**实验五 光线追踪综合实验**

姓名 蒋文强 学号 21013017 专业班级 计211 成绩\_\_\_\_\_\_\_\_\_

实验日期 2023.12.4 实验地点 信息楼215 指导教师(签名)

一．实验目的

1、 学习光线追踪原理。

2、 利用GLSL编程实现对模型的渲染。

二．实验工具与设备

**Visual Studio 2019.**

三、实验内容

在本科学习平台（s.ecust.edu.cn）资料栏下，下载以下文件：“实验五资料.zip”

1. 学习着色器编程

利用“实验五资料.zip”中的“实验五-环境安装指南.docx”配置glew，再配置好FreeGLUT，结合“ShaderProgram.ppt”，运行“ShaderTest.rar”实例，总结出着色器编程的实现要点；编写绘制一个四边形的着色器编程实例，给出修改要点，并截图。

**四边形的着色器编程实例:**

1. 修改display中的代码：

void display2()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

//四边形

glDrawArrays(GL\_TRIANGLE\_STRIP, 0, 4);

//三角形

//glDrawArrays(GL\_TRIANGLES,0,3);

glFlush();

}

GL\_TRIANGLE\_STRIP是一种特殊的多边形类型，使用连续的顶点来生成一个三角形带，可以用来绘制多个三角形组成的复杂形状。四边形被视为两个三角形，每个三角形使用三个连续的顶点。

1. **修改顶点坐标：**

void Model::InitModel()

{ /\*三角形

GLfloat vertices[21] =

{

0,0,-5, 1,0,0,1,

0,5,-5, 0,1,0,1,

5,0,-5, 0,0,1,1,

};\*/

GLfloat vertices[28] = {

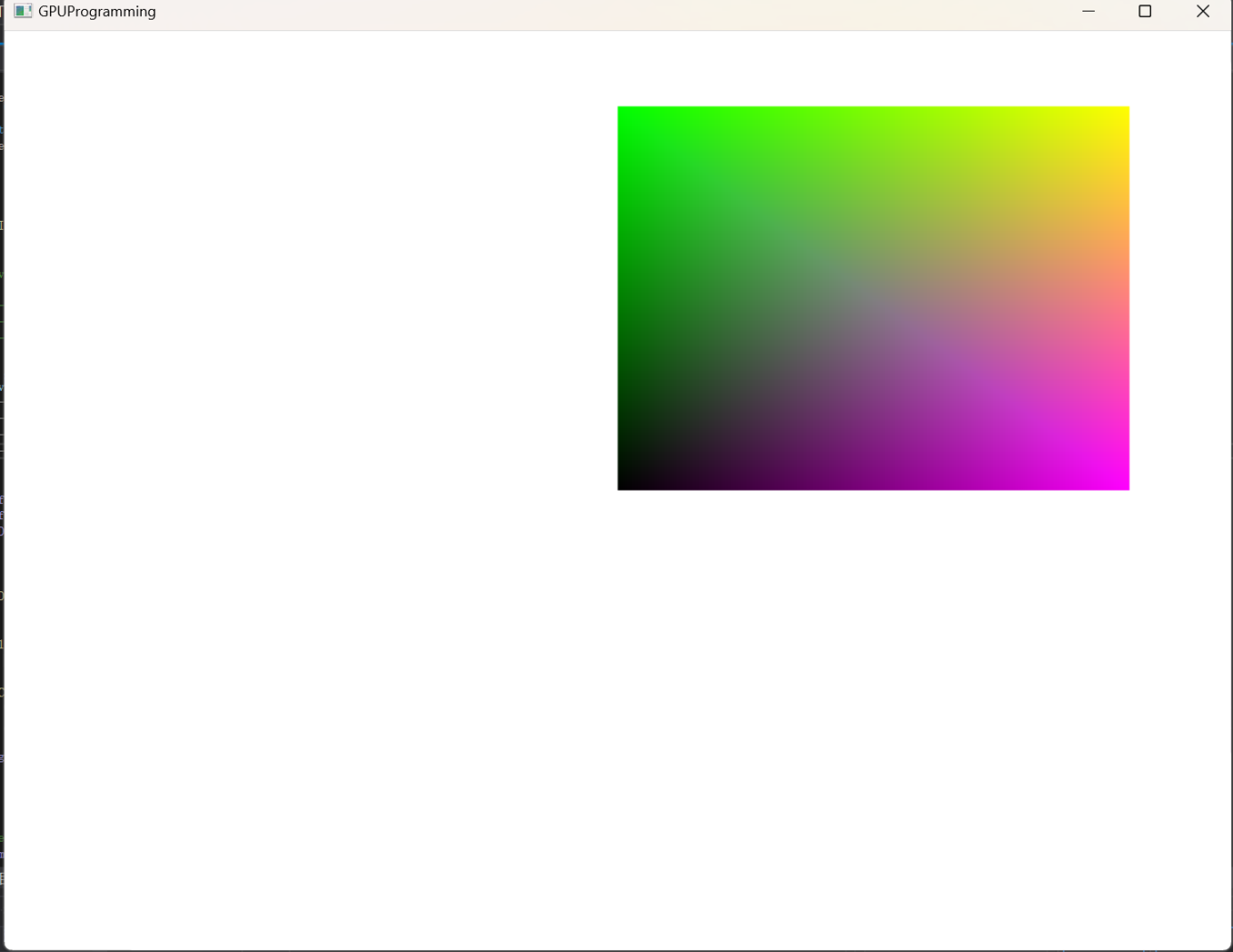
0,0,-5, 1,0,0,1,

0,5,-5, 0,1,0,1,

5,0,-5, 0,0,1,1,

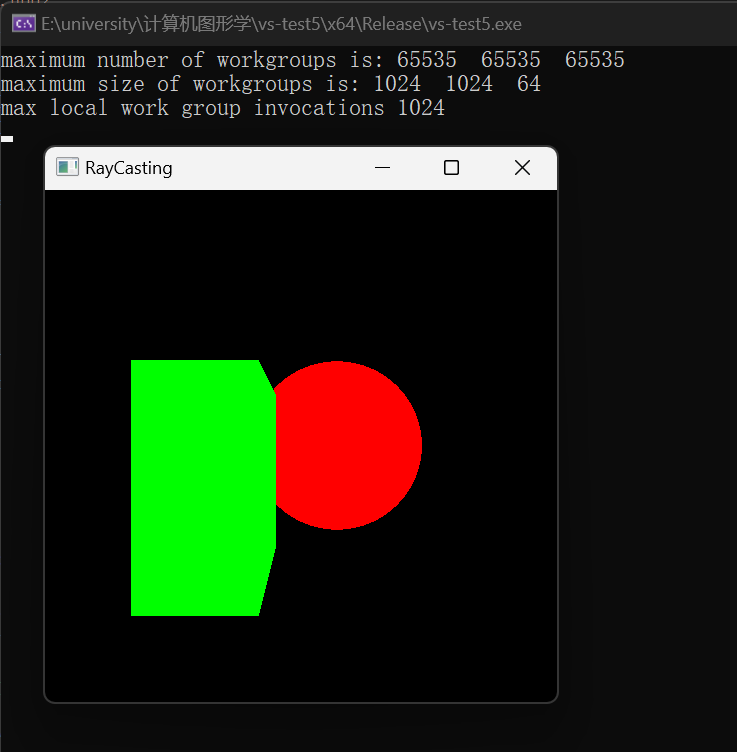
5,5,-5, 0,1,1,1,

};



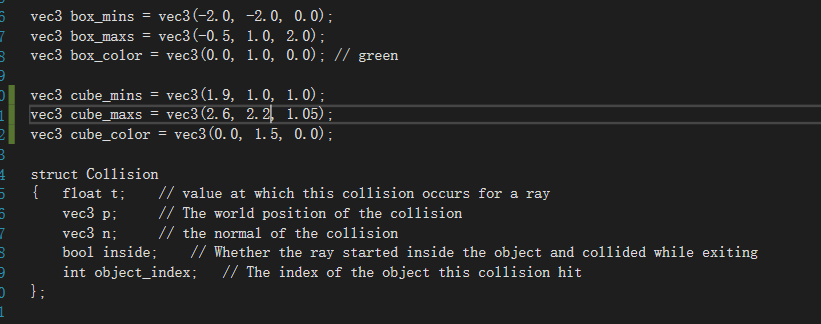
1. RayCasting编程

利用“实验五资料.zip”中的“实验五-环境安装指南.docx”和“RayCasting”代码，分析代码过程，简单写出程序的实现要点；在原程序界面球体右方再添加一个物体（长方体、正方体、球体都可），并给出修改要点，并截图。



**在原程序界面球体右方再添加一个物体:**

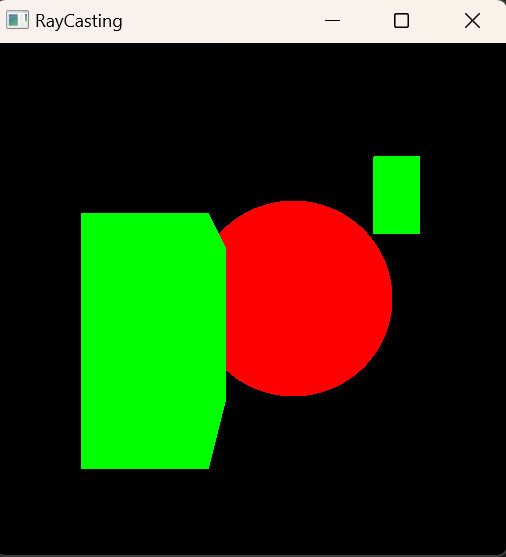
1. **打开.glsl文件**
2. **修改坐标，增加正方体**

****

1. **增加函数如下：**

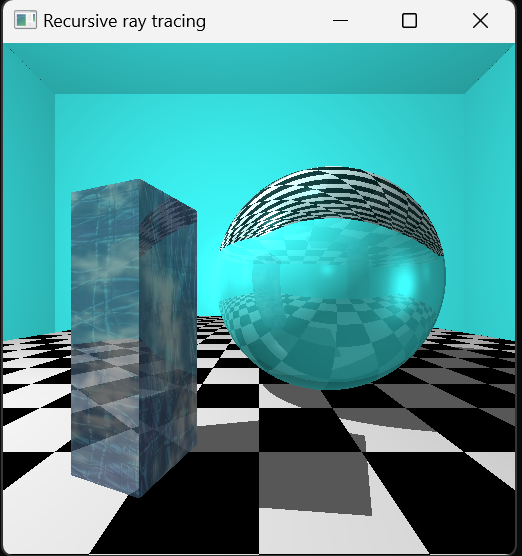


**实验结果：**



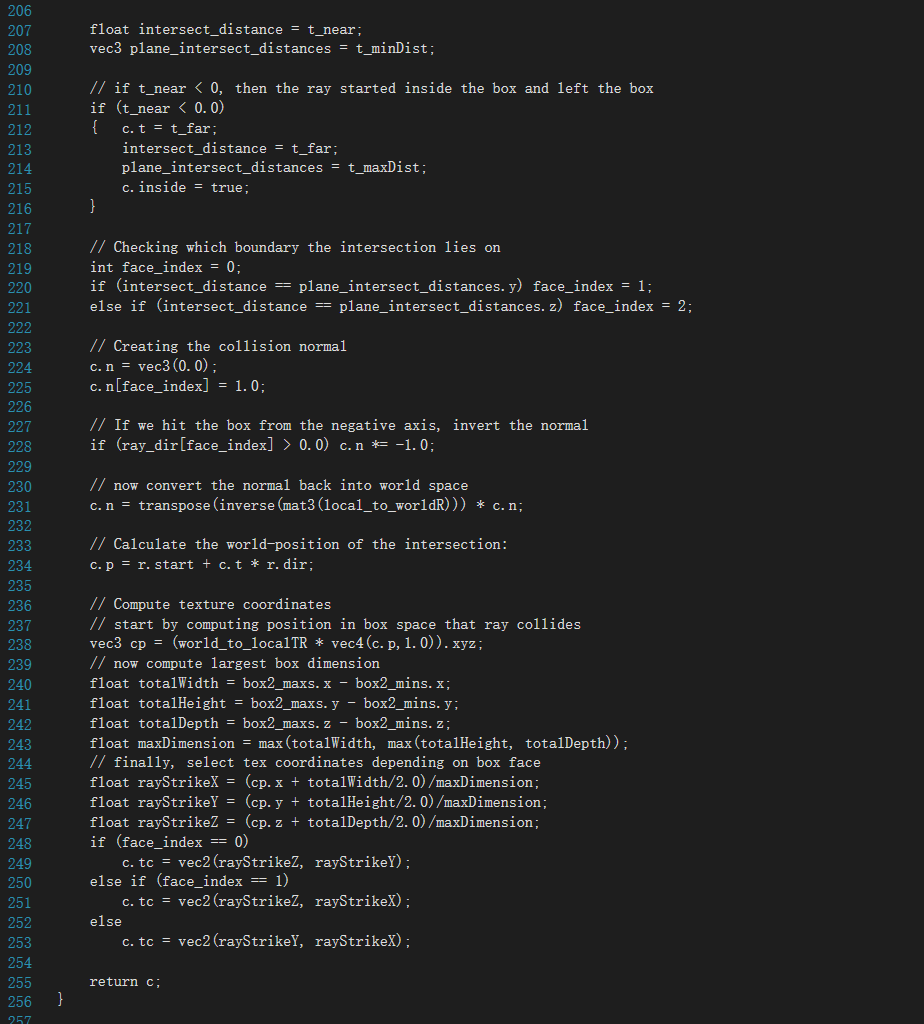
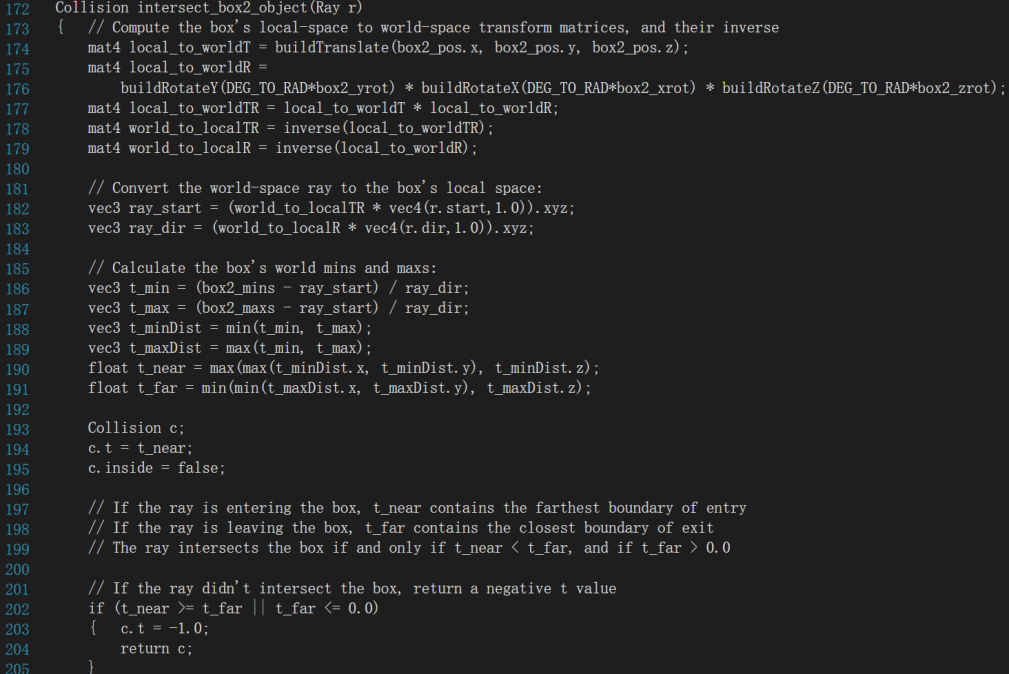
1. Recursive ray tracing编程

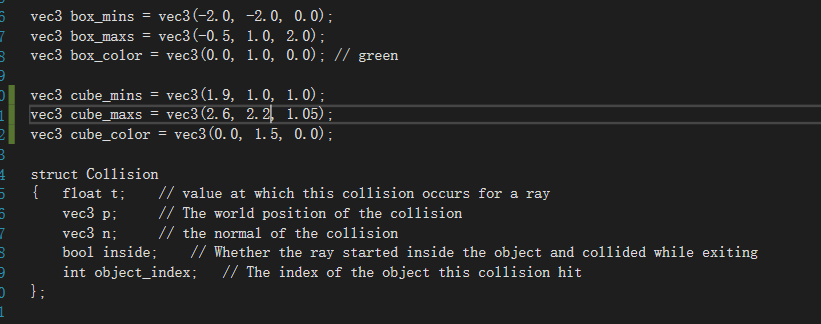
利用“实验五资料.zip”中的“实验五-环境安装指南.docx”和“Recursive ray tracing”代码，分析代码过程，简单写出程序的实现要点；在原程序界面球体右方再添加一个物体（长方体、正方体、球体都可），并给出修改要点，并截图。



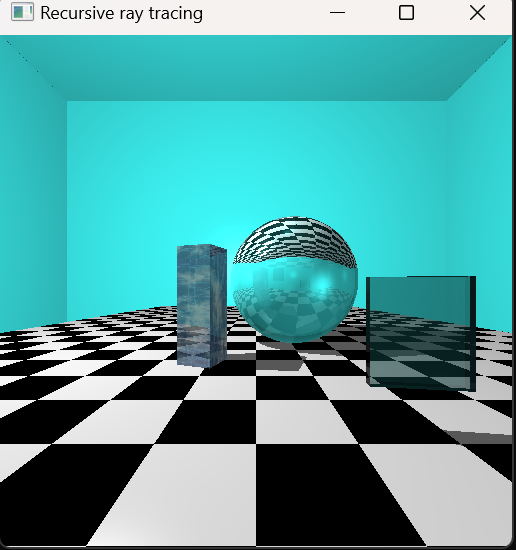
**在原程序界面球体右方再添加一个物体:**

1. **打开.glsl文件**
2. **修改坐标，增加长方体**
3. **增加函数如下：**





**实验结果：**



四、思考题

写OpenGL程序时，这些库各自有不同的作用。FreeGLUT提供了方便的工具，能够创建和管理窗口，处理用户的输入（比如键盘和鼠标事件），并在屏幕上绘制图形。它还支持一些特殊的OpenGL功能，比如创建剪切窗口和管理多重缓冲。

GLEW（OpenGL扩展库）处理OpenGL的扩展和核心功能，帮助轻松地使用最新的OpenGL特性。它允许加载和使用OpenGL的扩展功能，能够使用最新的OpenGL功能，而不必担心兼容性问题。

GLFW用于创建和管理窗口，并处理用户的输入和事件。它简化了窗口和OpenGL上下文的创建，同时处理用户输入和窗口事件，比如键盘输入或窗口大小变化。

GLAD（OpenGL辅助库）用于加载和使用OpenGL的核心和扩展功能。它简化了OpenGL初始化过程，自动生成加载OpenGL函数的代码，使得使用OpenGL更加方便。

**2.利用着色器实现光线追踪的优缺点**？

优点：

实时渲染：着色器可用于实时渲染，使其在需要实时渲染的应用程序中非常有用，如游戏和交互式应用。

灵活性和可扩展性：着色器语言（如GLSL、HLSL等）非常灵活，容易扩展和修改。这使得着色器可以用于实现各种复杂的光线追踪算法，包括路径追踪、光子映射等。

易于并行处理：着色器能够轻松并行处理，充分利用GPU的并行计算能力，提高光线追踪的计算效率。

缺点：

计算量大：光线追踪需要大量计算，特别是在处理复杂场景和高质量渲染效果时。这可能导致着色器计算时间较长，从而影响渲染速度。

内存带宽限制：着色器需要从GPU内存中读取数据，可能受到内存带宽的限制。如果内存带宽不足，可能导致着色器计算速度变慢。

着色器编程复杂：着色器编程需要一定的经验和技能，尤其是对于光线追踪等复杂的算法。这可能增加开发的难度和成本。

不适用于所有场景：尽管着色器可以实现实时的光线追踪效果，但对于某些特定场景和需求，可能更适合使用其他渲染技术，如光栅化或混合渲染技术。