RenderMan and OpenGL Shaders

CS557

Final Project

Chao Zhang

1. Source listings

```
##OpenGL GLIB
LookAt  0 0 -20  0 0 0  0 1 0

Perspective 70


Vertex   chaozhang.vert
Fragment chaozhang.frag
Program  OvalNoise                                          \
        Ad <.01 1. .5>  Bd <.01 .1 .5>                      \
        NoiseAmp <0. 10. 10.>  NoiseFreq <0. 1. 10.>            \
        Alpha <0. 1. 1.>   mColor {1., .5, 0. }                                 \
        Tol <0. 0. 1.>                                      \
                UseChromaDepth <false> \
                ChromaRed <-10. -5. -1.>  ChromaBlue <-10. -3. -1.>     \

PushMatrix
                Translate -5 0 -5                       # move away from the eye (= gluLookAt)
                Rotate -45  0. 1. 0.
Obj deer.obj
PopMatrix


Vertex  chaozhang1.vert
Fragment        chaozhang1.frag
Program    Coscos                                       \
                Flat <false>                            \
                A <-2. .2 2.>                           \
                B <-10. 10. 10.>                                \
                C <-10. 10. 10.>                                \
        NoiseAmp <0. 2.5 10.>                    \
        NoiseFreq <0. 1. 10.>                     \
        Ka <0. 0.5 1.0>                          \
        Kd <0. 0.6 1.0>                          \
        Ks <0. 0.3 1.0>                          \
        Shininess <1. 10. 50.>                  \
        LightX <-20. 5. 20.>                      \
        LightY <-20. 10. 20.>                     \
        LightZ <-20. 20. 20.>                     \
        uColor {1. .7 0. 1.}                       \
        SpecularColor {1. 1. 1. 1.}
PushMatrix

                Translate 5 0 -5                        # move away from the eye (= gluLookAt)
                Rotate 45  0. 1. 0.
Obj deer1.obj
PopMatrix
```

```
Vertex         tes.vert
TessControl    tes.tcs
TessEvaluation tes.tes
Geometry       tes.geom
Fragment       tes.frag
Program         MidTriangle \
                        uOuter01 <1 1 20> \
                        uOuter12 <1 1 20> \
                        uOuter20 <1 1 20> \
                        uInner <1 1 20> \
                        uZ01 <-5 1 10.> \
                        uZ12 <-5 2 10.> \
                        uZ20 <-5 2 10> \
                        uAdaptToZs <false> \
                        uShrink <0. 0.6 1.> \
                        uKa <0. 0.6 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 2.
NumPatchVertices 3
glBegin gl_patches
Obj deer2.obj
        glVertex 0. 0. 0.
        glVertex 2. 0. 0.
        glVertex 0. 2. 0.
glEnd
```

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.

```
#version 330 compatibility


out vec3  vMCposition;
out vec4  vColor;
out float vLightIntensity;
out vec2  vST;
out float z;

const vec3 LIGHTPOS = vec3( -2., 0., 10. );

void
main( )
{

    vec3 tnorm = normalize( gl_NormalMatrix * gl_Normal );
    vec3 ECposition = vec3( gl_ModelViewMatrix * gl_Vertex ).xyz;
    vLightIntensity  = abs( dot( normalize(LIGHTPOS - ECposition), tnorm )  );
    z = ECposition.z;
    vMCposition = gl_Vertex.xyz;
    vColor = gl_Color;
    vST = gl_MultiTexCoord0.st;
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

chaozhang.vert

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

```
#version 330 compatibility

in vec3  vMCposition;
in vec4  vColor;
in float vLightIntensity;
in vec2  vST;
in float z;

uniform bool  UseDiscard;
uniform float Ad;
uniform float Bd;
uniform vec4 mColor;
uniform float NoiseAmp;
uniform float NoiseFreq;
uniform sampler3D Noise3;
uniform float Tol;
uniform float Alpha;
uniform bool UseChromaDepth;
uniform float ChromaRed;
uniform float ChromaBlue;
```

Part1 chaozhang.freq

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

```
ChromaDepth( float t )
{
    t = clamp( t, 0., 1. );

    float r = 1.;
    float g = 0.0;
    float b = 1.  -  6. * ( t - (5./6.) );

        if( t <= (5./6.) )
        {
                r = 6. * ( t - (4./6.) );
                g = 0.;
                b = 1.;
        }

        if( t <= (4./6.) )
        {
                r = 0.;
                g = 1.  -  6. * ( t - (3./6.) );
                b = 1.;
        }

        if( t <= (3./6.) )
        {
                r = 0.;
                g = 1.;
                b = 6. * ( t - (2./6.) );
        }

        if( t <= (2./6.) )
        {
                r = 1.  -  6. * ( t - (1./6.) );
                g = 1.;
                b = 0.;
        }

        if( t <= (1./6.) )
        {
                r = 1.;
                g = 6. * t;
        }

    return vec3( r, g, b );
}
```

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.

```
void
main( )
{
    vec4 nv = texture3D( Noise3, NoiseFreq * vMCposition );
    float n = nv.r + nv.g + nv.b + nv.a;    // 1. -> 3.
    n = n - 2.;             // -1. -> 1.
    float delta = NoiseAmp * n;
    float s = vST.s;
    float t = vST.t;
    float up = 2. * s;
    float vp = t;
    up += delta;
    vp += delta;
    int numins = int( up / Ad);
    int numint = int( vp / Bd);
    float Ar = Ad/2;
    float Br = Bd/2;

    gl_FragColor = vColor;      // default color
```

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.

```
gl_FragColor = vColor;          // default color

if( ( (numins+numint) % 1 ) == 0. )
{
        float uc = numins*2. * Ar + Ar;
        float vc = numint*2. * Br + Br;
        up = (up - uc)/Ar;
        vp = (vp - vc)/Br;
        float d = up * up + vp * vp;
        if( abs(d - 1) <= Tol )
        {
                float m = smoothstep( 1 - Tol, 1 + Tol, d);
                gl_FragColor = mix( mColor, vColor, m);
        }
        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;
                }
        }
        if( d >= 1.+Tol )
        {
                if( UseDiscard )
                {
                        discard;
                }
                else
                {
                        gl_FragColor = vec4( 1, 1, 0, Alpha);
                }
        }
}

gl_FragColor.rgb *= vLightIntensity;    // apply lighting model
```

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.

```
#version 330 compatibility

uniform float LightX, LightY, LightZ;
uniform float A;
uniform float B;
uniform float C;

flat out vec3 vNf;
     out vec3 vNs;
flat out vec3 vLf;
     out vec3 vLs;
flat out vec3 vEf;
     out vec3 vEs;
out vec3  vMCposition;
vec3 eyeLightPosition = vec3( LightX, LightY, LightZ );


void
main( )
{

        vec4 nVertex = gl_Vertex;
        float z = A * cos(B * gl_Vertex.x) * cos(C * gl_Vertex.y);
        nVertex.z = z;
        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);

        vec3 Tx = vec3(1., 0., dzdx);
        vec3 Ty = vec3(0., 1., dzdy);
        vec3 nNormal = normalize(cross(Tx, Ty));
        vec4 ECposition = gl_ModelViewMatrix * nVertex;
        vMCposition = nVertex.xyz;
        vNf = normalize( gl_NormalMatrix * nNormal );   // surface normal vector
        vNs = vNf;

        vLf = eyeLightPosition - ECposition.xyz;                // vector from the point
                                                                        // to the light position
        vLs = vLf;

        vEf = vec3( 0., 0., 0. ) - ECposition.xyz;              // vector from the point
                                                                        // to the eye position
        vEs = vEf;

        gl_Position = gl_ModelViewProjectionMatrix * nVertex;
}
```

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;

```
#version 330 compatibility

flat in vec3 vNf;
     in vec3 vNs;
flat in vec3 vLf;
     in vec3 vLs;
flat in vec3 vEf;
     in vec3 vEs;

uniform float Ka, Kd, Ks;

uniform vec4 uColor;
uniform vec4 SpecularColor;
uniform float NoiseAmp;
uniform float NoiseFreq;
uniform sampler3D Noise3;
uniform float Shininess;

uniform bool Flat;
uniform bool uHalf;
in vec3  vMCposition;

vec3
RotateNormal( float angx, float angy, vec3 n )
{
        float cx = cos( angx );
        float sx = sin( angx );
        float cy = cos( angy );
        float sy = sin( angy );

        // rotate about x:
        float yp =  n.y*cx - n.z*sx;    // y'
        n.z      =  n.y*sx + n.z*cx;    // z'
        n.y      =  yp;
        // n.x      =  n.x;

        // rotate about y:
        float xp =  n.x*cy + n.z*sy;    // x'
        n.z      = -n.x*sy + n.z*cy;    // z'
        n.x      =  xp;
        // n.y      =  n.y;

        return normalize( n );
}
```

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.

```
void
main( )
{
        vec3 Normal;
        vec3 Light;
        vec3 Eye;

        vec4 nvx = NoiseAmp * texture3D( Noise3, NoiseFreq*vMCposition );
    vec4 nvy = NoiseAmp * texture3D( Noise3, NoiseFreq*vec3(vMCposition.xy,vMCposition.z+0.5) );
        float angx = nvx.r + nvx.g + nvx.b + nvx.a;     // 1. -> 3.
        angx = angx - 2.;
        angx *= NoiseAmp;
        float angy = nvy.r + nvy.g + nvy.b + nvy.a;     // 1. -> 3.
        angy = angy - 2.;
        angy *= NoiseAmp;


        if(Flat)
        {
                Normal = RotateNormal(angx, angy, vNf);
                Light = normalize(vLf);
                Eye = normalize(vEf);
        }
        else
        {
                Normal = RotateNormal(angx, angy, vNs);
                Light = normalize(vLs);
                Eye = normalize(vEs);
        }

        vec4 ambient = Ka * uColor;

        float d = max( dot(Normal,Light), 0. );
        vec4 diffuse = Kd * d * uColor;

        float s = 0.;
        if( dot(Normal,Light) > 0. )            // only do specular if the light can see the point
        {
                // use the reflection-vector:
                vec3 ref = normalize( 2. * Normal * dot(Normal,Light) - Light );
                s = pow( max( dot(Eye,ref),0. ), Shininess );
        }
        vec4 specular = Ks * s * SpecularColor;

        gl_FragColor = vec4( ambient.rgb + diffuse.rgb + specular.rgb, 1. );

}
```

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.

```
#version 330 compatibility


out vec3  vMCposition;
out vec4  vColor;
out float vLightIntensity;
out vec2  vST;
out float z;

const vec3 LIGHTPOS = vec3( -2., 0., 10. );

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

tes.vert

only the gl_position included.

```
#version 400 compatibility
in vec3 gNs;
in vec3 gLs;
in vec3 gEs;
uniform float uKa, uKd, uKs;
uniform float uShininess;


void
main( )
{

        vec3 Normal;
        vec3 Light;
        vec3 Eye;

        Normal = normalize(gNs);
        Light = normalize(gLs);
        Eye = normalize(gEs);

        vec4 ambient = uKa * vec4 (1., 0.5, 0., 1.);

        float d = max( dot(Normal, Light), 0.);
        vec4 diffuse = uKd * d * vec4 (1., 0.5, 0., 1.);

        float s = 0.;
        if(dot(Normal, Light) > 0.)
        {
                vec3 ref = normalize(2. * Normal * dot(Normal, Light) - Light);
                s = pow(max(dot(Eye, ref), 0.), uShininess);
        }


        vec4 specular = uKs * s * vec4(1., 1., 1., 1.);

        gl_FragColor = vec4 (ambient.rgb + diffuse.rgb + specular.rgb, 1.);
}
```

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.

```
#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;

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 );
}
```

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.

```
#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] = float( uOuter12 ) + uZ12;
		gl_TessLevelOuter[1] = float( uOuter20 ) + uZ20;
		gl_TessLevelOuter[2] = float( uOuter01 ) + uZ01;
		gl_TessLevelInner[0] = gl_TessLevelInner[1] = float(uInner) + (uZ12 + uZ20 + uZ01);
	}
	else
	{
		gl_TessLevelOuter[0] = float(uOuter12);
		gl_TessLevelOuter[1] = float(uOuter20);
		gl_TessLevelOuter[2] = float(uOuter01);
		gl_TessLevelInner[0] = gl_TessLevelInner[1] = float(uInner);
	}
}
```

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.

```
#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 ((p0.x + p1.x)/2,(p0.y + p1.y)/2, uZ01, 1.);
        vec4 p12 = vec4 ((p1.x + p2.x)/2,(p1.y + p2.y)/2, uZ12, 1.);
        vec4 p20 = vec4 ((p2.x + p0.x)/2,(p2.y + p0.y)/2, uZ20, 1.);



        float u = gl_TessCoord.x;
        float v = gl_TessCoord.y;
        float w = gl_TessCoord.z;
        float b0 = u * u;
        float b1 = v * v;
       float b2 = w * w;
        float b01 = 2 * u * v;
        float b12 = 2 * v * w;
        float b20 = 2 * w * u;
        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 ) );
}^M
```

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.

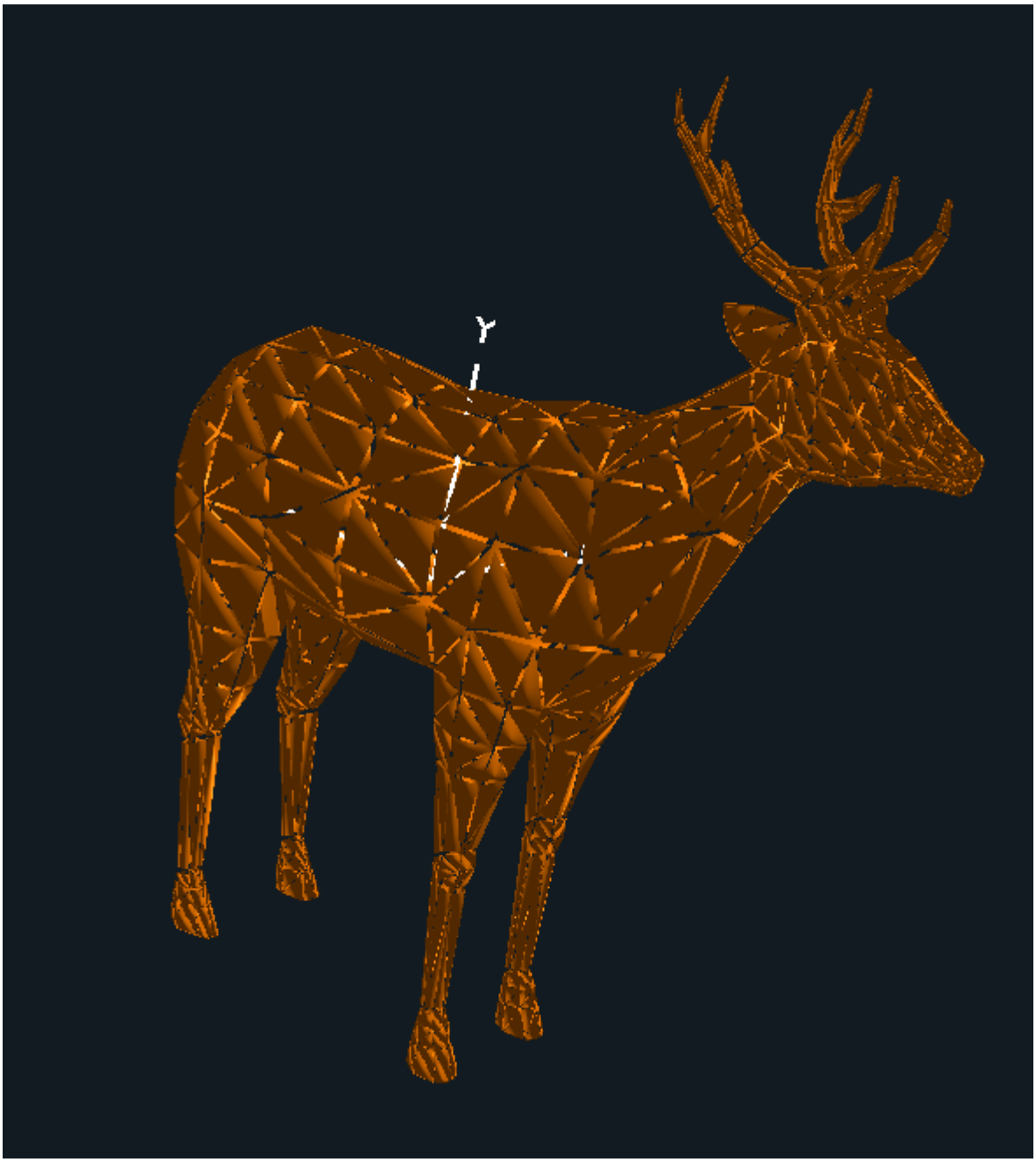
teECposition = (gl_ModelViewMatrix * (b0*p0 + b01*p01 + b1*p1 + b12*p12 + b2*p2 + b20*p20)).xyz;

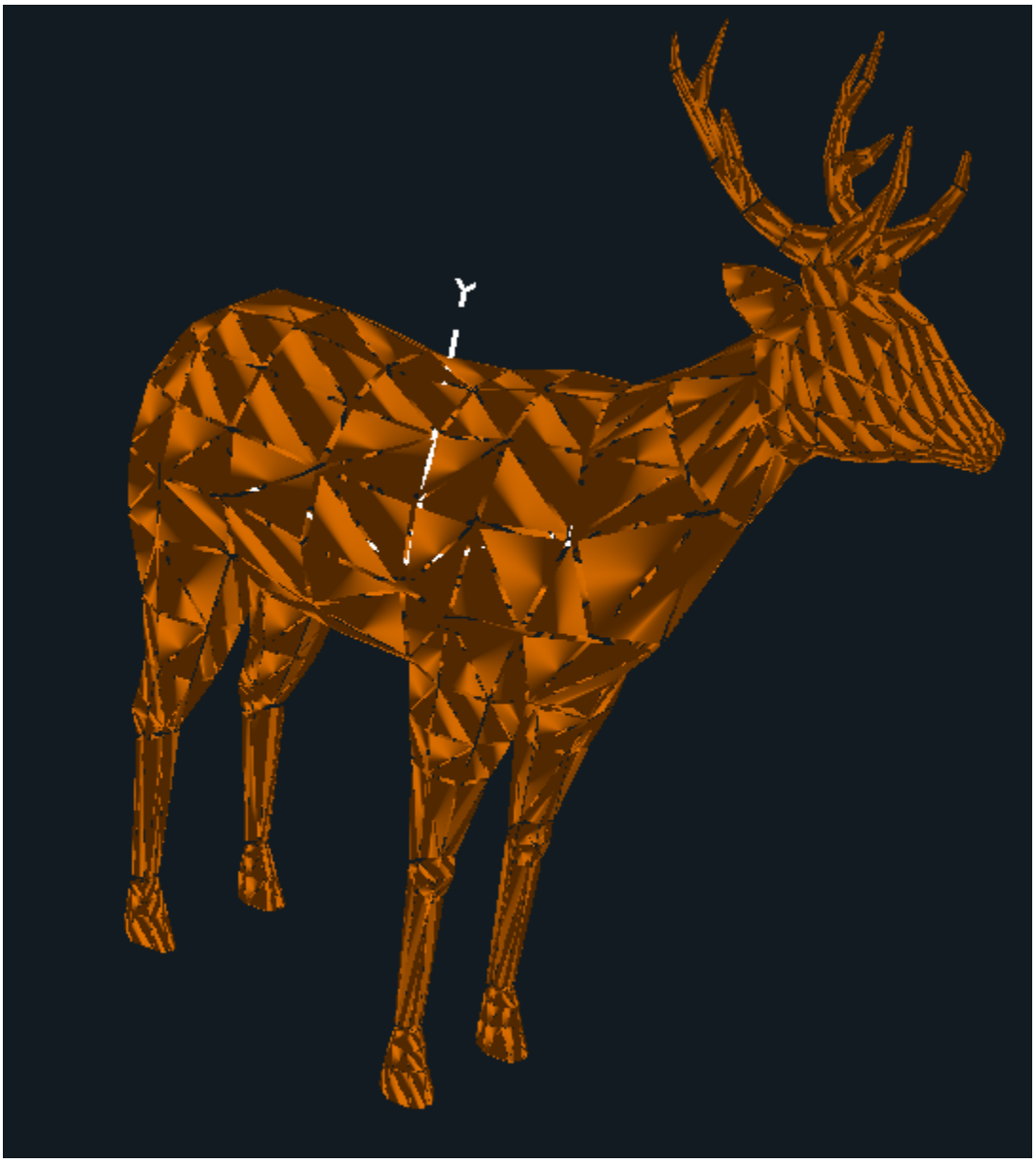
gl_Position = vec4(teECposition, 1.);

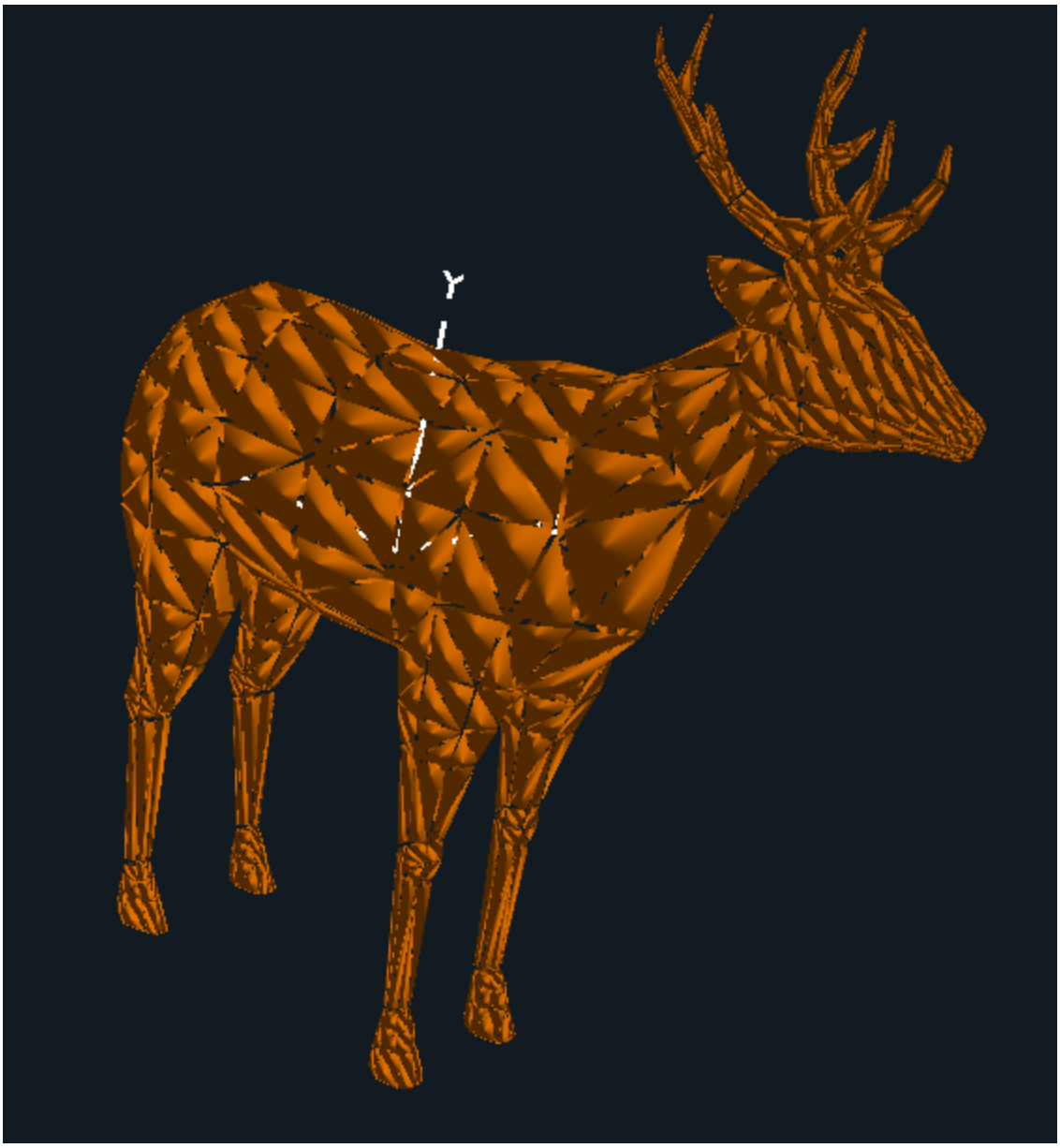
```
    vec4 dpdu = db0du*p0 + db01du*p01 + db1du*p1 + db12du*p12 + db2du*p2 + db20du*p20;
    vec4 dpdv = db0dv*p0 + db01dv*p01 + db1dv*p1 + db12dv*p12 + db2dv*p2 + db20dv*p20;
    teNormal = gl_NormalMatrix * normalize( cross( dpdu.xyz, dpdv.xyz ) );
}
```

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.

2. Results




Y


٢