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。   1. 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 2. 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) |  1. 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.    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    然后将漫反射系数kd设置为texture\_color    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 | | |
| 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    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); | | | |