// Lighthouse3D.com OpenGL 3.3 + GLSL 3.3 Sample

// http://www.lighthouse3d.com/cg-topics/code-samples/importing-3d-models-with-assimp/

// Lighthouse3D.com OpenGL 3.3 Loading an Image File and Creating a Texture

// http://www.lighthouse3d.com/2013/01/loading-and-image-file-and-creating-a-texture/

// C++ 11 Multithreading Tutorial

// https://solarianprogrammer.com/2011/12/16/cpp-11-thread-tutorial/

// AruCo Camera Calibration

// https://github.com/opencv/opencv\_contrib/blob/master/modules/aruco/samples/calibrate\_camera.cpp

//866 和1463

#ifdef \_WIN32

#pragma comment(lib,"assimp.lib")

#pragma comment(lib,"devil.lib")

#pragma comment(lib,"glew32.lib")

#endif

//OpenCV Includes

#include <opencv2/opencv.hpp>

#include <opencv2/aruco.hpp>

#include <opencv2/core/opengl.hpp>

// include DevIL for image loading 包含DevIL进行图像加载

#include <IL/il.h>

// include GLEW to access OpenGL 3.3 functions 包含GLEW以访问OpenGL 3.3功能

#include <GL/glew.h>

// GLUT is the toolkit to interface with the OS GLUT是与操作系统交互的工具包

#include <GL/freeglut.h>

// auxiliary C file to read the shader text files

//辅助C文件以读取着色器文本文件

#include "textfile.h"

// assimp include files. These three are usually needed. assimp包含文件，通常需要这三个。

#include "assimp/Importer.hpp" //OO version Header!

#include "assimp/postprocess.h"

#include "assimp/scene.h"

#include <math.h>

#include <fstream>

#include <map>

#include <string>

#include <vector>

#include <iostream>

#include <thread> // std::thread

#include <map>

#define WINDOW\_NAME "AR demo"

using namespace std;

#include<GL/glut.h>

#include<windows.h>

#include<math.h>

#include<stdlib.h>

#include<iostream>

#define C(j) cos((3.1415926/180)\*2\*(j))

#define S(j) sin((3.1415926/180)\*2\*(j))

#define B 32

#pragma comment(lib, "glut32.lib")

double ratio2 = 0.01;//血管的缩放比例

int upperbound = 1354;

GLfloat roate = 0.0;// 设置旋转速率

GLfloat rote = 0.0;//旋转角度

GLfloat anglex =-203.0;//X 轴旋转

GLfloat angley = 150.0;//Y 轴旋转

GLfloat anglez = -15.0;//Z 轴旋转

GLint WinW = 400;

GLint WinH = 400;

GLfloat oldx;//当左键按下时记录鼠标坐标

GLfloat oldy;

int di = 10;//精确度，di越小越精确，di是每次跳过几个点

float PI = 3.1415926f;

double thick = 1.5;

int begini = 1;//renderscene里面画血管，循环开始的第i个点

double slice = 10.0;//血管的slices数

double stack = 4.0;//血管的stack数

double dx = 0.0, dy = 0.0, dz = 0.0;

//文件

FILE\* fp = nullptr;

int timePin=1;//随时间变化，模型会改变

//Scene and view info for each model

//每个模型的场景和视图信息

struct MyModel

{

aiScene\* scene; // the global Assimp scene object for each model 每个模型的全局Assimp场景对象

std::vector<cv::Mat> viewMatrix = { cv::Mat::zeros(4, 4, CV\_32F),cv::Mat::zeros(4, 4, CV\_32F),cv::Mat::zeros(4, 4, CV\_32F) };

int seenlast = 0;

bool seennow = false;

Assimp::Importer importer; // Create an instance of the Importer class创建Importer类的实例

std::vector<struct MyMesh> myMeshes;

float scaleFactor; // scale factor for the model to fit in the window 模型的比例因子以适合窗口

int marker; //marker corresponding to this model 与此模型对应的标记

};

// Information to render each assimp node 渲染每个assimp节点的信息

struct MyMesh

{

GLuint vao;

GLuint texIndex;

GLuint uniformBlockIndex;

int numFaces;

};

// This is for a shader uniform block 这是用于着色器统一块

struct MyMaterial

{

float diffuse[4];

float ambient[4];

float specular[4];

float emissive[4];

float shininess;

int texCount;

};

class point3d {

public:

float x;

float y;

float z;

point3d() {}

void assignValue(float x, float y, float z);

~point3d() {}

};

//血管的点集

point3d\* pt = new point3d[1400];

//Window Default size

int windowWidth = 512, windowHeight = 512;

//int tx\_origin[2] = { 546,1000 }, ty\_origin[2] = { 1146 ,1000 }, tz\_origin[2] = { 1000,1000 };//坐标的初始值

int tx[2] = { 839,1000 }, ty[2] = { 1439 ,1000 }, tz[2] = { 1000,1000 };//改变量,第一个是血管模型，第二个是

int scalex[2] = { 1834,2000 }, scaley[2] = { 1093,2000 }, scalez[2] = { 2615,2000 };

int rotatexyz[2] = { 0,0 }, rotatex[2] = { 0,0 }, rotatey[2]= { 0,0 }, rotatez[2]= { 0,0 };

//GLdouble anglex=0.0, angley =0.0, anglez = 0.0;//血管摄像头的角度

int angle[2] = { 180, 90};

/\*

int tx\_dot = 1000, ty\_dot = 1000, tz\_dot = 1000;//改变量

int scalex\_dot = 2000, scaley\_dot = 2000, scalez\_dot = 2000;

int rotatexyz\_dot = 0, rotatex\_dot, rotatey\_dot, rotatez\_dot;

int angle\_dot = 90;\*/

// Model Matrix (part of the OpenGL Model View Matrix)

float modelMatrix[16];

// For push and pop matrix

std::vector<float\*> matrixStack;

// Vertex Attribute Locations 顶点属性位置

GLuint vertexLoc = 0, normalLoc = 1, texCoordLoc = 2;

// Uniform Bindings Points 统一绑定点

GLuint matricesUniLoc = 1, materialUniLoc = 2;

// The sampler uniform for textured models 用于纹理模型的采样器制服

// we are assuming a single texture so this will 我们假设一个纹理

//always be texture unit 0 始终为纹理单位0

GLuint texUnit = 0;

// Uniform Buffer for Matrices 矩阵的统一缓冲区

// this buffer will contain 3 matrices: projection, view and model 此缓冲区将包含3个矩阵：投影，视图和模型

// each matrix is a float array with 16 components 每个矩阵都是具有16个组件的float

GLuint matricesUniBuffer;

#define MatricesUniBufferSize sizeof(float) \* 16 \* 3

#define ProjMatrixOffset 0

#define ViewMatrixOffset sizeof(float) \* 16

#define ModelMatrixOffset sizeof(float) \* 16 \* 2

#define MatrixSize sizeof(float) \* 16

// Program and Shader Identifiers 程序和着色器标识符

GLuint program, vertexShader, fragmentShader;

GLuint p, vertexShader2D, fragmentShader2D;

// holder for the vertex array object id 顶点数组对象ID的持有人

GLuint vao, textureID;

// Shader Names

char\* vertexFileName = (char\*)"dirlightdiffambpix.vert";

char\* fragmentFileName = (char\*)"dirlightdiffambpix.frag";

std::map<int, MyModel> models;

// images / texture

// map image filenames to textureIds

// pointer to texture Array

//图片/纹理

//将图像文件名映射到textureIds

//指向纹理数组的指针

std::map<std::string, GLuint> textureIdMap;

// Replace the model name by your model's filename

//static const std::string modelname = "jeep1.ms3d";

std::map<int, string> modelMap;

static const std::string modelDir = "F:/project/ar/models/"; //3d文件路径

//our aruco variables

cv::Ptr<cv::aruco::Dictionary> dictionary = cv::aruco::getPredefinedDictionary(cv::aruco::DICT\_4X4\_100);

cv::Ptr<cv::aruco::DetectorParameters> detectorParams = cv::aruco::DetectorParameters::create();

std::vector< int > markerIds;

std::vector< std::vector<cv::Point2f> > markerCorners, rejectedCandidates;

//cv::VideoCapture cap("D:/profiles/心血管比赛相关资料/video\_20201103\_132337.mp4");

cv::VideoCapture cap("F:/project/test6.mp4");

bool flipped = false;

double K\_[3][3] =

{ { 6.5423193332302969e+02, 0., 3.4382422833024395e+02 },

{ 0.,6.5358529192792446e+02, 2.7602128088484807e+02 },

{ 0, 0, 1 } };

cv::Mat K = cv::Mat(3, 3, CV\_64F, K\_).clone();

const float markerLength = 3.00;

// Distortion coeffs (fill in your actual values here). 失真系数（在此处填写您的实际值）。

double dist\_[] = { 0, 0, 0, 0, 0 };

cv::Mat distCoeffs = cv::Mat(5, 1, CV\_64F, dist\_).clone();

cv::Mat imageMat;

cv::Mat imageMatGL;

#ifndef M\_PI

#define M\_PI 3.14159265358979323846f

#endif

static inline float

DegToRad(float degrees)

{

return (float)(degrees \* (M\_PI / 180.0f));

};

// Frame counting and FPS computation 帧计数和FPS计算

long timet, timebase = 0, frame = 0;

char s[32];

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

// Print for OpenGL errors

// Returns 1 if an OpenGL error occurred, 0 otherwise.

#define printOpenGLError() printOglError(\_\_FILE\_\_, \_\_LINE\_\_)

int printOglError(char\* file, int line)

{

GLenum glErr;

int retCode = 0;

glErr = glGetError();

if (glErr != GL\_NO\_ERROR)

{

printf("glError in file %s @ line %d: %s\n",

file, line, gluErrorString(glErr));

retCode = 1;

}

return retCode;

}

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

// MATRIX STUFF

//

// Push and Pop for modelMatrix

void pushMatrix()

{

float\* aux = (float\*)malloc(sizeof(float) \* 16);

memcpy(aux, modelMatrix, sizeof(float) \* 16);

matrixStack.push\_back(aux);

}

void popMatrix() {

float\* m = matrixStack[matrixStack.size() - 1];

memcpy(modelMatrix, m, sizeof(float) \* 16);

matrixStack.pop\_back();

free(m);

}

// sets the square matrix mat to the identity matrix, 将平方矩阵垫设置为单位矩阵，

// size refers to the number of rows (or columns) size是指行（或列）数

void setIdentityMatrix(float\* mat, int size) {

// fill matrix with 0s

for (int i = 0; i < size \* size; ++i)

mat[i] = 0.0f;

// fill diagonal with 1s 对角线

for (int i = 0; i < size; ++i)

mat[i + i \* size] = 0.3f; //调整模型的大小 size

}

// a = a \* b;

void multMatrix(float\* a, float\* b)

{

float res[16];

for (int i = 0; i < 4; ++i) {

for (int j = 0; j < 4; ++j) {

res[j \* 4 + i] = 0.0f;

for (int k = 0; k < 4; ++k) {

res[j \* 4 + i] += a[k \* 4 + i] \* b[j \* 4 + k];

}

}

}

memcpy(a, res, 16 \* sizeof(float));

}

// Defines a transformation matrix mat with a translation 定义带有转换的转换矩阵

void setTranslationMatrix(float\* mat, float x, float y, float z)

{

setIdentityMatrix(mat, 4);

mat[12] = x;

mat[13] = y;

mat[14] = z;

}

// Defines a transformation matrix mat with a scale 用比例尺定义转换矩阵

void setScaleMatrix(float\* mat, float sx, float sy, float sz)

{

setIdentityMatrix(mat, 4);

mat[0] = sx;

mat[5] = sy;

mat[10] = sz;

}

// Defines a transformation matrix mat with a rotation 定义旋转矩阵

// angle alpha and a rotation axis (x,y,z) 角度alpha和旋转轴（x，y，z）

void setRotationMatrix(float\* mat, float angle, float x, float y, float z) {

float radAngle = DegToRad(angle);

float co = cos(radAngle);

float si = sin(radAngle);

float x2 = x \* x;

float y2 = y \* y;

float z2 = z \* z;

mat[0] = x2 + (y2 + z2) \* co;

mat[4] = x \* y \* (1 - co) - z \* si;

mat[8] = x \* z \* (1 - co) + y \* si;

mat[12] = 0.0f;

mat[1] = x \* y \* (1 - co) + z \* si;

mat[5] = y2 + (x2 + z2) \* co;

mat[9] = y \* z \* (1 - co) - x \* si;

mat[13] = 0.0f;

mat[2] = x \* z \* (1 - co) - y \* si;

mat[6] = y \* z \* (1 - co) + x \* si;

mat[10] = z2 + (x2 + y2) \* co;

mat[14] = 0.0f;

mat[3] = 0.0f;

mat[7] = 0.0f;

mat[11] = 0.0f;

mat[15] = 1.0f;

}

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

// Model Matrix

//

// Copies the modelMatrix to the uniform buffer 将modelMatrix复制到统一缓冲区

void setModelMatrix()

{

glBindBuffer(GL\_UNIFORM\_BUFFER, matricesUniBuffer);

glBufferSubData(GL\_UNIFORM\_BUFFER, ModelMatrixOffset, MatrixSize, modelMatrix);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

}

// The equivalent to glTranslate applied to the model matrix

void translate(float x, float y, float z)

{

float aux[16];

setTranslationMatrix(aux, x, y, z);

multMatrix(modelMatrix, aux);

setModelMatrix();

}

// The equivalent to glRotate applied to the model matrix

void rotate(float angle, float x, float y, float z) {

float aux[16];

setRotationMatrix(aux, angle, x, y, z);

multMatrix(modelMatrix, aux);

setModelMatrix();

}

// The equivalent to glScale applied to the model matrix

void scale(float x, float y, float z) {

float aux[16];

setScaleMatrix(aux, x, y, z);

multMatrix(modelMatrix, aux);

setModelMatrix();

}

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

// Projection Matrix

// Computes the projection Matrix and stores it in the uniform buffer

void buildProjectionMatrix(float fov, float ratio, float nearp, float farp) {

float projMatrix[16];

float f = 1.0f / tan(fov \* (M\_PI / 360.0f));

setIdentityMatrix(projMatrix, 4);

projMatrix[0] = f / ratio;

projMatrix[1 \* 4 + 1] = f;

projMatrix[2 \* 4 + 2] = (farp + nearp) / (nearp - farp);

projMatrix[3 \* 4 + 2] = (2.0f \* farp \* nearp) / (nearp - farp);

projMatrix[2 \* 4 + 3] = -1.0f;

projMatrix[3 \* 4 + 3] = 0.0f;

glBindBuffer(GL\_UNIFORM\_BUFFER, matricesUniBuffer);

glBufferSubData(GL\_UNIFORM\_BUFFER, ProjMatrixOffset, MatrixSize, projMatrix);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

}

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

// View Matrix

//

// Computes the viewMatrix and stores it in the uniform buffer

// note: it assumes the camera is not tilted,

// i.e. a vertical up vector along the Y axis (remember gluLookAt?

//注意：假设相机没有倾斜，

//即沿Y轴的垂直向上矢量

void setCamera(cv::Mat viewMatrix) {

//Set these to make the view matrix happy

viewMatrix.at<float>(0, 3) = 0;

viewMatrix.at<float>(1, 3) = 0;

viewMatrix.at<float>(2, 3) = 0;

viewMatrix.at<float>(3, 3) = 1;

glBindBuffer(GL\_UNIFORM\_BUFFER, matricesUniBuffer);

glBufferSubData(GL\_UNIFORM\_BUFFER, ViewMatrixOffset, MatrixSize, (float\*)viewMatrix.data);

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

}

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

#define aisgl\_min(x,y) (x<y?x:y)

#define aisgl\_max(x,y) (y>x?y:x)