**《计算机图形学》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2022计算机科学与技术（卓越）01** | | | **姓名** |  |
| **实验题目** | **OpenGL 基本使用** | | | | | |
| **实验时间** | **2025/3/24** | | **实验地点** | **DS3305** | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  1. 了解 OpenGL 的性质、功能，形成图形学的初步概念；  2. 掌握计算机图形学中的几何变换、着色模型、纹理使用等 | | | | | | |
| 二、实验项目内容  1． 搭建 OpenGL 的基础程序框架，并设计场景和交互界面；  2． 实现几何物体的基础渲染，实现不同着色模型的界面交互选择；  3． 实现几何物体的通过鼠标进行几何变换；  4． 实现纹理贴图； | | | | | | |
| 三、实验过程或算法（源程序）  3.1实验过程   1. OpenGL框架搭建与场景设计  * 环境初始化：使用GLFW创建800x600窗口，初始化GLAD加载OpenGL函数指针  1. glfwInit(); 2. GLFWwindow\* window = glfwCreateWindow(...); 3. gladLoadGLLoader(...);  * ImGui集成：创建上下文并初始化GLFW/OpenGL3后端  1. ImGui::CreateContext(); 2. ImGui\_ImplGlfw\_InitForOpenGL(...); 3. ImGui\_ImplOpenGL3\_Init(...);  * 场景构建：   立方体：使用标准立方体顶点数据（包含位置、法线、纹理坐标）  地面：使用100x100平面，顶点包含位置和法线信息  阴影：通过投影矩阵实现简单平面阴影   1. 基础渲染与着色模型选择  * Lambert漫反射：  1. float diff = max(dot(norm, lightDir), 0.0); 2. diffuse = diff \* lightColor;  * Phong镜面反射：  1. vec3 reflectDir = reflect(-lightDir, norm); 2. float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32.0);  * Blinn-Phong优化：  1. vec3 halfwayDir = normalize(lightDir + viewDir); 2. float spec = pow(max(dot(norm, halfwayDir), 0.0), 32.0);   着色器通过shadingModel uniform控制算法分支, UI面板提供单选按钮切换模型：   1. ImGui::RadioButton("Lambert", &currentShadingModel, 0); 2. ImGui::RadioButton("Blinn-Phong", &currentShadingModel, 1); 3. ImGui::RadioButton("Phong", &currentShadingModel, 2);   3. 几何变换实现   * 矩阵变换体系：  1. glm::mat4 model = glm::rotate(glm::mat4(1.0f), 2. glm::radians(rotationAngle), 3. glm::vec3(0.3f, 0.5f, 0.0f));  * 当前通过UI滑块演示变换  1. ImGui::SliderFloat("Rotation", &rotationAngle, 0.0f, 360.0f);   4. 纹理贴图实现   * 纹理加载：使用stb\_image库加载图片  1. GLuint loadTexture(const char\* path) { 2. stbi\_set\_flip\_vertically\_on\_load(true); 3. *// ...生成纹理对象并绑定数据* 4. }  * 纹理绑定：渲染时激活纹理单元  1. glActiveTexture(GL\_TEXTURE0); 2. glBindTexture(GL\_TEXTURE\_2D, textures[selectedTexture]);  * 多重纹理支持：提供4种材质选择  1. std::vector<std::string> textureNames = 2. { "stone", "Metal", "Fabric","porcelain"};   3.2 关键代码实现  1. 阴影投影矩阵计算   1. glm::mat4 computeShadowMatrix(glm::vec4 plane, glm::vec3 lightPos) { 2. float d = glm::dot(plane, glm::vec4(lightPos, 1.0f)); 3. glm::mat4 shadowMat = glm::mat4( 4. d - lightPos.x \* plane.x, -lightPos.x \* plane.y, -lightPos.x \* plane.z, -lightPos.x \* plane.w, 5. -lightPos.y \* plane.x, d - lightPos.y \* plane.y, -lightPos.y \* plane.z, -lightPos.y \* plane.w, 6. -lightPos.z \* plane.x, -lightPos.z \* plane.y, d - lightPos.z \* plane.z, -lightPos.z \* plane.w, 7. -plane.x, -plane.y, -plane.z, d 8. ); 9. return shadowMat; 10. }   2.光照计算（GLSL实现）   1. *// Lambert漫反射* 2. float diff = max(dot(norm, lightDir), 0.0); 3. diffuse = diff \* lightColor; 4. *// Phong镜面反射* 5. vec3 reflectDir = reflect(-lightDir, norm); 6. float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32.0); 7. specular = 2.0 \* spec \* lightColor; 8. *// Blinn-Phong优化* 9. vec3 halfwayDir = normalize(lightDir + viewDir); 10. float spec = pow(max(dot(norm, halfwayDir), 0.0), 32.0); 11. specular = 1.5 \* spec \* lightColor;   3.纹理采样（GLSL实现）   1. *// 片段着色器纹理采样* 2. if(useTexture) { 3. vec4 texColor = texture(texture1, TexCoord); 4. finalColor = vec4(lighting, 1.0) \* texColor; 5. } else { 6. finalColor = vec4(lighting \* objectColor, 1.0); 7. }   4.几何变换矩阵生成   1. glm::mat4 model = glm::rotate( 2. glm::mat4(1.0f), 3. glm::radians(rotationAngle), 4. glm::vec3(0.3f, 0.5f, 0.0f) 5. ); 6. *// 视图矩阵生成代码* 7. glm::mat4 view = glm::lookAt( 8. glm::vec3(2.0f, 2.0f, 3.0f), *// 相机位置* 9. glm::vec3(0.0f, 0.0f, 0.0f), *// 观察目标* 10. glm::vec3(0.0f, 1.0f, 0.0f)  *// 上方向* 11. ); 12. *// 投影矩阵生成代码* 13. glm::mat4 projection = glm::perspective( 14. glm::radians(70.0f),        *// FOV* 15. 800.0f / 600.0f,           *// 宽高比* 16. 0.1f,                      *// 近平面* 17. 500.0f                     *// 远平面* 18. );   5.渲染管线配置   1. *// 初始化阶段* 2. void setupCube() { 3. glGenVertexArrays(1, &cubeVAO); 4. glGenBuffers(1, &cubeVBO); 5. *// ...顶点数据配置* 6. } 7. *// 每帧渲染循环* 8. while (!glfwWindowShouldClose(window)) { 9. *// 清空缓冲区* 10. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); 11. *// 更新矩阵* 12. glm::mat4 model = ...; 13. glm::mat4 view = ...; 14. glm::mat4 projection = ...; 15. *// 绘制地面* 16. glUseProgram(groundShaderProgram); 17. glBindVertexArray(groundVAO); 18. glDrawElements(...); 19. *// 绘制阴影* 20. glUseProgram(shadowShaderProgram); 21. glDrawArrays(...); 22. *// 绘制物体* 23. glUseProgram(shaderProgram); 24. glUniformMatrix4fv(...); 25. glDrawArrays(...); 26. *// UI渲染* 27. ImGui::Render(); 28. ImGui\_ImplOpenGL3\_RenderDrawData(...); 29. } | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  1. 基础渲染效果验证  Lambert模型：  呈现均匀的漫反射效果，物体表面无明显高光区域，不同旋转角度下光照过渡平滑    Phong模型：  出现明显镜面高光，高光区域随视角移动变化    Blinn-Phong模型：  高光区域比Phong更柔和，视角移动时高光过渡更自然    2.几何变换测试  通过控制rotation值的大小实现立方体绕轴旋转的几何变换    3. 纹理系统测试   |  |  | | --- | --- | | 测试项 | 实现效果 | | 石材纹理 | 显示石块肌理 | | 金属纹理 | 高反射特性 | | 织物纹理 | 显示布纹细节 | | 瓷器纹理 | 显示瓷器纹理细节 |           **源程序调试过程**  1.纹理映射异常   * 纹理显示为纯色 * 控制台输出Failed to load texture: textures/metal.png   调试步骤：   * 使用绝对路径测试：loadTexture("D:/project/textures/metal.png") * 验证stb\_image返回的nrChannels值（正确应为3） * 发现纹理文件实际命名为metal.jpg   2.ImGUI控制失效   * 滑动条操作无响应 * RadioButton状态不更新   调试步骤：   * 在UI渲染代码后添加调试输出：  1. std::cout << "Current shading model: " << currentShadingModel << std::endl;  * 发现变量值未同步到着色器 * 补充Uniform更新：  1. glUniform1i(glGetUniformLocation(shaderProgram, "shadingModel"), currentShadingModel); | | | | | | |