**深 圳 大 学 实 验 报 告**

**课程名称：­ 计算机图形学**

**实验项目名称： 实验一 OpenGL基本绘制**

**学院： 计算机与软件学院**

**专业： 软件工程（腾班）**

**指导教师： 熊卫丹**

**报告人： 黄亮铭 学号：2022155028 班级： 腾班**

**实验时间：2024年 09月04日 -- 2024年 09月30 日**

**实验报告提交时间： 2024年 09月12 日**

**教务部制**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验目的与要求：   1. 掌握Visual Studio Community 2019集成开发环境的安装；掌握CMake跨平台构建工具的安装；掌握Git版本控制工具的安装；掌握vcpkg库管理工具的安装；掌握系统环境变量的设置；了解和掌握OpenGL的环境配置；掌握OpenGL工程项目的建立和基本设置。 2. 理解OpenGL的原理；了解和熟悉OpenGL着色语言；掌握基于OpenGL的C++程序结构；掌握OpenGL中若干基本二维图形的绘制；了解顶点着色器的使用；了解片元着色器的使用。 3. 使用现代OpenGL中的着色器，绘制多个简单的二维图形，形状内容不限，自己发挥。   实验过程及内容：  Visual Studio 2022集成开发环境安装   1. 因为本地已经安装Visual Studio 2022，因此无需再从网上下载安装包安装。只需要依次点击工具=》获取工具和功能安装部分缺失的组件即可。     图1   1. 根据实验配置文档勾选相应的“使用C++的桌面开发”的相应组件。     图2   1. 然后在“语言包“一栏中选择中文和英语。     图3  CMake跨平台构建工具的安装   1. 根据实验文档打开网址<https://cmake.org/download/>，下载Windows平台的CMake安装包。然后按流程安装CMake即可（这里我选择了让CMake安装工具帮我配置环境变量）。 2. 在PowerShell中输入命令‘cmake’查看是否成功安装。经查看，确认安装成功。     图4  Git的安装   1. 打开网址<https://git-scm.com/>，下载对应平台的Git安装包。     图5   1. 按照安装流程安装Git即可，在选择编辑器时选择VsCode作为默认编辑工具。     图6   1. 在PowerShell中输入命令‘cmake’查看是否成功安装。经查看，确认安装成功.     图7  vcpkg的安装   1. 打开网址<https://github.com/microsoft/vcpkg/>直接下载当前版本的源代码解压到安装目录。     图8   1. 进入vcpkg-master文件，以管理员身份打开PowerShell。运行命令‘.\bootstrap-vcpkg.bat’和‘.\vcpkg integrate install’       图9   1. 配置环境变量（Key-Value形式给出）：VCPKG\_ROOT：C:\src\vcpkg-master；新建VCPKG\_DEFAULT\_TRIPLE： x64-windows；新建Path：%VCPKG\_ROOT%。       图10   1. 验证路径是否添加成功。打开任意一个文件夹（除C:\src\vcpkg-master外）的PowerShell，输入命令‘vcpkg’，如果没有弹出错误或者弹出使用说明则验证路径成功设置。     图11  GLFW，GLAD，GLM的安装   1. 在任意文件夹下输入命令“vcpkg install glfw3 glad glm”，等待下载编译完成即可。     图12  构建并运行实验1.1   1. 进入到实验1.1的参考代码文件夹下，然后打开命令行输入命令“cmake -B . ”     图13   1. 打开main.sln文件，将“解决方案管理器”的main.cpp设置为启动项。 2. 最后点击菜单栏下方的“本地Windows调试器”编译运行程序，运行程序后看到三角形，说明配置成功。     图14  执行并修改实验1.2   1. 按照执行实验1.1代码的顺序，对实验1.2执行相同的代码。即：在对应文件夹内执行命令“cmake -B .”和在Visual Studio里面将main.cpp设置为启动项即可。执行成功的结果如图所示。     图15   1. 通过修改给定代码中生成三角形和生成正方形的函数：generateTrianglePoints和generateSquarePoints，得实验文档要求的实验结果（逆时针旋转）。 2. 修改generateTrianglePoints函数：观察图15可知，红色顶点的角度为0，蓝色的为120，绿色的为240。要实现文档要求的结果，我们需要满足以下条件：红色x坐标为0，y坐标为正；蓝色的x坐标为负，y坐标为负；绿色x坐标为正，y坐标为负。为了满足上述条件，我将代码修改为如下形式。  |  | | --- | | // @TODO: 在此函数中修改三角形的顶点位置  **for** **(**int i **=** 0**;** i **<** 3**;** i**++)** **{**  // 当前顶点对应的角度  double currentAngle **=** getTriangleAngle**(**i**);**  // 根据角度及三角形中心计算顶点坐标  vertices**[**startVertexIndex **+** i**]** **=** glm**::**vec2**(-**sin**(**currentAngle**),** cos**(**currentAngle**))** **\*** scale **+** center**;**  **}** |   图16   1. 点击“本地Windows调试器“运行代码，发现修改正确。     图17   1. 接下来修改generateSquarePoints函数：观察实验文档给出的期望结果，发现纯白色的正方形需要变成白黑相间的正方形（三白三黑）。因此，我们只需要为正方形顶点的坐标乘上不同系数即可实现不同边长的正方形堆叠。这里我是用数组存储不同的系数实现。因为黑色视觉上会更窄，因此我黑色正方形系数会稍微比白色的大一点。此外，循环中的colors[]也需要随着奇偶变换。修改后代码如下图所示。  |  | | --- | | // @TODO: 在此函数中修改，生成多个嵌套正方形  // 根据正方形及顶点对应角度，计算当前正方形四个顶点坐标  float alpha**[**6**];**  alpha**[**0**]** **=** 1.0**,** alpha**[**5**]** **=** 0.16666667**;**  **for** **(**int i **=** 4**;** i **>=** 1**;** i**--)** **{**  **if** **(**i **%** 2**)** **{**  alpha**[**i**]** **=** alpha**[**i **+** 1**]** **+** 0.16666668**;**  **}**  **else** **{**  alpha**[**i**]** **=** alpha**[**i **+** 1**]** **+** 0.16666665**;**  **}**    **}**  **for** **(**int i **=** 0**;** i **<** 6**;** i**++)** **{**  **for** **(**int j **=** 0**;** j **<** 4**;** j**++)**  **{**  double currentAngle **=** getSquareAngle**(**j**);**  vertices**[**vertexIndex**]** **=** alpha**[**i**]** **\*** glm**::**vec2**(**cos**(**currentAngle**),** sin**(**currentAngle**))** **\*** scale **+** center**;**  colors**[**vertexIndex**]** **=** i **%** 2 **?** BLACK **:** WHITE**;**  vertexIndex**++;**  **}**  **}** |   图18   1. 点击“本地Windows调试器“运行代码，发现修改正确。     图19  实验1留空代码填充  理想效果如下图所示（来源：实验文档）    图20  实现步骤：   1. 定义圆形和椭圆的点。     图21   1. 调用生成形状顶点位置的函数，根据已有的函数的参数要求填写参数。     图22   1. 初始化圆和椭圆的数据。  |  | | --- | |  | |  |   图23   1. 绘制圆和椭圆。     图24  实验一留空代码填充结果展示    图25  绘制简单的二维图形（1）——梯形实现思路  考虑矩形和梯形的形状大致相似，因此可以仿照矩形的绘制方法，绘制梯形。具体方法为：   1. 在display函数中绑定梯形的顶点数据对象，同时调用glDrawArrays函数绘制图像。 2. 在init函数中定义梯形的四个顶点，然后调用生成梯形的顶点位置的函数生成顶点。 3. 在init函数中初始化梯形顶点数据：1）创建顶点数组对象并绑定；2）创建顶点缓存对象存储梯形的四个顶点的位置以及颜色。 4. 在generateTrapeziumPoints生成梯形每个点：1）设置梯形的尺寸及中心的位置；2）计算梯形顶点的角度：这里我设置四个顶点的角度依次为0，PI/3，2PI/3，PI。3）第二和第三个顶点距离中心的距离应该更近，因此，我将他们的距离乘上一个系数0.7；4）颜色则设置随机。   绘制简单的二维图形（1）——梯形核心代码  核心代码如下所示。   1. 计算梯形的四个顶点的坐标。  |  | | --- | | // 生成梯形每个点  void generateTrapeziumPoints(glm::vec2 vertices[], glm::vec3 colors[], int startVertexIndex)  {  // 最大梯形尺寸  glm::vec2 scale(0.8, 0.8);  // 梯形中心位置  glm::vec2 center(0.0, 0.4);  glm::vec3 currentColor[4] = { WHITE, RED, BLUE, GREEN };  int vertexIndex = startVertexIndex;  // 根据梯形及顶点对应角度，计算当前梯形四个顶点坐标  for (int j = 0; j < 4; j++)  {  double currentAngle = getTrapeziumAngle(j);  vertices[vertexIndex] = glm::vec2(cos(currentAngle), sin(currentAngle)) \* scale + center;  if (j == 1 || j == 2) {  vertices[vertexIndex] \*= 0.7;  }  //srand((unsigned)time(NULL));  colors[vertexIndex] = currentColor[rand() % 4];  vertexIndex++;  }  } |  1. 获得顶点和梯形中心的角度。  |  | | --- | | const int TRAPEZIUM\_NUM\_POINTS = 4;  // 获得梯形的每个角度  double getTrapeziumAngle(int point) {  return M\_PI / 3 \* point;  } |   绘制简单的二维图形（1）——梯形运行结果  代码运行结果如下图所示。    图26  绘制简单的二维图形（2）——圆形实现思路  考虑圆形可以由多个三角形组成，因此可以仿照三角形的绘制方法，绘制圆形，只需要注意每个三角形的角度即可。这里我选择使用360个三角形拼接成圆形，具体方法为：   1. 在display函数中绑定圆形的顶点数据对象，同时调用glDrawArrays函数绘制图像。 2. 在init函数中定义圆形的360个顶点，然后调用生成梯形的顶点位置的函数生成顶点。 3. 在init函数中初始化梯形顶点数据：1）创建顶点数组对象并绑定；2）创建顶点缓存对象存储圆形的四个顶点的位置以及颜色。 4. 在generateTrapeziumPoints生成圆形每个点：1）设置圆形的半径及中心的位置；2）计算圆形顶点的角度：公式为*M\_PI / 2 + M\_PI / 180 ∗ point，point为顶点的索引（从0开始）*。3）颜色设置随机。   绘制简单的二维图形（2）——圆形核心代码  代码运行结果如下图所示。   1. 计算圆形的顶点。  |  | | --- | | // 生成圆形的每个点  void generateCirclePoints(glm::vec2 vertices[], glm::vec3 colors[], int startVertexIndex)  {  // 圆形尺寸  glm::vec2 scale(0.3, 0.3);  // 圆形中心位置  glm::vec2 center(0.0, 0.0);  // @TODO: 在此函数中修改圆形的顶点位置  for (int i = 0; i < CIRCLE\_NUM\_POINTS; i++) {  // 当前顶点对应的角度  double currentAngle = getCircleAngle(i);  // 根据角度及圆形中心计算顶点坐标  vertices[startVertexIndex + i] = glm::vec2(cos(currentAngle), sin(currentAngle)) \* scale + center;  colors[startVertexIndex + i] = WHITE;  }  } |  1. 计算每个顶点和梯形中心的角度。  |  | | --- | | const int CIRCLE\_NUM\_POINTS = 360;  // 获得圆形的每个角度  double getCircleAngle(int point) {  return M\_PI / 2 + M\_PI / 180 \* point;  } |   绘制简单的二维图形（2）——圆形圆形运行结果  代码运行结果如下图所示。    图27  设计一个包含各种简单图元的几何形状图片——房屋  设计思路   1. 考虑房屋及其周围元素。房屋需要有主体（长方形），屋顶（三角形或梯形）、门（长方形）和窗户（正方形）。房屋周围有天空（蓝色背景）和地面（绿色矩形），天空上还会有太阳（圆形）、白云（椭圆）。 2. 房屋和太阳的放置位置应该在图像的两个对角，这样的图像的主体会更加均衡。 3. 天空和地面的分界线即地平线应当在房屋主体的中间。 4. 此外，我们还需要考虑各种图形的覆盖问题。在编写代码的时候，需要注意显示图像的先后顺序，即注意图层问题。   代码实现  代码实现基于实验1.2的代码。   1. 梯形和圆形的实现思路和代码实现还有运行结果已经在上面进行展示，只需要在上面的代码的基础上修改位置即可。 2. 这里以房屋主体为例，展示矩形的绘制方法。 3. 定义矩形的点。     图28   1. 调用生成形状顶点位置的函数，然后将其初始化。因为这里的代码多为重复，因此我在这里重构了代码，将初始化的代码封装成函数以方便复用。       图29   1. 其余形状按上述房屋主体绘制的思路调用初始化函数即可。     图30  运行结果    图31 |

|  |
| --- |
| 实验结论：   1. 我成功地掌握了Visual Studio Community 2019、CMake、Git、vcpkg等工具的安装与配置，并熟悉了OpenGL的环境配置。 2. 在绘制二维图形的过程中，我学会了如何使用现代OpenGL中的着色器来创建和修改图形。通过修改顶点着色器和片元着色器，我能够控制图形的显示效果，如颜色、形状和旋转等。在实验中，我成功地绘制了三角形、正方形、梯形和圆形等基本图形，并通过调整参数实现了图形的逆时针旋转和颜色变化。 3. 在解决实验中遇到的问题时，我学会了如何通过查阅文档和搜索网络资源来找到解决方案。例如，在配置环境变量和安装库文件时，我遇到了一些困难，但通过仔细阅读安装指南和实验文档，我能够顺利解决这些问题。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。