Computer Graphics Assignment 3

Student ID: 312551077
Name: 薛祖恩

Implementation

1. TODO#1

```
void main() {
  textureCoordinate = position_in;
  // TODO1: Set gl_Position
  // Hint:
  //   1. We want skybox infinite far from us. So the z should be 1.0 after perspective division.
  //   2. We don't want the skybox moving when we move. So the translation in view matrix should be removed.

  gl_Position = (projection * mat4(mat3(view)) * vec4(position_in.x, position_in.y, position_in.z, 1.0)).xyww;
}
```

```
void main() {
  // TODO1:

  FragColor = texture(skybox, textureCoordinate);
}
```

The skybox to be infinite far from us, thus the parameter should be set to 1.0. Moreover, I should remove the translation in the view matrix since I want the skybox be fixed. After the *.xyww*, the w should be 1.

For the fragment shader, I use the texture function to setup the fragment color of the skybox.

2. TODO#2

```
vec3 I = normalize(fs_in.position - fs_in.viewPosition);
vec3 N = normalize(fs_in.normal);
vec3 R = reflect(I, N);
vec3 refractColor = vec3(0);

for (int i = 0; i < 3; i++) {
  vec3 T = refract(I, N, Eta[i]);
  refractColor += texture(skybox, T).rgb;
}
refractColor /= 3.0;

vec3 reflectColor = texture(skybox, R).rgb;
float fresnel = clamp((1 - fresnelBias) + fresnelScale * pow(1.0 + dot(I, N), fresnelPower), 0.0, 1.0);

vec3 finalColor = mix(reflectColor, refractColor, fresnel);

FragColor = vec4(finalColor, 1.0);
```

This shader code implements the fresnel refraction and fresnel reflection. First, I calculate the normalized incident light direction vector and normalize the surface normal vector, then the reflected light direction can be calculated using *reflect* function. Next, I iterate through 3 channels to compute refracted colors using refract function and add to *refractColor*. At last, I calculate the Fresnel by the formula.

3. TODO#3

```
vec2 fragCoord = gl_FragCoord.xy;

vec2 dx = vec2(delta, 0.0);
vec2 dy = vec2(0.0, delta);
vec3 p0 = vec3(fragCoord, sin(offset - 0.1 * fragCoord.y));
vec3 p1 = vec3(fragCoord + dx, sin(offset - 0.1 * (fragCoord.y + delta)));
vec3 p2 = vec3(fragCoord + dy, sin(offset - 0.1 * (fragCoord.y + delta)));
vec3 p3 = vec3(fragCoord + dx + dy, sin(offset - 0.1 * (fragCoord.y + delta)));

vec3 normal1 = normalize(cross(p1 - p0, p2 - p0));
vec3 normal2 = normalize(cross(p3 - p1, p2 - p1));

vec3 avgNormal = normalize(normal1 + normal2) * 0.5 + 0.5;

height = (p0.z + p1.z + p2.z + p3.z) * 0.25 * 0.5 + 0.5;

normal = vec4(avgNormal, 1.0);
```

For this vertex shader file, I fetch the screen space position of the fragment, then I declare two variables to record the bias of x and y, and create 4 points to store the 4 points separately. The next step is the calculate the normal vector by the 4 points, the cross value generates a triangle, the normal vector must be normalized. Moreover, I calculate the *avgNormal* from range [-1, 1] to [0, 1], for the height, I calculate the arithmetic average and then map it from [-1, 1] to [0, 1].

4. TODO#4

```
vec3 T = normalize(vec3(modelMatrix * vec4(tangent_in, 1.0)));
vec3 B = normalize(vec3(modelMatrix * vec4(bitangent_in, 1.0)));
vec3 N = normalize(vec3(modelMatrix * normalMatrix * vec4(normal_in, 1.0)));
mat3 TBN = mat3(T, B, N);

vs_out.lightDirection = TBN * lightDirection;
vs_out.viewPosition = TBN * viewPosition.xyz;
vs_out.position = TBN * position_in;

vs_out.textureCoordinate = textureCoordinate_in;
vec3 displacementVector = vec3(0);
if (useDisplacementMapping) {
  // TODO (Bonus-Displacement): Set displacementVector, you should scale the height query from heightTexture by depthScale.

}
gl_Position = viewProjectionMatrix * (modelMatrix * vec4(position_in + displacementVector, 1.0));
```

```
vec3 normal = texture(normalTexture, textureCoordinate).rgb;
normal = normalize(normal * 2.0 - 1.0);

float ks = 0.75;
float kd = 0.75;
float shininess = 8.0;

vec3 halfwayDirection = normalize(fs_in.lightDirection + viewDirectionection);
float specular = ks * pow(max(dot(normal, halfwayDirection), 0.0), shininess);
float diffuse = kd * max(dot(normal, fs_in.lightDirection), 0.0);

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

For the first figure, which is the vert file, the Tangent-Bitangent-Normal (TBN) matrix is computed by transforming the tangent, bitangent, and normal vectors between object

space and world space. This matrix is for converting the light direction, view position, and vertex position into tangent space. Finally, the transformed and displaced vertex position is computed in view space and then transformed to clip space using the view projection matrix and model matrix.
For the frag file, the normal vector is sampled from the normal texture and converted from RGB [0, 1] to normal coordinates in the range [-1, 1]. Then I apply Blinn-Phong Shading by parameter kd = ks = 0.8 and shininess = 0.8. The halfway direction between the light direction and view direction is calculated, and specular and diffuse components are computed using the Blinn-Phong shading equations. Lastly, we set the lighting to be the sum of ambient, diffuse, and specular.

Bonus
The height in TODO3 is modified by calculating the arithmetic average of z of the four points of the two triangles, then map it from [-1, 1] to [0, 1]. The implementation are as follows:

$$\text{height} = (\text{p0.z} + \text{p1.z} + \text{p2.z} + \text{p3.z}) * 0.25 * 0.5 + 0.5;$$

Compare sphere and cube with two scenes
For the sphere, it can more accurately and comprehensively reflect the appearance of the skybox. On the other hand, for the cube, it will only display the appearance of the skybox facing each of its six faces.

Regardless of whether it's a sphere or a cube, the higher the Fresnel bias and Fresnel power, the clearer the reflection will be. A larger Fresnel scale, on the other hand, will result in a less distinct reflection. To be more precise, when scale and power is fixed, changes in bias may influence the distribution of reflectance; when scale and bias is fixed, changes in power may adjust the variation in reflectance near the incident angle; when bias and power is fixed, changes in scale may affect the slope of the Fresnel equation, consequently influencing the rate of change of reflectance with incident angle.

Problems I encountered
1. The ".xyww" in TODO1

If ".xyww" is not used, the sphere and the plane will be blocked, the reason is because the ".xyww" makes z to be 1. I thought the 1.0 in vec4 was the meaning "set z to 1" but I was wrong. By applying this method, I can finally see the plane and the sphere.

2. The formula in TODO2

The formula mentioned in the note is:

$$\text{clamp}(\text{fresnelBias} + \text{fresnelScale} * \text{pow}(1 + \text{dot}(I, N), \text{fresnelPower}), 0.0, 1.0);$$

Unfortunately, if the implementation follows the formula above, the effect of the bias will be opposite, therefore, I changed the bias from “fresnelBias” to “1-fresnelBias” to solve the issue.

3. The normal vector of the triangle in TODO3

The parameters of normal vector of the triangle calculated in calculatenormal.frag matters if the normal vector of the two triangles are not in the same direction, if the direction is not the same or the two parameters swap, the output of the normal map will not be the same as the normal shown in the class video. It tool me a while to try and error and finally solve the problem.

4. Calculating TBN in TODO4

At the beginning, I implement the TBN matrix all on my own without finding any references on the internet, there are some mistakes made by me, and it took me a long time to debug since for TODO4, two files must be implemented correctly to see the result. After finding the examples on learnopengl.com, I recognized the code I implement was totally wrong. Then I implement the code by seeing the examples in the website mentioned above, after that, the issue is perfectly solved.

Results