**OpenGL之鼠标拾取和模型控制**

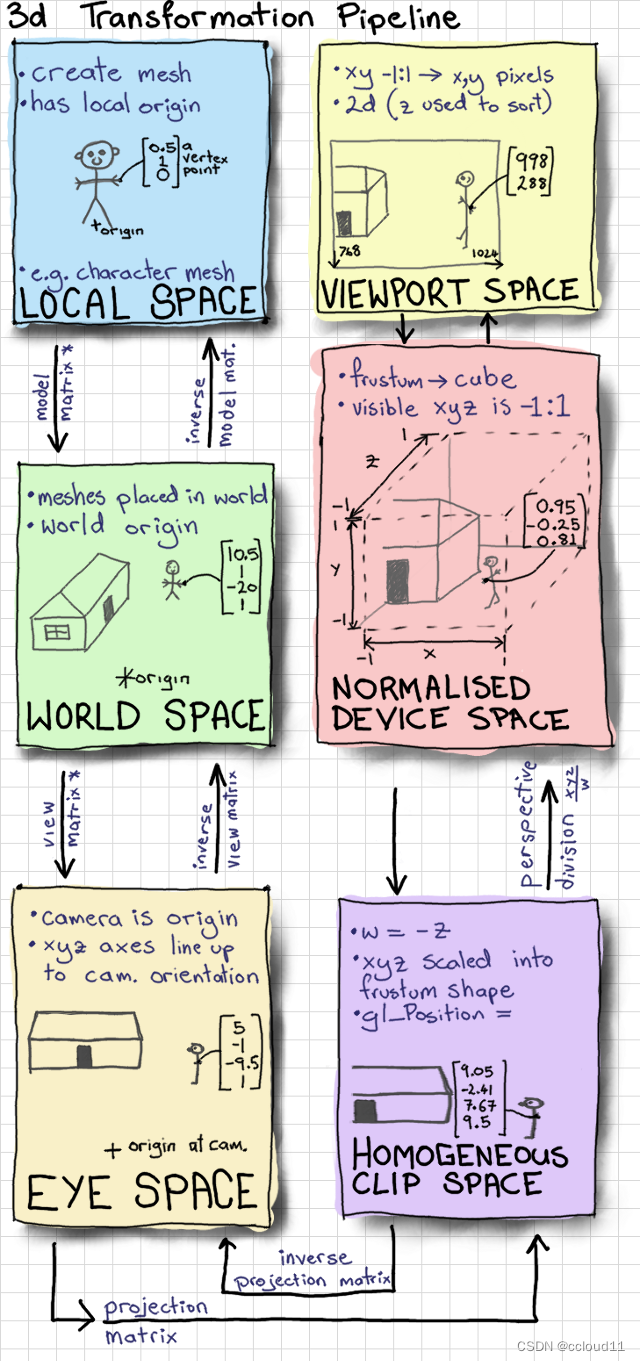
文章详细介绍了如何在3D场景中通过鼠标点击来拾取对象的过程，包括从2D视口坐标到3D规范化设备坐标，再到4D剪辑、眼和世界坐标的转换。此外，还展示了源代码示例，用于处理鼠标点击事件并计算点击位置的世界坐标。

**文章目录**

* [鼠标拾取](https://blog.csdn.net/qq_45526401/article/details/131273614#_1)
  + [转化步骤](https://blog.csdn.net/qq_45526401/article/details/131273614#_2)
    - [步骤 0：2D 视口坐标](https://blog.csdn.net/qq_45526401/article/details/131273614#_02D__5)
    - [步骤 1：3D 规范化设备坐标](https://blog.csdn.net/qq_45526401/article/details/131273614#_13D__12)
    - [步骤2：4d 均匀剪辑坐标](https://blog.csdn.net/qq_45526401/article/details/131273614#24d__22)
    - [步骤3：4D 眼（相机）坐标](https://blog.csdn.net/qq_45526401/article/details/131273614#34D__31)
    - [步骤4：4d 世界坐标](https://blog.csdn.net/qq_45526401/article/details/131273614#44d__42)
  + [源码](https://blog.csdn.net/qq_45526401/article/details/131273614#_52)
* [模型控制](https://blog.csdn.net/qq_45526401/article/details/131273614#_463)
  + [源码](https://blog.csdn.net/qq_45526401/article/details/131273614#_466)

**鼠标拾取**

**转化步骤**

 使用鼠标单击或“选择”场景中的 3D 对象可能很有用 光标。一种方法是从鼠标投射 3D 射线， 通过摄像机，进入场景，然后检查该光线是否与任何对象。这通常称为光线投射。  


**步骤 0：2D 视口坐标**

 范围 [0：宽度、高度：0]  
 我们从鼠标光标坐标开始。这些是 2d，并且在视口坐标系中。首先我们需要获取鼠标 x，y 像素 坐标。您可能已经设置了一个回调函数（例如 [GLFW](https://so.csdn.net/so/search?q=GLFW&spm=1001.2101.3001.7020) 或 GLUT） 像这样：

void mouse\_click\_callback(int b, int s, int mouse\_x, int mouse\_y);

* 1

 这给了我们一个 x 在 0：width 和 y 从高度：0 开始。请记住，0 位于此处的屏幕顶部，因此 y 轴方向与其他坐标系中的方向相反。

**步骤 1：3D 规范化设备坐标**

范围 [-1：1， -1：1， -1：1]  
下一步是将其转换为 3D 规范化设备坐标。 这应该在 x [-1：1] y [-1：1] 和 z [-1：1] 的范围内。我们有一个 x 和 y已经，所以我们缩放它们的范围，并反转y的方向。

float x = (2.0f \* mouse\_x) / width - 1.0f;

float y = 1.0f - (2.0f \* mouse\_y) / height;

float z = 1.0f;

vec3 ray\_nds = vec3(x, y, z);

我们实际上还不需要指定一个 z，但我放了一个（对于 craic）。

**步骤2：4d 均匀剪辑坐标**

范围 [-1：1， -1：1， -1：1， -1：1]  
 我们希望我们的射线的z指向前方 - 这通常是 OpenGL 样式中的负 z 方向。我们可以添加一个 ，这样我们就有一个 4d 向量。w

vec4 ray\_clip = vec4(ray\_nds.xy, -1.0, 1.0);

* 1

注意：我们不需要在这里反转透视划分，因为 这是一条没有内在深度的射线。有关光线投射的其他教程 会错误地告诉您这样做。不要理会假先知！我们会做的 仅在点的特殊情况下，对于某些特殊效果。

**步骤3：4D 眼（相机）坐标**

范围 [-x：x， -y：y， -z：z， -w：w]

 通常，为了从眼睛空间进入剪辑空间，我们将向量乘以 投影矩阵。我们可以通过乘以这个的倒数来倒退矩阵。

vec4 ray\_eye = inverse(projection\_matrix) \* ray\_clip;

* 1

 现在，我们只需要取消投影零件，所以让我们手动设置 表示“前进，而不是点”的部分。x,yz,w

ray\_eye = vec4(ray\_eye.xy, -1.0, 0.0);

* 1

**步骤4：4d 世界坐标**

范围 [-x：x， -y：y， -z：z， -w：w]

同样，回到转换管道中的另一个步骤。记得 我们手动为 z 分量指定了 -1，这意味着我们的光线 未规范化。我们应该在使用它之前这样做。

vec3 ray\_wor = (inverse(view\_matrix) \* ray\_eye).xyz;

// don't forget to normalise the vector at some point

ray\_wor = normalise(ray\_wor);

* 1
* 2
* 3

这应该平衡上下、左右和前进的组件 安全信息因此，假设我们的相机直接沿着-Z世界轴看，我们 当鼠标位于 屏幕，以及鼠标在屏幕上移动时不太重要的 z 值。 这将取决于纵横比，以及视图中定义的视野和投影矩阵。

**源码**

main.cpp

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <iostream>

#include <cmath>

#include "../shader.h"

#include "../camera.h"

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include "../model.h"

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height);

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset);

void processInput(GLFWwindow\* window);

unsigned int loadTexture(const char\* path);

void mouseClick\_callback(GLFWwindow\* window, int button, int action, int mods);

// settings

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

float \_near = 0.1, \_far = 100;

// camera

Camera camera(glm::vec3(0.0f, 0.0f, 3.0f));

float lastX = (float)SCR\_WIDTH / 2.0;

float lastY = (float)SCR\_HEIGHT / 2.0;

bool firstMouse = true;

// timing

float deltaTime = 0.0f;

float lastFrame = 0.0f;

glm::mat4 view;

glm::mat4 projection;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

#endif

// glfw window creation

// --------------------

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT, "LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

glfwSetCursorPosCallback(window, mouse\_callback);

glfwSetScrollCallback(window, scroll\_callback);

glfwSetMouseButtonCallback(window, mouseClick\_callback);

// tell GLFW to capture our mouse

//glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LESS); // always pass the depth test (same effect as glDisable(GL\_DEPTH\_TEST))

// build and compile shaders

// -------------------------

Shader shader("shaders/shader.vs", "shaders/shader.fs");

// set up vertex data (and buffer(s)) and configure vertex attributes

// ------------------------------------------------------------------

float cubeVertices[] = {

// positions // texture Coords

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 1.0f, 1.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f

};

float planeVertices[] = {

// positions // texture Coords (note we set these higher than 1 (together with GL\_REPEAT as texture wrapping mode). this will cause the floor texture to repeat)

5.0f, -0.5f, 5.0f, 2.0f, 0.0f,

-5.0f, -0.5f, 5.0f, 0.0f, 0.0f,

-5.0f, -0.5f, -5.0f, 0.0f, 2.0f,

5.0f, -0.5f, 5.0f, 2.0f, 0.0f,

-5.0f, -0.5f, -5.0f, 0.0f, 2.0f,

5.0f, -0.5f, -5.0f, 2.0f, 2.0f

};

// cube VAO

unsigned int cubeVAO, cubeVBO;

glGenVertexArrays(1, &cubeVAO);

glGenBuffers(1, &cubeVBO);

glBindVertexArray(cubeVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, cubeVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(cubeVertices), &cubeVertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glBindVertexArray(0);

// plane VAO

unsigned int planeVAO, planeVBO;

glGenVertexArrays(1, &planeVAO);

glGenBuffers(1, &planeVBO);

glBindVertexArray(planeVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, planeVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(planeVertices), &planeVertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glBindVertexArray(0);

// load textures

// -------------

unsigned int cubeTexture = loadTexture("../resources/container2.png");

unsigned int floorTexture = loadTexture("../resources/rock.jpg");

// shader configuration

// --------------------

shader.use();

shader.setInt("texture1", 0);

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = static\_cast<float>(glfwGetTime());

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

// input

// -----

processInput(window);

// render

// ------

glClearColor(0.1f, 0.1f, 0.1f, 1.0f);

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

shader.use();

glm::mat4 model = glm::mat4(1.0f);

view = camera.GetViewMatrix();

projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

shader.setMat4("view", view);

shader.setMat4("projection", projection);

// cubes

glBindVertexArray(cubeVAO);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, cubeTexture);

shader.setMat4("model", model);

glDrawArrays(GL\_TRIANGLES, 0, 36);

// floor

glBindVertexArray(planeVAO);

glBindTexture(GL\_TEXTURE\_2D, floorTexture);

shader.setMat4("model", glm::mat4(1.0f));

glDrawArrays(GL\_TRIANGLES, 0, 6);

glBindVertexArray(0);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// optional: de-allocate all resources once they've outlived their purpose:

// ------------------------------------------------------------------------

glDeleteVertexArrays(1, &cubeVAO);

glDeleteVertexArrays(1, &planeVAO);

glDeleteBuffers(1, &cubeVBO);

glDeleteBuffers(1, &planeVBO);

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

camera.ProcessKeyboard(FORWARD, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

camera.ProcessKeyboard(BACKWARD, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

camera.ProcessKeyboard(LEFT, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

camera.ProcessKeyboard(RIGHT, deltaTime);

}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes

// ---------------------------------------------------------------------------------------------

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xposIn, double yposIn)

{

float xpos = static\_cast<float>(xposIn);

float ypos = static\_cast<float>(yposIn);

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

lastX = xpos;

lastY = ypos;

camera.ProcessMouseMovement(xoffset, yoffset);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

camera.ProcessMouseScroll(static\_cast<float>(yoffset));

}

// utility function for loading a 2D texture from file

// ---------------------------------------------------

unsigned int loadTexture(char const\* path)

{

unsigned int textureID;

glGenTextures(1, &textureID);

int width, height, nrComponents;

unsigned char\* data = stbi\_load(path, &width, &height, &nrComponents, 0);

if (data)

{

GLenum format;

if (nrComponents == 1)

format = GL\_RED;

else if (nrComponents == 3)

format = GL\_RGB;

else if (nrComponents == 4)

format = GL\_RGBA;

glBindTexture(GL\_TEXTURE\_2D, textureID);

glTexImage2D(GL\_TEXTURE\_2D, 0, format, width, height, 0, format, GL\_UNSIGNED\_BYTE, data);

glGenerateMipmap(GL\_TEXTURE\_2D);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

stbi\_image\_free(data);

}

else

{

std::cout << "Texture failed to load at path: " << path << std::endl;

stbi\_image\_free(data);

}

return textureID;

}

void mouseClick\_callback(GLFWwindow\* window, int button, int action, int mods) {

if ((action == GLFW\_PRESS) && (button == GLFW\_MOUSE\_BUTTON\_RIGHT)) {

double winX, winY;

float winZ;

glfwGetCursorPos(window, &winX, &winY);

glReadPixels(

(int)winX,

(int)SCR\_HEIGHT - (int)winY

, 1, 1

, GL\_DEPTH\_COMPONENT, GL\_FLOAT

, &winZ);

float x = (2.0f \* winX) / SCR\_WIDTH - 1.0f;

float y = 1.0f - (2.0f \* winY) / SCR\_HEIGHT;

float z = winZ \* 2.0 - 1.0f;

//有像素被点中

if (winZ < 1.0f) {

//float w = (2.0 \* \_near \* \_far) / (\_far + \_near - z \* (\_far - \_near));

float w = \_near \* \_far / (\_near \* winZ - \_far \* winZ + \_far);

glm::vec4 wolrdPostion(x, y, z, 1);

wolrdPostion \*= w;

wolrdPostion = glm::inverse(view) \* glm::inverse(projection) \* wolrdPostion;

cout << " x:" << wolrdPostion.r

<< " y:" << wolrdPostion.y

<< " z:" << wolrdPostion.z

<< " w:" << w << endl;

}

else {

cout << "没有像素被选中" << endl;;

}

}

}

shader.vs

#version 330 core

layout (location = 0) in vec3 aPos;

layout (location = 1) in vec2 aTexCoords;

out vec2 TexCoords;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

TexCoords = aTexCoords;

gl\_Position = projection \* view \* model \* vec4(aPos, 1.0);

}

shader.fs

#version 330 core

out vec4 FragColor;

float near = 0.1;

float far = 100.0;

float LinearizeDepth(float depth) {

float z\_ndc = depth \* 2.0 - 1.0; // back to NDC

return (2.0 \* near \* far) / (far + near - z\_ndc \* (far - near));

}

in vec2 TexCoords;

uniform sampler2D texture1;

void main() {

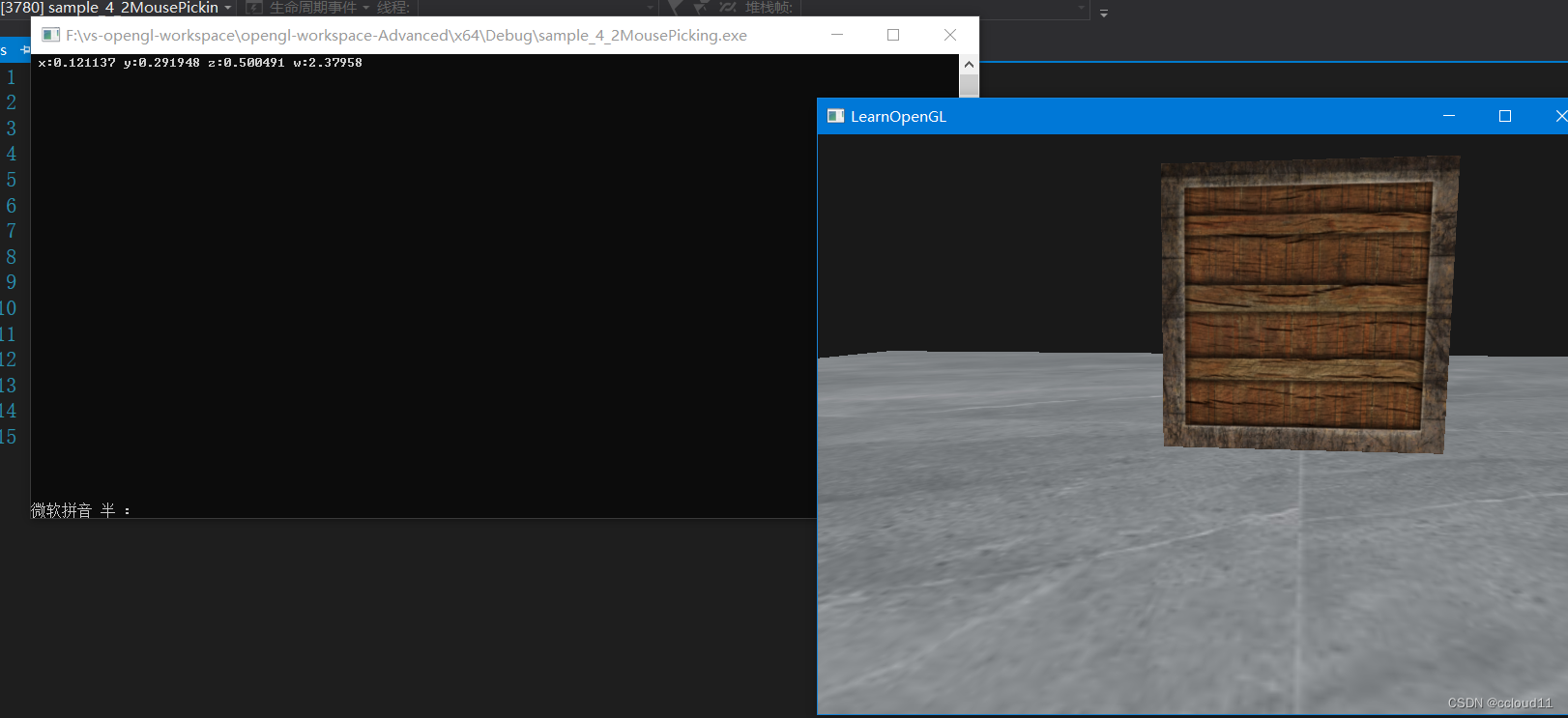
//float depth = LinearizeDepth(gl\_FragCoord.z) / far; // divide by far for demonstration

//FragColor = vec4(vec3(depth), 1.0);

FragColor = texture(texture1,TexCoords);

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

运行效果  
  
点击鼠标右键打印出该屏幕坐标的世界坐标。

模型**控制**

 主要要实现的效果是右击鼠标时可以进行拖动模型。  
实现思路：点击屏幕坐标后将屏幕坐标转化为世界坐标与模型的世界坐标进行判断，当小于2.5的时候就认为该模型被拾取到了，然后将该模型的位置进行移动。

**源码**

main.cpp

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <iostream>

#include <cmath>

#include "../shader.h"

#include "../camera.h"

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

#include "../model.h"

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height);

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos);

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset);

void processInput(GLFWwindow\* window);

unsigned int loadTexture(const char\* path);

void mouseClick\_callback(GLFWwindow\* window, int button, int action, int mods);

glm::vec3 worldPosFromViewPort(int winX, int winY);

// settings

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

float \_near = 0.1, \_far = 100;

// camera

Camera camera(glm::vec3(0.0f, 0.0f, 8.0f));

float lastX = (float)SCR\_WIDTH / 2.0;

float lastY = (float)SCR\_HEIGHT / 2.0;

bool firstMouse = true;

// timing

float deltaTime = 0.0f;

float lastFrame = 0.0f;

glm::mat4 view;

glm::mat4 projection;

bool LeftButtonHolding = false;

bool RightButtonHolding = false;

struct ModelInfo {

Model\* model;

glm::vec3 worldPos;

float pitch;

float yaw;

float roll;

bool isSelected;

string name;

};

map<string, ModelInfo> \_Models;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

#endif

// glfw window creation

// --------------------

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT, "LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

glfwSetCursorPosCallback(window, mouse\_callback);

glfwSetScrollCallback(window, scroll\_callback);

glfwSetMouseButtonCallback(window, mouseClick\_callback);

// tell GLFW to capture our mouse

//glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LESS); // always pass the depth test (same effect as glDisable(GL\_DEPTH\_TEST))

// build and compile shaders

// -------------------------

Shader shader("shaders/shader.vs", "shaders/shader.fs");

// set up vertex data (and buffer(s)) and configure vertex attributes

// ------------------------------------------------------------------

float cubeVertices[] = {

// positions // texture Coords

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 1.0f, 1.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

0.5f, -0.5f, -0.5f, 1.0f, 1.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f,

0.5f, 0.5f, -0.5f, 1.0f, 1.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f

};

float planeVertices[] = {

// positions // texture Coords (note we set these higher than 1 (together with GL\_REPEAT as texture wrapping mode). this will cause the floor texture to repeat)

5.0f, -0.5f, 5.0f, 2.0f, 0.0f,

-5.0f, -0.5f, 5.0f, 0.0f, 0.0f,

-5.0f, -0.5f, -5.0f, 0.0f, 2.0f,

5.0f, -0.5f, 5.0f, 2.0f, 0.0f,

-5.0f, -0.5f, -5.0f, 0.0f, 2.0f,

5.0f, -0.5f, -5.0f, 2.0f, 2.0f

};

// cube VAO

unsigned int cubeVAO, cubeVBO;

glGenVertexArrays(1, &cubeVAO);

glGenBuffers(1, &cubeVBO);

glBindVertexArray(cubeVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, cubeVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(cubeVertices), &cubeVertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glBindVertexArray(0);

// plane VAO

unsigned int planeVAO, planeVBO;

glGenVertexArrays(1, &planeVAO);

glGenBuffers(1, &planeVBO);

glBindVertexArray(planeVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, planeVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(planeVertices), &planeVertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glBindVertexArray(0);

// load textures

// -------------

unsigned int cubeTexture = loadTexture("../resources/container2.png");

unsigned int floorTexture = loadTexture("../resources/rock.jpg");

// shader configuration

// --------------------

shader.use();

shader.setInt("texture1", 0);

Model Zhang3 = Model("../resources/backpack/backpack.obj");

\_Models["zhang3"] = ModelInfo{ &Zhang3,glm::vec3(-3,0,0),0.0,0.0,0.0,false,"Zhang3" };

Model Li4 = Model("../resources/backpack/backpack.obj");

\_Models["Li4"] = ModelInfo{ &Li4,glm::vec3(3,0,0),0.0,0.0,0.0,false,"Li4" };

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = static\_cast<float>(glfwGetTime());

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

// input

// -----

processInput(window);

// render

// ------

glClearColor(0.1f, 0.1f, 0.1f, 1.0f);

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

shader.use();

glm::mat4 model = glm::mat4(1.0f);

view = camera.GetViewMatrix();

projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

shader.setMat4("view", view);

shader.setMat4("projection", projection);

//model

map<string, ModelInfo>::iterator \_modelsIter;

for (\_modelsIter = \_Models.begin(); \_modelsIter != \_Models.end(); \_modelsIter++)

{

model = glm::mat4(1.0f);

model = glm::translate(model, \_modelsIter->second.worldPos);

model = glm::rotate(model, \_modelsIter->second.pitch, glm::vec3(1.0f, 0.0f, 0.0f));

model = glm::rotate(model, \_modelsIter->second.yaw, glm::vec3(0.0f, 1.0f, 0.0f));

model = glm::rotate(model, \_modelsIter->second.roll, glm::vec3(0.0f, 0.0f, 1.0f));

shader.setMat4("model", model);

\_modelsIter->second.model->Draw(shader);

}

// floor

glBindVertexArray(planeVAO);

glBindTexture(GL\_TEXTURE\_2D, floorTexture);

shader.setMat4("model", glm::mat4(1.0f));

glDrawArrays(GL\_TRIANGLES, 0, 6);

glBindVertexArray(0);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// optional: de-allocate all resources once they've outlived their purpose:

// ------------------------------------------------------------------------

glDeleteVertexArrays(1, &cubeVAO);

glDeleteVertexArrays(1, &planeVAO);

glDeleteBuffers(1, &cubeVBO);

glDeleteBuffers(1, &planeVBO);

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frame and react accordingly

// ---------------------------------------------------------------------------------------------------------

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

camera.ProcessKeyboard(FORWARD, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

camera.ProcessKeyboard(BACKWARD, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

camera.ProcessKeyboard(LEFT, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

camera.ProcessKeyboard(RIGHT, deltaTime);

if (glfwGetKey(window, GLFW\_KEY\_LEFT\_CONTROL) == GLFW\_PRESS)

{

for (map<string, ModelInfo>::iterator \_modelsIter = \_Models.begin(); \_modelsIter != \_Models.end(); \_modelsIter++)

{

if (\_modelsIter->second.isSelected) {

\_modelsIter->second.yaw += 0.1 \* deltaTime;

break;

}

}

}

}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes

// ---------------------------------------------------------------------------------------------

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

camera.ProcessMouseScroll(static\_cast<float>(yoffset));

}

// utility function for loading a 2D texture from file

// ---------------------------------------------------

unsigned int loadTexture(char const\* path)

{

unsigned int textureID;

glGenTextures(1, &textureID);

int width, height, nrComponents;

unsigned char\* data = stbi\_load(path, &width, &height, &nrComponents, 0);

if (data)

{

GLenum format;

if (nrComponents == 1)

format = GL\_RED;

else if (nrComponents == 3)

format = GL\_RGB;

else if (nrComponents == 4)

format = GL\_RGBA;

glBindTexture(GL\_TEXTURE\_2D, textureID);

glTexImage2D(GL\_TEXTURE\_2D, 0, format, width, height, 0, format, GL\_UNSIGNED\_BYTE, data);

glGenerateMipmap(GL\_TEXTURE\_2D);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

stbi\_image\_free(data);

}

else

{

std::cout << "Texture failed to load at path: " << path << std::endl;

stbi\_image\_free(data);

}

return textureID;

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

void mouse\_callback(GLFWwindow\* window, double xposIn, double yposIn)

{

if (LeftButtonHolding) {

float xpos = static\_cast<float>(xposIn);

float ypos = static\_cast<float>(yposIn);

float xoffset = xpos - lastX;

float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to top

lastX = xpos;

lastY = ypos;

camera.ProcessMouseMovement(xoffset, yoffset);

}

if (RightButtonHolding) {

for (map<string, ModelInfo>::iterator \_modelsIter = \_Models.begin(); \_modelsIter != \_Models.end(); \_modelsIter++)

{

if (\_modelsIter->second.isSelected) {

\_modelsIter->second.worldPos = worldPosFromViewPort((int)xposIn, (int)yposIn);

break;

}

}

}

}

void mouseClick\_callback(GLFWwindow\* window, int button, int action, int mods) {

double winX, winY;

glfwGetCursorPos(window, &winX, &winY);

if ((action == GLFW\_PRESS) && (button == GLFW\_MOUSE\_BUTTON\_LEFT))

{

lastX = (float)winX;

lastY = (float)winY;

LeftButtonHolding = true; //只有按下鼠标左键时，才会改变摄像机角度

}

else

LeftButtonHolding = false;

if ((action == GLFW\_PRESS) && (button == GLFW\_MOUSE\_BUTTON\_RIGHT)) {

RightButtonHolding = true;

glm::vec3 wolrdPostion = worldPosFromViewPort((int)winX, (int)winY);

for (map<string, ModelInfo>::iterator \_modelsIter = \_Models.begin(); \_modelsIter != \_Models.end(); \_modelsIter++)

{

float \_distance = glm::distance(\_modelsIter->second.worldPos, glm::vec3(wolrdPostion));

if (\_distance < 2.5) {

cout << \_modelsIter->first << "模型被选中..." << endl;

\_modelsIter->second.isSelected = true;

break;

}

else

\_modelsIter->second.isSelected = false;

}

}

else

RightButtonHolding = false;

}

glm::vec3 worldPosFromViewPort(int winX, int winY) {

float winZ;

glReadPixels(

winX,

(int)SCR\_HEIGHT - winY

, 1, 1

, GL\_DEPTH\_COMPONENT, GL\_FLOAT

, &winZ);

float x = (2.0f \* (float)winX) / SCR\_WIDTH - 1.0f;

float y = 1.0f - (2.0f \* (float)winY) / SCR\_HEIGHT;

float z = winZ \* 2.0f - 1.0f;

//有像素被点中

//float w = (2.0 \* \_near \* \_far) / (\_far + \_near - z \* (\_far - \_near));

float w = \_near \* \_far / (\_near \* winZ - \_far \* winZ + \_far);

glm::vec4 wolrdPostion(x, y, z, 1);

wolrdPostion \*= w;

wolrdPostion = glm::inverse(view) \* glm::inverse(projection) \* wolrdPostion;

return wolrdPostion;

}

shader.vs

#version 330 core

layout (location = 0) in vec3 aPos;

layout (location = 1) in vec2 aTexCoords;

out vec2 TexCoords;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main()

{

TexCoords = aTexCoords;

gl\_Position = projection \* view \* model \* vec4(aPos, 1.0);

}

shader.fs

#version 330 core

out vec4 FragColor;

float near = 0.1;

float far = 100.0;

float LinearizeDepth(float depth) {

float z\_ndc = depth \* 2.0 - 1.0; // back to NDC

return (2.0 \* near \* far) / (far + near - z\_ndc \* (far - near));

}

in vec2 TexCoords;

uniform sampler2D texture1;

void main() {

//float depth = LinearizeDepth(gl\_FragCoord.z) / far; // divide by far for demonstration

//FragColor = vec4(vec3(depth), 1.0);

FragColor = texture(texture1,TexCoords);

}

[[](https://blog.csdn.net/qq_45526401)](https://blog.csdn.net/qq_45526401)