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：洪子敬 学号：2022155033 班级：腾班

实验时间：2024 年 09 月 11 日 -- 2024 年 09 月 30 日

实验报告提交时间：2024 年 09 月 15 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：

1. OpenGL 在 Visual Studio Community 2024 上的环境配置和实验构建

（1）环境配置

按照操作指引，下载安装好 VS2024 上需要的包、CMake、vcpkg，配置好对应的环境变量后，最后安装 OpenGL 库（GLFW、GLAD、GLM），在终端内执行以下命令即可进行安装：（注意开梯子即翻墙软件才不容易掉线）

vcpkg install glad glfw3 glm

但实际安装时却出现以下问题：

```
error: building glfw3:x64-windows failed with: BUILD_FAILED
See https://learn.microsoft.com/vcpkg/troubleshoot/build-failures?WT.mc_id=vcpkg_inprod
Elapsed time to handle glfw3:x64-windows: 7.1 s
Please ensure you're using the latest port files with `git pull` and `vcpkg update`.
Then check for known issues at:
  https://github.com/microsoft/vcpkg/issues?q=is%3Aissue+is%3Aopen+in%3Atitle+glfw3
You can submit a new issue at:
  https://github.com/microsoft/vcpkg/issues/new?title=[glfw3]+Build+error+on+x64-window
```

看上述出现问题的主要是 glfw3 的安装，通过多方努力才发现是 glfw3 包太大，需要梯子的网络比较稳定才能进行下载，后面也是换了个稳定的环境才下载成功。

```
PS D:\Vcpkg> vcpkg install glfw3 glad glm
Computing installation plan...
The following packages are already installed:
    glad[core,loader]:x64-windows@0.1.36
    glfw3:x64-windows@3.4
    glm:x64-windows@1.0.1#3
glad:x64-windows is already installed
glfw3:x64-windows is already installed
glm:x64-windows is already installed
Total install time: 1.21 ms
```

HZJ

（2）实验 1.1 的构建

首先解压实验 1.1 代码并进入相应的文件夹，并在该文件夹下用 cmd 打开，执行命令：

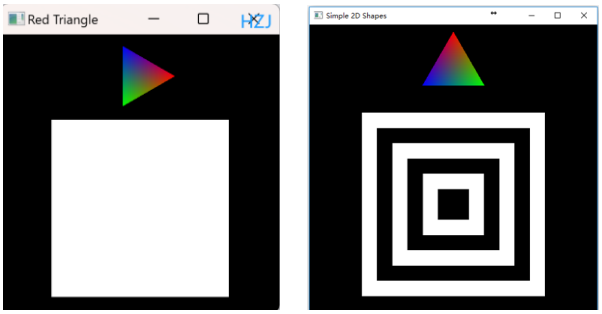
cmake -B.(注意后面有个点)

```
D:\实验报告\大三上\计算机图形学\实验1.1\实验1.1_参考代码>cmake -B.
-- Building for: Visual Studio 17 2022
-- Selecting Windows SDK version 10.0.22621.0 to target Windows 10.0.22631.
-- The C compiler identification is MSVC 19.37.32825.0
-- The CXX compiler identification is MSVC 19.37.32825.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: D:/visual studio/visual studio IDE/VC/Tools
kipped
```

接着打开生成的 main.sln 文件，将 main 文件设置为启动项，运行“本地 windows 调试器”得到以下结果：

2. OpenGL 与着色器编程

阅读所给的 OpenGL 编程教学文件，了解所给函数的具体释义和用法后，完成课堂练习。运行所给文件可以得到下面左图的结果：



题目要求我们修改代码得到上述右图，通过观察我们发现是将三角形逆时针旋转了 90 度（当然修改点的位置也可以），并且在正方形里面嵌入了多个不同大小和颜色的正方形。**针对三角形**，我在原代码的基础上上对该三角形进行了旋转，旋转角度为 90 度（逆时针为正值）。通过数学推导不难得到，旋转后顶点坐标为：

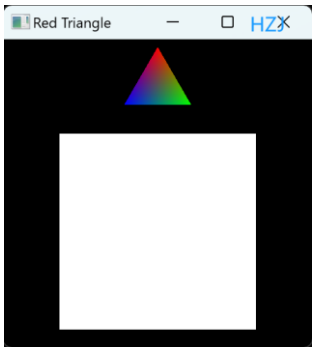
$$\begin{cases} x_{rotated} = x * \cos(angle) - y * \sin(angle) \\ y_{rotated} = x * \sin(angle) + y * \cos(angle) \end{cases}$$

其中 angle 为旋转角度，逆时针为正方向；

此外，要注意的是由于我们是要在原三角形中心进行旋转而非原点，因此，我们需要将中心先移到原点再进行旋转，旋转后再将中心移回原位。具体代码如下面左图所示：

```
// @TODO: 在此函数中修改三角形的顶点位置
//设置旋转角度为逆时针90度
double angle = M_PI / 2;
//设置旋转中心
double cx = 0.0;
double cy = 0.70;
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初始顶点坐标
    glm::vec2 originVertex = glm::vec2(cos(currentAngle), sin(currentAngle)) * scale + center;
    //平移坐标
    double x_tmp = originVertex.x - cx;
    double y_tmp = originVertex.y - cy;
    //旋转顶点
    double x_rotated = x_tmp * cos(angle) - y_tmp * sin(angle);
    double y_roated = x_tmp * sin(angle) + y_tmp * cos(angle);
    vertices[startVertexIndex + i] = glm::vec2(x_rotated+cx, y_roated+cy);
}
```

洪子敬
2022155033
腾班



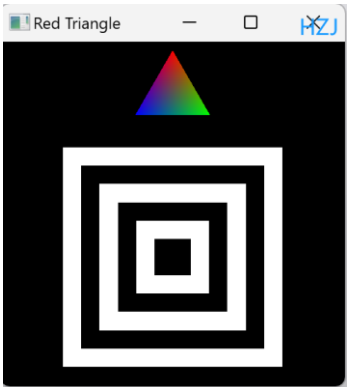
通过上面代码运行后可初步得到上面右图的结果：

针对正方形，我在原代码的基础上加上了一层外层循环，进行多个正方形绘制，并且对 scale 每次画完一个正方形后进行一定比例的缩短，同时我们需要判断绘制正方形的颜色，这里是通过外层循环计数的奇偶情况来进行颜色区分。具体代码如下面左图所示：

```
// @TODO: 在此函数中修改，生成多个嵌套正方形
// 定义每次的缩放比例
double scaleFactor = 0.15;

for (int i = 0; i < squareNum; i++) {
    //定义正方形尺寸
    glm::vec2 scale(0.90-i*0.15, 0.90-i*0.15);
    // 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
    for (int j = 0; j < 4; j++)
    {
        double currentAngle = getSquareAngle(j);
        vertices[vertexIndex] = glm::vec2(cos(currentAngle), sin(currentAngle)) * scale + center;
        //分情况画不一样颜色的图
        if (i % 2 == 0) {
            glm::vec3 currentColor = WHITE;
            colors[vertexIndex] = currentColor;
        }
        else{
            glm::vec3 currentColor = BLACK;
            colors[vertexIndex] = currentColor;
        }
        vertexIndex++;
    }
}
```

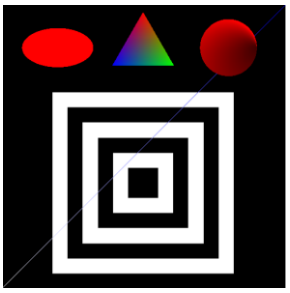
洪子敬
2022155033
腾班



通过上述代码编程，成功得到目标结果，如上面右图所示：

3. OpenGL 基本绘制

（1）在前面结果的基础上，增加绘制线、椭圆、圆的函数，绘制出以下结果：



通过观察所给代码的运行结果，发现我们所需要做的即绘制圆和椭圆；在理解了绘制圆和椭圆的函数之后，我们需要做的为以下几个部分：

一是数据点和颜色的定义：

```
// @TODO: 生成圆形和椭圆上的点和颜色
//定义圆形的点
glm::vec2 circle_vertices[CIRCLE_NUM_POINTS];
glm::vec3 circle_colors[CIRCLE_NUM_POINTS];
//定义椭圆的点
glm::vec2 ellipse_vertices[ELLIPSE_NUM_POINTS];
glm::vec3 ellipse_colors[ELLIPSE_NUM_POINTS];
//定义中心
glm::vec2 circle_center(0.6, 0.7);
glm::vec2 ellipse_center(-0.6, 0.7);
//调用生成形状顶点位置的函数
generateEllipsePoints(circle_vertices, circle_colors, 0, CIRCLE_NUM_POINTS, circle_center, 0.2, 1.0);
generateEllipsePoints(ellipse_vertices, ellipse_colors, 0, ELLIPSE_NUM_POINTS, ellipse_center, 0.25, 0.6);
```

洪子敬
2022155033

这里我们定义圆心和椭圆圆心的位置，以及各自的点和颜色数组，并调用生成顶点的函数。这里要注意，最后两个参数，第一个代表半径，而第二个代表 y 方向上的比例，如果不为 1，则代表椭圆（实际上对颜色也有影响，为 1 则为渐变红色，否则为全红色）。

二是数据的初始化：

```
//初始化圆的数据
glGenVertexArrays(1, &circleVao);
glBindVertexArray(circleVao);

glGenBuffers(1, &vbo[0]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(circle_vertices), circle_vertices, GL_STATIC_DRAW);
location = glGetAttribLocation(program, "vPosition");
glEnableVertexAttribArray(location);
glVertexAttribPointer(
    location,
    2,
    GL_FLOAT,
    GL_FALSE,
    sizeof(glm::vec2),
    BUFFER_OFFSET(0));

glGenBuffers(1, &vbo[1]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(circle_colors), circle_colors, GL_STATIC_DRAW);
cLocation = glGetAttribLocation(program, "vColor");
```

洪子敬
2022155033

（篇幅原因，只给出圆初始化部分代码）

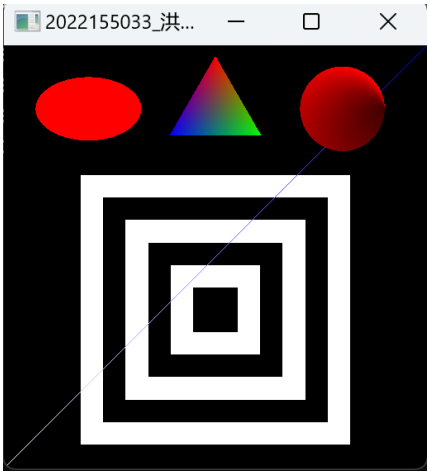
仿造其他图形的初始化进行即可，不过要注意，这里要自己定义各自的 dingdianshuzu1 对象 GLuint，缓冲区数组则可以不用，可以一起使用。

三是图形的展示：

```
// @TODO: 绘制圆
glBindVertexArray(circleVao);
glDrawArrays(GL_TRIANGLE_FAN, 0, CIRCLE_NUM_POINTS);
// @TODO: 绘制椭圆
glBindVertexArray(ellipseVao);
glDrawArrays(GL_TRIANGLE_FAN, 0, ELLIPSE_NUM_POINTS);
                                                    洪子敬
glFlush();                                          2022155033
```

这个部分较为简单，不过要注意绘制选项的选择，这里选择“**GL_TRIANGLE_FAN**”才行，这样才能保证图形内部是填充的，而非空心的。

通过上述三部分，最后可以得到以下结果，实验成功：



（2）个人设计一幅与前面不同的二维几何形状图片

个人想法是在蓝天下画一个太阳，在草地上画一个房子，天空中漂浮着一朵云。有了此想法后，将其进行分解为以下几个部分：

一是蓝天的绘制：

这个想法也比较简单，只需要将 init 函数中最后背景颜色设置为天空蓝即可，这里颜色的设置为：

glClearColor(0.529, 0.807, 0.922, 1.0);

二是太阳的绘制：

这个想法也不难，前面我们已经知道圆的画法，所以我们只需要更改一下参数即可。这里设置圆心在左上角，半径设为 0.2，颜色设置为黄色（RGB：1.0，1.0，0）。

三是房子的绘制：

为了实施方便，我们抛弃之前按角的绘制方法，直接进行顶点绘制。先确定房顶三角形的三个点，在确定房体正方形的四个点，并用不同的颜色绘制即可。这里设置房顶颜色为黄色，房体颜色为暗黄色（RGB：1.0，0.9，0）。此外，我们还绘制了一个门，这里也是规定四个点，并将其颜色设置为深黄色（RGB：0.8，0.7，0）。

四是草地的绘制：

这个也比较简单，这需要绘制一块绿色的矩形即可，不过值得注意的是，我们为了草地不遮挡到房子，我们需要**先展示草地，再展示房子**。

最后是云朵的绘制：

这个是最难画的，云朵实际上是多个图形的混合。这里为了简便，我是在圆的基础上对这些顶点进行**随机扰动**，这也很符合常理，毕竟云大多情况下是**不规则的**。云的扰动代

码设置如下：

vertices[i] = center + glm::vec2(x + (rand() % 20 - 10) * 0.01, y + (rand() % 20 - 10) * 0.01);

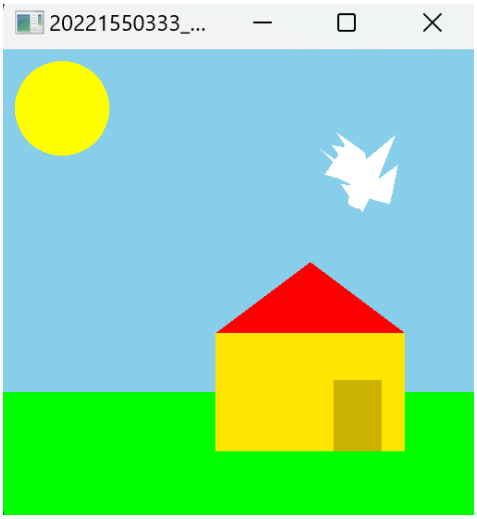
这里只展示绘制云朵的代码，其他绘制代码都与之前差不多，这里不多加展示，详细请见提交代码：

```
//生成云朵的点
void generateCloudPoints(glm::vec2 vertices[], glm::vec3 colors[], int numPoints, glm::vec2 center, float scale) {
    for (int i = 0; i < numPoints; ++i) {
        float angle = (2.0f * M_PI / numPoints) * i;
        float x = cos(angle) * scale;
        float y = sin(angle) * scale;

        vertices[i] = center + glm::vec2(x + (rand() % 20 - 10) * 0.01, y + (rand() % 20 - 10) * 0.01);
        colors[i] = WHITE; // 云朵颜色为白色
    }
}
```

洪子敬
2022155033

最后运行代码可以得到以下结果：



我们可以看到整体上还是不错的（除了这个云有点奇形怪状外），实验圆满结束。

实验结论：

本次实验进行 OpenGL 在 VS 上的环境配置，初步利用 OpenGL 进行给定图形的绘制，还进行个人 DIY 图形设计，实验结果十分成功，图形全部成功显示，也学习到了不少关于 OpenGL 在 VS 上的使用方法和注意事项，本次实验圆满结束。

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。