Computer Graphics HW3 Report

310551067吳岱容

- Implementation
  - Sky Box

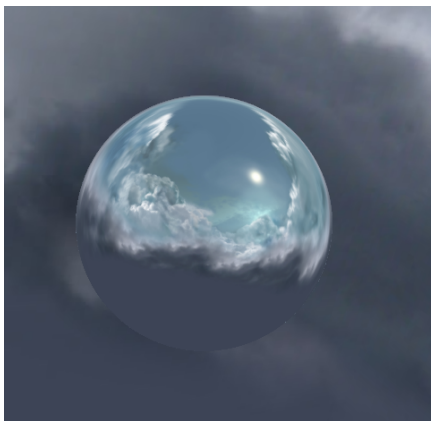
```
mat4 newView = mat4(mat3(view));
vec4 tmp = projection * newView * vec4(position_in, 1.0);
```

We did most of the implementation of the sky box in its vertex shader. In the vertex shader, we need to set gl_Position. Since we don't want the sky box to move when we move the camera, we need to remove the translation matrix from the view matrix. In order to do this, we only need to transform the view matrix into mat3 type and turn it back to mat4 type.

```
gl_Position = tmp.xyww;
```

Since the sky box needs to be infinitely far, we let gl_Position be tmp.xyww.

  - Environment Mapping




(Visual effect under different parameters)

```
vec3 Eta = vec3(1/ 1.39, 1 / 1.44, 1 / 1.47);
vec3 N = normalize(fs_in.normal);
vec3 I = normalize(fs_in.position-fs_in.viewPosition);
vec3 R = reflect(I,N);
vec3 TRed = refract(I,N,Eta.x);
vec3 TGreen = refract(I,N,Eta.y);
vec3 TBlue = refract(I,N,Eta.z);
```

We need to implement Fresnel effect on the sphere in this part. To implement this effect we need to consider reflection, refraction and chromatic dispersion. We can use the reflection function to get the

reflection direction. As for refraction, since we need to implement chromatic dispersion, and different colors have their own refraction rate, we need to calculate them separately.

```
vec4 reflectedColor = texture(skybox,R);
vec4 refractedColor = vec4(texture(skybox,TRed).r,texture(skybox,TGreen).g,texture(skybox,TBlue).b,0.75);
float fresnelIdx = clamp(fresnelBias + fresnelScale * pow(1 + dot(I, N), fresnelPower), 0.0, 1.0);
vec4 finalColor = mix(refractedColor,reflectedColor,fresnelIdx);
```

Then, we use the direction we get to query the texture color. Finally we mix the refraction and reflection effect.

- Normal Mapping

(Result)

```
const float delta = 0.01;
float x = gl_FragCoord.x;
float y = gl_FragCoord.y;

float x1 = x-delta;
float x2 = x+delta;
float y1 = y-delta;
float y2 = y+delta;
float z1 = sin(offset-0.1*y1);
float z2 = sin(offset-0.1*y2);

vec3 n1 = cross(vec3(0,y2-y1,z2-z1),vec3(x2-x1,0,0));
vec3 n2 = cross(vec3(0,y1-y2,z1-z2),vec3(x1-x2,0,0));
vec3 n =(n1+n2)/2;
normal = vec4(n*0.5+0.5,1.0);
```

We need to generate a sine-wave-like normal map in the fragment shader. First, we can get the fragment position by calling gl_FragCoord, and then we form two triangles by slightly moving the x and y positions. We can get the surface normal from these two triangles. In this part, we use the sine function to calculate z values (depth), and we get a sine-wave-like normal map.

```
vec3 T = normalize(vec3(modelMatrix*vec4(tangent_in,0.0)));
vec3 B = normalize(vec3(modelMatrix*vec4(bitangent_in,0.0)));
vec3 N = normalize(vec3(modelMatrix*vec4(normal_in,0.0)));

mat3 TBN = transpose(mat3(T,B,N));
vs_out.lightDirection = TBN*lightDirection;
vs_out.viewPosition = TBN*vec3(viewPosition);
vs_out.position = TBN*position_in;
```

To transform the light direction, view position, and position to the tangent space, we need to know the inverse of the tangent space transform matrix. We can get the tangent space transform matrix from multiplying the model matrix with the given parameters, and we can get the inverse matrix by transposing it because it is orthogonal.

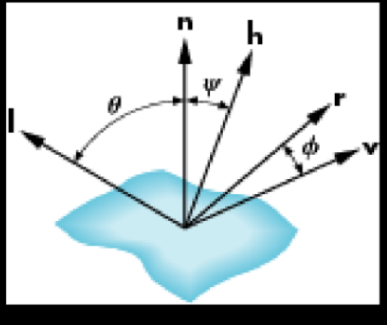
```
vec4 normal = texture(normalTexture,textureCoordinate);
vec4 tmp = normalize((normal-0.5)/5);
normal = tmp;
diffuse = 0.75*max(dot(fs_in.lightDirection,vec3(normal)),0.0);
vec3 viewDir =normalize(fs_in.position-fs_in.viewPosition);
vec3 halfway = normalize(fs_in.lightDirection + viewDir);
specular = 0.75*pow(max(dot(normalize(vec3(normal)),halfway),0.0),8.0);


float lighting = ambient + diffuse + specular;
FragColor = vec4(lighting * diffuseColor, 1.0);
```

In the fragment shader, we finish Blin-Phong shading. It is very similar to what we have done in HW2.

$$I = I_{ambient} + I_{diffuse} + I_{specular}$$
$$= k_a L_a + k_d (l \cdot n) L_d + k_s L_s (v \cdot r)^{\alpha}$$

We use the formula above to calculate the diffuse and specular colors.

$$h = (l + v)/|l + v|$$



In specular color, we use the halfway direction to replace (v·r).

- Problems and Discussion
    - Problems
        - Sine-wave-like normal mapping

            It took me some time to understand this task. At first, I finished the Blinn-Phong shading but the sine wave didn't show correctly, and then I realized that the z values I had assigned were wrong.
        - Texture problem

            Sometimes, the texture doesn't show correctly. This question was posted in the forum on E3. I have encountered the same issue. However, most of the time, the causes are very simple. The problem can be caused by the shaders aren't implemented in the right way, and the compiler fails to compile. This will cause the texture problems.
    - Discussion

        We have learnt advanced mapping in the homework this time. I think it is fun and felt very good when I finished the Fresnel effect and saw the pretty glass effect. Also we have learnt normal mapping, and I think it is very useful when we need to render landscape texture.

        We have learnt the concept in class, but through homework, we can get more familiar with these techniques.