CS 557 -- Winter Quarter 2016

Final Project Report

Rabbit in the Forest Views the Modern City

Instructor: Prof. Mike Bailey

Name: Kai Shi

ID: 932-504-512

March 2016

(1) Proposal

In final project, I'm going to create a rabbit (or other animals) by .obj file, which surface is smooth, and put it in the cube map environment. Also, I will add the sun and it's the lighting of the whole environment.

The final picture is like a rabbit in the forest under the sun.

Cube mapping: I will create a cube-mapping environment first. The background is a forest.

Cube map for reflection and refraction: I will implement reflection and refraction effect of the cube map environment among the surface of the animal.

Bump mapping: I will apply bump mapping in the sun and add noise effect.

Lighting: I will use the check box for choosing using per-vertex lighting or per-fragment lighting. Also, the lighting from sun can be changed according to the different "Time" of the day.

Morphing: I will also try to add morphing effect and animation. It makes the rabbit looks like involved in a black hole by some mathematical and physical formulas.

Fog Effect: I will add fog effect among the whole environment to make the rabbit blurry by the mixing function.

(2) Implement

In final project, I use GLSL in glman to create a rabbit standing in the riverside in the forest. The rabbit is overlooking at the city in the other side of the river. The sun is in the sky. I realized all parts in my proposal.

Cube mapping	**Done**
Reflection and Refraction	**Done**
Bump mapping	**Done**
Lighting	**Done**
Morphing	**Done**
Fog Effect	**Done**

(3) Code

glib:

```
##OpenGL GLIB
Perspective 60
LookAt 2 1 -7  0 0 0  0 1 0

Vertex   texture.vert
Fragment texture.frag
Program  Texture  TexUnit 6

Texture2D  6  posx.bmp
QuadYZ 8. 8.

Texture2D  6  negx.bmp
QuadYZ -8. 8.

Texture2D  6  posy.bmp
QuadXZ 8. 8.

Texture2D  6  negy.bmp
QuadXZ -8. 8.

Texture2D  6  posz.bmp
QuadXY 8. 8.

Texture2D  6  negz.bmp
QuadXY -8. 8.

CubeMap 7 posx.bmp negx.bmp  posy.bmp negy.bmp   posz.bmp negz.bmp
CubeMap 8 posx.bmp negx.bmp  posy.bmp negy.bmp   posz.bmp negz.bmp

Vertex kaishi.vert
Fragment kaishi.frag
Program kaishi   \
        uLightX <-30. 1. 30.>    \
        uLightY <-30. 28. 30.>    \
        uLightZ <-30. 3. 30.>    \
```

```
        uKa <0. 0.1 1.0>        \
        uKd <0. 0.7 1.0>        \
        uKs <0. 0.2 1.0>        \
        uDayTime <-40. 1. 40.>   \
        uForce <0. 0. 100.> \
        uFogStep <1 100 100>   \
        uUseBumpmap <false>  \
        uAng <-3.14159 0.785398 3.14159>          \
        uBumpDensity <5. 16. 100.>                          \
        uAmbient <0. 0.1 .4>                          \
        uUseReflect <false>  \
        uUseRefract <false>   \
        uReflectUnit 7   \
        uRefractUnit 8

Color 0.6 0.6 0.6
Translate -1. -8 5.
Scale 2. 2. 2.
Obj Rabbit.obj

Scale 0.2 0.2 0.2
Translate 1. 28. 3.
Rotate -180  1 0 0
Color 1. 0.8 0.
Sphere
```

vert:

```
#version 330 compatibility

in vec3 Tangent;
out vec3 vBTNx, vBTNy, vBTNz;
out vec2 vST;
out vec3 vDirToLight;
out vec4 vColor;

out vec3 vNormal, vLight, vEye;
out vec3 vReflectVector;

out vec3 vRefractVector;
out vec3 MCposition, ECposition;
```

```
uniform float uForce;
uniform float uLightX, uLightY, uLightZ;
uniform bool uUseReflect;
uniform bool uUseBumpmap;
uniform bool uUseRefract;
uniform float uDayTime;

void main()
{
	float lightx;
	vST = gl_MultiTexCoord0.st;

	lightx = uLightX + uDayTime;

	vec3 LIGHTPOS = vec3(lightx, uLightY, uLightZ);
	if (uUseBumpmap)
	{
		vec3 N = normalize( gl_NormalMatrix * gl_Normal );
		vec3 T;
		vec3 B;
	#define GRAM_SCHMIDT_METHOD

	#ifdef HAVE_TANGENT_METHOD
		T = normalize(  vec3( gl_ModelViewMatrix*vec4(Tangent,0.) )  );
		B = normalize( cross(T,N) );
	#endif

	#ifdef GRAM_SCHMIDT_METHOD
		T = vec3( 0.,1.,0.);
		float d = dot( T, N );
		T = normalize( T - d*N );
		B = normalize( cross(T,N) );
	#endif

	#ifdef CROSS_PRODUCT_METHOD
		T = vec3( 0.,1.,0.);
		B = normalize( cross(T,N) );
		T = normalize( cross(N,B) );
	#endif

		vBTNx = vec3( B.x, T.x, N.x );
		vBTNy = vec3( B.y, T.y, N.y );
		vBTNz = vec3( B.z, T.z, N.z );

		vColor = gl_Color;
```

```
        vec3 LightPosition = vec3( uLightX, uLightY, uLightZ );
        vec3 ECposition = ( gl_ModelViewMatrix * gl_Vertex ).xyz;
        vDirToLight = normalize( LightPosition - ECposition );

        gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

}


else if (uUseReflect)
{
        vec3 ECposition = vec3( gl_ModelViewMatrix * gl_Vertex );

        vec3 eyeDir = ECposition - vec3(0.,0.,0.);
        vec3 normal = normalize( gl_NormalMatrix * gl_Normal );

        vReflectVector = reflect( eyeDir, normal );
        gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

}

        else if (uUseRefract)
{
        vec3 ECposition = vec3( gl_ModelViewMatrix * gl_Vertex );

        vec3 eyeDir = ECposition - vec3(0.,0.,0.);
        vec3 normal = normalize( gl_NormalMatrix * gl_Normal );

        vRefractVector = refract( eyeDir, normal, 1.4 );
        gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

}

else
{

        float uX,uY,uZ;
        uX = -2.;
        uY = 0.8;
        uZ = 0.5;
        vec4 vertex0 = vec4(uX,uY,uZ,1.);
        vec3 ECposition = vec3( gl_ModelViewMatrix * gl_Vertex );

        vColor = gl_Color;
        vNormal = normalize(gl_NormalMatrix * gl_Normal);
```

```
		vLight  = LIGHTPOS - ECposition;
		vEye    = vec3(0.,0.,0.) - ECposition;

		gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;


		//morph
		float dis = distance(gl_Vertex.xyz, vertex0.xyz);
		vec4 positionA =  mix(gl_Vertex,vec4(uX,uY,uZ,1.), uForce/(dis*dis) );
		vec4 vertex1 = vec4(-uX,-uY,-uZ,1.);
		float dis1 = distance(positionA.xyz, vertex1.xyz);
		vec4 positionB =  gl_ModelViewProjectionMatrix
*mix(positionA,vec4(-uX,-uY,-uZ,1.), 0 );

		if(dot((mix(positionA,vec4(-uX,-uY,-uZ,1.),0 ).xyz-vertex0.xyz
),gl_Vertex.xyz-vertex0.xyz)<=0.0)
			gl_Position = gl_ModelViewProjectionMatrix
*vec4(uX,uY,uZ,1.);
		else
			{
			if(dot((mix(positionA,vec4(-uX,-uY,-uZ,1.), 0).xyz-vertex1.xyz
),gl_Vertex.xyz-vertex1.xyz)<=0.0)
				gl_Position = gl_ModelViewProjectionMatrix *vec4(-uX,-
uY,-uZ,1.);
			else
				gl_Position = positionB;
			}

		//fog
		MCposition = gl_Vertex.xyz;

	}

}
```

frag:

```
#version 330 compatibility
uniform float uKa, uKd, uKs;
uniform samplerCube uReflectUnit;
uniform samplerCube uRefractUnit;
uniform bool uUseReflect;
uniform bool uUseRefract;
uniform int uFogStep;
uniform float Timer;
```

```
uniform sampler3D Noise3;
uniform bool uUseBumpmap;
uniform float uBumpDensity;
uniform float uBumpSize;
uniform float uAng;
uniform float uAmbient;

in vec3 vNormal, vLight, vEye;
in vec3 MCposition, ECposition;
in vec3 vBTNx, vBTNy, vBTNz;
in vec2 vST;
in vec3 vDirToLight;
in vec4 vColor;
in vec3  vReflectVector;
in vec3 vRefractVector;

float Cang, Sang;
const float PI = 3.14159265;

vec3

ToXyz( vec3 sth )

{

  float xp = sth.x*Cang - sth.y*Sang;

  sth.y    = sth.x*Sang + sth.y*Cang;

  sth.x    = xp;

  // sth.z = sth.z;

  sth = normalize( sth );

  vec3 xyz;

  xyz.x = dot( vBTNx, sth );

  xyz.y = dot( vBTNy, sth );
```

```
  xyz.z = dot( vBTNz, sth );

  return normalize( xyz );

}

void main()

{
     if (vST.s>=0.5 && vST.t>=0.5)

       vec4 uColor = vec4 (0.2, 0.2, 0.2, 1.);

    else

       uColor = vColor;

if(uUseBumpmap)

  {

    float uBumpHeight;

    uBumpHeight = 0.005;

   vec2 st = vST.st; // locate the bumps based on (s,t)

    float Swidth  = 1. / uBumpDensity;
    float Theight = 1. / uBumpDensity;
    float numInS = floor( st.s /  Swidth );
    float numInT = floor( st.t / Theight );

    vec2 center;

    center.s = numInS * Swidth   +   Swidth/2.;
    center.t = numInT * Theight  +  Theight/2.;
    st -= center; // st is now wrt the center of the bump
```

```
	Cang = cos(uAng);
	Sang = sin(uAng);
	vec2 stp;   // st' = st rotated by -Ang
	stp.s =  st.s*Cang + st.t*Sang;
	stp.t = -st.s*Sang + st.t*Cang;
	float theta = atan( stp.t, stp.s );
	vec3 normal = ToXyz( vec3( 0., 0., 1. ) );

	if( abs(stp.s) > Swidth/4.  ||  abs(stp.t) > Theight/4. )

	{
		normal = ToXyz( vec3( 0., 0., 1. ) );
	}
	else
	{
		if( PI/4. <= theta  &&  theta <= 3.*PI/4. )
		{
			normal = ToXyz(  vec3( 0., uBumpHeight, Theight/4. )  );
		 }

		 else if( -PI/4. <= theta  &&  theta <= PI/4. )
		{
			normal = ToXyz(  vec3( uBumpHeight, 0., Swidth/4. )  );
		}
		else if( -3.*PI/4. <= theta  &&  theta <= -PI/4. )
		{
			normal = ToXyz(  vec3( 0., -uBumpHeight, Theight/4. )  );
		 }
		else if( theta >= 3.*PI/4.  ||  theta <= -3.*PI/4. )
		 {
			normal = ToXyz(  vec3( -uBumpHeight, 0., Swidth/4. )  );
		}
	}
	float intensity = uAmbient + (1.-uAmbient)*dot(normal, vDirToLight);
	vec3 litColor = uColor.rgb * intensity;
	gl_FragColor = vec4( litColor, 1. );
}
```

```
 else if(uUseReflect)
 {
  vec3 reflectcolor;
  reflectcolor= textureCube( uReflectUnit, vReflectVector ).rgb;
  gl_FragColor = vec4( reflectcolor, 1. );
 }

 else if(uUseRefract)
 {
  vec3 refractcolor;
  refractcolor = textureCube( uRefractUnit, vRefractVector ).rgb;
  gl_FragColor = vec4( refractcolor, 1. );
 }
 else
 {
  vec3 normLight = normalize(vLight);
  vec3 normEye   = normalize(vEye);
  vec4 ambient = uKa * uColor;
  float cd = max(dot(vNormal,normLight), 0.);
  vec4 diffuse = uKd * cd * uColor;
  float cs = 0.;
  if (dot(vNormal,normLight) > 0.)
  {
   vec3 ref = normalize(2. * vNormal * dot(vNormal,normLight) - normLight);
   cs = pow(max(dot(normEye,ref), 0.), 10.);
  }
  vec4 specular = uKs * cs * vec4 (1,0.6,0,1);
  vec4 Lightcolor =vec4(ambient.rgb + diffuse.rgb + specular.rgb, 1);

 //fog
  float fogDensity = 0.;
  vec3 sampleLoc = MCposition;
  fogDensity += float(texture3D(Noise3, 1*MCposition + 2*Timer)/(uFogStep));
  sampleLoc += normEye/uFogStep;
  gl_FragColor = mix(Lightcolor, vec4(0.7,0.7,0.7,1), fogDensity);
 }
}
```

texture.vert:

```
#version 330 compatibility
out vec2 vST;

void
main( )
{
	vST = gl_MultiTexCoord0.st;
	gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

texture.frag:

```
#version 330 compatibility
in vec2 vST;
uniform sampler2D TexUnit;
void
main( )
{
	vec3 newcolor = texture2D( TexUnit, vST ).rgb;
	gl_FragColor = vec4( newcolor.rgb, 1. );
}
```

(4) What I did and Reasons

First step is creating a cube-mapping environment. So, I used six pictures I found on the Internet and made their pixel be 2048*2048 with bmp format. Then, I applied the texture.vert file and frag file. In the gilb file, I set the TexUnit and draw six Quad, which is 8 wide. In gilb file to call the texture.vert and frag to do the cube mapping. After this, I got a cube mapping forest environment.

Next step is creating the rabbit and the Sun in the sky. Real rabbit wouldn't be one color, so, I set ST coordinates in vert shader and passed it to the frag shader. Then, I used a not clever way to find the ST coordinates of ears and back of the rabbit. The way is trying ST's range to see if it's the right place I wanted. Then, I painted ears and back of the rabbit deep gray color and other parts of rabbit are light gray. This is

the “skin” of the rabbit. Last is transforming the rabbit and the sun to the right place. The rabbit is standing in the riverside.

For the reflection and refraction part, I used two checkboxes to choose if using these two functions. In gilb file I defined uReflectUnit 7 and uRefractUnit 8 to store the information of pictures. And these 6 pictures are same as the pictures rendered walls. In vert shader, I used reflect() and refract() functions to create vReflectVector and vRefractVector and passed them to frag shader. In fragment shader, I assigned uReflectUnit and uRefractUnit. After this step, I got the reflected rabbit and refract rabbit.

I used bump mapping method to create some decorative patterns in the skin of the rabbit. I made some heights sticking up from skin. In vert shader, I defined a surface local coordinate system at each fragment with components N, T, B. In fragment shader, I created normal by the value of vLightDir. Then, transforming them to Eye Coordinates.

Lighting part I only chose to use per-fragment lighting, because I need to change the lighting according to the time of the day later. Doing ambient, diffuse, and specular lighting and only doing specular if the light can see the point. Last step is passing vec4(ambient.rgb + diffuse.rgb + specular.rgb, 1.) to the gl_FragColor. Then I could apply lighting, and then I transformed the point light to the sphere position. The sphere becomes the sun.

Next, I used a slide variable Time to simulate the change of time in a day. I used lightx = uLightX + uDayTime to change the position of the sun light. For example, when Time is small, it’s in the morning and the sun is in the east.

For the morphing part, I used once in proect6. First I set a point in the screen, it’s in the river and the rabbit would disappear from this point as it involved in a black hole. I use the formula F/d^2, d is the distance between gl_Vertex and the point I set.

This formula is similar to the Universal Gravitation Formula. But I simplified it by treating F of rabbit as 1. When I increased the F of black hole, the block hole would pull the rabbit in. I used mix(gl_Vertex,vec4(uX,uY,uZ,1.), F/(dis*dis)) function to make the “involving” effect.

Last part is fog effect. When applying this effect, the rabbit is like in a foggy forest. We can only see blurry skin of the rabbit. I did it in Model coordinates. In fragment shader, I defined the fog density. I used texture3D(Noise3, 1*MCposition + 2*Timer)/(uFogStep) to add noise in the fog density. The amplitude of the noise is set as 1. Also, I add the Timer function to make noise change from 0-1 in 10 seconds. This would make the fog move around the rabbit, which is more realistic.

(5) Results

Original Cube mapping and the rabbit:

Reflection and Refraction:

(Reflection)

(Refraction)

Bump mapping skin:

(Bump mapped skin)

Morning, noon, and evening:

(Morning)

(Noon)

(Evening)

Morphing by Black hole:

(When gravitation of black hole continues to increase)

When the gravitation of black hole continues to increase, the rabbit would disappear:

(Rabbit is involved in the black hole)

Fog effect, but static picture cannot show the moving fog effect:

(Rabbit in the Fog)

Through the fog to see the sun:

(The sun in the fog)