**FBO render to texture渲染到纹理**

订阅专栏

## 代码参考1：https://[github](https://so.csdn.net/so/search?q=github&spm=1001.2101.3001.7020).com/wld0595/FBO

*// Include standard headers*

#include <stdio.h>

*// Include GLEW*

#include <GL/glew.h>

*// Include GLFW*

#include <GLFW/glfw3.h>

GLFWwindow\* window;

#include <common/shader.hpp>

#define STB\_IMAGE\_IMPLEMENTATION

#include <stb\_image.h>

typedef struct

{

GLfloat x;

GLfloat y;

GLfloat z;

GLfloat u;

GLfloat v;

}Vertex;

int main( void )

{

if( !glfwInit() )

{

fprintf( stderr, "Failed to initialize GLFW.\n" );

getchar();

return -1;

}

glfwWindowHint(GLFW\_SAMPLES, 4);

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

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

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

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

window = glfwCreateWindow( 1024, 768, "Render To Texture", NULL, NULL);

if( window == NULL )

{

fprintf( stderr, "Failed to open GLFW window.\n" );

getchar();

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

int windowWidth,windowHeight;

glfwGetWindowSize(window,&windowWidth,&windowHeight);

glfwGetFramebufferSize(window, &windowWidth, &windowHeight);

glewExperimental = true;

if (glewInit() != GLEW\_OK)

{

fprintf(stderr, "Failed to initialize GLEW\n");

getchar();

glfwTerminate();

return -1;

}

glfwSetInputMode(window, GLFW\_STICKY\_KEYS, GL\_TRUE);

glEnable(GL\_TEXTURE\_2D);

*//vertex data*

Vertex g\_VertexBufferDataQ[]={

{-1.0,1.0,0.0, 0.0,1.0},

{1.0,1.0,0.0, 1.0,1.0},

{-1.0,-1.0,0.0, 0.0,0.0},

{1.0,-1.0,0.0, 1.0,0.0},

};

Vertex g\_VertexBufferDataT[]={

{-1.0,-1.0,0.0, 0.0,0.0},

{1.0,-1.0,0.0, 0.0,0.0},

{0.0,1.0,0.0, 0.0,0.0},

};

GLuint VertexArrayID;

*//===Generate the vertex array*

glGenVertexArrays(1,&VertexArrayID);

glBindVertexArray(VertexArrayID);

GLuint VertexBufferQ;

glGenBuffers(1,&VertexBufferQ);

glBindBuffer(GL\_ARRAY\_BUFFER,VertexBufferQ);

glBufferData(GL\_ARRAY\_BUFFER,sizeof(g\_VertexBufferDataQ),g\_VertexBufferDataQ,GL\_STATIC\_DRAW);

GLuint VertexBufferT;

glGenBuffers(1,&VertexBufferT);

glBindBuffer(GL\_ARRAY\_BUFFER,VertexBufferT);

glBufferData(GL\_ARRAY\_BUFFER,sizeof(g\_VertexBufferDataT),g\_VertexBufferDataT,GL\_STATIC\_DRAW);

GLuint ProgramIDT = LoadShaders("VertexShaderT.vertexshader","FragmentShaderT.fragmentshader");

GLint VertexPositionIDT=glGetAttribLocation(ProgramIDT,"vertexPosition\_modelspace");

GLuint ProgramIDQ = LoadShaders("VertexShaderQ.vertexshader","FragmentShaderQ.fragmentshader");

GLint VertexPositionIDQ=glGetAttribLocation(ProgramIDQ,"vertexPosition\_modelspace");

GLint UVIDQ=glGetAttribLocation(ProgramIDQ,"vertexUV");

GLint TextureIndexQ =glGetUniformLocation(ProgramIDQ, "myTextureSampler");

Create a texture

GLuint renderedTexture;

*//GLint texWidth=windowWidth,texHeight=windowHeight;*

GLint texWidth=832,texHeight=416;

glGenTextures(1, &renderedTexture);

glBindTexture(GL\_TEXTURE\_2D, renderedTexture);

glTexImage2D(GL\_TEXTURE\_2D, 0,GL\_RGB, texWidth, texHeight, 0,GL\_RGB, GL\_UNSIGNED\_BYTE, 0);

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

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

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

*//Create a FBO*

GLuint FramebufferName = 0;

glGenFramebuffers(1, &FramebufferName);

glBindFramebuffer(GL\_FRAMEBUFFER, FramebufferName);

glFramebufferTexture2D(GL\_FRAMEBUFFER,GL\_COLOR\_ATTACHMENT0,GL\_TEXTURE\_2D, renderedTexture,0);

if(glCheckFramebufferStatus(GL\_FRAMEBUFFER) != GL\_FRAMEBUFFER\_COMPLETE)

return false;

do{

glBindFramebuffer(GL\_FRAMEBUFFER, FramebufferName);

glClearColor(0, 0, 0, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0,0,texWidth,texHeight);

glUseProgram(ProgramIDT);

glEnableVertexAttribArray(VertexPositionIDT);

glBindBuffer(GL\_ARRAY\_BUFFER,VertexBufferT);

glVertexAttribPointer(VertexPositionIDT,3,GL\_FLOAT,GL\_FALSE,20,(void \*)0);

glDrawArrays(GL\_TRIANGLE\_STRIP,0,3);

glDisableVertexAttribArray(VertexPositionIDT);

*//=====äÖÈ¾µ½ÆÁÄ»ÉÏ*

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glClearColor( 1, 1, 1, 0 );

glClear( GL\_COLOR\_BUFFER\_BIT);

glViewport(0,0,windowWidth,windowHeight);

glUseProgram(ProgramIDQ);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, renderedTexture);

glUniform1i(TextureIndexQ,0);

glEnableVertexAttribArray(VertexPositionIDQ);

glBindBuffer(GL\_ARRAY\_BUFFER,VertexBufferQ);

glVertexAttribPointer(VertexPositionIDQ,3,GL\_FLOAT,GL\_FALSE,20,(void \*)0);

glEnableVertexAttribArray(UVIDQ);

*/\*glBindBuffer(GL\_ARRAY\_BUFFER,VertexBufferQ);\*/*

glVertexAttribPointer(UVIDQ,2,GL\_FLOAT,GL\_FALSE,20,(void \*)12);

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

glDisableVertexAttribArray(VertexPositionIDQ);

glDisableVertexAttribArray(UVIDQ);

*// Swap buffers*

glfwSwapBuffers(window);

glfwPollEvents();

}

while( glfwGetKey(window, GLFW\_KEY\_ESCAPE ) != GLFW\_PRESS &&

glfwWindowShouldClose(window) == 0 );

*/\*glDeleteFramebuffers(1, &FramebufferName);*

*glDeleteTextures(1, &renderedTexture);\*/*

glfwTerminate();

return 0;

}

## 代码参考2：https://github.com/mcdooda/gl-fbo-example

#include <stdlib.h>

#include <stdio.h>

#include <GL/glew.h>

#include <SDL2/SDL.h>

*/\* headers \*/*

GLuint compileProgram(GLuint fragmentShaderId, GLuint vertexShaderId);

void checkProgram(GLuint programId);

GLuint loadShader(GLenum shaderType, const GLchar\* source);

void checkShader(GLuint shaderId);

*/\* shaders \*/*

*/\* 1st pass \*/*

const GLchar\* vertexShaderSource1 =

"#version 130\n"

"in vec2 position;"

"void main()"

"{"

"gl\_Position = vec4(position, 0.0, 1.0);"

"}";

const GLchar\* fragmentShaderSource1 =

"#version 130\n"

"out vec4 outColor;"

"uniform vec4 color;"

"void main()"

"{"

"outColor = color;"

"}";

*/\* 2nd pass \*/*

const GLchar\* vertexShaderSource2 =

"#version 130\n"

"in vec2 position;"

"in vec2 texturePosition;"

"out vec2 texturePosition2;"

"void main()"

"{"

"gl\_Position = vec4(position, 0.0, 1.0);"

"texturePosition2 = texturePosition;"

"}";

const GLchar\* fragmentShaderSource2 =

"#version 130\n"

"in vec2 texturePosition2;"

"out vec4 outColor;"

"uniform sampler2D renderTexture;"

"void main()"

"{"

"ivec2 renderTextureSize = textureSize(renderTexture, 0);"

"float dx = 1.0 / renderTextureSize.x;"

"float dy = 1.0 / renderTextureSize.y;"

"float x = texturePosition2.x;"

"float y = texturePosition2.y;"

"vec3 color = vec3(0.0, 0.0, 0.0);"

"int blurRadius = 4;"

"for (int i = -blurRadius; i <= blurRadius; i++)"

"{"

"for (int j = -blurRadius; j <= blurRadius; j++)"

"{"

"color += texture(renderTexture, vec2(x + i \* dx, y + j \* dy)).xyz;"

"}"

"}"

"color /= (blurRadius \* 2 - 1) \* (blurRadius \* 2 - 1);"

"outColor = vec4(color, 1.0);"

"}";

*/\* vertices \*/*

*/\* 1st pass \*/*

const GLfloat redSquareVertices[] = {

-0.5f, -0.5f,

0.25f,-0.5f,

0.25f, 0.25f,

-0.5f, 0.25f

};

const GLfloat blueSquareVertices[] = {

-0.25f,-0.25f,

0.5f, -0.25f,

0.5f, 0.5f,

-0.25f, 0.5f

};

*/\* 2nd pass \*/*

const GLfloat screenVertices[] = {

-0.9f,-0.9f,

0.9f,-0.9f,

0.9f, 0.9f,

-0.9f, 0.9f

};

const GLfloat screenTextureCoordinates[] = {

0.0f, 0.0f,

1.0f, 0.0f,

1.0f, 1.0f,

0.0f, 1.0f

};

*/\* window size \*/*

#define WINDOW\_WIDTH 512

#define WINDOW\_HEIGHT 512

int main(int argc, char\* argv[])

{

SDL\_Init(SDL\_INIT\_VIDEO);

SDL\_Event e;

SDL\_Window\* window;

SDL\_Renderer\* renderer;

SDL\_CreateWindowAndRenderer(WINDOW\_WIDTH, WINDOW\_HEIGHT, SDL\_WINDOW\_OPENGL, &window, &renderer);

glewInit();

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

*/\* load programs \*/*

*/\* 1st pass \*/*

GLuint fragmentShaderId1 = loadShader(GL\_FRAGMENT\_SHADER, fragmentShaderSource1);

GLuint vertexShaderId1 = loadShader(GL\_VERTEX\_SHADER, vertexShaderSource1);

GLuint programId1 = compileProgram(fragmentShaderId1, vertexShaderId1);

*/\* 2nd pass \*/*

GLuint fragmentShaderId2 = loadShader(GL\_FRAGMENT\_SHADER, fragmentShaderSource2);

GLuint vertexShaderId2 = loadShader(GL\_VERTEX\_SHADER, vertexShaderSource2);

GLuint programId2 = compileProgram(fragmentShaderId2, vertexShaderId2);

*/\* retrieve attribs and uniforms \*/*

*/\* 1st pass \*/*

GLint positionAttribLocation1 = glGetAttribLocation(programId1, "position");

if (positionAttribLocation1 == -1)

fprintf(stderr, "warning: positionAttribLocation1 invalid\n");

GLint colorUniformLocation1 = glGetUniformLocation(programId1, "color");

if (colorUniformLocation1 == -1)

fprintf(stderr, "warning: colorUniformLocation1 invalid\n");

*/\* 2nd pass \*/*

GLint positionAttribLocation2 = glGetAttribLocation(programId2, "position");

if (positionAttribLocation2 == -1)

fprintf(stderr, "warning: positionAttribLocation2 invalid\n");

GLint texturePositionAttribLocation2 = glGetAttribLocation(programId2, "texturePosition");

if (texturePositionAttribLocation2 == -1)

fprintf(stderr, "warning: texturePositionAttribLocation2 invalid\n");

GLint renderTextureUniformLocation2 = glGetUniformLocation(programId2, "renderTexture");

if (renderTextureUniformLocation2 == -1)

fprintf(stderr, "warning: renderTextureUniformLocation2 invalid\n");

*/\* create FBO \*/*

GLuint fboId;

glGenFramebuffers(1, &fboId);

GLuint renderTextureId;

glGenTextures(1, &renderTextureId);

glBindTexture(GL\_TEXTURE\_2D, renderTextureId);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, WINDOW\_WIDTH, WINDOW\_HEIGHT, 0, GL\_RGBA, GL\_FLOAT, NULL);

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

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

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE);

glBindFramebuffer(GL\_FRAMEBUFFER, fboId);

glBindTexture(GL\_TEXTURE\_2D, renderTextureId);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, renderTextureId, 0);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

int keepRunning = 1;

while (keepRunning)

{

SDL\_PollEvent(&e);

if (e.type == SDL\_KEYDOWN && e.key.keysym.sym == SDLK\_ESCAPE)

keepRunning = 0;

*/\* 1st pass \*/*

glBindFramebuffer(GL\_FRAMEBUFFER, fboId);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

GLenum bufferlist [] = {

GL\_COLOR\_ATTACHMENT0

};

glDrawBuffers(1, bufferlist);

glClearColor(0.0f, 1.0f, 0.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(programId1);

glEnableVertexAttribArray(positionAttribLocation1);

glUniform4f(colorUniformLocation1, 1.0f, 0.0f, 0.0f, 1.0f);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 0, redSquareVertices);

glDrawArrays(GL\_QUADS, 0, 4);

glUniform4f(colorUniformLocation1, 0.0f, 0.0f, 1.0f, 1.0f);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 0, blueSquareVertices);

glDrawArrays(GL\_QUADS, 0, 4);

glDisableVertexAttribArray(positionAttribLocation1);

*/\* 2nd pass \*/*

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(programId2);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, renderTextureId);

glUniform1i(renderTextureUniformLocation2, 0);

glEnableVertexAttribArray(positionAttribLocation2);

glEnableVertexAttribArray(texturePositionAttribLocation2);

glVertexAttribPointer(positionAttribLocation2, 2, GL\_FLOAT, GL\_FALSE, 0, screenVertices);

glVertexAttribPointer(texturePositionAttribLocation2, 2, GL\_FLOAT, GL\_FALSE, 0, screenTextureCoordinates);

glDrawArrays(GL\_QUADS, 0, 4);

glDisableVertexAttribArray(positionAttribLocation2);

glDisableVertexAttribArray(texturePositionAttribLocation2);

glUseProgram(0);

glFlush();

SDL\_RenderPresent(renderer);

}

SDL\_Quit();

return 0;

}

GLuint compileProgram(GLuint fragmentShaderId, GLuint vertexShaderId)

{

GLuint programId = glCreateProgram();

glAttachShader(programId, vertexShaderId);

glAttachShader(programId, fragmentShaderId);

glLinkProgram(programId);

checkProgram(programId);

return programId;

}

void checkProgram(GLuint programId)

{

GLint result = GL\_FALSE;

glGetProgramiv(programId, GL\_LINK\_STATUS, &result);

if (!result)

{

GLint infoLogLength;

glGetProgramiv(programId, GL\_INFO\_LOG\_LENGTH, &infoLogLength);

GLchar\* message = alloca(infoLogLength \* sizeof(GLchar));

glGetProgramInfoLog(programId, infoLogLength, NULL, message);

fprintf(stderr, "Warning: %s\n", message);

}

}

GLuint loadShader(GLenum shaderType, const GLchar\* source)

{

GLuint shaderId = glCreateShader(shaderType);

glShaderSource(shaderId, 1, &source, NULL);

glCompileShader(shaderId);

checkShader(shaderId);

return shaderId;

}

void checkShader(GLuint shaderId)

{

GLint result = GL\_FALSE;

glGetShaderiv(shaderId, GL\_COMPILE\_STATUS, &result);

if (!result)

{

GLint infoLogLength;

glGetShaderiv(shaderId, GL\_INFO\_LOG\_LENGTH, &infoLogLength);

GLchar\* message = alloca(infoLogLength \* sizeof(GLchar));

glGetShaderInfoLog(shaderId, infoLogLength, NULL, message);

fprintf(stderr, "Warning: %s\n", message);

}

}

## 代码参考3：https://github.com/SweeneyChoi/FBO-in-OpenGL

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include "stb\_image.h"

#include <glm/glm.hpp>

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

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

#include "Shader.h"

#include "Camera.h"

#include "Model.h"

#include <iostream>

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);

const unsigned int SCR\_WIDTH = 1280;

const unsigned int SCR\_HEIGHT = 720;

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;

float deltaTime = 0.0f;

float lastFrame = 0.0f;

int main()

{

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 4 );

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

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

#ifdef \_\_APPLE\_\_

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

#endif

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

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

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

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

return -1;

}

glEnable(GL\_DEPTH\_TEST);

Shader shader("vshader.glsl", "fshader.glsl");

Shader screenShader("screenvShader.glsl", "screenfShader.glsl");

float cubeVertices[] = {

-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[] = {

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

};

float quadVertices[] = {

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

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

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

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

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

1.0f, 1.0f, 1.0f, 1.0f

};

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)));

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)));

unsigned int quadVAO, quadVBO;

glGenVertexArrays(1, &quadVAO);

glGenBuffers(1, &quadVBO);

glBindVertexArray(quadVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, quadVBO);

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

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 4 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 4 \* sizeof(float), (void\*)(2 \* sizeof(float)));

unsigned int cubeTexture = loadTexture("container.jpg");

unsigned int floorTexture = loadTexture("metal.png");

shader.use();

shader.setInt("texture1", 0);

screenShader.use();

screenShader.setInt("screenTexture", 0);

unsigned int frameBuffer;

glGenFramebuffers(1, &frameBuffer);

glBindFramebuffer(GL\_FRAMEBUFFER, frameBuffer);

unsigned int textureColorBufferMultiSampled;

glGenTextures(1, &textureColorBufferMultiSampled);

glBindTexture(GL\_TEXTURE\_2D\_MULTISAMPLE, textureColorBufferMultiSampled);

glTexImage2DMultisample(GL\_TEXTURE\_2D\_MULTISAMPLE, 4, GL\_RGB, SCR\_WIDTH, SCR\_HEIGHT, GL\_TRUE);

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D\_MULTISAMPLE, textureColorBufferMultiSampled, 0);

unsigned int rbo;

glGenRenderbuffers(1, &rbo);

glBindRenderbuffer(GL\_RENDERBUFFER, rbo);

glRenderbufferStorageMultisample(GL\_RENDERBUFFER, 4, GL\_DEPTH24\_STENCIL8, SCR\_WIDTH, SCR\_HEIGHT);

glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_STENCIL\_ATTACHMENT, GL\_RENDERBUFFER, rbo);

if (glCheckFramebufferStatus(GL\_FRAMEBUFFER) != GL\_FRAMEBUFFER\_COMPLETE) {

cout << "ERROR: FRAMENBUFFER:: Framebuffer is not complete!" << endl;

}

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

unsigned int intermediateFBO;

glGenFramebuffers(1, &intermediateFBO);

glBindFramebuffer(GL\_FRAMEBUFFER, intermediateFBO);

unsigned int screenTexture;

glGenTextures(1, &screenTexture);

glBindTexture(GL\_TEXTURE\_2D, screenTexture);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, SCR\_WIDTH, SCR\_HEIGHT, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, NULL);

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

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

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, screenTexture, 0);

if (glCheckFramebufferStatus(GL\_FRAMEBUFFER) != GL\_FRAMEBUFFER\_COMPLETE) {

cout << "ERROR: FRAMENBUFFER:: Framebuffer is not complete!" << endl;

}

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

while (!glfwWindowShouldClose(window))

{

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

processInput(window);

glBindFramebuffer(GL\_FRAMEBUFFER, frameBuffer);

glEnable(GL\_DEPTH\_TEST);

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 view = camera.GetViewMatrix();

glm::mat4 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);

glBindVertexArray(cubeVAO);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, cubeTexture);

model = glm::translate(model, glm::vec3(-1.0f, 0.0f, -1.0f));

shader.setMat4("model", model);

glDrawArrays(GL\_TRIANGLES, 0, 36);

model = glm::mat4();

model = glm::translate(model, glm::vec3(2.0f, 0.0f, 0.0f));

shader.setMat4("model", model);

glDrawArrays(GL\_TRIANGLES, 0, 36);

glBindVertexArray(planeVAO);

glBindTexture(GL\_TEXTURE\_2D, floorTexture);

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

glDrawArrays(GL\_TRIANGLES, 0, 6);

glBindVertexArray(0);

glBindFramebuffer(GL\_READ\_FRAMEBUFFER, frameBuffer);

glBindFramebuffer(GL\_DRAW\_FRAMEBUFFER, intermediateFBO);

glBlitFramebuffer(0, 0, SCR\_WIDTH, SCR\_HEIGHT, 0, 0, SCR\_WIDTH, SCR\_HEIGHT, GL\_COLOR\_BUFFER\_BIT, GL\_NEAREST);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

glDisable(GL\_DEPTH\_TEST);

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

glClear(GL\_COLOR\_BUFFER\_BIT);

screenShader.use();

glBindVertexArray(quadVAO);

glBindTexture(GL\_TEXTURE\_2D, screenTexture);

glDrawArrays(GL\_TRIANGLES, 0, 6);

glfwSwapBuffers(window);

glfwPollEvents();

}

glDeleteVertexArrays(1, &cubeVAO);

glDeleteVertexArrays(1, &planeVAO);

glDeleteVertexArrays(1, &quadVAO);

glDeleteBuffers(1, &cubeVBO);

glDeleteBuffers(1, &planeVBO);

glDeleteBuffers(1, &quadVBO);

glfwTerminate();

return 0;

}

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);

}

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

{

glViewport(0, 0, width, height);

}

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

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

camera.ProcessMouseMovement(xoffset, yoffset);

}

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

{

camera.ProcessMouseScroll(yoffset);

}

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;

}

## 代码参考4：https://github.com/robbor78/OpenGL\_BasicFBO

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include "shader\_utils.h"

void Initialize();

void InitGL();

void InitProgramFBO();

void InitProgramScreen();

void InitBuffer();

void InitFBO();

void Loop();

void RenderToFBO();

void RenderToScreen();

void Shutdown();

void OnWindowResize(GLFWwindow\* window, int width, int height);

GLFWwindow\* window;

int screenWidth = 640;

int screenHeight = 480;

GLuint render2FBOProgram;

GLuint render2ScreenProgram;

GLuint vao;

GLuint vbo;

GLuint fbo;

GLuint color\_texture;

int main() {

Initialize();

Loop();

Shutdown();

return 0;

}

void Initialize() {

InitGL();

InitProgramScreen();

InitProgramFBO();

InitBuffer();

InitFBO();

}

void InitGL() {

glfwInit();

window = glfwCreateWindow(screenWidth, screenHeight, "FBO Demo", NULL, NULL);

glfwMakeContextCurrent(window);

glewInit();

glClearColor(0.0f, 0.0f, 0.1f, 1.0f);

}

void InitProgramFBO() {

GLuint vs;

GLuint fs;

render2FBOProgram = create\_program("vs.glsl", "fbo.fs.glsl", vs, fs);

glDeleteShader(vs);

glDeleteShader(fs);

}

void InitProgramScreen() {

GLuint vs;

GLuint fs;

render2ScreenProgram = create\_program("vs.glsl", "screen.fs.glsl", vs, fs);

glDeleteShader(vs);

glDeleteShader(fs);

}

void InitBuffer() {

*//define the square made up of 2 triangles*

static const GLfloat points[] = {

*//x y z texcoord u and v*

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

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

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

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

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

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

};

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

*//create buffer for points*

glGenBuffers(1, &vbo);

glBindBuffer(GL\_ARRAY\_BUFFER, vbo);

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

*//tell opengl how to find the coordinate data*

glVertexAttribPointer (0, 3, GL\_FLOAT, GL\_FALSE, 5\*sizeof(GLfloat), (GLubyte\*)NULL);

glEnableVertexAttribArray(0);

*//tell opengl how to find the texcoord data*

glVertexAttribPointer (1, 2, GL\_FLOAT, GL\_FALSE, 5\*sizeof(GLfloat), (GLvoid \*)(3 \* sizeof(GLfloat)));

glEnableVertexAttribArray(1);

}

void InitFBO() {

*//create a framebuffer*

glGenFramebuffers(1, &fbo);

glBindFramebuffer(GL\_FRAMEBUFFER, fbo);

*//create a texture as the backing store for the framebuffer*

glGenTextures(1, &color\_texture);

glBindTexture(GL\_TEXTURE\_2D, color\_texture);

glTexStorage2D(GL\_TEXTURE\_2D, 9, GL\_RGBA8, 512, 512); *//1 = mipmap levels*

*//mip map filtering*

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

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

*//attach the texture as the color attachment of the framebuffer*

glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0, GL\_TEXTURE\_2D, color\_texture, 0);

*//tell opengl to draw into the color attachment*

static const GLenum draw\_buffers[] = { GL\_COLOR\_ATTACHMENT0 };

glDrawBuffers(1, draw\_buffers);

}

void Loop() {

*//glBindVertexArray(vao);*

*//glBindBuffer(GL\_ARRAY\_BUFFER, vbo);*

while (!glfwWindowShouldClose(window)) {

RenderToFBO();

RenderToScreen();

glfwSwapBuffers(window);

glfwPollEvents();

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

glfwSetWindowShouldClose(window, 1);

}

}

}

void RenderToFBO() {

static const GLfloat green[] = { 0.0f, 1.0f, 0.0f, 1.0f }; *//texture background is green*

glBindFramebuffer(GL\_FRAMEBUFFER, fbo);

glViewport(0, 0, 512, 512); *//set view port to texture size*

glClearBufferfv(GL\_COLOR, 0, green);

glUseProgram(render2FBOProgram);

glDrawArrays(GL\_TRIANGLES, 0, 6);

glBindFramebuffer(GL\_FRAMEBUFFER, 0);

}

void RenderToScreen() {

static const GLfloat blue[] = { 0.0f, 0.0f, 1.0f, 1.0f }; *//screen background is blue*

glViewport(0, 0, screenWidth, screenHeight);

glClearBufferfv(GL\_COLOR, 0, blue);

glBindTexture(GL\_TEXTURE\_2D, color\_texture);

glUseProgram(render2ScreenProgram);

glDrawArrays(GL\_TRIANGLES, 0, 6);

glBindTexture(GL\_TEXTURE\_2D, 0);

}

void Shutdown() {

glUseProgram(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteProgram(render2FBOProgram);

glDeleteProgram(render2ScreenProgram);

glfwTerminate();

}

*// a call-back function*

void OnWindowResize(GLFWwindow\* window, int width, int height) {

screenWidth = width;

screenHeight = height;

glViewport(0, 0, screenWidth, screenHeight);

}

## 代码参考5：

GLuint VolRenCore::RendrToFBO()

{

if (!m\_Data || !m\_Data->GetData()) return 0;

if (tex)

{

glDeleteTexturesEXT(1, &tex);

tex = 0;

}

*//init gl*

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_DEPTH\_TEST);

*//fbo*

GLuint fb;

glGenFramebuffersEXT(1, &fb);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, fb);

*//texture buffer*

glGenTextures(1, &tex);

glActiveTexture(GL\_TEXTURE0);

glBindTexture(GL\_TEXTURE\_2D, tex);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);

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

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

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

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, m\_ViewPort[0], m\_ViewPort[1], 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, 0);

*//render buffer*

GLuint rbo;

glGenRenderbuffersEXT(1, &rbo);

glBindRenderbufferEXT(GL\_RENDERBUFFER, rbo);

glRenderbufferStorageEXT(GL\_RENDERBUFFER, GL\_DEPTH\_COMPONENT, m\_ViewPort[0], m\_ViewPort[1]);

*//glBindRenderbufferEXT(GL\_RENDERBUFFER, 0);*

*//attach*

glFramebufferTexture2DEXT(GL\_FRAMEBUFFER\_EXT, GL\_COLOR\_ATTACHMENT0\_EXT, GL\_TEXTURE\_2D, tex, 0);

glFramebufferRenderbufferEXT(GL\_FRAMEBUFFER\_EXT,GL\_DEPTH\_ATTACHMENT, GL\_RENDERBUFFER, rbo);

glDrawBuffer(GL\_COLOR\_ATTACHMENT0\_EXT);

*//检查FBO状态*

bool status = checkFramebufferStatus();

if (!status)

return 0;

if(!Render()) return 0;

*//*

*// glReadBuffer(GL\_COLOR\_ATTACHMENT0\_EXT);*

*// //glReadBuffer(GL\_DEPTH\_ATTACHMENT);*

*// int w = m\_ViewPort[0];*

*// int h = m\_ViewPort[1];*

*// QImage img(w,h, QImage::Format\_ARGB32);*

*// glReadPixels(0,0,w,h,GL\_RGBA, GL\_UNSIGNED\_BYTE, img.bits());*

*// mDisplayDialog fbo(img);*

*// fbo.exec();*

*//*

glBindTexture(GL\_TEXTURE\_2D, 0);

glBindFramebufferEXT(GL\_FRAMEBUFFER\_EXT, 0);

glDeleteFramebuffersEXT(1, &fb);

glDeleteRenderbuffersEXT(1, &rbo);

return tex;

}

void VRDisplayWidget::paintGL()

{

GLuint tex1;

GLuint tex2;

if (m\_Core)

{

tex1 = m\_Core->RendrToFBO();

m\_Core->m\_VolTextureNeedUpdate = true;

tex2 = m\_Core1->RendrToFBO();

m\_Core1->m\_VolTextureNeedUpdate = true;

if (!tex1) return;

if(!tex2) return;

glClearColor(0.0, 0.0, 0.0, 0.0f);

glClearDepth(1.0f);

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

glFrontFace(GL\_CCW);

*//gl*

glEnable(GL\_BLEND);

*//glBlendFunc(GL\_SRC\_ALPHA,GL\_ONE\_MINUS\_SRC\_ALPHA);*

glBlendFunc(GL\_SRC\_ALPHA,GL\_SRC\_ALPHA);

*//texture0*

glActiveTexture(GL\_TEXTURE0\_ARB);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, tex1);

*//texture1*

glActiveTexture(GL\_TEXTURE1\_ARB);

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, tex2);

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

glEnable(GL\_TEXTURE\_2D);

glBegin(GL\_QUADS);

glMultiTexCoord2f(GL\_TEXTURE0\_ARB, 0.0, 0.0);glMultiTexCoord2f(GL\_TEXTURE1\_ARB, 0.0, 0.0); glVertex2f(0.0, 0.0);

glMultiTexCoord2f(GL\_TEXTURE0\_ARB, 0.0, 1.0);glMultiTexCoord2f(GL\_TEXTURE1\_ARB, 0.0, 1.0); glVertex2f(0.0, hei);

glMultiTexCoord2f(GL\_TEXTURE0\_ARB, 1.0, 1.0);glMultiTexCoord2f(GL\_TEXTURE1\_ARB, 1.0, 1.0); glVertex2f(wid, hei);

glMultiTexCoord2f(GL\_TEXTURE0\_ARB, 1.0, 0.0);glMultiTexCoord2f(GL\_TEXTURE1\_ARB, 1.0, 0.0); glVertex2f(wid, 0.0);

glEnd();

*/\*glFlush();\*/*

glPushAttrib(GL\_ALL\_ATTRIB\_BITS);

glColor3f(1.0f,0.0f,0.0f);

renderText(30,40,QString::number(m\_fps)+"FPS",QFont("Arial",20));

glPopAttrib();

}

}