

# 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： 计算机科学与技术

指导教师： 周虹

报告人： 林宪亮 学号： 202222150130 班级： 国际班

实验时间： 2024 年 09 月 19 日 -- 2024 年 09 月 21 日

实验报告提交时间： 2024 年 09 月 21 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：

### 一、实验环境的安装

#### 1. 集成开发环境：Visual Studio Community 2022 的安装

需要安装“使用 C++ 的桌面开发”，“用于 Windows 的 C++ C++ CMake 工具”、“英语语言包”三项。



图 1 Visual Studio Community 2022 的安装

因为我已经安装过 VS，所以点击修改按钮新增所需的拓展包即可。

#### 2. CMake 的安装

打开网址 <https://cmake.org/download/>，下载对应平台的 CMake 安装包。打开安装包，按流程安装 CMake，其中有修改系统变量的选项选择第二项。

名称	修改日期	类型	大小
cmake	2024/9/9 20:42	文件夹	
vcpkg	2024/9/9 12:22	文件夹	

图 2 CMake 的安装

安装结果如图。

#### 3. Git 的安装

打开网址 <https://git-scm.com/>，下载对应平台的 Git 安装包。打开安装包，按流程安装 Git，其中有选择编辑器的选项。

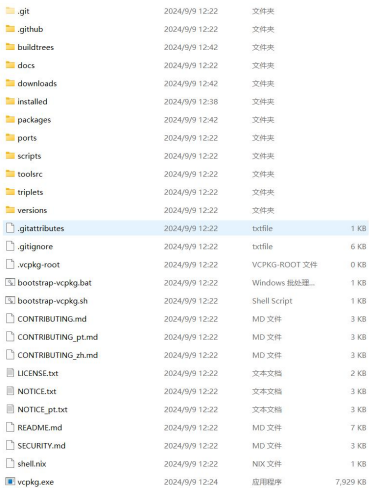
#### 4. vcpkg 的安装

打开网址 <https://github.com/microsoft/vcpkg/>，使用 Git 克隆仓库到安装目录。

进入到 vcpkg 文件，点击左上角文件，使用管理员身份打开 Powershell。

运行目录下的 bootstrap 引导脚本，执行 `.\bootstrap-vcpkg.bat`，构建 vcpkg。

执行 `.\vcpkg integrate install` 命令，将 vcpkg 聚合到 visual studio，这样用 vcpkg 安装的库就可以直接在 visual studio 中使用了。



.git	2024/9/9 12:22	文件夹
.github	2024/9/9 12:22	文件夹
buildtrees	2024/9/9 12:42	文件夹
docs	2024/9/9 12:22	文件夹
downloads	2024/9/9 12:42	文件夹
installed	2024/9/9 12:38	文件夹
packages	2024/9/9 12:42	文件夹
ports	2024/9/9 12:22	文件夹
scripts	2024/9/9 12:22	文件夹
toolsrc	2024/9/9 12:22	文件夹
triplets	2024/9/9 12:22	文件夹
versions	2024/9/9 12:22	文件夹
.gitattributes	2024/9/9 12:22	textfile 1 KB
.gitignore	2024/9/9 12:22	textfile 6 KB
vcpkg-root	2024/9/9 12:22	VCPKG-ROOT 文件 0 KB
bootstrap-vcpkg.bat	2024/9/9 12:22	Windows 批处理... 1 KB
bootstrap-vcpkg.sh	2024/9/9 12:22	Shell Script 1 KB
CONTRIBUTING.md	2024/9/9 12:22	MD 文件 3 KB
CONTRIBUTING_gst.md	2024/9/9 12:22	MD 文件 3 KB
CONTRIBUTING_ri.md	2024/9/9 12:22	MD 文件 3 KB
LICENSE.txt	2024/9/9 12:22	文本文档 2 KB
NOTICE.txt	2024/9/9 12:22	文本文档 3 KB
NOTICE_gst.txt	2024/9/9 12:22	文本文档 3 KB
README.md	2024/9/9 12:22	MD 文件 7 KB
SECURITY.md	2024/9/9 12:22	MD 文件 3 KB
shell.nix	2024/9/9 12:22	NIX 文件 1 KB
vcpkg.exe	2024/9/9 12:24	应用程序 7,929 KB

图 3 vcpkg 的安装

如图我安装好了 vcpkg 库。

#### 5. OpenGL 库安装: GLFW, GLAD, GLM

在随便一个路径下打开命令行，在命令行中输入命令 `vcpkg install glfw3 glad glm`。

```
C:\Users\22237>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#1
  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: 890 us
glad provides CMake targets:

# this is heuristically generated, and may not be correct
find_package(glad CONFIG REQUIRED)
target_link_libraries(main PRIVATE glad::glad)

glfw3 provides CMake targets:

# this is heuristically generated, and may not be correct
find_package(glfw3 CONFIG REQUIRED)
target_link_libraries(main PRIVATE glfw)

glfw3 provides pkg-config modules:
```

图 4 OpenGL 库安装

如图，我下载好了三个所需的库。

## 二、绘制二维图形

由于三角形，正方形，线在提供的代码中均已经画好，我只需要绘制圆形和椭圆形即可。

## 分析绘制椭圆和圆形的函数 generateEllipsePoints:

```
// 获得椭圆/圆的每个顶点
void generateEllipsePoints(glm::vec2 vertices[], glm::vec3 colors[], int startVertexIndex, int numPoints,
                           glm::vec2 center, double scale, double verticalScale)
{
    double angleIncrement = (2 * M_PI) / numPoints;
    double currentAngle = M_PI / 2;

    for (int i = startVertexIndex; i < startVertexIndex + numPoints; ++i) {
        vertices[i] = getEllipseVertex(center, scale, verticalScale, currentAngle);
        if (verticalScale == 1.0) {
            colors[i] = glm::vec3(generateAngleColor(currentAngle), 0.0, 0.0);
        } else {
            colors[i] = RED;
        }
        currentAngle += angleIncrement;
    }
}
```

图 5 函数 generateEllipsePoints

这个函数用于生成圆形和椭圆形的顶点和颜色数据。具体来说，函数的作用是根据指定的中心位置、半径、以及点数来计算图形的各个顶点，并生成对应的颜色信息。

下面是函数的一些**参数**的介绍：

**glm::vec2 vertices[]**：传入的顶点数组，用来存储生成的顶点坐标。每个顶点对应椭圆或圆的一个点。

**glm::vec3 colors[]**：传入的颜色数组，用来存储生成的顶点颜色。

**int startVertexIndex**：顶点数组的起始位置，表示从这个索引开始存储顶点。

**int numPoints**：要生成的顶点数，也就是用于近似绘制椭圆或圆的点的数量。值越大，图形越精细、平滑。

**glm::vec2 center**：椭圆或圆的中心坐标，表示图形的中心点位置。

**double scale**：椭圆或圆的横向半径。对于圆，它是半径。

**double verticalScale**：椭圆的纵向比例。对于椭圆，它与 **scale** 一起控制形状的纵横比；对于圆，这个值通常设为 1.0。

### 详细的步骤：

(1)  $\text{angleIncrement} = (2 * M\_PI) / \text{numPoints};$

计算每两个相邻点之间的角度增量，确保生成的点均匀分布在椭圆或圆的周围。一个圆的周长是  $2 * M\_PI$ （以弧度表示），通过将其除以点数 **numPoints**，确定每个点之间的角度间隔。

(2)  $\text{currentAngle} = M\_PI / 2;$

初始化当前角度为  $M\_PI / 2$ （即 90 度，位于圆顶端）。这是椭圆/圆生成时的起始点。

(3) **for** 循环生成顶点：

**for (int i = startVertexIndex; i < startVertexIndex + numPoints; ++i)**：遍历从起始顶点索引到 **numPoints** 的范围。

**vertices[i] = getEllipseVertex(center, scale, verticalScale, currentAngle);** 调用 **getEllipseVertex** 函数，根据当前的角度 **currentAngle**、中心点 **center**、横向半径 **scale** 和纵向比例 **verticalScale** 来计算椭圆/圆的顶点坐标。生成的顶点存储在 **vertices[i]** 中。

**if (verticalScale == 1.0)** 如果 **verticalScale == 1.0**，则表明当前生成的是圆形，而不是椭圆。因此，顶点的颜色根据顶点的角度 **currentAngle** 来生成。

**colors[i]=glm::vec3(generateAngleColor(currentAngle),0.0,0.0);generateAngleColor(currentAngle)** 返回一个值，用于生成颜色的红色分量，随角度变化，颜色会逐渐改变。其余颜色分量为 0，颜色为不同强度的红色。

else 如果 verticalScale 不等于 1.0, 则表明生成的是椭圆。此时, 顶点的颜色统一设为 红色 (RED = glm::vec3(1.0, 0.0, 0.0)), 不根据角度生成颜色。

currentAngle += angleIncrement; 递增 currentAngle, 为下一个顶点计算新的角度。

知道了这些我就可以开始绘制我们的圆形和椭圆型了, 主要是两个函数需要变动, 一个是 **init** 函数, 另一个是 **display** 函数。

对于这两个函数, 我只需要根据绘制三角形的函数代码编写绘制椭圆和圆形的代码即可。

**init 函数:**

```
// 生成圆形和椭圆型上的顶点和颜色
generateEllipsePoints(circle_vertices, circle_colors, 0, CIRCLE_NUM_POINTS, glm::vec2(0.6, 0.7), 0.20, 1.0); // 圆形
generateEllipsePoints(ellipse_vertices, ellipse_colors, 0, ELLIPSE_NUM_POINTS, glm::vec2(-0.6, 0.7), 0.20, 0.6); // 椭圆型

// 初始化圆形数据
glGenVertexArrays(1, &vao[3]);
glBindVertexArray(vao[3]);

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");
glEnableVertexAttribArray(clocation);
glVertexAttribPointer(
    clocation,
    3,
    GL_FLOAT,
    GL_FALSE,
    sizeof(glm::vec3),
    BUFFER_OFFSET(0));
```

图 6 绘制圆形的代码

如图 6, 是绘制圆形的代码, 除了调用函数 generateEllipsePoints 的参数不一样, 其余部分和绘制三角形的代码都是一致的, 所以就不过多阐述。

```
// 初始化椭圆数据
glGenVertexArrays(1, &vao[4]);
glBindVertexArray(vao[4]);

glGenBuffers(1, &vbo[0]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(ellipse_vertices), ellipse_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(ellipse_colors), ellipse_colors, GL_STATIC_DRAW);
clocation = glGetAttribLocation(program, "vColor");
glEnableVertexAttribArray(clocation);
glVertexAttribPointer(
    clocation,
    3,
    GL_FLOAT,
    GL_FALSE,
    sizeof(glm::vec3),
    BUFFER_OFFSET(0));
```

图 7 绘制椭圆的代码

如图 7, 绘制椭圆和绘制圆形只有位置参数和比例参数不同, 其余部分都一致。

**display 函数:**

```
// @TODO: 绘制圆
glBindVertexArray(vao[3]);
glDrawArrays(GL_TRIANGLE_FAN, 0, CIRCLE_NUM_POINTS);
// @TODO: 绘制椭圆

glBindVertexArray(vao[4]);
glDrawArrays(GL_TRIANGLE_FAN, 0, ELLIPSE_NUM_POINTS);
```

图 8 显示图形

与显示三角形和正方形一样，先使用 `glBindVertexArray` 函数绑定 VAO，再使用 `glDrawArrays` 函数绘制图形。

运行程序，绘制的图形如下图所示，和实验的要求一致。

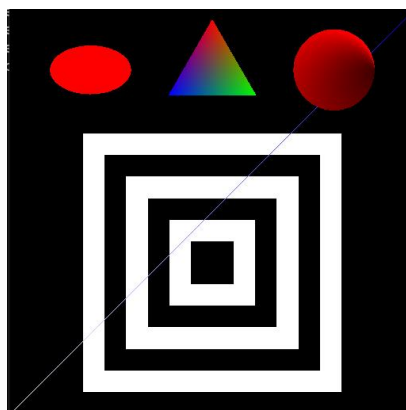


图 9 二维图片绘制结果

### 三、绘制自己的图形

我绘制的图形为“奔驰 S1000 公路救援车”。本图形从设计到实现均有本人独自完成，不存在任何抄袭与网上借鉴。

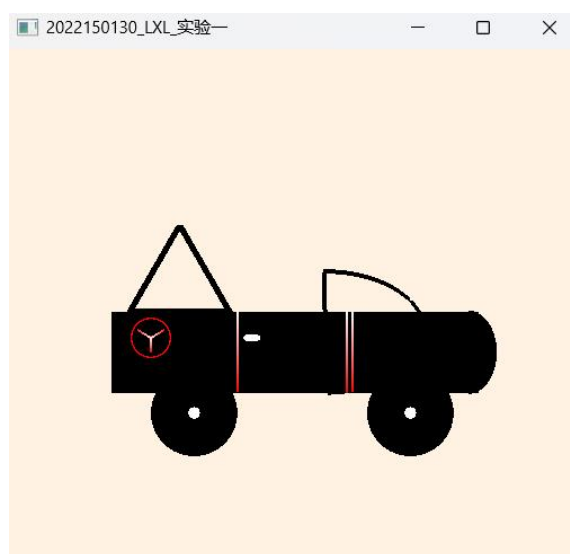


图 10 奔驰 S1000 公路救援车

#### 1. 设计理念：

这款救险车的设计灵感源自基本几何元素，如三角形、长方形、圆形、椭圆形及线条等。具体而言，车身主体采用了长方形的结构，车头则运用了半椭圆形的设计，赋予其流畅的外观感。挡风玻璃采用了空心的四分之一椭圆形，巧妙呼应整体造型。车门通过简洁的线条勾勒，体现简约实用的美学。公路救险标志则通过三角形元素进行呈现，直观且具备标识性。此外，车尾的奔驰标志则结合圆形和线条设计，精准展现品牌特征与车辆功能的完美融合。

#### 2. 实现过程

## 2.1 实现的基本函数

我对原来提供的基本元素绘制的函数进行了一些改写，使其更契合我设计的需求。

### 2.1.1 三角形

```
void generateTrianglePoints2(glm::vec2 vertices[], glm::vec3 colors[], int
startVertexIndex, glm::vec2 scale, glm::vec2 center)
{
    for (int i = 0; i < 3; ++i) {
        double currentAngle = getTriangleAngle(i);
        vertices[startVertexIndex + i] = glm::vec2(sin(currentAngle),
cos(currentAngle)) * scale + center;
    }
    colors[startVertexIndex] = BLACK;
    colors[startVertexIndex + 1] = BLACK;
    colors[startVertexIndex + 2] = BLACK;
}
```

对于这个函数实现，我主要修改了颜色的实现，从原来的彩色变成了黑色，其它地方就没有变动。

### 2.1.2 长方形

```
void generateSquarePoints2(glm::vec2 vertices[], glm::vec3 colors[], int
squareNumber, int startVertexIndex, glm::vec2 scale, glm::vec2 center, const
glm::vec3 color)
{
    int vertexIndex = startVertexIndex;
    for (int i = 0; i < 4; ++i) {
        double currentAngle = getSquareAngle(i);
        vertices[vertexIndex] = glm::vec2(sin(currentAngle),
cos(currentAngle)) * scale + center;
        colors[vertexIndex] = color;
        vertexIndex++;
    }
}
```

对于绘制长方形的函数，我加入了一些参数，如长方形的大小，中心点位置，颜色，这样使得这个函数有更多的功能，在绘制图案的时候更加方便。

### 2.1.3 圆形和椭圆型

```
void generateEllipsePoints2(glm::vec2 vertices[], glm::vec3 colors[], int
startVertexIndex, int numPoints,
    glm::vec2 center, double scale, double verticalScale, glm::vec3 color)
{
    double angleIncrement = (2 * M_PI) / numPoints;
    double currentAngle = M_PI / 2;
    for (int i = startVertexIndex; i < startVertexIndex + numPoints; ++i)
    {
        vertices[i] = getEllipseVertex(center, scale, verticalScale,
```

```

currentAngle);
    if (1) {
        colors[i] = color;
    }
    currentAngle += angleIncrement;
}
}

```

同样的，对于圆形和椭圆的绘制函数，我也加入了一些参数，如颜色，这样绘制就能绘制不同颜色的基本元素了。

```

void generateEllipsePoints3(glm::vec2 vertices[], glm::vec3 colors[], int
startVertexIndex, int numPoints,
    glm::vec2 center, double scale, double verticalScale, const glm::vec3
color)
{
    // 半圆范围的角度增量: 0 到 PI
    double angleIncrement = M_PI / numPoints; // 半圆的角度增量
    double currentAngle = 0.0; // 从 0 开始 (上半圆)

    for (int i = startVertexIndex; i < startVertexIndex + numPoints; ++i)
    {
        vertices[i] = getEllipseVertex(center, scale, verticalScale,
currentAngle);
        if (verticalScale == 1.0) {
            colors[i] = color;
        }
        currentAngle += angleIncrement;
    }
}

```

同时，我还加入了绘制半圆的函数，当然绘制四分之一圆函数也是同理，就不过多阐述。

#### 2.1.4 线条

```

void generateLinePoints2(glm::vec2 vertices[], glm::vec3 colors[], int
startVertexIndex, glm::vec2 start, glm::vec2 end)
{
    vertices[startVertexIndex] = start;
    vertices[startVertexIndex + 1] = end;

    colors[startVertexIndex] = WHITE;
    colors[startVertexIndex + 1] = RED;
}

```

我对绘制线条的函数加入了起始点和终点两个参数，同时我设计了起始点和终点为不一样的颜色，这样就可以画出渐变的效果。

#### 2.2 VAO 的准备



### 2.2.1 车轮

```
//车轮
glm::vec2 car_circle_vertices1[CIRCLE_NUM_POINTS];
glm::vec3 car_circle_colors1[CIRCLE_NUM_POINTS];

glm::vec2 car_circle_vertices2[CIRCLE_NUM_POINTS];
glm::vec3 car_circle_colors2[CIRCLE_NUM_POINTS];

generateEllipsePoints2(car_circle_vertices1, car_circle_colors1, 0,
CIRCLE_NUM_POINTS, glm::vec2(0.4, -0.26), 0.15, 1.0, BLACK); // 车轮 1
generateEllipsePoints2(car_circle_vertices2, car_circle_colors2, 0,
CIRCLE_NUM_POINTS, glm::vec2(-0.35, -0.26), 0.15, 1.0, BLACK); // 车轮 2
//车轮上的白点
glm::vec2 car_circle_vertices3[CIRCLE_NUM_POINTS];
glm::vec3 car_circle_colors3[CIRCLE_NUM_POINTS];
generateEllipsePoints2(car_circle_vertices3, car_circle_colors3, 0,
CIRCLE_NUM_POINTS, glm::vec2(0.4, -0.26), 0.02, 1.0, WHITE); // 白点 1
generateEllipsePoints2(car_circle_vertices4, car_circle_colors4, 0,
CIRCLE_NUM_POINTS, glm::vec2(-0.35, -0.26), 0.02, 1.0, WHITE); // 白点 2
```

这一部分代码生成了救险车的两个车轮及其中心的白点，以增强视觉效果。

### 2.2.2 车身

```
//车身
glm::vec2 car_square_vertices[4];
glm::vec3 car_square_colors[4];
glm::vec2 car_scale1(0.90, 0.20);

glm::vec2 car_center1(0.0, -0.05);
generateSquarePoints2(car_square_vertices, car_square_colors, 4, 0,
car_scale1, car_center1, BLACK);//生成车身点和颜色
```

这一段代码创建了车身的长方形结构，提供车辆的主体外形。

### 2.2.3 挡风玻璃

```
//挡风玻璃
glm::vec2 car_ellipse_vertices[ELLIPSE_NUM_POINTS];
glm::vec3 car_ellipse_colors[ELLIPSE_NUM_POINTS];
generateEllipsePoints3(car_ellipse_vertices, car_ellipse_colors, 0,
CIRCLE_NUM_POINTS, glm::vec2(0.1, 0.02), 0.35, 0.60, BLACK); // 生成挡风
玻璃的点和颜色
```

此代码段负责生成挡风玻璃的形状，为车辆提供视野。

### 2.2.4 车头

```
//车头
glm::vec2 car_ellipse_vertices2[ELLIPSE_NUM_POINTS];
glm::vec3 car_ellipse_colors2[ELLIPSE_NUM_POINTS];
generateEllipsePoints3(car_ellipse_vertices2, car_ellipse_colors2, 0,
CIRCLE_NUM_POINTS, glm::vec2(0.6, -0.05), 0.1, 1.45, BLACK); // 生成车头
点和颜色
```

这一部分代码生成了车头的形状，增强了车辆的外观特征。

### 2.2.5 车门

```
glm::vec2 car_circle_vertices5[CIRCLE_NUM_POINTS];
glm::vec3 car_circle_colors5[CIRCLE_NUM_POINTS];
generateEllipsePoints2(car_circle_vertices5, car_circle_colors5, 0,
CIRCLE_NUM_POINTS, glm::vec2(-0.15,-0.0), 0.03, 0.45, WHITE); // 门把手
```

```
//车门的线条
glm::vec2 line_vertices1[LINE_NUM_POINTS];
glm::vec3 line_colors1[LINE_NUM_POINTS];
glm::vec2 line_vertices2[LINE_NUM_POINTS];
glm::vec3 line_colors2[LINE_NUM_POINTS];
glm::vec2 line_vertices3[LINE_NUM_POINTS];
glm::vec3 line_colors3[LINE_NUM_POINTS];
generateLinePoints2(line_vertices1, line_colors1, 0, glm::vec2(-0.20,
0.09), glm::vec2(-0.20, -0.19));
generateLinePoints2(line_vertices2, line_colors2, 0, glm::vec2(0.2, 0.09),
glm::vec2(0.2, -0.19));
generateLinePoints2(line_vertices3, line_colors3, 0, glm::vec2(0.18,
0.09), glm::vec2(0.18, -0.19));
```

这部分代码定义了车门的把手和轮廓线条，增加了车门的细节。

### 2.2.6 奔驰车标

//外围的大圈和中心点

```
glm::vec2 car_circle_vertices7[100];
glm::vec3 car_circle_colors7[100];
glm::vec2 car_circle_vertices8[100];
glm::vec3 car_circle_colors8[100];
glm::vec2 car_circle_vertices9[100];
glm::vec3 car_circle_colors9[100];
generateEllipsePoints2(car_circle_vertices7, car_circle_colors7, 0, 100,
glm::vec2(-0.5, -0.0), 0.07, 1.0, RED);
generateEllipsePoints2(car_circle_vertices8, car_circle_colors8, 0, 100,
glm::vec2(-0.5, -0.0), 0.065, 1.0, BLACK);
```

```

generateEllipsePoints2(car_circle_vertices9, car_circle_colors9, 0, 100,
glm::vec2(-0.5, -0.0), 0.007, 1.0, RED);

//三条横线
glm::vec2 line_vertices7[LINE_NUM_POINTS];
glm::vec3 line_colors7[LINE_NUM_POINTS];
glm::vec2 line_vertices8[LINE_NUM_POINTS];
glm::vec3 line_colors8[LINE_NUM_POINTS];
glm::vec2 line_vertices9[LINE_NUM_POINTS];
glm::vec3 line_colors9[LINE_NUM_POINTS];
generateLinePoints2(line_vertices7, line_colors7, 0, glm::vec2(-0.5,
-0.0), glm::vec2(-0.55, 0.03));
generateLinePoints2(line_vertices8, line_colors8, 0, glm::vec2(-0.5,
-0.0), glm::vec2(-0.45, 0.03));
generateLinePoints2(line_vertices9, line_colors9, 0, glm::vec2(-0.5,
-0.0), glm::vec2(-0.5, -0.05));

```

此段代码绘制了奔驰车标的外圈、中心点和横线，增强了品牌标识的表现。

#### 2.2.7 三角指示牌

// 指示牌

```

glm::vec2 triangle_vertices2[TRIANGLE_NUM_POINTS];
glm::vec3 triangle_colors2[TRIANGLE_NUM_POINTS];
generateTrianglePoints2(triangle_vertices2, triangle_colors2, 0,
glm::vec2(0.2,0.2), glm::vec2(-0.4,0.19));

```

此部分代码生成了公路救险的三角指示牌，增加了安全警示的元素。

#### 2.3 绘制车辆

```

glBindVertexArray(vao[x]);
glDrawArrays(GL_TRIANGLE_FAN, 0, NUM_POINTS);

```

使用这两个函数，逐个把 VAO 绑定然后绘制即可。

### 实验结论:

在本次实验中，我成功搭建了 OpenGL 所需的开发环境，并绘制了预期的二维图形，设计并实现了“奔驰 S1000 公路救援车”。这一过程充满了挑战和收获。

首先，在环境配置阶段，我遇到了诸多困难。由于网络不稳定、版本不匹配和安装路径的问题，多个依赖库无法正确下载和安装。经过反复尝试、逐一检查错误信息，并在助教的指导下，我最终成功解决了这些问题。这个过程让我深刻意识到，与环境配置相关的文件最好安装在系统盘上，以减少潜在的兼容性问题。

在绘制指定图形时，最大的挑战在于理解数百行代码的具体功能，并在此基础上进行修改，以实现预期的视觉效果。特别是在设计我自己的图形时，这一过程显得尤为复杂。设计不仅需要创意，还涉及到细致的调整与反复实验，这让我花费了大量时间进行优化与修改。尽管最终的成果可能未尽完美，但我依然感到无比满足，因为我成功地将抽象的设计理念转化为实际的图形表现。

总的来说，这次实验让我不仅学到了 OpenGL 的绘图技巧，还提高了我的问题解决能力和设计思维，虽然经历了不少波折，但我最终成功完成了实验，收获颇丰。

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。