计算机学院实验报告

实验题目：实验六：Assignment3		学号：202000120101
日期：2022.12.19	班级：20.2	姓名：尹国泰
Email：1018693208@qq.com		

实验目的：

1.实现Blinn-Phong 模型计算Fragment Color.

2.在实现Blinn-Phong的基础上，实现Texture Shading Fragment Shader.

3.在实现Blinn-Phong 的基础上，实现Bump mapping.

4.在实现Bump mapping 的基础上，实现displacement mapping.

实验环境介绍：

操作系统：Window10

编译器环境：MinGW，VSCode

OpenGL 环境：freeglut

Opencv2

Eigen3

解决问题的主要思路：

1. 在phong_fragment_shader()函数中，需要计算环境光颜色ambient，漫反射光颜色diffuse，镜面反射光颜色specular后，将三者加到物体本身的颜色中，作为呈现出来的颜色Fragment Color

具体来说，可以使用如下公式来计算ambient，diffuse 和 specular的颜色

（1）ambient = ka * ambientLightColor

（2）diffuse = kd * lightColor * max(dot(normal, lightDirection), 0.0)

（3）specular = ks * specularColor * pow(max(dot(viewDirection, reflect(-lightDirection, normal)), 0.0), specularExponent)

其中，ka，kd，ks 分别表示环境光反射系数，漫反射反射系数，镜面反射系数，ambientLightColor 表示环境光的颜色，lightColor 表示光源的颜色，normal 表示表面法线，lightDirection 表示光源的方向，viewDirection 表示视线方向，specularColor 表示镜面反射的颜色，specularExponent 表示镜面反射的指数。

2. 在texture_fragment_shader()中，需要首先textur.png中模型对应的颜色填充到texture_color，映射关系在加载模型时已经设置在tex_coords中，再将漫反射系数kd设置为texture_color，然后用phong_fragment_shader中的方法计算光照影响，得到最终的Fragment Color
3. 在bump_fragment_shader()中，根据注释内容实现代码，对表面法线

normal进行修改，实现模拟凹凸表面细节的技术

```
    // TODO: Implement bump mapping here
    // Let n = normal = (x, y, z)
    // Vector t =
(x*y/sqrt(x*x+z*z),sqrt(x*x+z*z),z*y/sqrt(x*x+z*z))
    // Vector b = n cross product t
    // Matrix TBN = [t b n]
    // dU = kh * kn * (h(u+1/w,v)-h(u,v))
    // dV = kh * kn * (h(u,v+1/h)-h(u,v))
    // Vector ln = (-dU, -dV, 1)
    // Normal n = normalize(TBN * ln)
```

4. 在displacement_fragment_shader()中，相较于bump_fragment_shader()多了一步对物体顶点point重新计算的步骤

```
    // Position p = p + kn * n * h(u,v)
```

这可以看做是对于物体表面顶点的位移，实际对物体形状进行改变，从而模拟凹凸细节，而在bump mapping只是调整表面法线来模拟凹凸效果，并没有对物体形状进行实际改变

实验步骤：

1.在 phong_fragment_shader()函数中，修改要求实现的内容，计算得出 ambient, diffuse, specular 后，再计算出最终的 Fragment Color.

```
for (auto& light : lights)
{
    // TODO: For each light source in the code, calculate what the *ambient*, *diffuse*, and *specular*
    // components are. Then, accumulate that result on the *result_color* object.
    Vector3f ambient(0,0,0);
    Vector3f diffsue(0,0,0);
    Vector3f specular(0,0,0);

    Vector3f light_dir =  (light.position -point).normalized();
    Vector3f view_dir = (eye_pos - point).normalized();
    Vector3f h = (view_dir + light_dir).normalized();
    float rr =  ( light.position -point).squaredNorm();
    for (size_t i = 0; i < 3; i++)
    {
        ambient[i] = amb_light_intensity[i] * ka[i];
        diffsue[i] = kd[i] * (light.intensity[i]/rr) * std::max(0.0f,normal.dot(light_dir));
        specular[i] = ks[i] * (light.intensity[i]/rr) * std::pow(std::max(0.0f,normal.dot(h)),p);
    }
    result_color += ambient;
    result_color += diffsue;
    result_color += specular;
}
```

2. 修改函数texture_fragment_shader() in main.cpp: 在实现Blinn-Phong的基础上，将纹理颜色视为公式中的kd，实现Texture Shading Fragment Shader。

首先获取模型对应的材质颜色 texture_color

映射关系在加载模型时已经设置在 tex_coords 中

```
if (payload.texture)
{
    // TODO: Get the texture value at the texture coordinates of the current fragment
    return_color = payload.texture->getColor(payload.tex_coords.x(),payload.tex_coords.y());
}
Eigen::Vector3f texture_color;
texture_color << return_color.x(), return_color.y(), return_color.z();
```

然后将漫反射系数 kd 设置为 texture_color

```
Eigen::Vector3f kd = texture_color / 255.f;
```

计算光照对 Fragment Color 的部分与 phong_fragment_shader 函数中的完全相同

3. 修改函数bump_fragment_shader() in main.cpp: 在实现Blinn-Phong 的基础上，仔细阅读该函数中的注释，实现 Bump mapping.
依据注释中的内容，重新计算 normal 的值，实现凹凸贴图

```
    // TODO: Implement bump mapping here
    // Let n = normal = (x, y, z)
    // Vector t =
(x*y/sqrt(x*x+z*z),sqrt(x*x+z*z),z*y/sqrt(x*x+z*z))
    // Vector b = n cross product t
    // Matrix TBN = [t b n]
    // dU = kh * kn * (h(u+1/w,v)-h(u,v))
    // dV = kh * kn * (h(u,v+1/h)-h(u,v))
    // Vector ln = (-dU, -dV, 1)
    // Normal n = normalize(TBN * ln)

    Vector3f n = normal;
    float x = normal.x();
    float y = normal.y();
    float z = normal.z();
    Vector3f t(x*y/sqrt(x*x+z*z),sqrt(x*x+z*z),z*y/sqrt(x*x+z*z));
    Vector3f b = n.cross(t);

    Matrix3f TBN;
    TBN.col(0) = t.normalized();
    TBN.col(1) = b.normalized();
    TBN.col(2) = n.normalized();

    int w = payload.texture->width;
    int h = payload.texture->height;
    float u = payload.tex_coords.x();
    float v = payload.tex_coords.y();
```

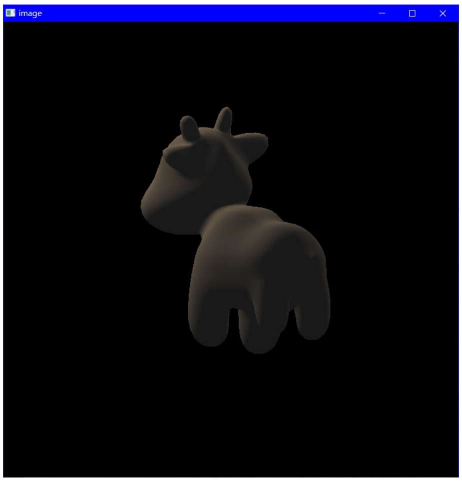
```
    float dU = kh * kn * (height(u+1.0/w,v,payload)-
height(u,v,payload));
    float dV = kh * kn * (height(u,v+1.0/h,payload)-
height(u,v,payload));

    Vector3f ln(-dU,-dV,1);
    normal = (TBN * ln).normalized();
```

4. 修改函数displacement_fragment_shader() in main.cpp: 在实现Bump mapping 的基础上，实现displacement mapping.
依据注释中的内容，写出代码，对normal重新赋值，与bump_fragment_shader（）函数中的内容整体相同，只是多了一步对物体的顶点坐标point重新赋值的操作，可认为该算法对于模型的凹凸进行了实际修改

```
// TODO: Implement displacement mapping here
    // Let n = normal = (x, y, z)
    // Vector t =
(x*y/sqrt(x*x+z*z),sqrt(x*x+z*z),z*y/sqrt(x*x+z*z))
    // Vector b = n cross product t
    // Matrix TBN = [t b n]
    // dU = kh * kn * (h(u+1/w,v)-h(u,v))
    // dV = kh * kn * (h(u,v+1/h)-h(u,v))
    // Vector ln = (-dU, -dV, 1)
    // Position p = p + kn * n * h(u,v)
    // Normal n = normalize(TBN * ln)
    Vector3f n = normal;
    float x = normal.x();
    float y = normal.y();
    float z = normal.z();
    Vector3f t(x*y/sqrt(x*x+z*z),sqrt(x*x+z*z),z*y/sqrt(x*x+z*z));
    Vector3f b = n.cross(t);

    Matrix3f TBN;
    TBN.col(0) = t.normalized();
    TBN.col(1) = b.normalized();
    TBN.col(2) = n.normalized();

    int w = payload.texture->width;
    int h = payload.texture->height;
    float u = payload.tex_coords.x();
    float v = payload.tex_coords.y();
```

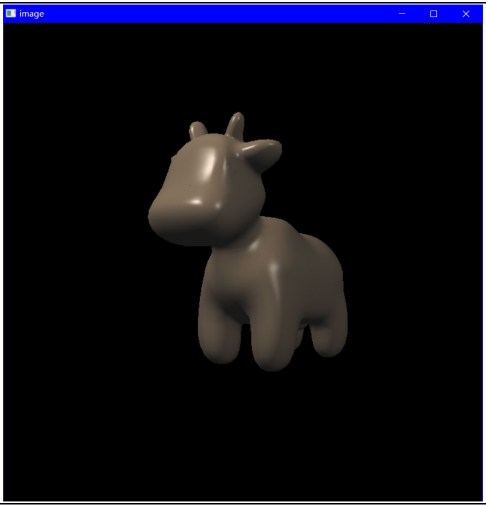
```
    float dU = kh * kn * (height(u+1.0/w,v,payload)-height(u,v,payload));
    float dV = kh * kn * (height(u,v+1.0/h,payload)-height(u,v,payload));

    Vector3f ln(-dU,-dV,1);
    point = point +  kn * n * height(u,v,payload);

    normal = (TBN * ln).normalized();
```

5. 经过测试，发现呈现的图片中模型是头朝后的，与指导书里的预期结果并不相符，尝试对计算投影矩阵的 `get_projection_matrix` 函数进行修改



修改后的代码如下，对两处的符号进行修改，修改部分原内容见注释

```
Eigen::Matrix4f get_projection_matrix(float eye_fov, float aspect_ratio, float zNear, float zFar)
{
    // TODO: Use the same projection matrix from the previous assignments
    Eigen::Matrix4f projection = Eigen::Matrix4f::Identity();
    Matrix4f squish;
    squish << zNear, 0, 0, 0,
        0, zNear, 0, 0,
        0, 0, zNear + zFar, (-1.0 * zNear * zFar),
        0, 0, 1, 0;          // 0, 0, -1, 0
    projection = squish * projection;

    Matrix4f ortho_translate, ortho_scale;
    double halfEyeRadian = eye_fov * MY_PI / 2 / 180.0;
    double top = -zNear * tan(halfEyeRadian); // zNear * tan(halfEyeRadian)
    double bottom = -top;
    double right = top * aspect_ratio;
    double left = -right;
    ortho_translate << 1, 0, 0, -(right + left) / 2,
        0, 1, 0, -(top + bottom) / 2,
        0, 0, 1, -(zNear + zFar) / 2,
        0, 0, 0, 1;
    ortho_scale << 2 / (right - left), 0, 0, 0,
        0, 2 / (top - bottom), 0, 0,
        0, 0, 2 / (zNear - zFar), 0,
        0, 0, 0, 1;
    projection = ortho_scale * ortho_translate * projection;
    return projection;
}
```



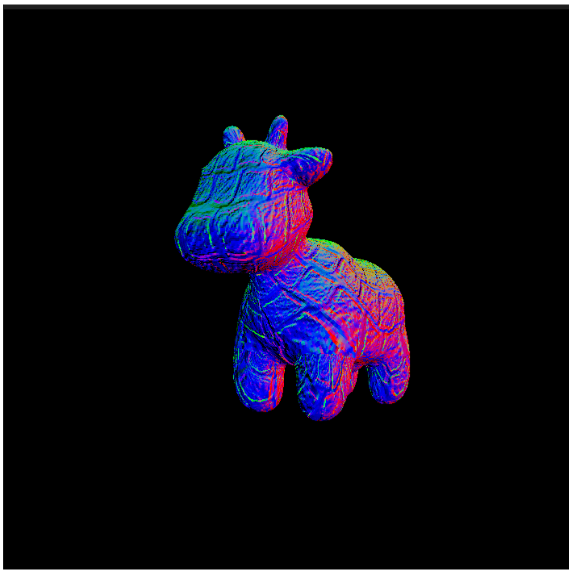
实验结果展示及分析：

1. normal（法向量模型）

2. phong 模型

3. Texture Shading Fragment Shader.

4. Bump mapping.



5. displacement mapping.

实验中存在的问题及解决:

1. 在编译时，出现以下错误

```
C:\Users\Sunstrider\Desktop\opengl\Assignment3\Code\rasterizer.hpp:106:14: error: 'optional' in namespace 'std' does not name a template type
         std::optional<Texture> texture;
              ^~~~~~~~
```

这是因为 optional 是在 C++17 中才出现的关键字，而编译时使用 C++11 版本，在编译时添加-std=c++17 参数即可

2. 生成图片时，发现模型并不是预期的那样朝前，而是头朝后，该怎么修改？

这是因为获取投影矩阵的函数 get_projection_matrix()并没有如预期的指导书中的结果那样计算投影矩阵，对其进行一些修改后可以实现实验指导书中的预期结果，以下是进行修改的内容，注释部分为原来的代码

```
//  squish << zNear, 0, 0, 0,
//      0, zNear, 0, 0,
//      0, 0, zNear + zFar, (-1.0 * zNear * zFar),
//      0, 0, -1, 0;
    squish << zNear, 0, 0, 0,
        0, zNear, 0, 0,
        0, 0, zNear + zFar, (-1.0 * zNear * zFar),
        0, 0, 1, 0;

//  double top = zNear * tan(halfEyeRadian);
    double top = -zNear * tan(halfEyeRadian);
```