**旋转的地球**

项目说明文档

郭建章 20192131038

1. 项目名称

旋转的地球

1. 项目概况

用软件展现一个不断旋转的地球（使用c++ opengl图形接口实现）

1. 项目功能介绍
2. 用OpenGL定义一个三维球体
3. 将世界地图作为该球体的表面“纹理”贴上
4. 不断将该球体进行旋转和展现，即形成旋转的地球的效果。
5. 点击鼠标可以暂停旋转，松开鼠标继续旋转
6. 代码实现

先根据数学公式建立球的顶点矩阵，再建立索引矩阵，选取顶点组成三角形，然后绑定贴图并采样，通过着色器将颜色渲染在球模型上，通过计时器和旋转矩阵使模型旋转

#include<GL/glew.h>

#include <GLFW/glfw3.h>

#include<iostream>

#include<fstream>

#include<string>

#include<sstream>

#include<math.h>

#include<vector>

#include"glm/glm.hpp"

#include"glm/gtc/matrix\_transform.hpp"

#include"GL/glut.h"

#include "Renderer.h"

#include "VertexBuffer.h"

#include "IndexBuffer.h"

#include "VertexArray.h"

#include "VertexBufferLayout.h"

#include "Shader.h"

#include "Texture.h"

#include "MouseEvent.h"

#define M\_PI 3.14159265358979323846

int main(void)

{

GLFWwindow\* window;

/\* Initialize the library \*/

if (!glfwInit())

return -1;

width = 960.0f; height = 900.0f;

/\* Create a windowed mode window and its OpenGL context \*/

window = glfwCreateWindow(width, height, "Hello World", NULL, NULL);

if (!window)

{

glfwTerminate();

return -1;

}

glfwSetKeyCallback(window, key\_callback);

glfwSetMouseButtonCallback(window, mouse\_button\_callback);

glfwSetCursorPosCallback(window, cursor\_position\_callback);

glfwSetScrollCallback(window, scroll\_callback);

/\* Make the window's context current \*/

glfwMakeContextCurrent(window);

glfwSwapInterval(1);

if (glewInit() != GLEW\_OK)

{

std::cout << "Error!" << std::endl;

}

std::cout << glGetString(GL\_VERSION) << std::endl;

{

float radius = 500.0f;

std::vector<float> vertexs;

unsigned int angleSpan = 1;

unsigned int row = (180 / angleSpan) + 1;// 球面切分的行数

unsigned int col = 360 / angleSpan;// 球面切分的列数

for (int rowAngle = -90; rowAngle <= 90; rowAngle+= angleSpan)

{

double rowA = rowAngle \* 2 \* M\_PI / 360;

for (int colAngle = 0; colAngle < 360; colAngle += angleSpan)

{

double colA = colAngle \* 2 \* M\_PI / 360;

double xozlength = radius \* cos(rowA);

float x = (float)xozlength \* cos(colA);

float z = (float)xozlength \* sin(colA);

float y = (float)radius \* sin(rowA);

vertexs.push\_back(x);

vertexs.push\_back(y);

vertexs.push\_back(z);

int k = (rowAngle + 90) / angleSpan + 1;

//if (k != 20 && k != 21)

{

int i = (rowAngle + 90) / angleSpan;

int j = colAngle / angleSpan;

vertexs.push\_back( float(j) / float(col));

vertexs.push\_back( float(i) / float(row));

//std::cout << float(j) / float(col) << " " << float(i) / float(row) << std::endl;

}

}

}

std::vector<unsigned int> indices;

for (unsigned int i = 0; i < row; i++)// 对每一行循环

{

if (i > 0 && i < row - 1) { // 中间行

for (unsigned int j = 0; j < col; j++) {// 中间行的两个相邻点与下一行的对应点构成三角形

unsigned int k = i \* col + j;

// 第1个三角形顶点下标

indices.push\_back(k + col);

// 第2个三角形顶点下标

unsigned int tmp = k + 1;

if (j == col - 1) {

tmp = (i)\*col;

}

indices.push\_back(tmp);

// 第3个三角形顶点下标

indices.push\_back(k);

indices.push\_back(k + col);

indices.push\_back(tmp + col);

indices.push\_back(tmp);

}

}

}

GLCall(glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA));

VertexArray va;

VertexBuffer vb(vertexs.data(), vertexs.size() \* sizeof(float));

VertexBufferLayout layout;

layout.Push<float>(3);

layout.Push<float>(2);

va.AddBuffer(vb, layout);

IndexBuffer ib(indices.data(), indices.size());

glm::mat4 proj = glm::ortho(-500.0f, 500.0f, -500.0f, 500.0f, -500.0f, 500.0f);

glm::mat4 view = glm::translate(glm::mat4(1.0f), glm::vec3(0, 0, 500));

glm::mat4 model = glm::rotate(glm::mat4(1.0f), 0.0f, glm::vec3(0.0f, 1.0f, 0.0f));

glm::mat4 mvp = proj \* view \* model;

Shader shader("res/shaders/Basic.shader");

shader.Bind();

shader.SetUniformMat4f("u\_MVP", mvp);

Texture texture("res/earth.jpg");

texture.Bind();

shader.SetUniform1i("u\_Texture", 0);

va.Unbind();

vb.Unbind();

ib.Unbind();

shader.Unbind();

float lastTime = 0.0f;

float delta = 0.0f;

float deltax = 0.0f;

float deltay = 0.0f;

bool isLeftFirst = true;

/\* Loop until the user closes the window \*/

while (!glfwWindowShouldClose(window))

{

/\* Render here \*/

glClear(GL\_COLOR\_BUFFER\_BIT);

texture.Bind();

shader.Bind();

//std::cout << (GLfloat)glfwGetTime() << std::endl;

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

if (mouseRightDown)

{

model = glm::rotate(model, lastTime\*72 , glm::vec3(0.0f, 1.0f, 0.0f));

isFirst = true;

}

else

{

if (isFirst)

{

delta = (GLfloat)glfwGetTime() - lastTime;

isFirst = false;

}

model = glm::rotate(model, ((GLfloat)glfwGetTime() - delta) \* 72 , glm::vec3(0.0f, 1.0f, 0.0f));

lastTime = (GLfloat)glfwGetTime()-delta ;

}

glm::mat4 mvp = proj \* view \* model;

shader.SetUniformMat4f("u\_MVP", mvp);

/\*

float t = 0.0f;

if (mouseLeftDown)

{

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

t = mouseX;

model = glm::rotate(model, mouseX+t, glm::vec3(0.0f, -1.0f, 0.0f));

std::cout << "鼠标坐标：" << mouseX << " " << mouseY << std::endl;

glm::mat4 mvp = proj \* view \* model;

shader.SetUniformMat4f("u\_MVP", mvp);

//model = glm::rotate(model, cameraAngleX / 4 , glm::vec3(1.0f, 0.0f, 0.0f));

//mvp = proj \* view \* model;

//shader.SetUniformMat4f("u\_MVP", mvp);

}

\*/

va.Bind();

ib.Bind();

GLCall(glDrawElements(GL\_TRIANGLES, indices.size(), GL\_UNSIGNED\_INT, nullptr));

/\* Swap front and back buffers \*/

glfwSwapBuffers(window);

/\* Poll for and process events \*/

glfwPollEvents();

}

}

glfwTerminate();

return 0;

}

1. 软件测试

屏幕截图

