

实验： 立方体旋转

华中科技大学软件学院 万琳





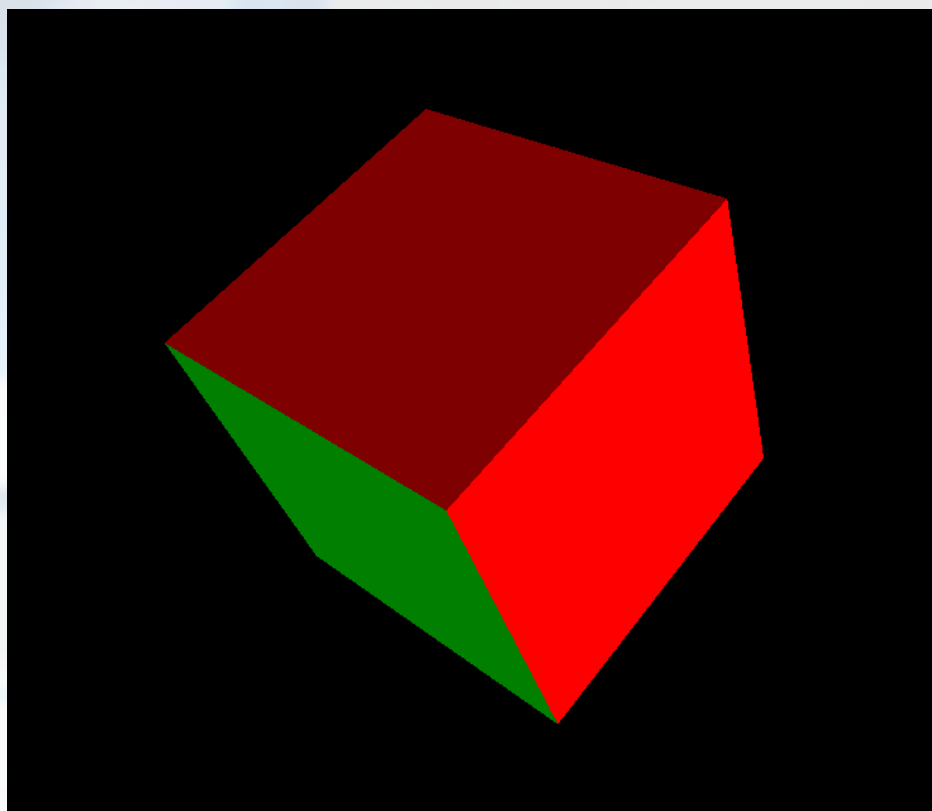
提纲

- ① 实验要求
- ② 程序流程
- ③ 要点解析
- ④ 程序演示



实验要求

在窗口中绘制一个旋转的立方体。



2

程序流程

在窗口中绘制一个旋转的立方体。





要点解析

➤问题分析



立方体



旋转

3

要点解析

➤问题分析

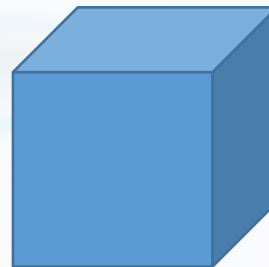


立方体

之前：



现在：



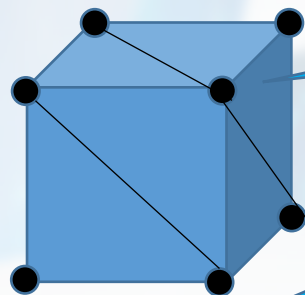
3

要点解析



顶点数据定义

现在：



六个面，每个面两个三角形，
每个三角形三个顶点

顶点的局部坐标

一共36行
对应12个三角形

```
float vertices[] = {
```

```
-0.5f, -0.5f, -0.5f,
```

```
0.5f, -0.5f, -0.5f,
```

```
0.5f, 0.5f, -0.5f,
```

```
0.5f, 0.5f, -0.5f,
```

```
-0.5f, 0.5f, -0.5f,
```

```
.....
```

```
}
```

顶点的颜色值

```
1.0f, 0.0f, 0.0f,
```

```
1.0f, 0.0f, 0.0f,
```

```
1.0f, 0.0f, 0.0f,
```

```
1.0f, 0.0f, 0.0f,
```

```
1.0f, 0.0f, 0.0f,
```

3

要点解析

➤问题分析



旋转

怎么转起来呢？



模型
变换



视图
变换



投影
变换



屏幕
映射

Model

View

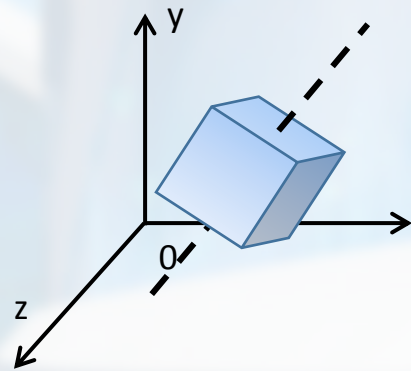
Projection

PVM矩阵

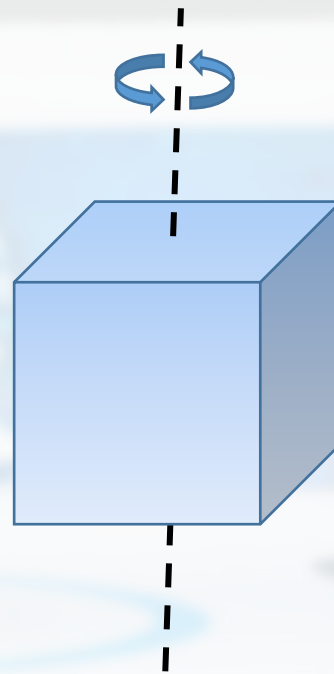
3

要点解析

➤ 一个比较自然的方法：在模型变换中想办法

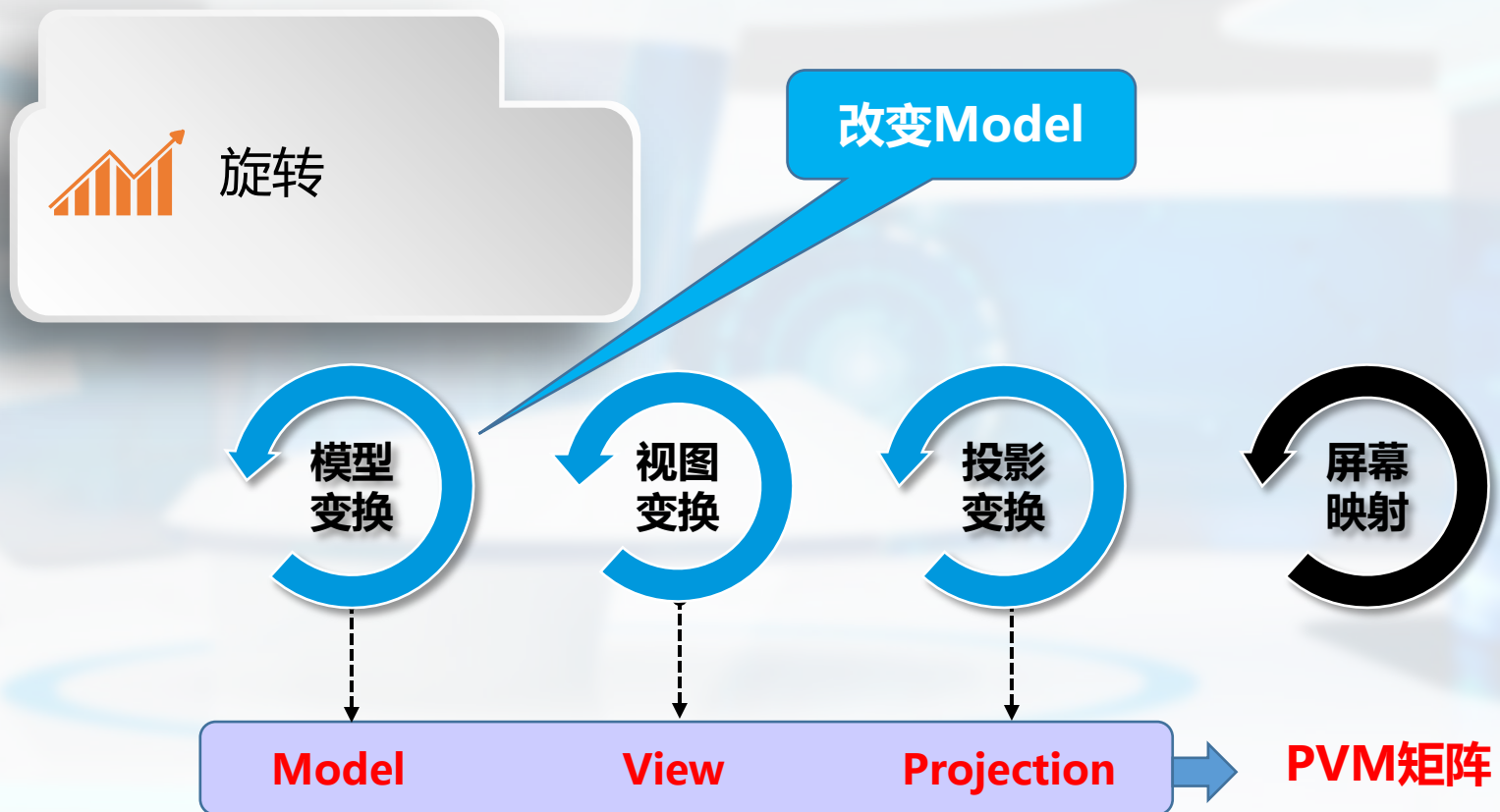


世界坐标系



3

要点解析



3

要点解析

改变Model

模型
变换

Model

// Transform坐标变换矩阵

```
glm::mat4 model(1); // model矩阵, 局部坐标变换至世界坐标  
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f)); // 先移动再旋转  
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.5f, 1.0f, 0.0f));
```

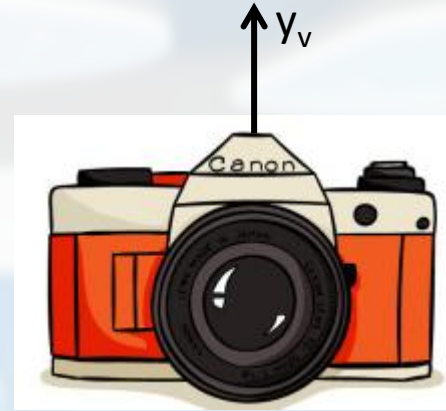
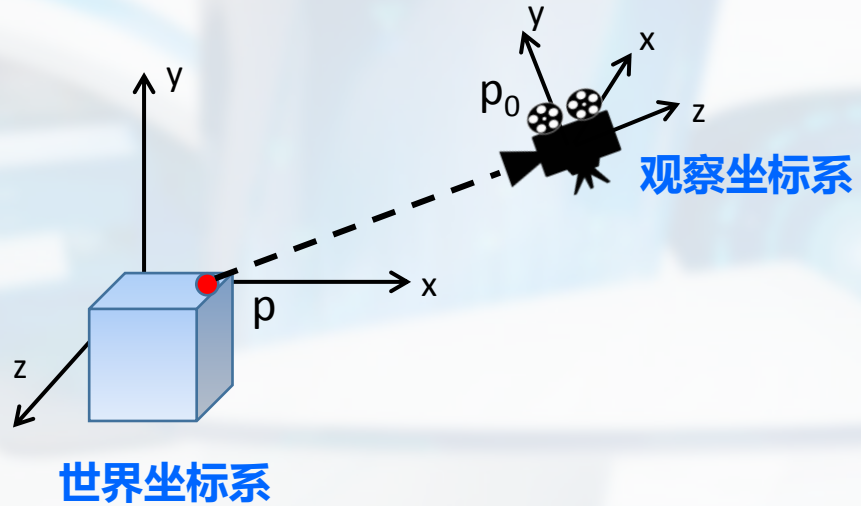
```
glm::mat4 view(1); // view矩阵, 世界坐标变换至观察坐标系  
view = glm::lookAt(camera_position, camera_position + camera_front,  
camera_up);
```

```
glm::mat4 projection(1); // projection矩阵, 投影矩阵  
projection = glm::perspective(glm::radians(fov), (float)screen_width /  
screen_height, 0.1f, 100.0f);
```

3

要点解析

➤ 另一种方法：在视图变换中想办法



3

要点解析



旋转

改变View

模型
变换

视图
变换

投影
变换

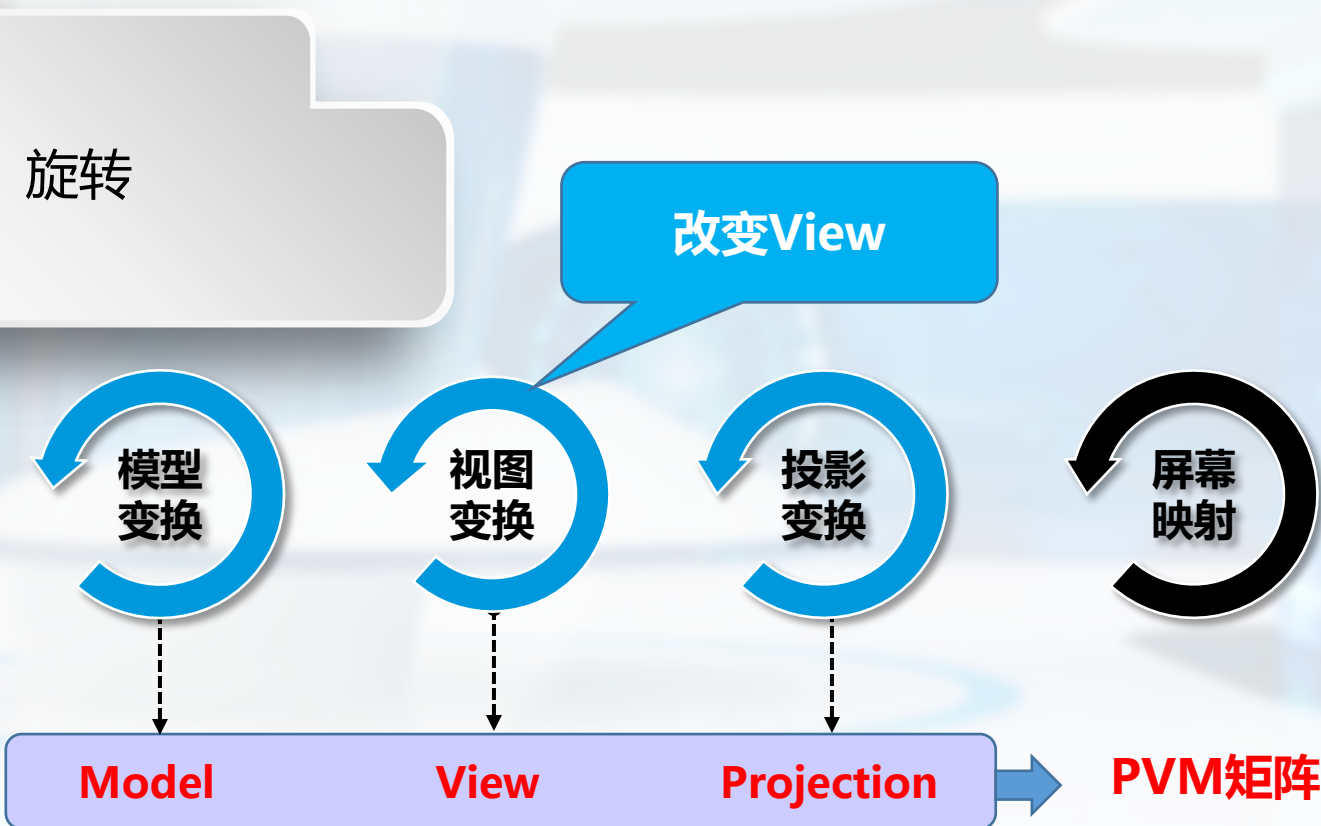
屏幕
映射

Model

View

Projection

PVM矩阵



3

要点解析

改变View

视图
变换

View

如何改变？

如何改变？

// Transform坐标变换矩阵

```
glm::mat4 model(1); // model矩阵，局部坐标变换至世界坐标  
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.5f, 1.0f, 0.0f));  
glm::mat4 view(1); // view矩阵，世界坐标变换至观察坐标系
```

```
view = glm::lookAt(camera_position, camera_position + camera_front,  
camera_up);
```

glm::mat4 projection(1); // projection矩阵，投影矩阵

```
projection = glm::perspective(glm::radians(fov), (float)screen_width /  
screen_height, 0.1f, 100.0f);
```

3

要点解析



//根据键盘输入修改camara的参数

```
void ProcessInput(GLFWwindow* window)
{
    .....
    float camera_speed = 2.5f * delta_time;
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
    {
        camera_position += camera_speed * camera_front;
    }
    .....
}
```

3

要点解析

改变View

视图
变换

View

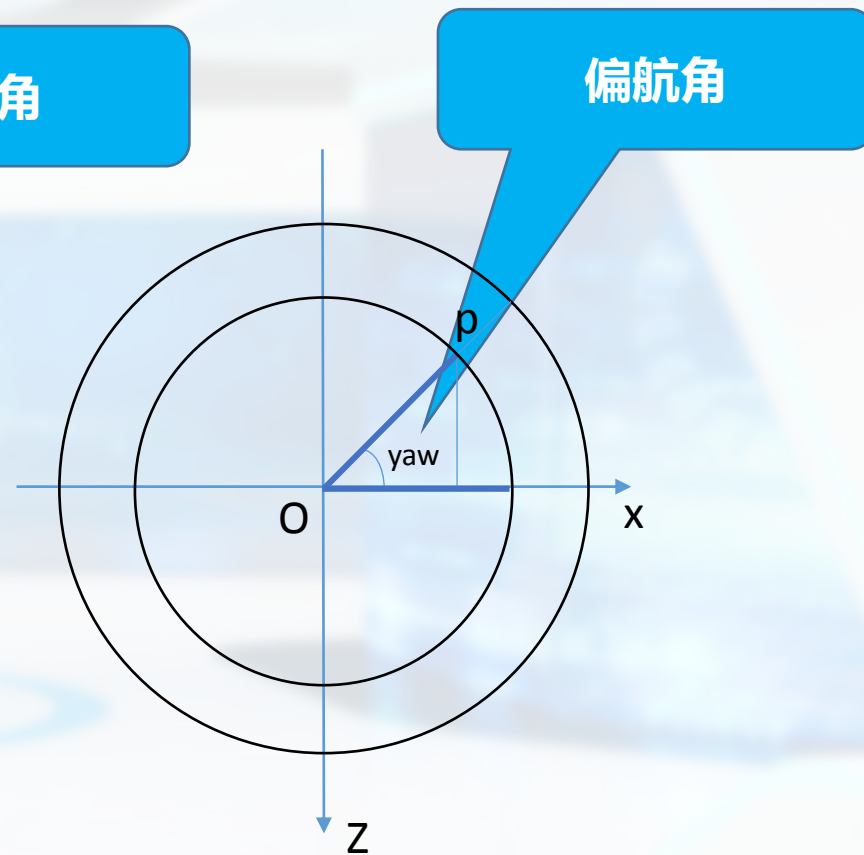
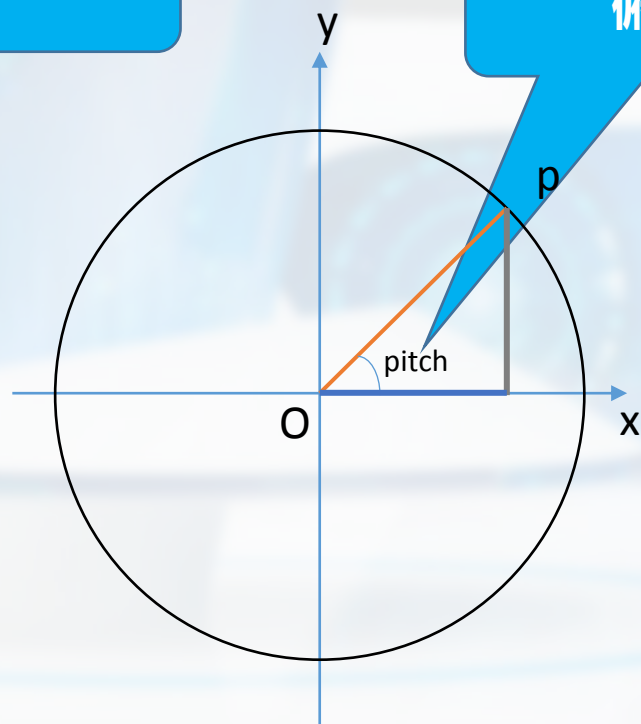
如何改变相机方向？

回忆：单位球体上一顶点的坐标可以通过 α 角和 β 角表示出来
是否可以使用两个角度去表示相机的方向？

```
//根据鼠标移动修改camara的参数  
void MouseCallback(GLFWwindow*window, double xpos, double ypos)  
{  
    .....  
    glm::vec3 front;  
    front.x = cos(glm::radians(yaw))*cos(glm::radians(pitch));  
    front.y = sin(glm::radians(pitch));  
    front.z = sin(glm::radians(yaw)*cos(glm::radians(pitch)));  
    camera_front = glm::normalize(front);  
}
```


3

要点解析



说明：语音中“从x轴看”应该为“从y轴看”。



要点解析

➤问题分析



立方体



旋转

3

要点解析

➤传入摄像机

最后，每次绘制都要将model，view和projection矩阵传入着色器

```
int model_location = glGetUniformLocation(our_shader.ID, "model");  
//获取着色器内某个参数的位置  
glUniformMatrix4fv(model_location, 1, GL_FALSE, glm::value_ptr(model));  
//写入参数值
```

3

要点解析

➤ 其余重要问题

Shader
封装

深度
测试

键盘鼠标
响应



3

要点解析

➤ 其余重要问题

Shader 封装

将读取创建绑定Shader的操作封装成一个类

//根据路径读取创建着色器

```
Shader(const GLchar* vertex_shader_path, const GLchar*  
fragment_shader_path);
```

//绑定着色器

```
void Use();
```

//用法

```
Shader our_shader("res/shader/task2.vs", "res/shader/task2.fs");//加载着色器
```

```
our_shader.Use();//使用our_shader着色器对象
```

```
our_shader.ID//着色器的ID值
```

3

要点解析

➤ 其余重要问题

深度 测试

如何开启深度测试？

```
glEnable(GL_DEPTH_TEST); // 开启深度测试
```

设置深度测试方法

```
glDepthFunc(GL_LESS); // 深度测试 输入的深度值小于参考值，则通过
```



3

要点解析

➤ 其余重要问题

两种方式实现响应

键盘鼠标 响应

绘制循环中，每绘制一次检测某个按键是否按下

```
void ProcessInput(GLFWwindow* window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, true); // 关闭窗口
    }
}
```

在窗口创建成功后，通过glfw注册鼠标响应

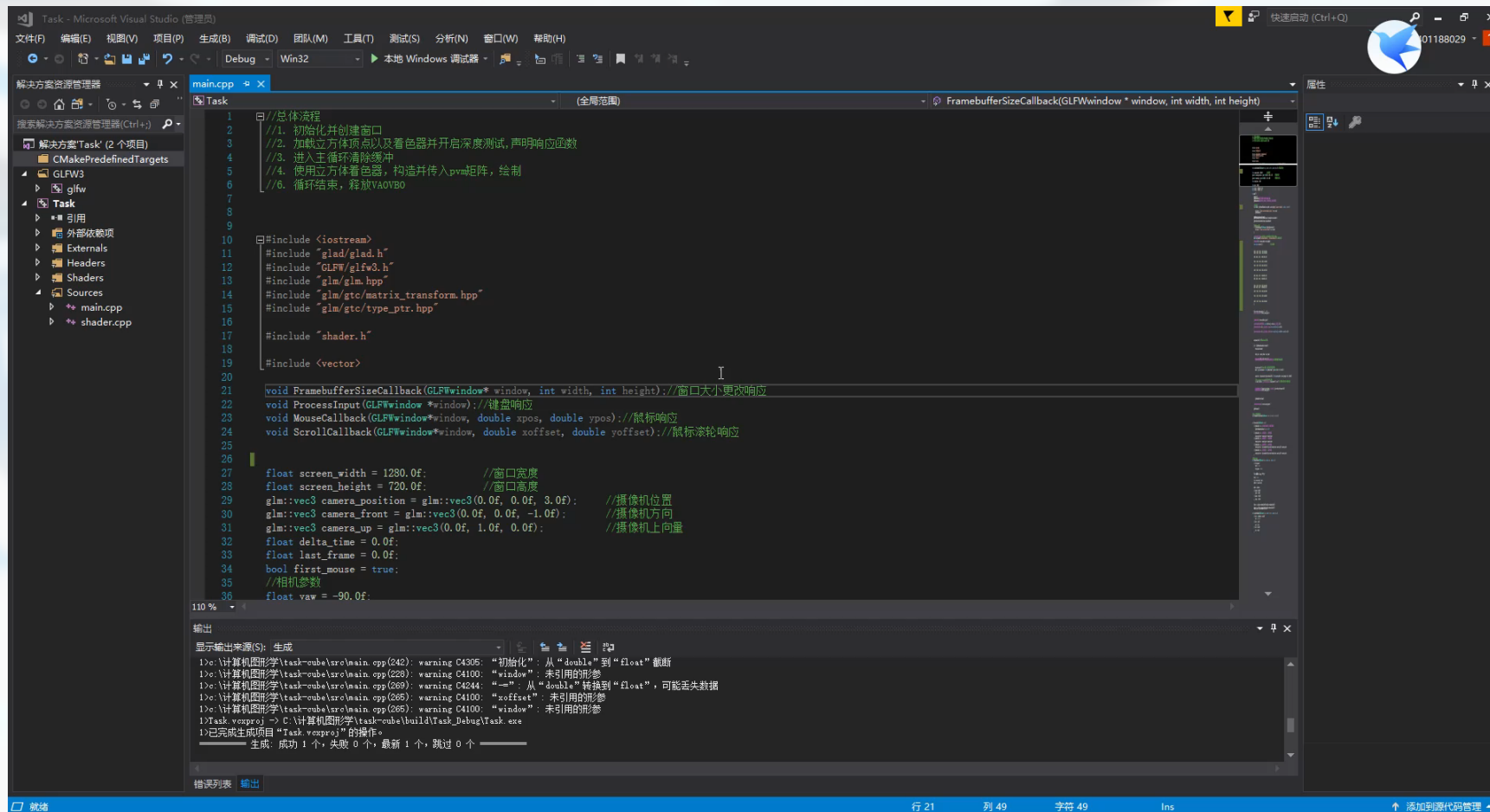
```
glfwSetCursorPosCallback(window, MouseCallback);
void MouseCallback(GLFWwindow* window, double xpos, double ypos)
{
    .....
}
```

```
// 绘制主循环
while (!glfwWindowShouldClose(window)) {
    ProcessInput(window);
}
```


4

程序演示

【运行】 ➡ 【移动视角】 ➡ 【关闭立方体旋转】



```
1 //总体流程
2 //1. 初始化并创建窗口
3 //2. 加载立方体顶点以及着色器并开启深度测试, 声明响应函数
4 //3. 进入主循环清除缓冲
5 //4. 使用立方体着色器, 构造并传入pvm矩阵, 绘制
6 //6. 循环结束, 释放VAO/VBO
7
8
9
10 #include <iostream>
11 #include "glad/glad.h"
12 #include "GLFW/glfw3.h"
13 #include "glm/glm.hpp"
14 #include "glm/gtc/matrix_transform.hpp"
15 #include "glm/gtc/type_ptr.hpp"
16
17 #include "shader.h"
18
19 #include <vector>
20
21 void framebufferSizeCallback(GLFWwindow* window, int width, int height) //窗口大小更改响应
22 void processInput(GLFWwindow *window) //键盘响应
23 void mouseCallback(GLFWwindow*window, double xpos, double ypos) //鼠标响应
24 void scrollCallback(GLFWwindow*window, double xoffset, double yoffset) //鼠标滚轮响应
25
26
27 float screen_width = 1280.0f; //窗口宽度
28 float screen_height = 720.0f; //窗口高度
29 glm::vec3 camera_position = glm::vec3(0.0f, 0.0f, 3.0f); //摄像机位置
30 glm::vec3 camera_front = glm::vec3(0.0f, 0.0f, -1.0f); //摄像机方向
31 glm::vec3 camera_up = glm::vec3(0.0f, 1.0f, 0.0f); //摄像机上向量
32
33 float delta_time = 0.0f;
34 float last_frame = 0.0f;
35 bool first_mouse = true;
36 //相机参数
37 float yaw = -90.0f;
```

输出

显示输出来源(S): 生成

1> C:\计算机图形学\task-cube\src\main.cpp(242): warning C4305: "初始化": 从 "double" 到 "float" 截断

1> C:\计算机图形学\task-cube\src\main.cpp(228): warning C4100: "window": 未引用的形参

1> C:\计算机图形学\task-cube\src\main.cpp(269): warning C4244: "x": 从 "double" 转换到 "float", 可能丢失数据

1> C:\计算机图形学\task-cube\src\main.cpp(265): warning C4100: "xoffset": 未引用的形参

1> C:\计算机图形学\task-cube\src\main.cpp(265): warning C4100: "yoffset": 未引用的形参

1> task_vcpkg --> C:\计算机图形学\task-cube\build\task_Debug\task.exe

1> 已完成生成项目 "Task -> vcpkg" 的操作。

生成: 成功 1 个, 失败 0 个, 最新 1 个, 跳过 0 个



谢谢

软件学院 万琳