计算机图形学课程报告

计算机科学与技术学院

班　　级： **CS2108**

学　　号： **I202120037**

姓　　名： **郑澍频**

指导教师： **何云峰老师**

完成日期： **2023 年 12 月 11 日**

1 简答

（1）你选修计算机图形学课程，想得到的是什么知识？现在课程结束，对于所得的知识是否满意？如果不满意，你准备如何寻找自己需要的知识。

选修计算机图形学这门课，主要是想学到怎么生成一图像，也想加深对游戏设计的理解。在课程过程中确实也学到了很多相关的知识，其中包括软件和框架的使用、图像和三维图形的生成，渲染等。除此之外，在课后作业中也浅试了一下运动插值、图形渲染、物理模拟等。这些也让我对图形学有了更深刻的理解。

在学习的过程中，我感觉自己接触到了很多新奇的知识，对我的未来增添了更多的可能性。

（2）你对计算机图形学课程中的哪一个部分的内容最感兴趣，请叙述一下，并谈谈你现在的认识。

在整个课程中，我对光中模型部分的知识印象最深刻，因为这是我认为最难的部分。光照模型分为两类，一类是局部光照模型，是本次课程中学习到的知识。局部光照模型主要考虑光照在物体表面的微观效应，忽略了光在整个场景中的全局传播。其中最经典的是冯·肖德模型（Phong Illumination Model），该模型包含三个主要部分：环境光、漫反射光、镜面反射光。环境光是由场景中的所有光源均匀发射的光线；面反射光描述的是物体表面对入射光的散射的光；镜面反射光顾名思义，就是物体表面微观凸起导致的镜面反射。这三种光照模型是最后一次实验中的重点。

另外一类，是全局光照模型。全局光照模型考虑了光在整个场景中的传播，以更准确地模拟真实世界中的光照效果。著名的全局光照算法包括光线追踪和辐射度传输。这种光照模型在课程中并没有提到很多，但是对我有着很大的吸引力，是打算之后在空闲时间里自己学习的知识。

（3）你对计算机图形学课程的教学内容和教学方法有什么看法和建议。

计算机图形学这个课堂非常的轻松，但也是我学到最多知识的一堂课。这堂课并没有其他课那样多的作业和实验，更多的是理论方面的知识，告诉我们计算机图形学可以做什么，做到什么程度。但是就是实践的部分需要自己探索。幸运的是，作业提交的时间不是很紧迫，在给我们自己探索的时间的同时，何老师的授课方式很轻松，更是提供了我们非常多的例子和配置文件。这对我们在实践方面的帮助是非常大的。

总而言之，是非常喜欢的一门课，虽然很抱歉作业可能做的不是那样的完美，但是非常开心能够在一堂课上找到了自己想深入的方向。如果可以的话，希望可以在课程的某个部分开设一节实验课带着同学们弄一个 demo 熟悉一下软件的使用就更好了。

2 论述

选择 A：实验报告：撰写实验作业的实验报告，报告需要填写的内容按照附录 1 完成。

3 课后作业

选择 B：

利用 OpenGL 框架，设计一个独孤信印章模型动画。要求如下：

（1）构建独孤信印章模型，该模型可以通过指定的外接球半径控制大小。

（2）增加纹理（凹凸纹理），可以使用木纹纹理或者其它图片纹理。

（3）构建观察空间，使用三维投影对模型进行观察，要求模型能够进行自转，通过键盘可以控制模型的大小和转速。

（4）增加光照处理，光源可以设在左上方。

（5）在上面的基础上，构建两个印章模型，一大一小，大印章自转，小印章绕着大印章旋转，要求处理好消隐。

2.1 实验内容

利用 OpenGL 框架，设计一个独孤信印章模型动画。要求如下：

（1）构建独孤信印章模型，该模型可以通过指定的外接球半径控制大小。

（2）增加纹理（凹凸纹理），可以使用木纹纹理或者其它图片纹理。

（3）构建观察空间，使用三维投影对模型进行观察，要求模型能够进行自转，通过键盘可以控制模型的大小和转速。

（4）增加光照处理，光源可以设在左上方。

（5）在上面的基础上，构建两个印章模型，一大一小，大印章自转，小印章绕着大印章旋转，要求处理好消隐。

2.2 实验方法和过程

（1）渲染模型

使用 blender 软件制作“独孤信印章”模型。该模型由 48 个棱，26 个面（其中 18 个正方形印面，8 个三角形印面），高 4.5 厘米，宽 4.35 厘米。

在 blender 软件中添加正方体，ctrl+b 快捷键进行倒角，调整模型至与上述相符，最后在模型表面添加准备好的纹理和纹理的发现贴图后进行 wavefront 格式导出保存即可。

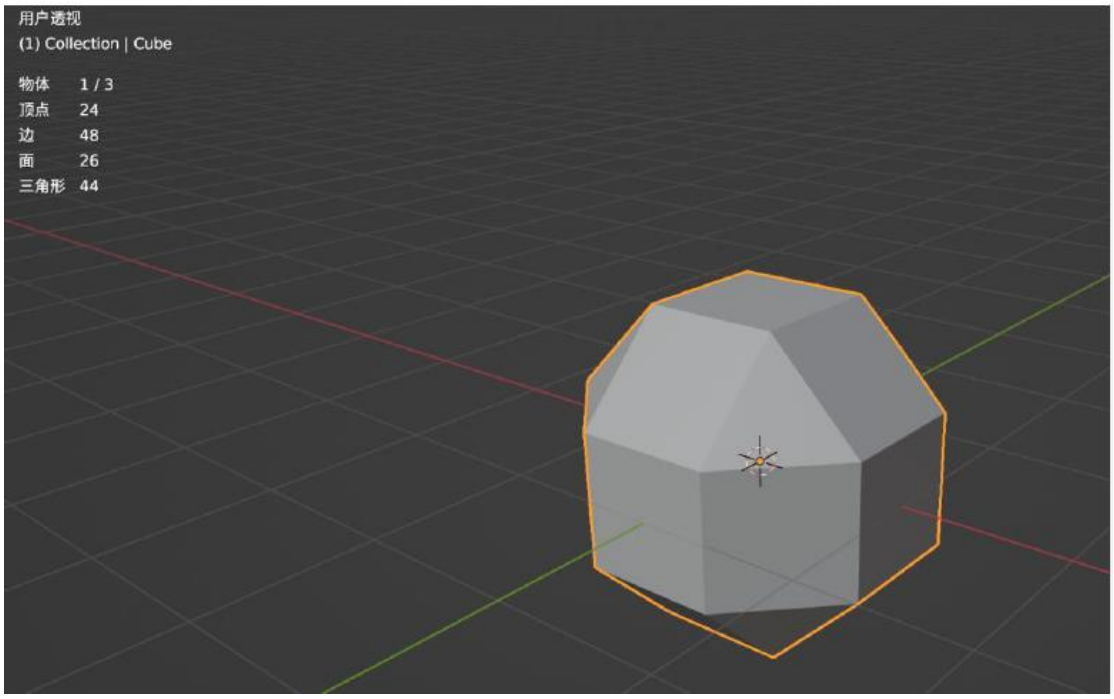

图 2.1 模型渲染

使用 Assimp 将模型从 wavefront 模式中读取顶点信息和贴图路径，再将这些数据转换成 opengl 的 VAO 和 VBO 的工具累。使用 stb 库读取和解码纹理贴图，再将数据转换成 opengl 使用的 rgb 数组。使用 modern opengl 接口渲染模型，GLSL 编写模型的顶点着色器和分段着色器。

（2）模型变换

实现缩放，将模型从 y 轴旋转 theta 度，在以 scale 倍进行缩放调节，最后再平移到模型当前坐标。

实现视图变换，使用 camera 类将当前的位置和角度记录下来，再通过 GetViewMatrix 返回对应的视图变换矩阵。使用透视投影计算矩阵的投影矩阵。最后将得到的三个矩阵相乘，即可得到 MVP 矩阵。

（3）光照模型

本次实验中使用的光照模型是 phong 模型。该模型分成了三种光照，分别是环境光，漫反射光和高光。在场景中设置两个光源，一个是上方的固定光源，另一个是类似手电筒的移动光源。

为了更贴近现实的光照，根据距离进行衰减计算。根据如下公式进行计算：

F_{att}=1.0/（K_c+K_l∗d+K_q∗d_2）

对于移动光源需要对光线角度进行一些特殊处理。比较简单的方式是将入射光线角度大于∅的反射光线衰减为 0。但是这样实现的手电筒光圈边缘十分锐利，和现实中不同。因此在∅附近需要做平滑处理，使得手电筒光圈变柔和，公式如下：

I=$Clamp$($\theta-\gamma/\emptyset-\gamma$)；

$Clamp(x)$={1 (x>1) ； x (1≥x>0)； 0 ($otherwise$)；

最后，将固定光源和移动光源叠加即可获得最终光照。

（4）凹凸纹理

在导入模型贴图时，同时加载图形和法线贴图。之前的光照模型中，默认纹理时平整的。因此需要导入法线贴图，在此处的法向量时是贴图的法向量。在贴图贴上后会出现一个初始方向，需要使用 TBN 变换，将顶点位置和入射方向等位置变换到正切空间。这样就可以看到凹凸不平的纹理了。

2.3 实验结果

开发环境：Visual Studio 2022

设备系统：Windows10

图中由两个印章、一个固定光源和一个移动光源组成。两个印章也是一个固定、一个移动，移动印章围绕着固定印章转动。

视觉方向控制：WASD

模型大小控制：UJ

公转速度控制：HK

自转速度控制：MN

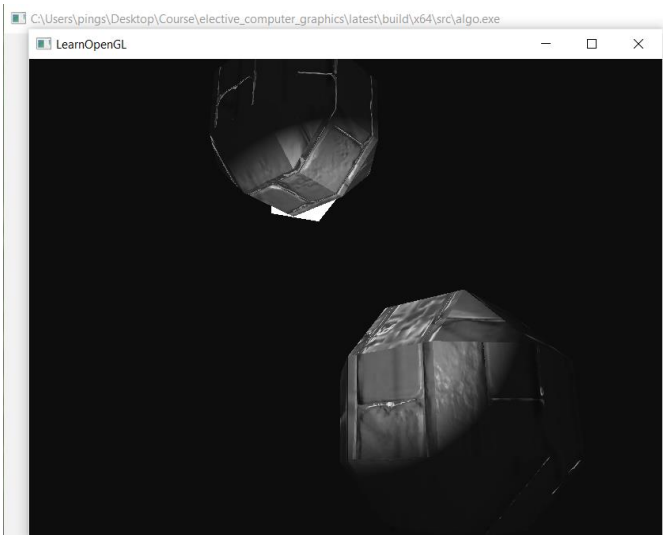

图 2.2 运行结果 1

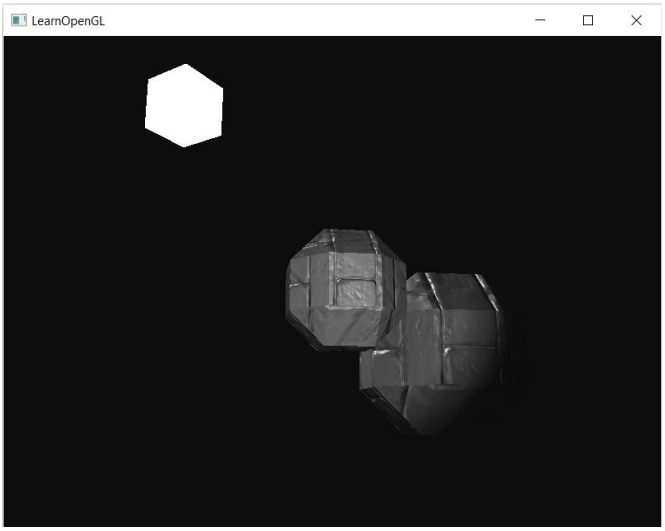

图 2.3 运行结果 2

2.4 心得体会

计算机图形学这门课真的很好玩，很有趣。在课堂中，我学到了通过图形学概述深入的了解了计算机图形学。在后续的课堂中也学到很多图形学多的基础知识，其中包括但不限于，三维图形、图形变换、光照模型、渲染模型等。这些基础知识对我未来自己更深入的去学习图形学有很大的帮助。

在实践的过程中，我也学到了很多除了基础知识外的东西，例如，环境的配置与调试方法、依赖库、格式转换等。说真的在理解环境配置的时间不比理解代码的时间短。

除此之外，当然少不了代码的理解能力。在实践的过程中真的学到了很多之前没见过的算法和知识点。比如：贴图，渲染等等，这些对我来说都是一些新奇的知识点。何老师有趣的授课方式也让我对这些知识的渴望更上一层楼。非常感谢老师的细心解答和提供了那么多的例子。

总而言之，在这堂课的理论和实践学习中真的获益匪浅啊！真的非常庆幸能够抢到这门课，在这堂课中学到的知识点，相信对我的未来会有很大的帮助。是非常喜欢的一堂课！

2.5 源代码

由于代码量有点大，这里只给出了主程序 main.cpp 的代码，其余代码请在提交文件中查看。

程序 1：main.cpp

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <algo/shader.h>
#include <algo/camera.h>
#include <algo/model.h>
#include <algo/cube.h>
#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
```

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

// camera
Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));
glm::vec3 pointPosition(-5.0f, 5.0f, 0.0f);

float lastX = SCR_WIDTH / 2.0f;
float lastY = SCR_HEIGHT / 2.0f;
bool firstMouse = true;

// timing
float deltaTime = 0.0f;
float lastFrame = 0.0f;

float current_scale = 1.0f;
float current_omega = glm::radians(50.0f);
float current_theta = 0.0f;

float current_fai = 0.0f;
float dfai_dt = glm::radians(30.f);

const auto axis = glm::normalize(glm::vec3 {-1, 1, 0});

struct draw_context {
    Model &model;
    Shader &shader;
    const glm::mat4 &view;
    const glm::mat4 &projection;
    glm::vec3 position;
    float scale;
    float theta;
};
```

```
void set_light(Shader& lightingShader);
void draw_model(const draw_context &context);

int main()
{
    // glfw: initialize and configure
    // ------------------------------
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE,
GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", nullptr, nullptr);
    if (window == nullptr)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // tell GLFW to capture our mouse
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
```

```
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}

// tell stb_image.h to flip loaded texture's on the y-axis (before loading model).
stbi_set_flip_vertically_on_load(true);

// configure global opengl state
// -----------------------------
glEnable(GL_DEPTH_TEST);

// build and compile shaders
// -------------------------
Shader ourShader("colors.vs", "colors.fs");
Shader pointShader("light_cube.vs", "light_cube.fs");

// load models
// -----------
Model ourModel("model/model.obj");
cube point(pointPosition);

// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // per-frame time logic
    // --------------------
    auto currentFrame = static_cast<float>(glfwGetTime());
    deltaTime = currentFrame - lastFrame;
    lastFrame = currentFrame;
    current_theta += current_omega * deltaTime;
    current_fai     += dfai_dt * deltaTime;

    // input
    // -----
    processInput(window);
```

```
        // render
        // ------
        glClearColor(0.05f, 0.05f, 0.05f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        const glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
        const glm::mat4 view = camera.GetViewMatrix();
        const auto r = 3.0f;

        draw_context context{ourModel, ourShader, view, projection};

        context.position = glm::vec3(0.0f);
        context.theta    = current_fai;
        context.scale    = current_scale;
        draw_model(context);

        const auto position = glm::rotate(glm::mat4(1.0f), current_theta, axis) * glm::vec4(0, 0, 1, 1);
//          context.position = glm::vec3(r * cos(current_theta), r * sin(current_theta), 0.0f);
        context.position = position * r;
        context.theta    = current_fai;
        context.scale    = 0.5f;
        draw_model(context);

        point.Draw(pointShader, view, projection);

        // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
        // -------------------------------------------------------------------------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
```

```
}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly
// ---------------------------------------------------------------------------------------------------------
void processInput(GLFWwindow *window)
{
    const auto stride = 1.0f;
    const auto rstride = 2.0f;

    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(FORWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
        current_scale = min(2.0f, current_scale + stride * deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS) {
        current_scale = max(0.3f, current_scale - stride * deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS) {
        current_omega = min(4.0f, current_omega + rstride * deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS) {
        current_omega = max(-4.0f, current_omega - rstride * deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_M) == GLFW_PRESS) {
        dfai_dt = min(4.0f, dfai_dt + rstride * deltaTime);
    }
    if (glfwGetKey(window, GLFW_KEY_N) == GLFW_PRESS) {
        dfai_dt = max(-4.0f, dfai_dt - rstride * deltaTime);
```

```
    }
}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes
// ---------------------------------------------------------------------------------------------
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    // make sure the viewport matches the new window dimensions; note that width and
    // height will be significantly larger than specified on retina displays.
    glViewport(0, 0, width, height);
}

// glfw: whenever the mouse moves, this callback is called
// -------------------------------------------------------
void mouse_callback(GLFWwindow* window, double xposIn, double yposIn)
{
    auto xpos = static_cast<float>(xposIn);
    auto ypos = static_cast<float>(yposIn);

    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

    lastX = xpos;
    lastY = ypos;

    camera.ProcessMouseMovement(xoffset, yoffset);
}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called
// ----------------------------------------------------------------------
```

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    camera.ProcessMouseScroll(static_cast<float>(yoffset));
}

void set_light(Shader& lightingShader) {
    lightingShader.setVec3("light.position", camera.Position);
    lightingShader.setVec3("light.direction", camera.Front);
    lightingShader.setFloat("light.cutOff", glm::cos(glm::radians(12.5f)));
    lightingShader.setFloat("light.outerCutOff", glm::cos(glm::radians(17.5f)));
    lightingShader.setVec3("viewPos", camera.Position);

    // light properties
    lightingShader.setVec3("light.ambient", 0.1f, 0.1f, 0.1f);
    // we configure the diffuse intensity slightly higher; the right lighting conditions differ with each lighting method and environment.
    // each environment and lighting type requires some tweaking to get the best out of your environment.
    lightingShader.setVec3("light.diffuse", 0.8f, 0.8f, 0.8f);
    lightingShader.setVec3("light.specular", 0.5f, 0.5f, 0.5f);
    lightingShader.setFloat("light.constant", 1.0f);
    lightingShader.setFloat("light.linear", 0.09f);
    lightingShader.setFloat("light.quadratic", 0.032f);


    lightingShader.setVec3("point.position", pointPosition);
    lightingShader.setVec3("point.ambient", 0.05f, 0.05f, 0.05f);
    lightingShader.setVec3("point.diffuse", 0.8f, 0.8f, 0.8f);
    lightingShader.setVec3("point.specular", 1.0f, 1.0f, 1.0f);
    lightingShader.setFloat("point.constant", 1.0f);
    lightingShader.setFloat("point.linear", 0.045f);
    lightingShader.setFloat("point.quadratic", 0.0075f);
}

void draw_model(const draw_context &context) {
    // don't forget to enable shader before setting uniforms
    context.shader.use();
    set_light(context.shader);
```

```
        context.shader.setMat4("projection", context.projection);
        context.shader.setMat4("view", context.view);

        // render the loaded model
        glm::mat4 model = glm::mat4(1.0f);
        model = glm::translate(model, context.position); // translate it down so it's at the center of the scene
        model = glm::scale(model, glm::vec3(context.scale));      // it's a bit too big for our scene, so scale it down
        model = glm::rotate(model, context.theta, glm::vec3(0.0f, 1.0f, 0.0f));
        context.shader.setMat4("model", model);

        context.model.Draw(context.shader);
}
```