ICG HW3 Report

111550108 吳佳諭

1. Bling-Phong Shader

Vertex shader changes normal from model coordinate to world coordinate, then pass texture coordinate, position, and normal to fragment shader.

Fragment shader first transforms the position from model coordinate to world coordinate and calculate the light direction and view direction to get the half way vector. Apply Phong shading model with the half way vector to calculate the final color, where $Ambient = L_{ambient} \times K_{ambient}$, $Diffuse = L_{diffuse} \times K_{diffuse} \times (L \cdot N)$, $Specular = L_{speculaar} \times K_{specular} \times (N \cdot H)^{a}$.

*Fragment shader

```
vec4 texcolor = texture(ourTexture, TexCoord);

vec3 lightDir = normalize(lightPos - FragPos);
vec3 viewDir = normalize(viewPos - FragPos);
vec3 halfway = normalize(lightDir + viewDir);

vec3 ambient = lightAmbient * materialAmbient * texcolor.rgb;
vec3 diffuse = lightDiffuse * materialDiffuse * max(dot(lightDir, Normal), 0.0) * texcolor.rgb;
vec3 specular = lightSpecular * materialSpecular * pow(max(dot(Normal, halfway), 0.0), materialGloss);

vec3 tot_color = ambient + diffuse + specular;
FragColor = vec4(tot_color, texcolor.a);
```

2. Gouraud Shader

In vertex shader, it transforms the normal and the position from model coordinate to world coordinate and calculate the light direction and view direction. Gouraud shading calculate the color per pixel, so it calculates the final color in vertex shader, where $final\ color = ambient + diffuse + specular$.

In fragment shader, it gets the colors passed from vertex shader. It then combines the colors and texture to output FragColor.

*Vertex shader

```
vec4 worldPos = model * vec4(aPos, 1.0);
vec3 normal = normalize(mat3(transpose(inverse(model))) * aNormal);

vec3 lightDir = normalize(lightPos - worldPos.xyz);
vec3 viewDir = normalize(viewPos - worldPos.xyz);

ambient = lightAmbient * materialAmbient;
diffuse = lightDiffuse * materialDiffuse * max(dot(lightDir, normal), 0.0);
specular = lightSpecular * materialSpecular * pow(max(dot(viewDir, reflect(-lightDir, normal)), 0.0), materialGloss);

// tot_color = ambient + diffuse + specular;

gl_Position = projection * view * model * vec4(aPos, 1.0);
TexCoord = aTexCoord;
```

3. Environment Cubemap

Assign cube map's texture color to the output. Notice that we divide the gl_Position's xyz coordinate by w.

```
TexCoord = aPos;
vec4 pos = projection * view * vec4(aPos, 1.0);
gl_Position = pos.xyww;
```

cubemap.vert

```
FragColor = texture(ourTexture, TexCoord);
```

cubemap.frag

4. Metallic Shader

Vertex shader transforms variables into world coordinate and passes them to the fragment shader. Fragment shader calculate $C_{final} = \alpha \times B \times C_{model\ color} + (1 - \alpha)C_{reflect}$.

In fragment shader, we first define some constants like bias, alpha, and light intensity and calculate some util vectors like lightDir, viewDir, and reflectDir. Notice that we don't call reflect() to calculate the reflect light, but calculate it by $R = I - 2 \cdot (I \cdot N) \cdot N$. Then we calculate the diffuse light by $B = B_d + bias$, where $B_d = \max((L \cdot N)I_l, 0)$. Finally, we can mix the reflect color and the original color by using $C_{final} = \alpha \times B \times C_{model\ color} + (1 - \alpha) \times C_{reflect}$, where model color is obtained by $(ambient + diffuse + specular) * texture\ color$, and reflect color is the color of the cubemap.

```
float bias = 0.2;
float alpha = 0.4;
float lightIntensity = 1.0;

vec4 texcolor = texture(ourTexture, TexCoord);

vec3 lightDir = normalize(lightPos - FragPos);
vec3 viewDir = normalize(viewPos - FragPos);
// vec3 reflectDir = normalize(lightDir - 2.0 * dot(lightDir, Normal) * Normal);
vec3 reflectDir = normalize(-viewDir - 2.0 * dot(-viewDir, Normal) * Normal);

// diffuse
float Bd = max(dot(lightDir, Normal), 0.0) * lightIntensity;
float B = Bd + bias;

// specular & ambient
vec3 ambient = lightAmbient * materialAmbient;
vec3 specular = lightSpecular * materialSpecular * pow(max(dot(viewDir, reflectDir), 0.0), materialGloss);

vec3 tot_color = ambient * texcolor.rgb + specular + B * texcolor.rgb;

vec3 reflect_color = texture(cubeMap, reflectDir).rgb;
vec3 C_final = alpha * B * tot_color + (1.0 - alpha) * reflect_color;

FragColor = vec4(C_final, texcolor.a);
```

5. Glass Shader

(1) Schlick

Once again, the vertex shader transforms the coordinate to world coordinate and pass data to fragment shader. Fragment shader calculate final color with $C_{final} = R_\theta C_{reflect} + (1 - R_\theta)C_{refract}$, where $R_\theta = R_0 + (1 - R_0) \times (1 + I \cdot N)^5$ and $R_0 = (\frac{n1-n2}{n1+n2})^2$.

In fragment shader, we calculate the refract direction T using $T = \eta I - (\eta(I \cdot N) + \sqrt{1 - \eta^2(1 - (I \cdot N)^2)})N$. If the result inside the square root is smaller than 0, then we set T to (0, 0, 0). The reflect color is texture(cubeMap, reflectDir), and the refract color is texture(cubeMap, T), then we can calculate the final color by the above formula.

```
float R0 = pow((GLASS_coeff - AIR_coeff) / (AIR_coeff + GLASS_coeff), 2);
float R_theta = R0 + (1 - R0) * pow((1.0 + dot(-viewDir, Normal)), 5);

float K = 1 - n * n * (1 - pow(dot(-viewDir, Normal), 2));
vec3 T;
if(K < 0){
    T = vec3(0.0, 0.0, 0.0);
}
else{
    T = n * (-viewDir) - (n * dot(-viewDir, Normal) + pow(K, 1/2)) * Normal;
}

vec3 reflect_color = texture(cubeMap, reflectDir).rgb;
vec3 refract_color = texture(cubeMap, T).rgb;

vec3 C_final = R_theta * reflect_color + (1 - R_theta) * refract_color;
// vec3 C_final = texcolor.rgb;
FragColor = vec4(C_final, texcolor.a);
```

(2) Empirical

The only different in this method and the previous one is the way they calculate R_θ. Here we have $R_\theta = \max(0, \min(1, bias + scale \times (1 + I \cdot N)^{power}))$. We calculate it in the fragment shader and mix the reflect color and refract color to get the final color.

```
float R_theta = max(0.0, min(1.0, bias + scale * pow((1 + dot(-viewDir, Normal)), power)));
```