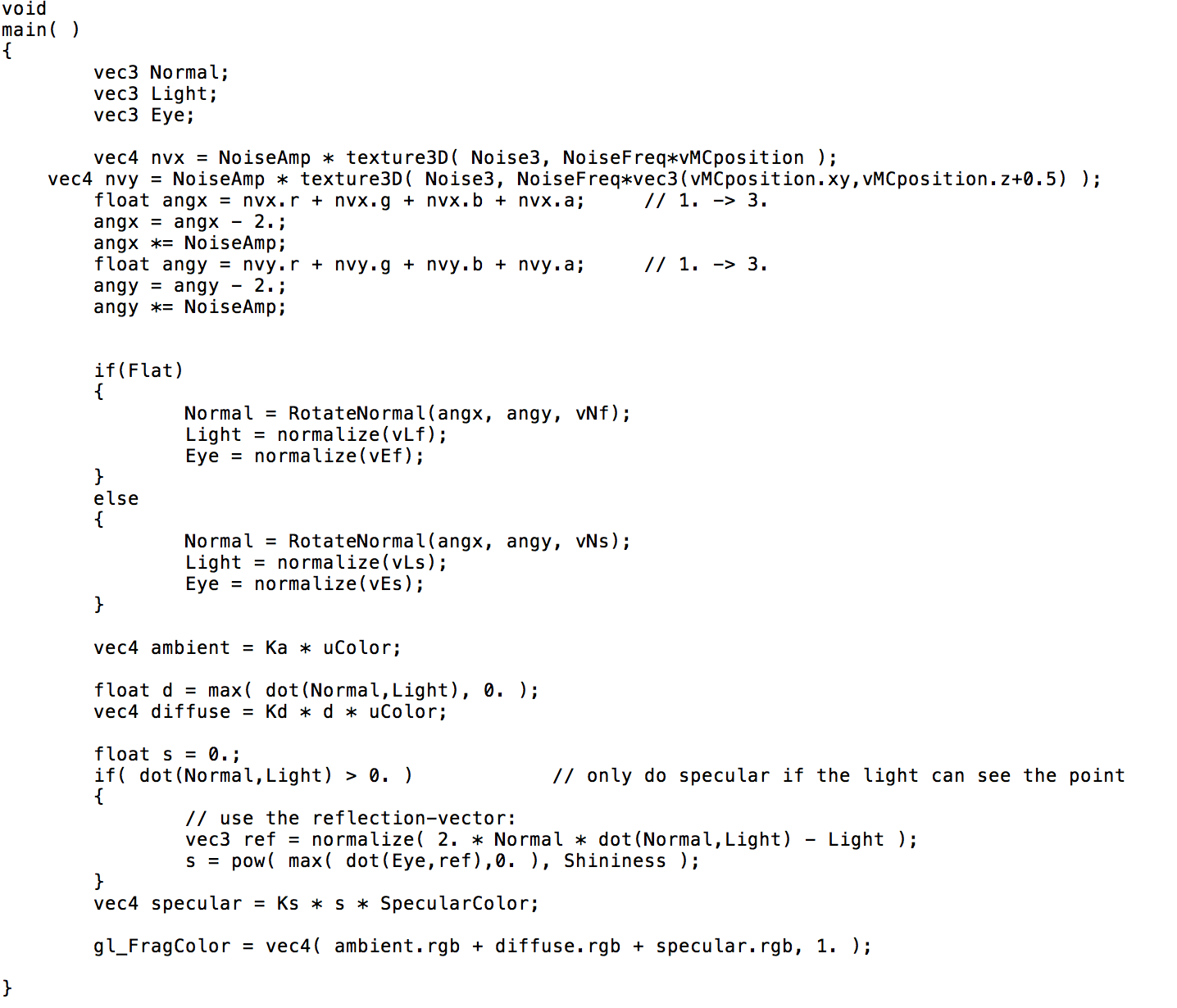
vNf = normalize( gl\_NormalMatrix \* nNormal );

gl\_Position = gl\_ModelViewProjectionMatrix \* nVertex;



part1 chaozhang1.frag

This is the first part of the fragment shader. This part do a rotate of the normal. The reason I do this is to implement the bump-mapping. This function is to rotate the normal in two ways, the angle with x axle and the angle with y axle.



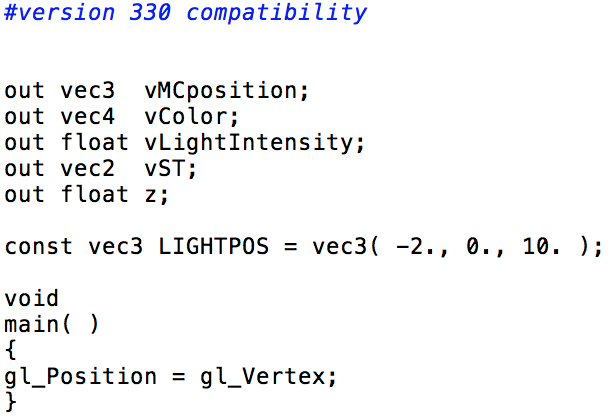
part2 chaozhang1.frag

This part is to implement the rotate normal.

vec4 nvx = uNoiseAmp \* texture3D( Noise3, uNoiseFreq\*vMC );

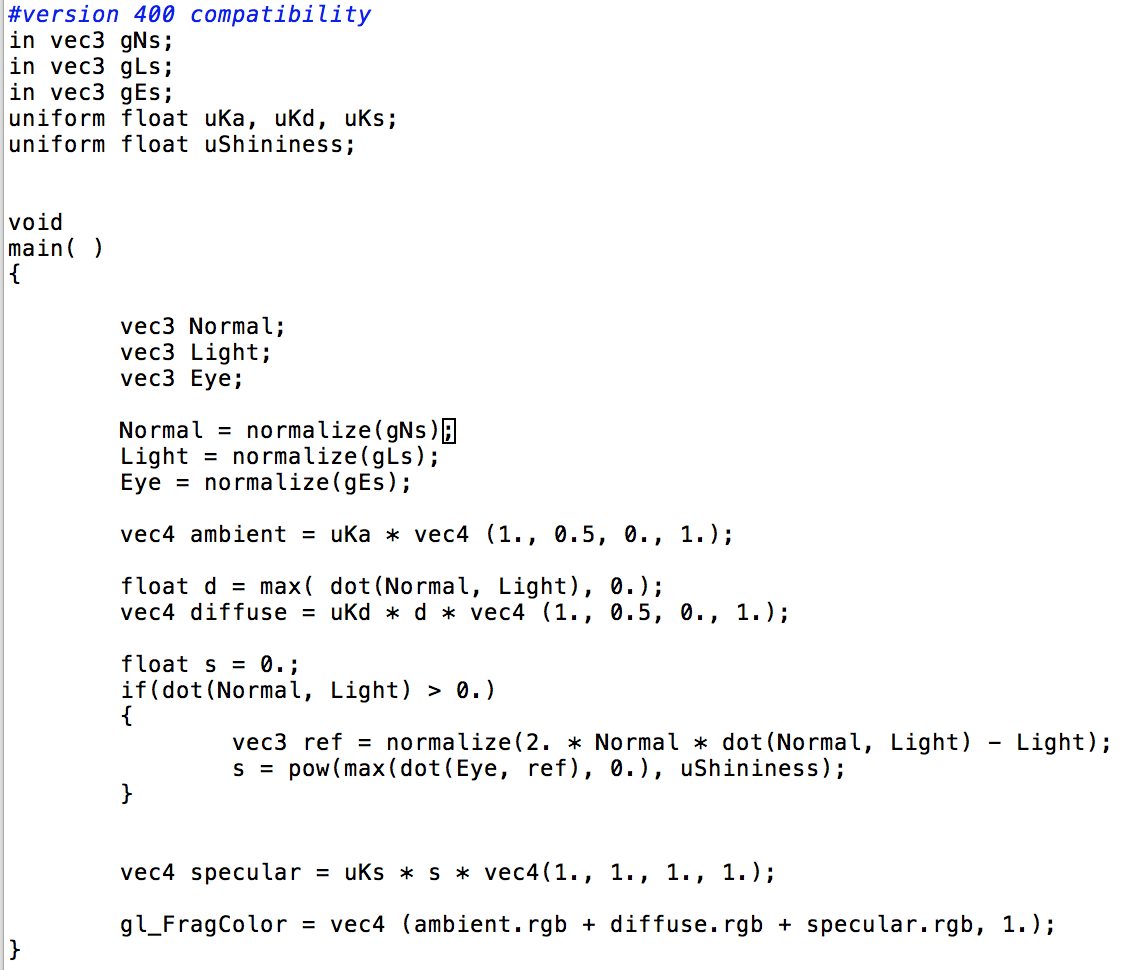
vec4 nvy = uNoiseAmp \* texture3D( Noise3, uNoiseFreq\*vec3(vMC.xy,vMC.z+0.5) );

this two line is to calculate the noise vectors. This part is the same as project 3 noise part. After that, I make the Normal = RotateNormal(angx, angy, vNf); to get the result.



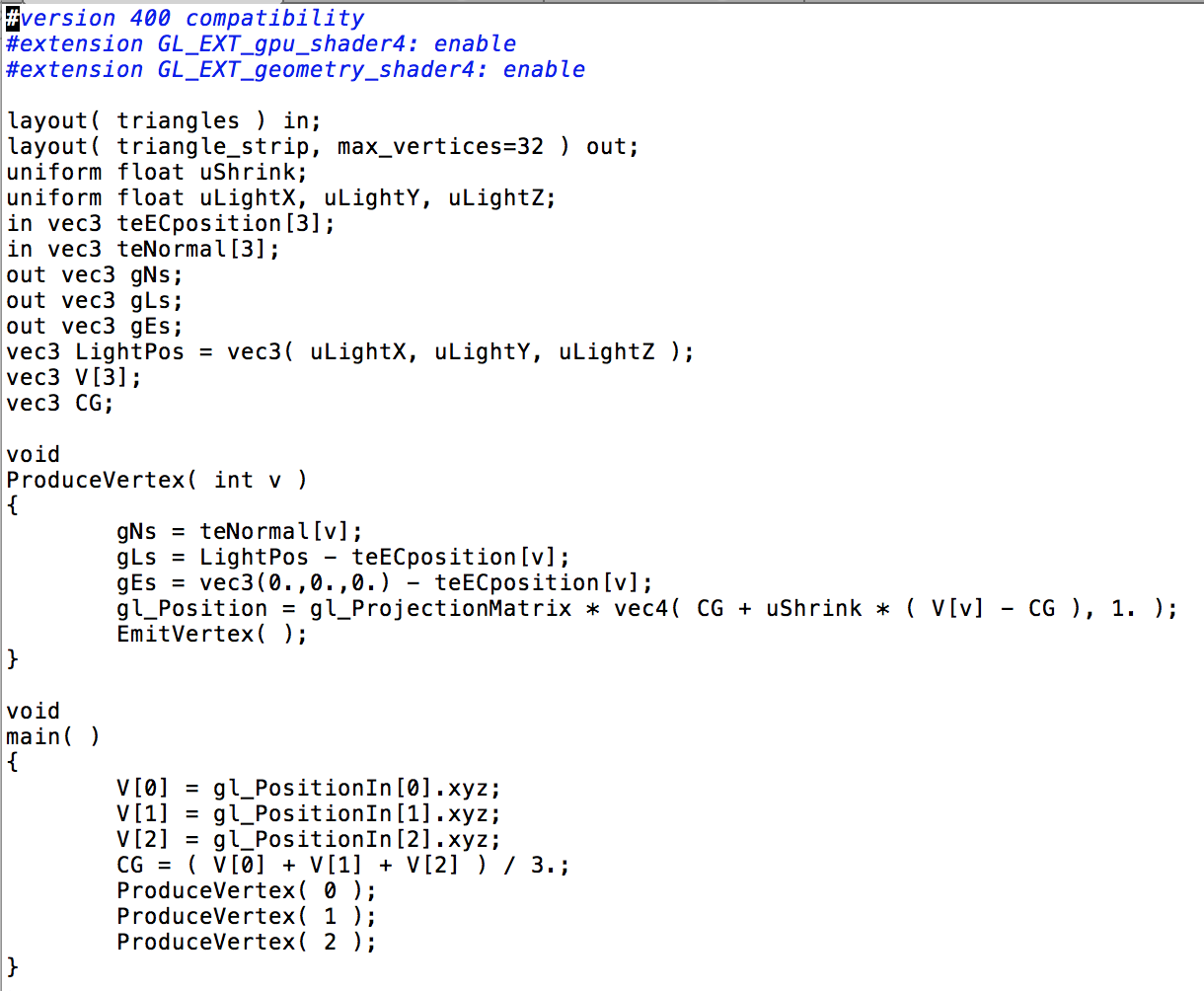
tes.vert

only the gl\_position included.



tes.freq

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



tes.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;

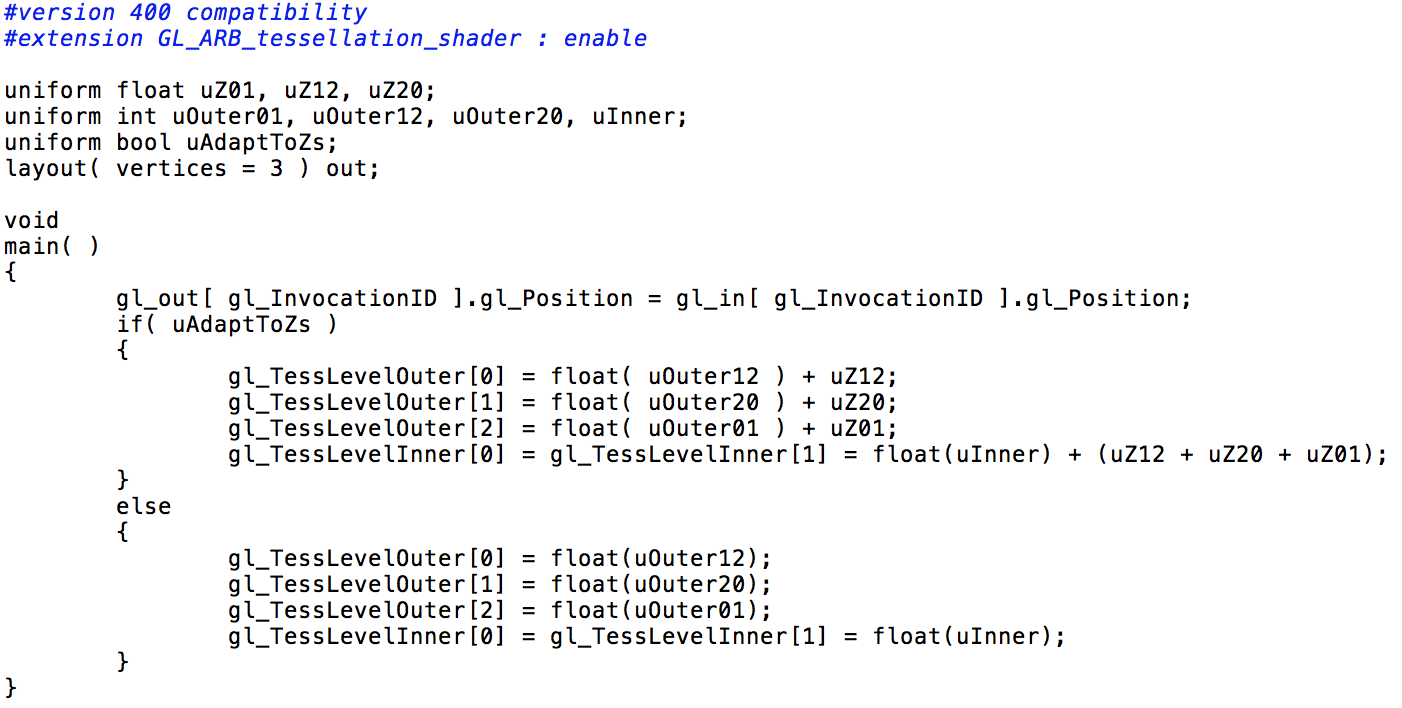
V[2] = gl\_PositionIn[2].xyz;

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

ProduceVertex( 0 );

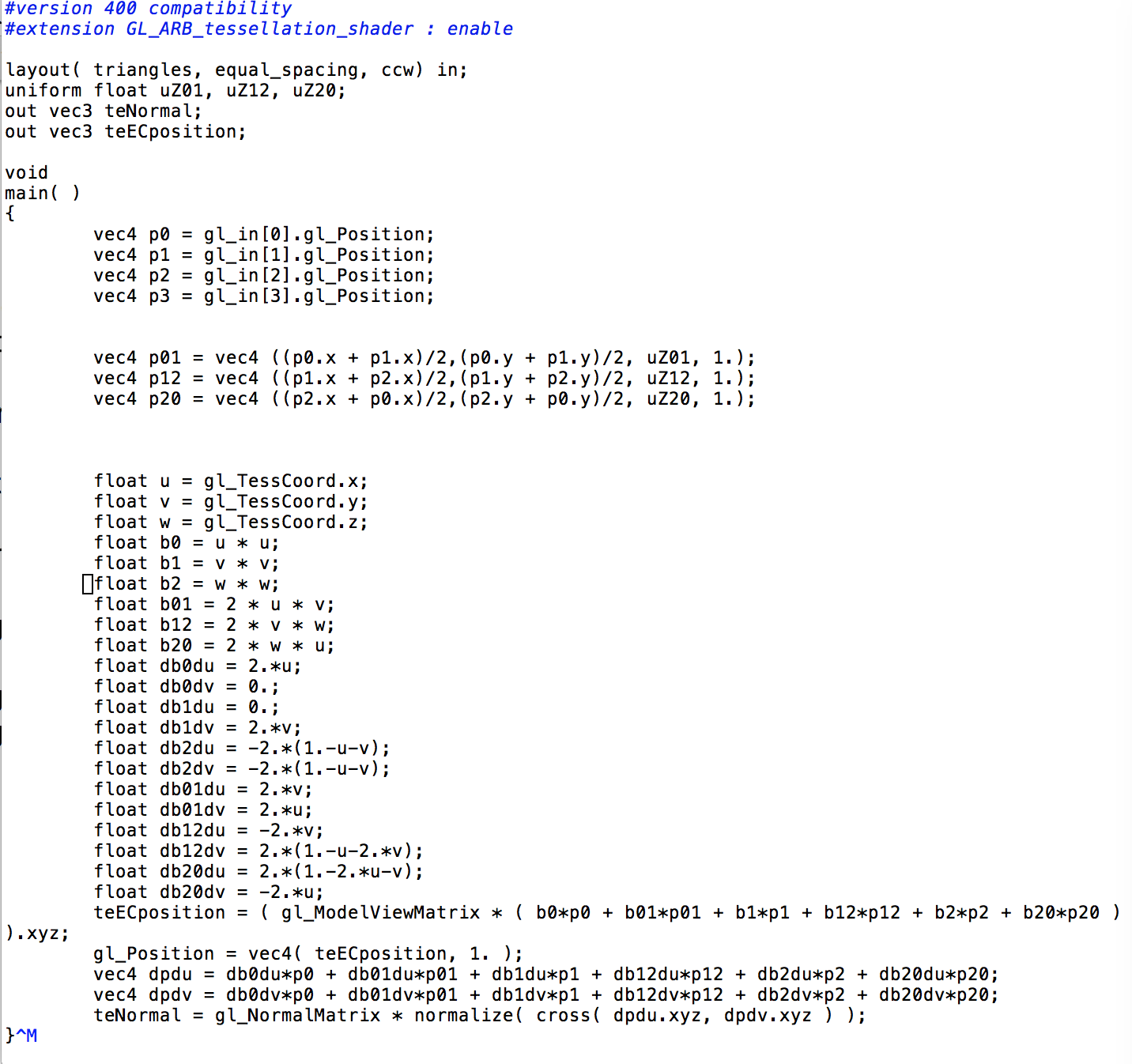
ProduceVertex( 1 );

ProduceVertex( 2 ); Those will finish the produce.



tes.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.



tes.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

