// 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;

//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;

};

struct point3d {

GLfloat x;

GLfloat y;

GLfloat z;

};

void assignValue(point3d pt, GLfloat x, GLfloat y, GLfloat z) {

pt.x = x;

pt.y = y;

pt.z = z;

}

//血管中心点的坐标数据

point3d\* assignBatchPoint() {

point3d pt[1500];

assignValue(pt[1], 343.5371, 352.0383, 13.112878);

assignValue(pt[2], 344.5371, 353.0383, 14.112878);

assignValue(pt[3], 345.5371, 354.0383, 15.112878);

assignValue(pt[4], 346.5371, 355.0383, 16.112878);

assignValue(pt[5], 347.5371, 356.0383, 17.112878);

assignValue(pt[6], 348.5371, 357.0383, 18.112878);

assignValue(pt[7], 349.5371, 358.0383, 19.112878);

assignValue(pt[8], 350.5371, 359.0383, 20.112878);

assignValue(pt[9], 351.5371, 360.0383, 21.112878);

assignValue(pt[10], 352.5371, 361.0383, 22.112878);

assignValue(pt[11], 353.5371, 362.0383, 23.112878);

assignValue(pt[12], 354.5371, 363.0383, 24.112878);

assignValue(pt[13], 355.5371, 364.0383, 25.112878);

assignValue(pt[14], 356.5371, 365.0383, 26.112878);

assignValue(pt[15], 357.5371, 366.0383, 27.112878);

assignValue(pt[16], 358.5371, 367.0383, 28.112878);

assignValue(pt[17], 359.5371, 368.0383, 29.112878);

assignValue(pt[18], 360.5371, 369.0383, 30.112878);

assignValue(pt[19], 361.5371, 370.0383, 31.112878);

assignValue(pt[20], 362.5371, 371.0383, 32.112878);

assignValue(pt[21], 363.5371, 372.0383, 33.112878);

assignValue(pt[22], 364.5371, 373.0383, 34.112878);

assignValue(pt[23], 365.5371, 374.0383, 35.112878);

assignValue(pt[24], 366.5371, 375.0383, 36.112878);

assignValue(pt[25], 367.5371, 376.0383, 37.112878);

assignValue(pt[26], 368.5371, 377.0383, 38.112878);

assignValue(pt[27], 369.5371, 378.0383, 39.112878);

assignValue(pt[28], 370.5371, 379.0383, 40.112878);

assignValue(pt[29], 371.5371, 380.0383, 41.112878);

assignValue(pt[30], 372.5371, 381.0383, 42.112878);

assignValue(pt[31], 373.5371, 382.0383, 43.112878);

assignValue(pt[32], 374.5371, 383.0383, 44.112878);

assignValue(pt[33], 375.5371, 384.0383, 45.112878);

assignValue(pt[34], 376.5371, 385.0383, 46.112878);

assignValue(pt[35], 377.5371, 386.0383, 47.112878);

assignValue(pt[36], 378.5371, 387.0383, 48.112878);

assignValue(pt[37], 379.5371, 388.0383, 49.112878);

assignValue(pt[38], 380.5371, 389.0383, 50.112878);

assignValue(pt[39], 381.5371, 390.0383, 51.112878);

assignValue(pt[40], 382.5371, 391.0383, 52.112878);

assignValue(pt[41], 383.5371, 392.0383, 53.112878);

assignValue(pt[42], 384.5371, 393.0383, 54.112878);

assignValue(pt[43], 385.5371, 394.0383, 55.112878);

assignValue(pt[44], 386.5371, 395.0383, 56.112878);

assignValue(pt[45], 387.5371, 396.0383, 57.112878);

assignValue(pt[46], 388.5371, 397.0383, 58.112878);

assignValue(pt[47], 389.5371, 398.0383, 59.112878);

assignValue(pt[48], 390.5371, 399.0383, 60.112878);

assignValue(pt[49], 391.5371, 400.0383, 61.112878);

assignValue(pt[50], 392.5371, 401.0383, 62.112878);

assignValue(pt[51], 393.5371, 402.0383, 63.112878);

assignValue(pt[52], 394.5371, 403.0383, 64.112878);

assignValue(pt[53], 395.5371, 404.0383, 65.112878);

assignValue(pt[54], 396.5371, 405.0383, 66.112878);

assignValue(pt[55], 397.5371, 406.0383, 67.112878);

assignValue(pt[56], 398.5371, 407.0383, 68.112878);

assignValue(pt[57], 399.5371, 408.0383, 69.112878);

assignValue(pt[58], 400.5371, 409.0383, 70.112878);

assignValue(pt[59], 401.5371, 410.0383, 71.112878);

assignValue(pt[60], 402.5371, 411.0383, 72.112878);

assignValue(pt[61], 403.5371, 412.0383, 73.112878);

assignValue(pt[62], 404.5371, 413.0383, 74.112878);

assignValue(pt[63], 405.5371, 414.0383, 75.112878);

assignValue(pt[64], 406.5371, 415.0383, 76.112878);

assignValue(pt[65], 407.5371, 416.0383, 77.112878);

assignValue(pt[66], 408.5371, 417.0383, 78.112878);

assignValue(pt[67], 409.5371, 418.0383, 79.112878);

assignValue(pt[68], 410.5371, 419.0383, 80.112878);

assignValue(pt[69], 411.5371, 420.0383, 81.112878);

assignValue(pt[70], 412.5371, 421.0383, 82.112878);

assignValue(pt[71], 413.5371, 422.0383, 83.112878);

assignValue(pt[72], 414.5371, 423.0383, 84.112878);

assignValue(pt[73], 415.5371, 424.0383, 85.112878);

assignValue(pt[74], 416.5371, 425.0383, 86.112878);

assignValue(pt[75], 417.5371, 426.0383, 87.112878);

assignValue(pt[76], 418.5371, 427.0383, 88.112878);

assignValue(pt[77], 419.5371, 428.0383, 89.112878);

assignValue(pt[78], 420.5371, 429.0383, 90.112878);

assignValue(pt[79], 421.5371, 430.0383, 91.112878);

assignValue(pt[80], 422.5371, 431.0383, 92.112878);

assignValue(pt[81], 423.5371, 432.0383, 93.112878);

assignValue(pt[82], 424.5371, 433.0383, 94.112878);

assignValue(pt[83], 425.5371, 434.0383, 95.112878);

assignValue(pt[84], 426.5371, 435.0383, 96.112878);

assignValue(pt[85], 427.5371, 436.0383, 97.112878);

assignValue(pt[86], 428.5371, 437.0383, 98.112878);

assignValue(pt[87], 429.5371, 438.0383, 99.112878);

assignValue(pt[88], 430.5371, 439.0383, 100.112878);

assignValue(pt[89], 431.5371, 440.0383, 101.112878);

assignValue(pt[90], 432.5371, 441.0383, 102.112878);

assignValue(pt[91], 433.5371, 442.0383, 103.112878);

assignValue(pt[92], 434.5371, 443.0383, 104.112878);

assignValue(pt[93], 435.5371, 444.0383, 105.112878);

assignValue(pt[94], 436.5371, 445.0383, 106.112878);

assignValue(pt[95], 437.5371, 446.0383, 107.112878);

assignValue(pt[96], 438.5371, 447.0383, 108.112878);

assignValue(pt[97], 439.5371, 448.0383, 109.112878);

assignValue(pt[98], 440.5371, 449.0383, 110.112878);

assignValue(pt[99], 441.5371, 450.0383, 111.112878);

assignValue(pt[100], 442.5371, 451.0383, 112.112878);

assignValue(pt[101], 443.5371, 452.0383, 113.112878);

assignValue(pt[102], 444.5371, 453.0383, 114.112878);

assignValue(pt[103], 445.5371, 454.0383, 115.112878);

assignValue(pt[104], 446.5371, 455.0383, 116.112878);

assignValue(pt[105], 447.5371, 456.0383, 117.112878);

assignValue(pt[106], 448.5371, 457.0383, 118.112878);

assignValue(pt[107], 449.5371, 458.0383, 119.112878);

assignValue(pt[108], 450.5371, 459.0383, 120.112878);

assignValue(pt[109], 451.5371, 460.0383, 121.112878);

assignValue(pt[110], 452.5371, 461.0383, 122.112878);

assignValue(pt[111], 453.5371, 462.0383, 123.112878);

assignValue(pt[112], 454.5371, 463.0383, 124.112878);

assignValue(pt[113], 455.5371, 464.0383, 125.112878);

assignValue(pt[114], 456.5371, 465.0383, 126.112878);

assignValue(pt[115], 457.5371, 466.0383, 127.112878);

assignValue(pt[116], 458.5371, 467.0383, 128.112878);

assignValue(pt[117], 459.5371, 468.0383, 129.112878);

assignValue(pt[118], 460.5371, 469.0383, 130.112878);

assignValue(pt[119], 461.5371, 470.0383, 131.112878);

assignValue(pt[120], 462.5371, 471.0383, 132.112878);

assignValue(pt[121], 463.5371, 472.0383, 133.112878);

assignValue(pt[122], 464.5371, 473.0383, 134.112878);

assignValue(pt[123], 465.5371, 474.0383, 135.112878);

assignValue(pt[124], 466.5371, 475.0383, 136.112878);

assignValue(pt[125], 467.5371, 476.0383, 137.112878);

assignValue(pt[126], 468.5371, 477.0383, 138.112878);

assignValue(pt[127], 469.5371, 478.0383, 139.112878);

assignValue(pt[128], 470.5371, 479.0383, 140.112878);

assignValue(pt[129], 471.5371, 480.0383, 141.112878);

assignValue(pt[130], 472.5371, 481.0383, 142.112878);

assignValue(pt[131], 473.5371, 482.0383, 143.112878);

assignValue(pt[132], 474.5371, 483.0383, 144.112878);

assignValue(pt[133], 475.5371, 484.0383, 145.112878);

assignValue(pt[134], 476.5371, 485.0383, 146.112878);

assignValue(pt[135], 477.5371, 486.0383, 147.112878);

assignValue(pt[136], 478.5371, 487.0383, 148.112878);

assignValue(pt[137], 479.5371, 488.0383, 149.112878);

assignValue(pt[138], 480.5371, 489.0383, 150.112878);

assignValue(pt[139], 481.5371, 490.0383, 151.112878);

assignValue(pt[140], 482.5371, 491.0383, 152.112878);

assignValue(pt[141], 483.5371, 492.0383, 153.112878);

assignValue(pt[142], 484.5371, 493.0383, 154.112878);

assignValue(pt[143], 485.5371, 494.0383, 155.112878);

assignValue(pt[144], 486.5371, 495.0383, 156.112878);

assignValue(pt[145], 487.5371, 496.0383, 157.112878);

assignValue(pt[146], 488.5371, 497.0383, 158.112878);

assignValue(pt[147], 489.5371, 498.0383, 159.112878);

assignValue(pt[148], 490.5371, 499.0383, 160.112878);

assignValue(pt[149], 491.5371, 500.0383, 161.112878);

assignValue(pt[150], 492.5371, 501.0383, 162.112878);

assignValue(pt[151], 493.5371, 502.0383, 163.112878);

assignValue(pt[152], 494.5371, 503.0383, 164.112878);

assignValue(pt[153], 495.5371, 504.0383, 165.112878);

assignValue(pt[154], 496.5371, 505.0383, 166.112878);

assignValue(pt[155], 497.5371, 506.0383, 167.112878);

assignValue(pt[156], 498.5371, 507.0383, 168.112878);

assignValue(pt[157], 499.5371, 508.0383, 169.112878);

assignValue(pt[158], 500.5371, 509.0383, 170.112878);

assignValue(pt[159], 501.5371, 510.0383, 171.112878);

assignValue(pt[160], 502.5371, 511.0383, 172.112878);

assignValue(pt[161], 503.5371, 512.0383, 173.112878);

assignValue(pt[162], 504.5371, 513.0383, 174.112878);

assignValue(pt[163], 505.5371, 514.0383, 175.112878);

assignValue(pt[164], 506.5371, 515.0383, 176.112878);

assignValue(pt[165], 507.5371, 516.0383, 177.112878);

assignValue(pt[166], 508.5371, 517.0383, 178.112878);

assignValue(pt[167], 509.5371, 518.0383, 179.112878);

assignValue(pt[168], 510.5371, 519.0383, 180.112878);

assignValue(pt[169], 511.5371, 520.0383, 181.112878);

assignValue(pt[170], 512.5371, 521.0383, 182.112878);

assignValue(pt[171], 513.5371, 522.0383, 183.112878);

assignValue(pt[172], 514.5371, 523.0383, 184.112878);

assignValue(pt[173], 515.5371, 524.0383, 185.112878);

assignValue(pt[174], 516.5371, 525.0383, 186.112878);

assignValue(pt[175], 517.5371, 526.0383, 187.112878);

assignValue(pt[176], 518.5371, 527.0383, 188.112878);

assignValue(pt[177], 519.5371, 528.0383, 189.112878);

assignValue(pt[178], 520.5371, 529.0383, 190.112878);

assignValue(pt[179], 521.5371, 530.0383, 191.112878);

assignValue(pt[180], 522.5371, 531.0383, 192.112878);

assignValue(pt[181], 523.5371, 532.0383, 193.112878);

assignValue(pt[182], 524.5371, 533.0383, 194.112878);

assignValue(pt[183], 525.5371, 534.0383, 195.112878);

assignValue(pt[184], 526.5371, 535.0383, 196.112878);

assignValue(pt[185], 527.5371, 536.0383, 197.112878);

assignValue(pt[186], 528.5371, 537.0383, 198.112878);

assignValue(pt[187], 529.5371, 538.0383, 199.112878);

assignValue(pt[188], 530.5371, 539.0383, 200.112878);

assignValue(pt[189], 531.5371, 540.0383, 201.112878);

assignValue(pt[190], 532.5371, 541.0383, 202.112878);

assignValue(pt[191], 533.5371, 542.0383, 203.112878);

assignValue(pt[192], 534.5371, 543.0383, 204.112878);

assignValue(pt[193], 535.5371, 544.0383, 205.112878);

assignValue(pt[194], 536.5371, 545.0383, 206.112878);

assignValue(pt[195], 537.5371, 546.0383, 207.112878);

assignValue(pt[196], 538.5371, 547.0383, 208.112878);

assignValue(pt[197], 539.5371, 548.0383, 209.112878);

assignValue(pt[198], 540.5371, 549.0383, 210.112878);

assignValue(pt[199], 541.5371, 550.0383, 211.112878);

assignValue(pt[200], 542.5371, 551.0383, 212.112878);

assignValue(pt[201], 543.5371, 552.0383, 213.112878);

assignValue(pt[202], 544.5371, 553.0383, 214.112878);

assignValue(pt[203], 545.5371, 554.0383, 215.112878);

assignValue(pt[204], 546.5371, 555.0383, 216.112878);

assignValue(pt[205], 547.5371, 556.0383, 217.112878);

assignValue(pt[206], 548.5371, 557.0383, 218.112878);

assignValue(pt[207], 549.5371, 558.0383, 219.112878);

assignValue(pt[208], 550.5371, 559.0383, 220.112878);

assignValue(pt[209], 551.5371, 560.0383, 221.112878);

assignValue(pt[210], 552.5371, 561.0383, 222.112878);

assignValue(pt[211], 553.5371, 562.0383, 223.112878);

assignValue(pt[212], 554.5371, 563.0383, 224.112878);

assignValue(pt[213], 555.5371, 564.0383, 225.112878);

assignValue(pt[214], 556.5371, 565.0383, 226.112878);

assignValue(pt[215], 557.5371, 566.0383, 227.112878);

assignValue(pt[216], 558.5371, 567.0383, 228.112878);

assignValue(pt[217], 559.5371, 568.0383, 229.112878);

assignValue(pt[218], 560.5371, 569.0383, 230.112878);

assignValue(pt[219], 561.5371, 570.0383, 231.112878);

assignValue(pt[220], 562.5371, 571.0383, 232.112878);

assignValue(pt[221], 563.5371, 572.0383, 233.112878);

assignValue(pt[222], 564.5371, 573.0383, 234.112878);

assignValue(pt[223], 565.5371, 574.0383, 235.112878);

assignValue(pt[224], 566.5371, 575.0383, 236.112878);

assignValue(pt[225], 567.5371, 576.0383, 237.112878);

assignValue(pt[226], 568.5371, 577.0383, 238.112878);

assignValue(pt[227], 569.5371, 578.0383, 239.112878);

assignValue(pt[228], 570.5371, 579.0383, 240.112878);

assignValue(pt[229], 571.5371, 580.0383, 241.112878);

assignValue(pt[230], 572.5371, 581.0383, 242.112878);

assignValue(pt[231], 573.5371, 582.0383, 243.112878);

assignValue(pt[232], 574.5371, 583.0383, 244.112878);

assignValue(pt[233], 575.5371, 584.0383, 245.112878);

assignValue(pt[234], 576.5371, 585.0383, 246.112878);

assignValue(pt[235], 577.5371, 586.0383, 247.112878);

assignValue(pt[236], 578.5371, 587.0383, 248.112878);

assignValue(pt[237], 579.5371, 588.0383, 249.112878);

assignValue(pt[238], 580.5371, 589.0383, 250.112878);

assignValue(pt[239], 581.5371, 590.0383, 251.112878);

assignValue(pt[240], 582.5371, 591.0383, 252.112878);

assignValue(pt[241], 583.5371, 592.0383, 253.112878);

assignValue(pt[242], 584.5371, 593.0383, 254.112878);

assignValue(pt[243], 585.5371, 594.0383, 255.112878);

assignValue(pt[244], 586.5371, 595.0383, 256.112878);

assignValue(pt[245], 587.5371, 596.0383, 257.112878);

assignValue(pt[246], 588.5371, 597.0383, 258.112878);

assignValue(pt[247], 589.5371, 598.0383, 259.112878);

assignValue(pt[248], 590.5371, 599.0383, 260.112878);

assignValue(pt[249], 591.5371, 600.0383, 261.112878);

assignValue(pt[250], 592.5371, 601.0383, 262.112878);

assignValue(pt[251], 593.5371, 602.0383, 263.112878);

assignValue(pt[252], 594.5371, 603.0383, 264.112878);

assignValue(pt[253], 595.5371, 604.0383, 265.112878);

assignValue(pt[254], 596.5371, 605.0383, 266.112878);

assignValue(pt[255], 597.5371, 606.0383, 267.112878);

assignValue(pt[256], 598.5371, 607.0383, 268.112878);

assignValue(pt[257], 599.5371, 608.0383, 269.112878);

assignValue(pt[258], 600.5371, 609.0383, 270.112878);

assignValue(pt[259], 601.5371, 610.0383, 271.112878);

assignValue(pt[260], 602.5371, 611.0383, 272.112878);

assignValue(pt[261], 603.5371, 612.0383, 273.112878);

assignValue(pt[262], 604.5371, 613.0383, 274.112878);

assignValue(pt[263], 605.5371, 614.0383, 275.112878);

assignValue(pt[264], 606.5371, 615.0383, 276.112878);

assignValue(pt[265], 607.5371, 616.0383, 277.112878);

assignValue(pt[266], 608.5371, 617.0383, 278.112878);

assignValue(pt[267], 609.5371, 618.0383, 279.112878);

assignValue(pt[268], 610.5371, 619.0383, 280.112878);

assignValue(pt[269], 611.5371, 620.0383, 281.112878);

assignValue(pt[270], 612.5371, 621.0383, 282.112878);

assignValue(pt[271], 613.5371, 622.0383, 283.112878);

assignValue(pt[272], 614.5371, 623.0383, 284.112878);

assignValue(pt[273], 615.5371, 624.0383, 285.112878);

assignValue(pt[274], 616.5371, 625.0383, 286.112878);

assignValue(pt[275], 617.5371, 626.0383, 287.112878);

assignValue(pt[276], 618.5371, 627.0383, 288.112878);

assignValue(pt[277], 619.5371, 628.0383, 289.112878);

assignValue(pt[278], 620.5371, 629.0383, 290.112878);

assignValue(pt[279], 621.5371, 630.0383, 291.112878);

assignValue(pt[280], 622.5371, 631.0383, 292.112878);

assignValue(pt[281], 623.5371, 632.0383, 293.112878);

assignValue(pt[282], 624.5371, 633.0383, 294.112878);

assignValue(pt[283], 625.5371, 634.0383, 295.112878);

assignValue(pt[284], 626.5371, 635.0383, 296.112878);

assignValue(pt[285], 627.5371, 636.0383, 297.112878);

assignValue(pt[286], 628.5371, 637.0383, 298.112878);

assignValue(pt[287], 629.5371, 638.0383, 299.112878);

assignValue(pt[288], 630.5371, 639.0383, 300.112878);

assignValue(pt[289], 631.5371, 640.0383, 301.112878);

assignValue(pt[290], 632.5371, 641.0383, 302.112878);

assignValue(pt[291], 633.5371, 642.0383, 303.112878);

assignValue(pt[292], 634.5371, 643.0383, 304.112878);

assignValue(pt[293], 635.5371, 644.0383, 305.112878);

assignValue(pt[294], 636.5371, 645.0383, 306.112878);

assignValue(pt[295], 637.5371, 646.0383, 307.112878);

assignValue(pt[296], 638.5371, 647.0383, 308.112878);

assignValue(pt[297], 639.5371, 648.0383, 309.112878);

assignValue(pt[298], 640.5371, 649.0383, 310.112878);

assignValue(pt[299], 641.5371, 650.0383, 311.112878);

assignValue(pt[300], 642.5371, 651.0383, 312.112878);

assignValue(pt[301], 643.5371, 652.0383, 313.112878);

assignValue(pt[302], 644.5371, 653.0383, 314.112878);

assignValue(pt[303], 645.5371, 654.0383, 315.112878);

assignValue(pt[304], 646.5371, 655.0383, 316.112878);

assignValue(pt[305], 647.5371, 656.0383, 317.112878);

assignValue(pt[306], 648.5371, 657.0383, 318.112878);

assignValue(pt[307], 649.5371, 658.0383, 319.112878);

assignValue(pt[308], 650.5371, 659.0383, 320.112878);

assignValue(pt[309], 651.5371, 660.0383, 321.112878);

assignValue(pt[310], 652.5371, 661.0383, 322.112878);

assignValue(pt[311], 653.5371, 662.0383, 323.112878);

assignValue(pt[312], 654.5371, 663.0383, 324.112878);

assignValue(pt[313], 655.5371, 664.0383, 325.112878);

assignValue(pt[314], 656.5371, 665.0383, 326.112878);

assignValue(pt[315], 657.5371, 666.0383, 327.112878);

assignValue(pt[316], 658.5371, 667.0383, 328.112878);

assignValue(pt[317], 659.5371, 668.0383, 329.112878);

assignValue(pt[318], 660.5371, 669.0383, 330.112878);

assignValue(pt[319], 661.5371, 670.0383, 331.112878);

assignValue(pt[320], 662.5371, 671.0383, 332.112878);

assignValue(pt[321], 663.5371, 672.0383, 333.112878);

assignValue(pt[322], 664.5371, 673.0383, 334.112878);

assignValue(pt[323], 665.5371, 674.0383, 335.112878);

assignValue(pt[324], 666.5371, 675.0383, 336.112878);

assignValue(pt[325], 667.5371, 676.0383, 337.112878);

assignValue(pt[326], 668.5371, 677.0383, 338.112878);

assignValue(pt[327], 669.5371, 678.0383, 339.112878);

assignValue(pt[328], 670.5371, 679.0383, 340.112878);

assignValue(pt[329], 671.5371, 680.0383, 341.112878);

assignValue(pt[330], 672.5371, 681.0383, 342.112878);

assignValue(pt[331], 673.5371, 682.0383, 343.112878);

assignValue(pt[332], 674.5371, 683.0383, 344.112878);

assignValue(pt[333], 675.5371, 684.0383, 345.112878);

assignValue(pt[334], 676.5371, 685.0383, 346.112878);

assignValue(pt[335], 677.5371, 686.0383, 347.112878);

assignValue(pt[336], 678.5371, 687.0383, 348.112878);

assignValue(pt[337], 679.5371, 688.0383, 349.112878);

assignValue(pt[338], 680.5371, 689.0383, 350.112878);

assignValue(pt[339], 681.5371, 690.0383, 351.112878);

assignValue(pt[340], 682.5371, 691.0383, 352.112878);

assignValue(pt[341], 683.5371, 692.0383, 353.112878);

assignValue(pt[342], 684.5371, 693.0383, 354.112878);

assignValue(pt[343], 685.5371, 694.0383, 355.112878);

assignValue(pt[344], 686.5371, 695.0383, 356.112878);

assignValue(pt[345], 687.5371, 696.0383, 357.112878);

assignValue(pt[346], 688.5371, 697.0383, 358.112878);

assignValue(pt[347], 689.5371, 698.0383, 359.112878);

assignValue(pt[348], 690.5371, 699.0383, 360.112878);

assignValue(pt[349], 691.5371, 700.0383, 361.112878);

assignValue(pt[350], 692.5371, 701.0383, 362.112878);

assignValue(pt[351], 693.5371, 702.0383, 363.112878);

assignValue(pt[352], 694.5371, 703.0383, 364.112878);

assignValue(pt[353], 695.5371, 704.0383, 365.112878);

assignValue(pt[354], 696.5371, 705.0383, 366.112878);

assignValue(pt[355], 697.5371, 706.0383, 367.112878);

assignValue(pt[356], 698.5371, 707.0383, 368.112878);

assignValue(pt[357], 699.5371, 708.0383, 369.112878);

assignValue(pt[358], 700.5371, 709.0383, 370.112878);

assignValue(pt[359], 701.5371, 710.0383, 371.112878);

assignValue(pt[360], 702.5371, 711.0383, 372.112878);

assignValue(pt[361], 703.5371, 712.0383, 373.112878);

assignValue(pt[362], 704.5371, 713.0383, 374.112878);

assignValue(pt[363], 705.5371, 714.0383, 375.112878);

assignValue(pt[364], 706.5371, 715.0383, 376.112878);

assignValue(pt[365], 707.5371, 716.0383, 377.112878);

assignValue(pt[366], 708.5371, 717.0383, 378.112878);

assignValue(pt[367], 709.5371, 718.0383, 379.112878);

assignValue(pt[368], 710.5371, 719.0383, 380.112878);

assignValue(pt[369], 711.5371, 720.0383, 381.112878);

assignValue(pt[370], 712.5371, 721.0383, 382.112878);

assignValue(pt[371], 713.5371, 722.0383, 383.112878);

assignValue(pt[372], 714.5371, 723.0383, 384.112878);

assignValue(pt[373], 715.5371, 724.0383, 385.112878);

assignValue(pt[374], 716.5371, 725.0383, 386.112878);

assignValue(pt[375], 717.5371, 726.0383, 387.112878);

assignValue(pt[376], 718.5371, 727.0383, 388.112878);

assignValue(pt[377], 719.5371, 728.0383, 389.112878);

assignValue(pt[378], 720.5371, 729.0383, 390.112878);

assignValue(pt[379], 721.5371, 730.0383, 391.112878);

assignValue(pt[380], 722.5371, 731.0383, 392.112878);

assignValue(pt[381], 723.5371, 732.0383, 393.112878);

assignValue(pt[382], 724.5371, 733.0383, 394.112878);

assignValue(pt[383], 725.5371, 734.0383, 395.112878);

assignValue(pt[384], 726.5371, 735.0383, 396.112878);

assignValue(pt[385], 727.5371, 736.0383, 397.112878);

assignValue(pt[386], 728.5371, 737.0383, 398.112878);

assignValue(pt[387], 729.5371, 738.0383, 399.112878);

assignValue(pt[388], 730.5371, 739.0383, 400.112878);

assignValue(pt[389], 731.5371, 740.0383, 401.112878);

assignValue(pt[390], 732.5371, 741.0383, 402.112878);

assignValue(pt[391], 733.5371, 742.0383, 403.112878);

assignValue(pt[392], 734.5371, 743.0383, 404.112878);

assignValue(pt[393], 735.5371, 744.0383, 405.112878);

assignValue(pt[394], 736.5371, 745.0383, 406.112878);

assignValue(pt[395], 737.5371, 746.0383, 407.112878);

assignValue(pt[396], 738.5371, 747.0383, 408.112878);

assignValue(pt[397], 739.5371, 748.0383, 409.112878);

assignValue(pt[398], 740.5371, 749.0383, 410.112878);

assignValue(pt[399], 741.5371, 750.0383, 411.112878);

assignValue(pt[400], 742.5371, 751.0383, 412.112878);

assignValue(pt[401], 743.5371, 752.0383, 413.112878);

assignValue(pt[402], 744.5371, 753.0383, 414.112878);

assignValue(pt[403], 745.5371, 754.0383, 415.112878);

assignValue(pt[404], 746.5371, 755.0383, 416.112878);

assignValue(pt[405], 747.5371, 756.0383, 417.112878);

assignValue(pt[406], 748.5371, 757.0383, 418.112878);

assignValue(pt[407], 749.5371, 758.0383, 419.112878);

assignValue(pt[408], 750.5371, 759.0383, 420.112878);

assignValue(pt[409], 751.5371, 760.0383, 421.112878);

assignValue(pt[410], 752.5371, 761.0383, 422.112878);

assignValue(pt[411], 753.5371, 762.0383, 423.112878);

assignValue(pt[412], 754.5371, 763.0383, 424.112878);

assignValue(pt[413], 755.5371, 764.0383, 425.112878);

assignValue(pt[414], 756.5371, 765.0383, 426.112878);

assignValue(pt[415], 757.5371, 766.0383, 427.112878);

assignValue(pt[416], 758.5371, 767.0383, 428.112878);

assignValue(pt[417], 759.5371, 768.0383, 429.112878);

assignValue(pt[418], 760.5371, 769.0383, 430.112878);

assignValue(pt[419], 761.5371, 770.0383, 431.112878);

assignValue(pt[420], 762.5371, 771.0383, 432.112878);

assignValue(pt[421], 763.5371, 772.0383, 433.112878);

assignValue(pt[422], 764.5371, 773.0383, 434.112878);

assignValue(pt[423], 765.5371, 774.0383, 435.112878);

assignValue(pt[424], 766.5371, 775.0383, 436.112878);

assignValue(pt[425], 767.5371, 776.0383, 437.112878);

assignValue(pt[426], 768.5371, 777.0383, 438.112878);

assignValue(pt[427], 769.5371, 778.0383, 439.112878);

assignValue(pt[428], 770.5371, 779.0383, 440.112878);

assignValue(pt[429], 771.5371, 780.0383, 441.112878);

assignValue(pt[430], 772.5371, 781.0383, 442.112878);

assignValue(pt[431], 773.5371, 782.0383, 443.112878);

assignValue(pt[432], 774.5371, 783.0383, 444.112878);

assignValue(pt[433], 775.5371, 784.0383, 445.112878);

assignValue(pt[434], 776.5371, 785.0383, 446.112878);

assignValue(pt[435], 777.5371, 786.0383, 447.112878);

assignValue(pt[436], 778.5371, 787.0383, 448.112878);

assignValue(pt[437], 779.5371, 788.0383, 449.112878);

assignValue(pt[438], 780.5371, 789.0383, 450.112878);

assignValue(pt[439], 781.5371, 790.0383, 451.112878);

assignValue(pt[440], 782.5371, 791.0383, 452.112878);

assignValue(pt[441], 783.5371, 792.0383, 453.112878);

assignValue(pt[442], 784.5371, 793.0383, 454.112878);

assignValue(pt[443], 785.5371, 794.0383, 455.112878);

assignValue(pt[444], 786.5371, 795.0383, 456.112878);

assignValue(pt[445], 787.5371, 796.0383, 457.112878);

assignValue(pt[446], 788.5371, 797.0383, 458.112878);

assignValue(pt[447], 789.5371, 798.0383, 459.112878);

assignValue(pt[448], 790.5371, 799.0383, 460.112878);

assignValue(pt[449], 791.5371, 800.0383, 461.112878);

assignValue(pt[450], 792.5371, 801.0383, 462.112878);

assignValue(pt[451], 793.5371, 802.0383, 463.112878);

assignValue(pt[452], 794.5371, 803.0383, 464.112878);

assignValue(pt[453], 795.5371, 804.0383, 465.112878);

assignValue(pt[454], 796.5371, 805.0383, 466.112878);

assignValue(pt[455], 797.5371, 806.0383, 467.112878);

assignValue(pt[456], 798.5371, 807.0383, 468.112878);

assignValue(pt[457], 799.5371, 808.0383, 469.112878);

assignValue(pt[458], 800.5371, 809.0383, 470.112878);

assignValue(pt[459], 801.5371, 810.0383, 471.112878);

assignValue(pt[460], 802.5371, 811.0383, 472.112878);

assignValue(pt[461], 803.5371, 812.0383, 473.112878);

assignValue(pt[462], 804.5371, 813.0383, 474.112878);

assignValue(pt[463], 805.5371, 814.0383, 475.112878);

assignValue(pt[464], 806.5371, 815.0383, 476.112878);

assignValue(pt[465], 807.5371, 816.0383, 477.112878);

assignValue(pt[466], 808.5371, 817.0383, 478.112878);

assignValue(pt[467], 809.5371, 818.0383, 479.112878);

assignValue(pt[468], 810.5371, 819.0383, 480.112878);

assignValue(pt[469], 811.5371, 820.0383, 481.112878);

assignValue(pt[470], 812.5371, 821.0383, 482.112878);

assignValue(pt[471], 813.5371, 822.0383, 483.112878);

assignValue(pt[472], 814.5371, 823.0383, 484.112878);

assignValue(pt[473], 815.5371, 824.0383, 485.112878);

assignValue(pt[474], 816.5371, 825.0383, 486.112878);

assignValue(pt[475], 817.5371, 826.0383, 487.112878);

assignValue(pt[476], 818.5371, 827.0383, 488.112878);

assignValue(pt[477], 819.5371, 828.0383, 489.112878);

assignValue(pt[478], 820.5371, 829.0383, 490.112878);

assignValue(pt[479], 821.5371, 830.0383, 491.112878);

assignValue(pt[480], 822.5371, 831.0383, 492.112878);

assignValue(pt[481], 823.5371, 832.0383, 493.112878);

assignValue(pt[482], 824.5371, 833.0383, 494.112878);

assignValue(pt[483], 825.5371, 834.0383, 495.112878);

assignValue(pt[484], 826.5371, 835.0383, 496.112878);

assignValue(pt[485], 827.5371, 836.0383, 497.112878);

assignValue(pt[486], 828.5371, 837.0383, 498.112878);

assignValue(pt[487], 829.5371, 838.0383, 499.112878);

assignValue(pt[488], 830.5371, 839.0383, 500.112878);

assignValue(pt[489], 831.5371, 840.0383, 501.112878);

assignValue(pt[490], 832.5371, 841.0383, 502.112878);

assignValue(pt[491], 833.5371, 842.0383, 503.112878);

assignValue(pt[492], 834.5371, 843.0383, 504.112878);

assignValue(pt[493], 835.5371, 844.0383, 505.112878);

assignValue(pt[494], 836.5371, 845.0383, 506.112878);

assignValue(pt[495], 837.5371, 846.0383, 507.112878);

assignValue(pt[496], 838.5371, 847.0383, 508.112878);

assignValue(pt[497], 839.5371, 848.0383, 509.112878);

assignValue(pt[498], 840.5371, 849.0383, 510.112878);

assignValue(pt[499], 841.5371, 850.0383, 511.112878);

assignValue(pt[500], 842.5371, 851.0383, 512.112878);

assignValue(pt[501], 843.5371, 852.0383, 513.112878);

assignValue(pt[502], 844.5371, 853.0383, 514.112878);

assignValue(pt[503], 845.5371, 854.0383, 515.112878);

assignValue(pt[504], 846.5371, 855.0383, 516.112878);

assignValue(pt[505], 847.5371, 856.0383, 517.112878);

assignValue(pt[506], 848.5371, 857.0383, 518.112878);

assignValue(pt[507], 849.5371, 858.0383, 519.112878);

assignValue(pt[508], 850.5371, 859.0383, 520.112878);

assignValue(pt[509], 851.5371, 860.0383, 521.112878);

assignValue(pt[510], 852.5371, 861.0383, 522.112878);

assignValue(pt[511], 853.5371, 862.0383, 523.112878);

assignValue(pt[512], 854.5371, 863.0383, 524.112878);

assignValue(pt[513], 855.5371, 864.0383, 525.112878);

assignValue(pt[514], 856.5371, 865.0383, 526.112878);

assignValue(pt[515], 857.5371, 866.0383, 527.112878);

assignValue(pt[516], 858.5371, 867.0383, 528.112878);

assignValue(pt[517], 859.5371, 868.0383, 529.112878);

assignValue(pt[518], 860.5371, 869.0383, 530.112878);

assignValue(pt[519], 861.5371, 870.0383, 531.112878);

assignValue(pt[520], 862.5371, 871.0383, 532.112878);

assignValue(pt[521], 863.5371, 872.0383, 533.112878);

assignValue(pt[522], 864.5371, 873.0383, 534.112878);

assignValue(pt[523], 865.5371, 874.0383, 535.112878);

assignValue(pt[524], 866.5371, 875.0383, 536.112878);

assignValue(pt[525], 867.5371, 876.0383, 537.112878);

assignValue(pt[526], 868.5371, 877.0383, 538.112878);

assignValue(pt[527], 869.5371, 878.0383, 539.112878);

assignValue(pt[528], 870.5371, 879.0383, 540.112878);

assignValue(pt[529], 871.5371, 880.0383, 541.112878);

assignValue(pt[530], 872.5371, 881.0383, 542.112878);

assignValue(pt[531], 873.5371, 882.0383, 543.112878);

assignValue(pt[532], 874.5371, 883.0383, 544.112878);

assignValue(pt[533], 875.5371, 884.0383, 545.112878);

assignValue(pt[534], 876.5371, 885.0383, 546.112878);

assignValue(pt[535], 877.5371, 886.0383, 547.112878);

assignValue(pt[536], 878.5371, 887.0383, 548.112878);

assignValue(pt[537], 879.5371, 888.0383, 549.112878);

assignValue(pt[538], 880.5371, 889.0383, 550.112878);

assignValue(pt[539], 881.5371, 890.0383, 551.112878);

assignValue(pt[540], 882.5371, 891.0383, 552.112878);

assignValue(pt[541], 883.5371, 892.0383, 553.112878);

assignValue(pt[542], 884.5371, 893.0383, 554.112878);

assignValue(pt[543], 885.5371, 894.0383, 555.112878);

assignValue(pt[544], 886.5371, 895.0383, 556.112878);

assignValue(pt[545], 887.5371, 896.0383, 557.112878);

assignValue(pt[546], 888.5371, 897.0383, 558.112878);

assignValue(pt[547], 889.5371, 898.0383, 559.112878);

assignValue(pt[548], 890.5371, 899.0383, 560.112878);

assignValue(pt[549], 891.5371, 900.0383, 561.112878);

assignValue(pt[550], 892.5371, 901.0383, 562.112878);

assignValue(pt[551], 893.5371, 902.0383, 563.112878);

assignValue(pt[552], 894.5371, 903.0383, 564.112878);

assignValue(pt[553], 895.5371, 904.0383, 565.112878);

assignValue(pt[554], 896.5371, 905.0383, 566.112878);

assignValue(pt[555], 897.5371, 906.0383, 567.112878);

assignValue(pt[556], 898.5371, 907.0383, 568.112878);

assignValue(pt[557], 899.5371, 908.0383, 569.112878);

assignValue(pt[558], 900.5371, 909.0383, 570.112878);

assignValue(pt[559], 901.5371, 910.0383, 571.112878);

assignValue(pt[560], 902.5371, 911.0383, 572.112878);

assignValue(pt[561], 903.5371, 912.0383, 573.112878);

assignValue(pt[562], 904.5371, 913.0383, 574.112878);

assignValue(pt[563], 905.5371, 914.0383, 575.112878);

assignValue(pt[564], 906.5371, 915.0383, 576.112878);

assignValue(pt[565], 907.5371, 916.0383, 577.112878);

assignValue(pt[566], 908.5371, 917.0383, 578.112878);

assignValue(pt[567], 909.5371, 918.0383, 579.112878);

assignValue(pt[568], 910.5371, 919.0383, 580.112878);

assignValue(pt[569], 911.5371, 920.0383, 581.112878);

assignValue(pt[570], 912.5371, 921.0383, 582.112878);

assignValue(pt[571], 913.5371, 922.0383, 583.112878);

assignValue(pt[572], 914.5371, 923.0383, 584.112878);

assignValue(pt[573], 915.5371, 924.0383, 585.112878);

assignValue(pt[574], 916.5371, 925.0383, 586.112878);

assignValue(pt[575], 917.5371, 926.0383, 587.112878);

assignValue(pt[576], 918.5371, 927.0383, 588.112878);

assignValue(pt[577], 919.5371, 928.0383, 589.112878);

assignValue(pt[578], 920.5371, 929.0383, 590.112878);

assignValue(pt[579], 921.5371, 930.0383, 591.112878);

assignValue(pt[580], 922.5371, 931.0383, 592.112878);

assignValue(pt[581], 923.5371, 932.0383, 593.112878);

assignValue(pt[582], 924.5371, 933.0383, 594.112878);

assignValue(pt[583], 925.5371, 934.0383, 595.112878);

assignValue(pt[584], 926.5371, 935.0383, 596.112878);

assignValue(pt[585], 927.5371, 936.0383, 597.112878);

assignValue(pt[586], 928.5371, 937.0383, 598.112878);

assignValue(pt[587], 929.5371, 938.0383, 599.112878);

assignValue(pt[588], 930.5371, 939.0383, 600.112878);

assignValue(pt[589], 931.5371, 940.0383, 601.112878);

assignValue(pt[590], 932.5371, 941.0383, 602.112878);

assignValue(pt[591], 933.5371, 942.0383, 603.112878);

assignValue(pt[592], 934.5371, 943.0383, 604.112878);

assignValue(pt[593], 935.5371, 944.0383, 605.112878);

assignValue(pt[594], 936.5371, 945.0383, 606.112878);

assignValue(pt[595], 937.5371, 946.0383, 607.112878);

assignValue(pt[596], 938.5371, 947.0383, 608.112878);

assignValue(pt[597], 939.5371, 948.0383, 609.112878);

assignValue(pt[598], 940.5371, 949.0383, 610.112878);

assignValue(pt[599], 941.5371, 950.0383, 611.112878);

assignValue(pt[600], 942.5371, 951.0383, 612.112878);

assignValue(pt[601], 943.5371, 952.0383, 613.112878);

assignValue(pt[602], 944.5371, 953.0383, 614.112878);

assignValue(pt[603], 945.5371, 954.0383, 615.112878);

assignValue(pt[604], 946.5371, 955.0383, 616.112878);

assignValue(pt[605], 947.5371, 956.0383, 617.112878);

assignValue(pt[606], 948.5371, 957.0383, 618.112878);

assignValue(pt[607], 949.5371, 958.0383, 619.112878);

assignValue(pt[608], 950.5371, 959.0383, 620.112878);

assignValue(pt[609], 951.5371, 960.0383, 621.112878);

assignValue(pt[610], 952.5371, 961.0383, 622.112878);

assignValue(pt[611], 953.5371, 962.0383, 623.112878);

assignValue(pt[612], 954.5371, 963.0383, 624.112878);

assignValue(pt[613], 955.5371, 964.0383, 625.112878);

assignValue(pt[614], 956.5371, 965.0383, 626.112878);

assignValue(pt[615], 957.5371, 966.0383, 627.112878);

assignValue(pt[616], 958.5371, 967.0383, 628.112878);

assignValue(pt[617], 959.5371, 968.0383, 629.112878);

assignValue(pt[618], 960.5371, 969.0383, 630.112878);

assignValue(pt[619], 961.5371, 970.0383, 631.112878);

assignValue(pt[620], 962.5371, 971.0383, 632.112878);

assignValue(pt[621], 963.5371, 972.0383, 633.112878);

assignValue(pt[622], 964.5371, 973.0383, 634.112878);

assignValue(pt[623], 965.5371, 974.0383, 635.112878);

assignValue(pt[624], 966.5371, 975.0383, 636.112878);

assignValue(pt[625], 967.5371, 976.0383, 637.112878);

assignValue(pt[626], 968.5371, 977.0383, 638.112878);

assignValue(pt[627], 969.5371, 978.0383, 639.112878);

assignValue(pt[628], 970.5371, 979.0383, 640.112878);

assignValue(pt[629], 971.5371, 980.0383, 641.112878);

assignValue(pt[630], 972.5371, 981.0383, 642.112878);

assignValue(pt[631], 973.5371, 982.0383, 643.112878);

assignValue(pt[632], 974.5371, 983.0383, 644.112878);

assignValue(pt[633], 975.5371, 984.0383, 645.112878);

assignValue(pt[634], 976.5371, 985.0383, 646.112878);

assignValue(pt[635], 977.5371, 986.0383, 647.112878);

assignValue(pt[636], 978.5371, 987.0383, 648.112878);

assignValue(pt[637], 979.5371, 988.0383, 649.112878);

assignValue(pt[638], 980.5371, 989.0383, 650.112878);

assignValue(pt[639], 981.5371, 990.0383, 651.112878);

assignValue(pt[640], 982.5371, 991.0383, 652.112878);

assignValue(pt[641], 983.5371, 992.0383, 653.112878);

assignValue(pt[642], 984.5371, 993.0383, 654.112878);

assignValue(pt[643], 985.5371, 994.0383, 655.112878);

assignValue(pt[644], 986.5371, 995.0383, 656.112878);

assignValue(pt[645], 987.5371, 996.0383, 657.112878);

assignValue(pt[646], 988.5371, 997.0383, 658.112878);

assignValue(pt[647], 989.5371, 998.0383, 659.112878);

assignValue(pt[648], 990.5371, 999.0383, 660.112878);

assignValue(pt[649], 991.5371, 1000.0383, 661.112878);

assignValue(pt[650], 992.5371, 1001.0383, 662.112878);

assignValue(pt[651], 993.5371, 1002.0383, 663.112878);

assignValue(pt[652], 994.5371, 1003.0383, 664.112878);

assignValue(pt[653], 995.5371, 1004.0383, 665.112878);

assignValue(pt[654], 996.5371, 1005.0383, 666.112878);

assignValue(pt[655], 997.5371, 1006.0383, 667.112878);

assignValue(pt[656], 998.5371, 1007.0383, 668.112878);

assignValue(pt[657], 999.5371, 1008.0383, 669.112878);

assignValue(pt[658], 1000.5371, 1009.0383, 670.112878);

assignValue(pt[659], 1001.5371, 1010.0383, 671.112878);

assignValue(pt[660], 1002.5371, 1011.0383, 672.112878);

assignValue(pt[661], 1003.5371, 1012.0383, 673.112878);

assignValue(pt[662], 1004.5371, 1013.0383, 674.112878);

assignValue(pt[663], 1005.5371, 1014.0383, 675.112878);

assignValue(pt[664], 1006.5371, 1015.0383, 676.112878);

assignValue(pt[665], 1007.5371, 1016.0383, 677.112878);

assignValue(pt[666], 1008.5371, 1017.0383, 678.112878);

assignValue(pt[667], 1009.5371, 1018.0383, 679.112878);

assignValue(pt[668], 1010.5371, 1019.0383, 680.112878);

assignValue(pt[669], 1011.5371, 1020.0383, 681.112878);

assignValue(pt[670], 1012.5371, 1021.0383, 682.112878);

assignValue(pt[671], 1013.5371, 1022.0383, 683.112878);

assignValue(pt[672], 1014.5371, 1023.0383, 684.112878);

assignValue(pt[673], 1015.5371, 1024.0383, 685.112878);

assignValue(pt[674], 1016.5371, 1025.0383, 686.112878);

assignValue(pt[675], 1017.5371, 1026.0383, 687.112878);

assignValue(pt[676], 1018.5371, 1027.0383, 688.112878);

assignValue(pt[677], 1019.5371, 1028.0383, 689.112878);

assignValue(pt[678], 1020.5371, 1029.0383, 690.112878);

assignValue(pt[679], 1021.5371, 1030.0383, 691.112878);

assignValue(pt[680], 1022.5371, 1031.0383, 692.112878);

assignValue(pt[681], 1023.5371, 1032.0383, 693.112878);

assignValue(pt[682], 1024.5371, 1033.0383, 694.112878);

assignValue(pt[683], 1025.5371, 1034.0383, 695.112878);

assignValue(pt[684], 1026.5371, 1035.0383, 696.112878);

assignValue(pt[685], 1027.5371, 1036.0383, 697.112878);

assignValue(pt[686], 1028.5371, 1037.0383, 698.112878);

assignValue(pt[687], 1029.5371, 1038.0383, 699.112878);

assignValue(pt[688], 1030.5371, 1039.0383, 700.112878);

assignValue(pt[689], 1031.5371, 1040.0383, 701.112878);

assignValue(pt[690], 1032.5371, 1041.0383, 702.112878);

assignValue(pt[691], 1033.5371, 1042.0383, 703.112878);

assignValue(pt[692], 1034.5371, 1043.0383, 704.112878);

assignValue(pt[693], 1035.5371, 1044.0383, 705.112878);

assignValue(pt[694], 1036.5371, 1045.0383, 706.112878);

assignValue(pt[695], 1037.5371, 1046.0383, 707.112878);

assignValue(pt[696], 1038.5371, 1047.0383, 708.112878);

assignValue(pt[697], 1039.5371, 1048.0383, 709.112878);

assignValue(pt[698], 1040.5371, 1049.0383, 710.112878);

assignValue(pt[699], 1041.5371, 1050.0383, 711.112878);

assignValue(pt[700], 1042.5371, 1051.0383, 712.112878);

assignValue(pt[701], 1043.5371, 1052.0383, 713.112878);

assignValue(pt[702], 1044.5371, 1053.0383, 714.112878);

assignValue(pt[703], 1045.5371, 1054.0383, 715.112878);

assignValue(pt[704], 1046.5371, 1055.0383, 716.112878);

assignValue(pt[705], 1047.5371, 1056.0383, 717.112878);

assignValue(pt[706], 1048.5371, 1057.0383, 718.112878);

assignValue(pt[707], 1049.5371, 1058.0383, 719.112878);

assignValue(pt[708], 1050.5371, 1059.0383, 720.112878);

assignValue(pt[709], 1051.5371, 1060.0383, 721.112878);

assignValue(pt[710], 1052.5371, 1061.0383, 722.112878);

assignValue(pt[711], 1053.5371, 1062.0383, 723.112878);

assignValue(pt[712], 1054.5371, 1063.0383, 724.112878);

assignValue(pt[713], 1055.5371, 1064.0383, 725.112878);

assignValue(pt[714], 1056.5371, 1065.0383, 726.112878);

assignValue(pt[715], 1057.5371, 1066.0383, 727.112878);

assignValue(pt[716], 1058.5371, 1067.0383, 728.112878);

assignValue(pt[717], 1059.5371, 1068.0383, 729.112878);

assignValue(pt[718], 1060.5371, 1069.0383, 730.112878);

assignValue(pt[719], 1061.5371, 1070.0383, 731.112878);

assignValue(pt[720], 1062.5371, 1071.0383, 732.112878);

assignValue(pt[721], 1063.5371, 1072.0383, 733.112878);

assignValue(pt[722], 1064.5371, 1073.0383, 734.112878);

assignValue(pt[723], 1065.5371, 1074.0383, 735.112878);

assignValue(pt[724], 1066.5371, 1075.0383, 736.112878);

assignValue(pt[725], 1067.5371, 1076.0383, 737.112878);

assignValue(pt[726], 1068.5371, 1077.0383, 738.112878);

assignValue(pt[727], 1069.5371, 1078.0383, 739.112878);

assignValue(pt[728], 1070.5371, 1079.0383, 740.112878);

assignValue(pt[729], 1071.5371, 1080.0383, 741.112878);

assignValue(pt[730], 1072.5371, 1081.0383, 742.112878);

assignValue(pt[731], 1073.5371, 1082.0383, 743.112878);

assignValue(pt[732], 1074.5371, 1083.0383, 744.112878);

assignValue(pt[733], 1075.5371, 1084.0383, 745.112878);

assignValue(pt[734], 1076.5371, 1085.0383, 746.112878);

assignValue(pt[735], 1077.5371, 1086.0383, 747.112878);

assignValue(pt[736], 1078.5371, 1087.0383, 748.112878);

assignValue(pt[737], 1079.5371, 1088.0383, 749.112878);

assignValue(pt[738], 1080.5371, 1089.0383, 750.112878);

assignValue(pt[739], 1081.5371, 1090.0383, 751.112878);

assignValue(pt[740], 1082.5371, 1091.0383, 752.112878);

assignValue(pt[741], 1083.5371, 1092.0383, 753.112878);

assignValue(pt[742], 1084.5371, 1093.0383, 754.112878);

assignValue(pt[743], 1085.5371, 1094.0383, 755.112878);

assignValue(pt[744], 1086.5371, 1095.0383, 756.112878);

assignValue(pt[745], 1087.5371, 1096.0383, 757.112878);

assignValue(pt[746], 1088.5371, 1097.0383, 758.112878);

assignValue(pt[747], 1089.5371, 1098.0383, 759.112878);

assignValue(pt[748], 1090.5371, 1099.0383, 760.112878);

assignValue(pt[749], 1091.5371, 1100.0383, 761.112878);

assignValue(pt[750], 1092.5371, 1101.0383, 762.112878);

assignValue(pt[751], 1093.5371, 1102.0383, 763.112878);

assignValue(pt[752], 1094.5371, 1103.0383, 764.112878);

assignValue(pt[753], 1095.5371, 1104.0383, 765.112878);

assignValue(pt[754], 1096.5371, 1105.0383, 766.112878);

assignValue(pt[755], 1097.5371, 1106.0383, 767.112878);

assignValue(pt[756], 1098.5371, 1107.0383, 768.112878);

assignValue(pt[757], 1099.5371, 1108.0383, 769.112878);

assignValue(pt[758], 1100.5371, 1109.0383, 770.112878);

assignValue(pt[759], 1101.5371, 1110.0383, 771.112878);

assignValue(pt[760], 1102.5371, 1111.0383, 772.112878);

assignValue(pt[761], 1103.5371, 1112.0383, 773.112878);

assignValue(pt[762], 1104.5371, 1113.0383, 774.112878);

assignValue(pt[763], 1105.5371, 1114.0383, 775.112878);

assignValue(pt[764], 1106.5371, 1115.0383, 776.112878);

assignValue(pt[765], 1107.5371, 1116.0383, 777.112878);

assignValue(pt[766], 1108.5371, 1117.0383, 778.112878);

assignValue(pt[767], 1109.5371, 1118.0383, 779.112878);

assignValue(pt[768], 1110.5371, 1119.0383, 780.112878);

assignValue(pt[769], 1111.5371, 1120.0383, 781.112878);

assignValue(pt[770], 1112.5371, 1121.0383, 782.112878);

assignValue(pt[771], 1113.5371, 1122.0383, 783.112878);

assignValue(pt[772], 1114.5371, 1123.0383, 784.112878);

assignValue(pt[773], 1115.5371, 1124.0383, 785.112878);

assignValue(pt[774], 1116.5371, 1125.0383, 786.112878);

assignValue(pt[775], 1117.5371, 1126.0383, 787.112878);

assignValue(pt[776], 1118.5371, 1127.0383, 788.112878);

assignValue(pt[777], 1119.5371, 1128.0383, 789.112878);

assignValue(pt[778], 1120.5371, 1129.0383, 790.112878);

assignValue(pt[779], 1121.5371, 1130.0383, 791.112878);

assignValue(pt[780], 1122.5371, 1131.0383, 792.112878);

assignValue(pt[781], 1123.5371, 1132.0383, 793.112878);

assignValue(pt[782], 1124.5371, 1133.0383, 794.112878);

assignValue(pt[783], 1125.5371, 1134.0383, 795.112878);

assignValue(pt[784], 1126.5371, 1135.0383, 796.112878);

assignValue(pt[785], 1127.5371, 1136.0383, 797.112878);

assignValue(pt[786], 1128.5371, 1137.0383, 798.112878);

assignValue(pt[787], 1129.5371, 1138.0383, 799.112878);

assignValue(pt[788], 1130.5371, 1139.0383, 800.112878);

assignValue(pt[789], 1131.5371, 1140.0383, 801.112878);

assignValue(pt[790], 1132.5371, 1141.0383, 802.112878);

assignValue(pt[791], 1133.5371, 1142.0383, 803.112878);

assignValue(pt[792], 1134.5371, 1143.0383, 804.112878);

assignValue(pt[793], 1135.5371, 1144.0383, 805.112878);

assignValue(pt[794], 1136.5371, 1145.0383, 806.112878);

assignValue(pt[795], 1137.5371, 1146.0383, 807.112878);

assignValue(pt[796], 1138.5371, 1147.0383, 808.112878);

assignValue(pt[797], 1139.5371, 1148.0383, 809.112878);

assignValue(pt[798], 1140.5371, 1149.0383, 810.112878);

assignValue(pt[799], 1141.5371, 1150.0383, 811.112878);

assignValue(pt[800], 1142.5371, 1151.0383, 812.112878);

assignValue(pt[801], 1143.5371, 1152.0383, 813.112878);

assignValue(pt[802], 1144.5371, 1153.0383, 814.112878);

assignValue(pt[803], 1145.5371, 1154.0383, 815.112878);

assignValue(pt[804], 1146.5371, 1155.0383, 816.112878);

assignValue(pt[805], 1147.5371, 1156.0383, 817.112878);

assignValue(pt[806], 1148.5371, 1157.0383, 818.112878);

assignValue(pt[807], 1149.5371, 1158.0383, 819.112878);

assignValue(pt[808], 1150.5371, 1159.0383, 820.112878);

assignValue(pt[809], 1151.5371, 1160.0383, 821.112878);

assignValue(pt[810], 1152.5371, 1161.0383, 822.112878);

assignValue(pt[811], 1153.5371, 1162.0383, 823.112878);

assignValue(pt[812], 1154.5371, 1163.0383, 824.112878);

assignValue(pt[813], 1155.5371, 1164.0383, 825.112878);

assignValue(pt[814], 1156.5371, 1165.0383, 826.112878);

assignValue(pt[815], 1157.5371, 1166.0383, 827.112878);

assignValue(pt[816], 1158.5371, 1167.0383, 828.112878);

assignValue(pt[817], 1159.5371, 1168.0383, 829.112878);

assignValue(pt[818], 1160.5371, 1169.0383, 830.112878);

assignValue(pt[819], 1161.5371, 1170.0383, 831.112878);

assignValue(pt[820], 1162.5371, 1171.0383, 832.112878);

assignValue(pt[821], 1163.5371, 1172.0383, 833.112878);

assignValue(pt[822], 1164.5371, 1173.0383, 834.112878);

assignValue(pt[823], 1165.5371, 1174.0383, 835.112878);

assignValue(pt[824], 1166.5371, 1175.0383, 836.112878);

assignValue(pt[825], 1167.5371, 1176.0383, 837.112878);

assignValue(pt[826], 1168.5371, 1177.0383, 838.112878);

assignValue(pt[827], 1169.5371, 1178.0383, 839.112878);

assignValue(pt[828], 1170.5371, 1179.0383, 840.112878);

assignValue(pt[829], 1171.5371, 1180.0383, 841.112878);

assignValue(pt[830], 1172.5371, 1181.0383, 842.112878);

assignValue(pt[831], 1173.5371, 1182.0383, 843.112878);

assignValue(pt[832], 1174.5371, 1183.0383, 844.112878);

assignValue(pt[833], 1175.5371, 1184.0383, 845.112878);

assignValue(pt[834], 1176.5371, 1185.0383, 846.112878);

assignValue(pt[835], 1177.5371, 1186.0383, 847.112878);

assignValue(pt[836], 1178.5371, 1187.0383, 848.112878);

assignValue(pt[837], 1179.5371, 1188.0383, 849.112878);

assignValue(pt[838], 1180.5371, 1189.0383, 850.112878);

assignValue(pt[839], 1181.5371, 1190.0383, 851.112878);

assignValue(pt[840], 1182.5371, 1191.0383, 852.112878);

assignValue(pt[841], 1183.5371, 1192.0383, 853.112878);

assignValue(pt[842], 1184.5371, 1193.0383, 854.112878);

assignValue(pt[843], 1185.5371, 1194.0383, 855.112878);

assignValue(pt[844], 1186.5371, 1195.0383, 856.112878);

assignValue(pt[845], 1187.5371, 1196.0383, 857.112878);

assignValue(pt[846], 1188.5371, 1197.0383, 858.112878);

assignValue(pt[847], 1189.5371, 1198.0383, 859.112878);

assignValue(pt[848], 1190.5371, 1199.0383, 860.112878);

assignValue(pt[849], 1191.5371, 1200.0383, 861.112878);

assignValue(pt[850], 1192.5371, 1201.0383, 862.112878);

assignValue(pt[851], 1193.5371, 1202.0383, 863.112878);

assignValue(pt[852], 1194.5371, 1203.0383, 864.112878);

assignValue(pt[853], 1195.5371, 1204.0383, 865.112878);

assignValue(pt[854], 1196.5371, 1205.0383, 866.112878);

assignValue(pt[855], 1197.5371, 1206.0383, 867.112878);

assignValue(pt[856], 1198.5371, 1207.0383, 868.112878);

assignValue(pt[857], 1199.5371, 1208.0383, 869.112878);

assignValue(pt[858], 1200.5371, 1209.0383, 870.112878);

assignValue(pt[859], 1201.5371, 1210.0383, 871.112878);

assignValue(pt[860], 1202.5371, 1211.0383, 872.112878);

assignValue(pt[861], 1203.5371, 1212.0383, 873.112878);

assignValue(pt[862], 1204.5371, 1213.0383, 874.112878);

assignValue(pt[863], 1205.5371, 1214.0383, 875.112878);

assignValue(pt[864], 1206.5371, 1215.0383, 876.112878);

assignValue(pt[865], 1207.5371, 1216.0383, 877.112878);

assignValue(pt[866], 1208.5371, 1217.0383, 878.112878);

assignValue(pt[867], 1209.5371, 1218.0383, 879.112878);

assignValue(pt[868], 1210.5371, 1219.0383, 880.112878);

assignValue(pt[869], 1211.5371, 1220.0383, 881.112878);

assignValue(pt[870], 1212.5371, 1221.0383, 882.112878);

assignValue(pt[871], 1213.5371, 1222.0383, 883.112878);

assignValue(pt[872], 1214.5371, 1223.0383, 884.112878);

assignValue(pt[873], 1215.5371, 1224.0383, 885.112878);

assignValue(pt[874], 1216.5371, 1225.0383, 886.112878);

assignValue(pt[875], 1217.5371, 1226.0383, 887.112878);

assignValue(pt[876], 1218.5371, 1227.0383, 888.112878);

assignValue(pt[877], 1219.5371, 1228.0383, 889.112878);

assignValue(pt[878], 1220.5371, 1229.0383, 890.112878);

assignValue(pt[879], 1221.5371, 1230.0383, 891.112878);

assignValue(pt[880], 1222.5371, 1231.0383, 892.112878);

assignValue(pt[881], 1223.5371, 1232.0383, 893.112878);

assignValue(pt[882], 1224.5371, 1233.0383, 894.112878);

assignValue(pt[883], 1225.5371, 1234.0383, 895.112878);

assignValue(pt[884], 1226.5371, 1235.0383, 896.112878);

assignValue(pt[885], 1227.5371, 1236.0383, 897.112878);

assignValue(pt[886], 1228.5371, 1237.0383, 898.112878);

assignValue(pt[887], 1229.5371, 1238.0383, 899.112878);

assignValue(pt[888], 1230.5371, 1239.0383, 900.112878);

assignValue(pt[889], 1231.5371, 1240.0383, 901.112878);

assignValue(pt[890], 1232.5371, 1241.0383, 902.112878);

assignValue(pt[891], 1233.5371, 1242.0383, 903.112878);

assignValue(pt[892], 1234.5371, 1243.0383, 904.112878);

assignValue(pt[893], 1235.5371, 1244.0383, 905.112878);

assignValue(pt[894], 1236.5371, 1245.0383, 906.112878);

assignValue(pt[895], 1237.5371, 1246.0383, 907.112878);

assignValue(pt[896], 1238.5371, 1247.0383, 908.112878);

assignValue(pt[897], 1239.5371, 1248.0383, 909.112878);

assignValue(pt[898], 1240.5371, 1249.0383, 910.112878);

assignValue(pt[899], 1241.5371, 1250.0383, 911.112878);

assignValue(pt[900], 1242.5371, 1251.0383, 912.112878);

assignValue(pt[901], 1243.5371, 1252.0383, 913.112878);

assignValue(pt[902], 1244.5371, 1253.0383, 914.112878);

assignValue(pt[903], 1245.5371, 1254.0383, 915.112878);

assignValue(pt[904], 1246.5371, 1255.0383, 916.112878);

assignValue(pt[905], 1247.5371, 1256.0383, 917.112878);

assignValue(pt[906], 1248.5371, 1257.0383, 918.112878);

assignValue(pt[907], 1249.5371, 1258.0383, 919.112878);

assignValue(pt[908], 1250.5371, 1259.0383, 920.112878);

assignValue(pt[909], 1251.5371, 1260.0383, 921.112878);

assignValue(pt[910], 1252.5371, 1261.0383, 922.112878);

assignValue(pt[911], 1253.5371, 1262.0383, 923.112878);

assignValue(pt[912], 1254.5371, 1263.0383, 924.112878);

assignValue(pt[913], 1255.5371, 1264.0383, 925.112878);

assignValue(pt[914], 1256.5371, 1265.0383, 926.112878);

assignValue(pt[915], 1257.5371, 1266.0383, 927.112878);

assignValue(pt[916], 1258.5371, 1267.0383, 928.112878);

assignValue(pt[917], 1259.5371, 1268.0383, 929.112878);

assignValue(pt[918], 1260.5371, 1269.0383, 930.112878);

assignValue(pt[919], 1261.5371, 1270.0383, 931.112878);

assignValue(pt[920], 1262.5371, 1271.0383, 932.112878);

assignValue(pt[921], 1263.5371, 1272.0383, 933.112878);

assignValue(pt[922], 1264.5371, 1273.0383, 934.112878);

assignValue(pt[923], 1265.5371, 1274.0383, 935.112878);

assignValue(pt[924], 1266.5371, 1275.0383, 936.112878);

assignValue(pt[925], 1267.5371, 1276.0383, 937.112878);

assignValue(pt[926], 1268.5371, 1277.0383, 938.112878);

assignValue(pt[927], 1269.5371, 1278.0383, 939.112878);

assignValue(pt[928], 1270.5371, 1279.0383, 940.112878);

assignValue(pt[929], 1271.5371, 1280.0383, 941.112878);

assignValue(pt[930], 1272.5371, 1281.0383, 942.112878);

assignValue(pt[931], 1273.5371, 1282.0383, 943.112878);

assignValue(pt[932], 1274.5371, 1283.0383, 944.112878);

assignValue(pt[933], 1275.5371, 1284.0383, 945.112878);

assignValue(pt[934], 1276.5371, 1285.0383, 946.112878);

assignValue(pt[935], 1277.5371, 1286.0383, 947.112878);

assignValue(pt[936], 1278.5371, 1287.0383, 948.112878);

assignValue(pt[937], 1279.5371, 1288.0383, 949.112878);

assignValue(pt[938], 1280.5371, 1289.0383, 950.112878);

assignValue(pt[939], 1281.5371, 1290.0383, 951.112878);

assignValue(pt[940], 1282.5371, 1291.0383, 952.112878);

assignValue(pt[941], 1283.5371, 1292.0383, 953.112878);

assignValue(pt[942], 1284.5371, 1293.0383, 954.112878);

assignValue(pt[943], 1285.5371, 1294.0383, 955.112878);

assignValue(pt[944], 1286.5371, 1295.0383, 956.112878);

assignValue(pt[945], 1287.5371, 1296.0383, 957.112878);

assignValue(pt[946], 1288.5371, 1297.0383, 958.112878);

assignValue(pt[947], 1289.5371, 1298.0383, 959.112878);

assignValue(pt[948], 1290.5371, 1299.0383, 960.112878);

assignValue(pt[949], 1291.5371, 1300.0383, 961.112878);

assignValue(pt[950], 1292.5371, 1301.0383, 962.112878);

assignValue(pt[951], 1293.5371, 1302.0383, 963.112878);

assignValue(pt[952], 1294.5371, 1303.0383, 964.112878);

assignValue(pt[953], 1295.5371, 1304.0383, 965.112878);

assignValue(pt[954], 1296.5371, 1305.0383, 966.112878);

assignValue(pt[955], 1297.5371, 1306.0383, 967.112878);

assignValue(pt[956], 1298.5371, 1307.0383, 968.112878);

assignValue(pt[957], 1299.5371, 1308.0383, 969.112878);

assignValue(pt[958], 1300.5371, 1309.0383, 970.112878);

assignValue(pt[959], 1301.5371, 1310.0383, 971.112878);

assignValue(pt[960], 1302.5371, 1311.0383, 972.112878);

assignValue(pt[961], 1303.5371, 1312.0383, 973.112878);

assignValue(pt[962], 1304.5371, 1313.0383, 974.112878);

assignValue(pt[963], 1305.5371, 1314.0383, 975.112878);

assignValue(pt[964], 1306.5371, 1315.0383, 976.112878);

assignValue(pt[965], 1307.5371, 1316.0383, 977.112878);

assignValue(pt[966], 1308.5371, 1317.0383, 978.112878);

assignValue(pt[967], 1309.5371, 1318.0383, 979.112878);

assignValue(pt[968], 1310.5371, 1319.0383, 980.112878);

assignValue(pt[969], 1311.5371, 1320.0383, 981.112878);

assignValue(pt[970], 1312.5371, 1321.0383, 982.112878);

assignValue(pt[971], 1313.5371, 1322.0383, 983.112878);

assignValue(pt[972], 1314.5371, 1323.0383, 984.112878);

assignValue(pt[973], 1315.5371, 1324.0383, 985.112878);

assignValue(pt[974], 1316.5371, 1325.0383, 986.112878);

assignValue(pt[975], 1317.5371, 1326.0383, 987.112878);

assignValue(pt[976], 1318.5371, 1327.0383, 988.112878);

assignValue(pt[977], 1319.5371, 1328.0383, 989.112878);

assignValue(pt[978], 1320.5371, 1329.0383, 990.112878);

assignValue(pt[979], 1321.5371, 1330.0383, 991.112878);

assignValue(pt[980], 1322.5371, 1331.0383, 992.112878);

assignValue(pt[981], 1323.5371, 1332.0383, 993.112878);

assignValue(pt[982], 1324.5371, 1333.0383, 994.112878);

assignValue(pt[983], 1325.5371, 1334.0383, 995.112878);

assignValue(pt[984], 1326.5371, 1335.0383, 996.112878);

assignValue(pt[985], 1327.5371, 1336.0383, 997.112878);

assignValue(pt[986], 1328.5371, 1337.0383, 998.112878);

assignValue(pt[987], 1329.5371, 1338.0383, 999.112878);

assignValue(pt[988], 1330.5371, 1339.0383, 1000.112878);

assignValue(pt[989], 1331.5371, 1340.0383, 1001.112878);

assignValue(pt[990], 1332.5371, 1341.0383, 1002.112878);

assignValue(pt[991], 1333.5371, 1342.0383, 1003.112878);

assignValue(pt[992], 1334.5371, 1343.0383, 1004.112878);

assignValue(pt[993], 1335.5371, 1344.0383, 1005.112878);

assignValue(pt[994], 1336.5371, 1345.0383, 1006.112878);

assignValue(pt[995], 1337.5371, 1346.0383, 1007.112878);

assignValue(pt[996], 1338.5371, 1347.0383, 1008.112878);

assignValue(pt[997], 1339.5371, 1348.0383, 1009.112878);

assignValue(pt[998], 1340.5371, 1349.0383, 1010.112878);

assignValue(pt[999], 1341.5371, 1350.0383, 1011.112878);

assignValue(pt[1000], 1342.5371, 1351.0383, 1012.112878);

assignValue(pt[1001], 1343.5371, 1352.0383, 1013.112878);

assignValue(pt[1002], 1344.5371, 1353.0383, 1014.112878);

assignValue(pt[1003], 1345.5371, 1354.0383, 1015.112878);

assignValue(pt[1004], 1346.5371, 1355.0383, 1016.112878);

assignValue(pt[1005], 1347.5371, 1356.0383, 1017.112878);

assignValue(pt[1006], 1348.5371, 1357.0383, 1018.112878);

assignValue(pt[1007], 1349.5371, 1358.0383, 1019.112878);

assignValue(pt[1008], 1350.5371, 1359.0383, 1020.112878);

assignValue(pt[1009], 1351.5371, 1360.0383, 1021.112878);

assignValue(pt[1010], 1352.5371, 1361.0383, 1022.112878);

assignValue(pt[1011], 1353.5371, 1362.0383, 1023.112878);

assignValue(pt[1012], 1354.5371, 1363.0383, 1024.112878);

assignValue(pt[1013], 1355.5371, 1364.0383, 1025.112878);

assignValue(pt[1014], 1356.5371, 1365.0383, 1026.112878);

assignValue(pt[1015], 1357.5371, 1366.0383, 1027.112878);

assignValue(pt[1016], 1358.5371, 1367.0383, 1028.112878);

assignValue(pt[1017], 1359.5371, 1368.0383, 1029.112878);

assignValue(pt[1018], 1360.5371, 1369.0383, 1030.112878);

assignValue(pt[1019], 1361.5371, 1370.0383, 1031.112878);

assignValue(pt[1020], 1362.5371, 1371.0383, 1032.112878);

assignValue(pt[1021], 1363.5371, 1372.0383, 1033.112878);

assignValue(pt[1022], 1364.5371, 1373.0383, 1034.112878);

assignValue(pt[1023], 1365.5371, 1374.0383, 1035.112878);

assignValue(pt[1024], 1366.5371, 1375.0383, 1036.112878);

assignValue(pt[1025], 1367.5371, 1376.0383, 1037.112878);

assignValue(pt[1026], 1368.5371, 1377.0383, 1038.112878);

assignValue(pt[1027], 1369.5371, 1378.0383, 1039.112878);

assignValue(pt[1028], 1370.5371, 1379.0383, 1040.112878);

assignValue(pt[1029], 1371.5371, 1380.0383, 1041.112878);

assignValue(pt[1030], 1372.5371, 1381.0383, 1042.112878);

assignValue(pt[1031], 1373.5371, 1382.0383, 1043.112878);

assignValue(pt[1032], 1374.5371, 1383.0383, 1044.112878);

assignValue(pt[1033], 1375.5371, 1384.0383, 1045.112878);

assignValue(pt[1034], 1376.5371, 1385.0383, 1046.112878);

assignValue(pt[1035], 1377.5371, 1386.0383, 1047.112878);

assignValue(pt[1036], 1378.5371, 1387.0383, 1048.112878);

assignValue(pt[1037], 1379.5371, 1388.0383, 1049.112878);

assignValue(pt[1038], 1380.5371, 1389.0383, 1050.112878);

assignValue(pt[1039], 1381.5371, 1390.0383, 1051.112878);

assignValue(pt[1040], 1382.5371, 1391.0383, 1052.112878);

assignValue(pt[1041], 1383.5371, 1392.0383, 1053.112878);

assignValue(pt[1042], 1384.5371, 1393.0383, 1054.112878);

assignValue(pt[1043], 1385.5371, 1394.0383, 1055.112878);

assignValue(pt[1044], 1386.5371, 1395.0383, 1056.112878);

assignValue(pt[1045], 1387.5371, 1396.0383, 1057.112878);

assignValue(pt[1046], 1388.5371, 1397.0383, 1058.112878);

assignValue(pt[1047], 1389.5371, 1398.0383, 1059.112878);

assignValue(pt[1048], 1390.5371, 1399.0383, 1060.112878);

assignValue(pt[1049], 1391.5371, 1400.0383, 1061.112878);

assignValue(pt[1050], 1392.5371, 1401.0383, 1062.112878);

assignValue(pt[1051], 1393.5371, 1402.0383, 1063.112878);

assignValue(pt[1052], 1394.5371, 1403.0383, 1064.112878);

assignValue(pt[1053], 1395.5371, 1404.0383, 1065.112878);

assignValue(pt[1054], 1396.5371, 1405.0383, 1066.112878);

assignValue(pt[1055], 1397.5371, 1406.0383, 1067.112878);

assignValue(pt[1056], 1398.5371, 1407.0383, 1068.112878);

assignValue(pt[1057], 1399.5371, 1408.0383, 1069.112878);

assignValue(pt[1058], 1400.5371, 1409.0383, 1070.112878);

assignValue(pt[1059], 1401.5371, 1410.0383, 1071.112878);

assignValue(pt[1060], 1402.5371, 1411.0383, 1072.112878);

assignValue(pt[1061], 1403.5371, 1412.0383, 1073.112878);

assignValue(pt[1062], 1404.5371, 1413.0383, 1074.112878);

assignValue(pt[1063], 1405.5371, 1414.0383, 1075.112878);

assignValue(pt[1064], 1406.5371, 1415.0383, 1076.112878);

assignValue(pt[1065], 1407.5371, 1416.0383, 1077.112878);

assignValue(pt[1066], 1408.5371, 1417.0383, 1078.112878);

assignValue(pt[1067], 1409.5371, 1418.0383, 1079.112878);

assignValue(pt[1068], 1410.5371, 1419.0383, 1080.112878);

assignValue(pt[1069], 1411.5371, 1420.0383, 1081.112878);

assignValue(pt[1070], 1412.5371, 1421.0383, 1082.112878);

assignValue(pt[1071], 1413.5371, 1422.0383, 1083.112878);

assignValue(pt[1072], 1414.5371, 1423.0383, 1084.112878);

assignValue(pt[1073], 1415.5371, 1424.0383, 1085.112878);

assignValue(pt[1074], 1416.5371, 1425.0383, 1086.112878);

assignValue(pt[1075], 1417.5371, 1426.0383, 1087.112878);

assignValue(pt[1076], 1418.5371, 1427.0383, 1088.112878);

assignValue(pt[1077], 1419.5371, 1428.0383, 1089.112878);

assignValue(pt[1078], 1420.5371, 1429.0383, 1090.112878);

assignValue(pt[1079], 1421.5371, 1430.0383, 1091.112878);

assignValue(pt[1080], 1422.5371, 1431.0383, 1092.112878);

assignValue(pt[1081], 1423.5371, 1432.0383, 1093.112878);

assignValue(pt[1082], 1424.5371, 1433.0383, 1094.112878);

assignValue(pt[1083], 1425.5371, 1434.0383, 1095.112878);

assignValue(pt[1084], 1426.5371, 1435.0383, 1096.112878);

assignValue(pt[1085], 1427.5371, 1436.0383, 1097.112878);

assignValue(pt[1086], 1428.5371, 1437.0383, 1098.112878);

assignValue(pt[1087], 1429.5371, 1438.0383, 1099.112878);

assignValue(pt[1088], 1430.5371, 1439.0383, 1100.112878);

assignValue(pt[1089], 1431.5371, 1440.0383, 1101.112878);

assignValue(pt[1090], 1432.5371, 1441.0383, 1102.112878);

assignValue(pt[1091], 1433.5371, 1442.0383, 1103.112878);

assignValue(pt[1092], 1434.5371, 1443.0383, 1104.112878);

assignValue(pt[1093], 1435.5371, 1444.0383, 1105.112878);

assignValue(pt[1094], 1436.5371, 1445.0383, 1106.112878);

assignValue(pt[1095], 1437.5371, 1446.0383, 1107.112878);

assignValue(pt[1096], 1438.5371, 1447.0383, 1108.112878);

assignValue(pt[1097], 1439.5371, 1448.0383, 1109.112878);

assignValue(pt[1098], 1440.5371, 1449.0383, 1110.112878);

assignValue(pt[1099], 1441.5371, 1450.0383, 1111.112878);

assignValue(pt[1100], 1442.5371, 1451.0383, 1112.112878);

assignValue(pt[1101], 1443.5371, 1452.0383, 1113.112878);

assignValue(pt[1102], 1444.5371, 1453.0383, 1114.112878);

assignValue(pt[1103], 1445.5371, 1454.0383, 1115.112878);

assignValue(pt[1104], 1446.5371, 1455.0383, 1116.112878);

assignValue(pt[1105], 1447.5371, 1456.0383, 1117.112878);

assignValue(pt[1106], 1448.5371, 1457.0383, 1118.112878);

assignValue(pt[1107], 1449.5371, 1458.0383, 1119.112878);

assignValue(pt[1108], 1450.5371, 1459.0383, 1120.112878);

assignValue(pt[1109], 1451.5371, 1460.0383, 1121.112878);

assignValue(pt[1110], 1452.5371, 1461.0383, 1122.112878);

assignValue(pt[1111], 1453.5371, 1462.0383, 1123.112878);

assignValue(pt[1112], 1454.5371, 1463.0383, 1124.112878);

assignValue(pt[1113], 1455.5371, 1464.0383, 1125.112878);

assignValue(pt[1114], 1456.5371, 1465.0383, 1126.112878);

assignValue(pt[1115], 1457.5371, 1466.0383, 1127.112878);

assignValue(pt[1116], 1458.5371, 1467.0383, 1128.112878);

assignValue(pt[1117], 1459.5371, 1468.0383, 1129.112878);

assignValue(pt[1118], 1460.5371, 1469.0383, 1130.112878);

assignValue(pt[1119], 1461.5371, 1470.0383, 1131.112878);

assignValue(pt[1120], 1462.5371, 1471.0383, 1132.112878);

assignValue(pt[1121], 1463.5371, 1472.0383, 1133.112878);

assignValue(pt[1122], 1464.5371, 1473.0383, 1134.112878);

assignValue(pt[1123], 1465.5371, 1474.0383, 1135.112878);

assignValue(pt[1124], 1466.5371, 1475.0383, 1136.112878);

assignValue(pt[1125], 1467.5371, 1476.0383, 1137.112878);

assignValue(pt[1126], 1468.5371, 1477.0383, 1138.112878);

assignValue(pt[1127], 1469.5371, 1478.0383, 1139.112878);

assignValue(pt[1128], 1470.5371, 1479.0383, 1140.112878);

assignValue(pt[1129], 1471.5371, 1480.0383, 1141.112878);

assignValue(pt[1130], 1472.5371, 1481.0383, 1142.112878);

assignValue(pt[1131], 1473.5371, 1482.0383, 1143.112878);

assignValue(pt[1132], 1474.5371, 1483.0383, 1144.112878);

assignValue(pt[1133], 1475.5371, 1484.0383, 1145.112878);

assignValue(pt[1134], 1476.5371, 1485.0383, 1146.112878);

assignValue(pt[1135], 1477.5371, 1486.0383, 1147.112878);

assignValue(pt[1136], 1478.5371, 1487.0383, 1148.112878);

assignValue(pt[1137], 1479.5371, 1488.0383, 1149.112878);

assignValue(pt[1138], 1480.5371, 1489.0383, 1150.112878);

assignValue(pt[1139], 1481.5371, 1490.0383, 1151.112878);

assignValue(pt[1140], 1482.5371, 1491.0383, 1152.112878);

assignValue(pt[1141], 1483.5371, 1492.0383, 1153.112878);

assignValue(pt[1142], 1484.5371, 1493.0383, 1154.112878);

assignValue(pt[1143], 1485.5371, 1494.0383, 1155.112878);

assignValue(pt[1144], 1486.5371, 1495.0383, 1156.112878);

assignValue(pt[1145], 1487.5371, 1496.0383, 1157.112878);

assignValue(pt[1146], 1488.5371, 1497.0383, 1158.112878);

assignValue(pt[1147], 1489.5371, 1498.0383, 1159.112878);

assignValue(pt[1148], 1490.5371, 1499.0383, 1160.112878);

assignValue(pt[1149], 1491.5371, 1500.0383, 1161.112878);

assignValue(pt[1150], 1492.5371, 1501.0383, 1162.112878);

assignValue(pt[1151], 1493.5371, 1502.0383, 1163.112878);

assignValue(pt[1152], 1494.5371, 1503.0383, 1164.112878);

assignValue(pt[1153], 1495.5371, 1504.0383, 1165.112878);

assignValue(pt[1154], 1496.5371, 1505.0383, 1166.112878);

assignValue(pt[1155], 1497.5371, 1506.0383, 1167.112878);

assignValue(pt[1156], 1498.5371, 1507.0383, 1168.112878);

assignValue(pt[1157], 1499.5371, 1508.0383, 1169.112878);

assignValue(pt[1158], 1500.5371, 1509.0383, 1170.112878);

assignValue(pt[1159], 1501.5371, 1510.0383, 1171.112878);

assignValue(pt[1160], 1502.5371, 1511.0383, 1172.112878);

assignValue(pt[1161], 1503.5371, 1512.0383, 1173.112878);

assignValue(pt[1162], 1504.5371, 1513.0383, 1174.112878);

assignValue(pt[1163], 1505.5371, 1514.0383, 1175.112878);

assignValue(pt[1164], 1506.5371, 1515.0383, 1176.112878);

assignValue(pt[1165], 1507.5371, 1516.0383, 1177.112878);

assignValue(pt[1166], 1508.5371, 1517.0383, 1178.112878);

assignValue(pt[1167], 1509.5371, 1518.0383, 1179.112878);

assignValue(pt[1168], 1510.5371, 1519.0383, 1180.112878);

assignValue(pt[1169], 1511.5371, 1520.0383, 1181.112878);

assignValue(pt[1170], 1512.5371, 1521.0383, 1182.112878);

assignValue(pt[1171], 1513.5371, 1522.0383, 1183.112878);

assignValue(pt[1172], 1514.5371, 1523.0383, 1184.112878);

assignValue(pt[1173], 1515.5371, 1524.0383, 1185.112878);

assignValue(pt[1174], 1516.5371, 1525.0383, 1186.112878);

assignValue(pt[1175], 1517.5371, 1526.0383, 1187.112878);

assignValue(pt[1176], 1518.5371, 1527.0383, 1188.112878);

assignValue(pt[1177], 1519.5371, 1528.0383, 1189.112878);

assignValue(pt[1178], 1520.5371, 1529.0383, 1190.112878);

assignValue(pt[1179], 1521.5371, 1530.0383, 1191.112878);

assignValue(pt[1180], 1522.5371, 1531.0383, 1192.112878);

assignValue(pt[1181], 1523.5371, 1532.0383, 1193.112878);

assignValue(pt[1182], 1524.5371, 1533.0383, 1194.112878);

assignValue(pt[1183], 1525.5371, 1534.0383, 1195.112878);

assignValue(pt[1184], 1526.5371, 1535.0383, 1196.112878);

assignValue(pt[1185], 1527.5371, 1536.0383, 1197.112878);

assignValue(pt[1186], 1528.5371, 1537.0383, 1198.112878);

assignValue(pt[1187], 1529.5371, 1538.0383, 1199.112878);

assignValue(pt[1188], 1530.5371, 1539.0383, 1200.112878);

assignValue(pt[1189], 1531.5371, 1540.0383, 1201.112878);

assignValue(pt[1190], 1532.5371, 1541.0383, 1202.112878);

assignValue(pt[1191], 1533.5371, 1542.0383, 1203.112878);

assignValue(pt[1192], 1534.5371, 1543.0383, 1204.112878);

assignValue(pt[1193], 1535.5371, 1544.0383, 1205.112878);

assignValue(pt[1194], 1536.5371, 1545.0383, 1206.112878);

assignValue(pt[1195], 1537.5371, 1546.0383, 1207.112878);

assignValue(pt[1196], 1538.5371, 1547.0383, 1208.112878);

assignValue(pt[1197], 1539.5371, 1548.0383, 1209.112878);

assignValue(pt[1198], 1540.5371, 1549.0383, 1210.112878);

assignValue(pt[1199], 1541.5371, 1550.0383, 1211.112878);

assignValue(pt[1200], 1542.5371, 1551.0383, 1212.112878);

assignValue(pt[1201], 1543.5371, 1552.0383, 1213.112878);

assignValue(pt[1202], 1544.5371, 1553.0383, 1214.112878);

assignValue(pt[1203], 1545.5371, 1554.0383, 1215.112878);

assignValue(pt[1204], 1546.5371, 1555.0383, 1216.112878);

assignValue(pt[1205], 1547.5371, 1556.0383, 1217.112878);

assignValue(pt[1206], 1548.5371, 1557.0383, 1218.112878);

assignValue(pt[1207], 1549.5371, 1558.0383, 1219.112878);

assignValue(pt[1208], 1550.5371, 1559.0383, 1220.112878);

assignValue(pt[1209], 1551.5371, 1560.0383, 1221.112878);

assignValue(pt[1210], 1552.5371, 1561.0383, 1222.112878);

assignValue(pt[1211], 1553.5371, 1562.0383, 1223.112878);

assignValue(pt[1212], 1554.5371, 1563.0383, 1224.112878);

assignValue(pt[1213], 1555.5371, 1564.0383, 1225.112878);

assignValue(pt[1214], 1556.5371, 1565.0383, 1226.112878);

assignValue(pt[1215], 1557.5371, 1566.0383, 1227.112878);

assignValue(pt[1216], 1558.5371, 1567.0383, 1228.112878);

assignValue(pt[1217], 1559.5371, 1568.0383, 1229.112878);

assignValue(pt[1218], 1560.5371, 1569.0383, 1230.112878);

assignValue(pt[1219], 1561.5371, 1570.0383, 1231.112878);

assignValue(pt[1220], 1562.5371, 1571.0383, 1232.112878);

assignValue(pt[1221], 1563.5371, 1572.0383, 1233.112878);

assignValue(pt[1222], 1564.5371, 1573.0383, 1234.112878);

assignValue(pt[1223], 1565.5371, 1574.0383, 1235.112878);

assignValue(pt[1224], 1566.5371, 1575.0383, 1236.112878);

assignValue(pt[1225], 1567.5371, 1576.0383, 1237.112878);

assignValue(pt[1226], 1568.5371, 1577.0383, 1238.112878);

assignValue(pt[1227], 1569.5371, 1578.0383, 1239.112878);

assignValue(pt[1228], 1570.5371, 1579.0383, 1240.112878);

assignValue(pt[1229], 1571.5371, 1580.0383, 1241.112878);

assignValue(pt[1230], 1572.5371, 1581.0383, 1242.112878);

assignValue(pt[1231], 1573.5371, 1582.0383, 1243.112878);

assignValue(pt[1232], 1574.5371, 1583.0383, 1244.112878);

assignValue(pt[1233], 1575.5371, 1584.0383, 1245.112878);

assignValue(pt[1234], 1576.5371, 1585.0383, 1246.112878);

assignValue(pt[1235], 1577.5371, 1586.0383, 1247.112878);

assignValue(pt[1236], 1578.5371, 1587.0383, 1248.112878);

assignValue(pt[1237], 1579.5371, 1588.0383, 1249.112878);

assignValue(pt[1238], 1580.5371, 1589.0383, 1250.112878);

assignValue(pt[1239], 1581.5371, 1590.0383, 1251.112878);

assignValue(pt[1240], 1582.5371, 1591.0383, 1252.112878);

assignValue(pt[1241], 1583.5371, 1592.0383, 1253.112878);

assignValue(pt[1242], 1584.5371, 1593.0383, 1254.112878);

assignValue(pt[1243], 1585.5371, 1594.0383, 1255.112878);

assignValue(pt[1244], 1586.5371, 1595.0383, 1256.112878);

assignValue(pt[1245], 1587.5371, 1596.0383, 1257.112878);

assignValue(pt[1246], 1588.5371, 1597.0383, 1258.112878);

assignValue(pt[1247], 1589.5371, 1598.0383, 1259.112878);

assignValue(pt[1248], 1590.5371, 1599.0383, 1260.112878);

assignValue(pt[1249], 1591.5371, 1600.0383, 1261.112878);

assignValue(pt[1250], 1592.5371, 1601.0383, 1262.112878);

assignValue(pt[1251], 1593.5371, 1602.0383, 1263.112878);

assignValue(pt[1252], 1594.5371, 1603.0383, 1264.112878);

assignValue(pt[1253], 1595.5371, 1604.0383, 1265.112878);

assignValue(pt[1254], 1596.5371, 1605.0383, 1266.112878);

assignValue(pt[1255], 1597.5371, 1606.0383, 1267.112878);

assignValue(pt[1256], 1598.5371, 1607.0383, 1268.112878);

assignValue(pt[1257], 1599.5371, 1608.0383, 1269.112878);

assignValue(pt[1258], 1600.5371, 1609.0383, 1270.112878);

assignValue(pt[1259], 1601.5371, 1610.0383, 1271.112878);

assignValue(pt[1260], 1602.5371, 1611.0383, 1272.112878);

assignValue(pt[1261], 1603.5371, 1612.0383, 1273.112878);

assignValue(pt[1262], 1604.5371, 1613.0383, 1274.112878);

assignValue(pt[1263], 1605.5371, 1614.0383, 1275.112878);

assignValue(pt[1264], 1606.5371, 1615.0383, 1276.112878);

assignValue(pt[1265], 1607.5371, 1616.0383, 1277.112878);

assignValue(pt[1266], 1608.5371, 1617.0383, 1278.112878);

assignValue(pt[1267], 1609.5371, 1618.0383, 1279.112878);

assignValue(pt[1268], 1610.5371, 1619.0383, 1280.112878);

assignValue(pt[1269], 1611.5371, 1620.0383, 1281.112878);

assignValue(pt[1270], 1612.5371, 1621.0383, 1282.112878);

assignValue(pt[1271], 1613.5371, 1622.0383, 1283.112878);

assignValue(pt[1272], 1614.5371, 1623.0383, 1284.112878);

assignValue(pt[1273], 1615.5371, 1624.0383, 1285.112878);

assignValue(pt[1274], 1616.5371, 1625.0383, 1286.112878);

assignValue(pt[1275], 1617.5371, 1626.0383, 1287.112878);

assignValue(pt[1276], 1618.5371, 1627.0383, 1288.112878);

assignValue(pt[1277], 1619.5371, 1628.0383, 1289.112878);

assignValue(pt[1278], 1620.5371, 1629.0383, 1290.112878);

assignValue(pt[1279], 1621.5371, 1630.0383, 1291.112878);

assignValue(pt[1280], 1622.5371, 1631.0383, 1292.112878);

assignValue(pt[1281], 1623.5371, 1632.0383, 1293.112878);

assignValue(pt[1282], 1624.5371, 1633.0383, 1294.112878);

assignValue(pt[1283], 1625.5371, 1634.0383, 1295.112878);

assignValue(pt[1284], 1626.5371, 1635.0383, 1296.112878);

assignValue(pt[1285], 1627.5371, 1636.0383, 1297.112878);

assignValue(pt[1286], 1628.5371, 1637.0383, 1298.112878);

assignValue(pt[1287], 1629.5371, 1638.0383, 1299.112878);

assignValue(pt[1288], 1630.5371, 1639.0383, 1300.112878);

assignValue(pt[1289], 1631.5371, 1640.0383, 1301.112878);

assignValue(pt[1290], 1632.5371, 1641.0383, 1302.112878);

assignValue(pt[1291], 1633.5371, 1642.0383, 1303.112878);

assignValue(pt[1292], 1634.5371, 1643.0383, 1304.112878);

assignValue(pt[1293], 1635.5371, 1644.0383, 1305.112878);

assignValue(pt[1294], 1636.5371, 1645.0383, 1306.112878);

assignValue(pt[1295], 1637.5371, 1646.0383, 1307.112878);

assignValue(pt[1296], 1638.5371, 1647.0383, 1308.112878);

assignValue(pt[1297], 1639.5371, 1648.0383, 1309.112878);

assignValue(pt[1298], 1640.5371, 1649.0383, 1310.112878);

assignValue(pt[1299], 1641.5371, 1650.0383, 1311.112878);

assignValue(pt[1300], 1642.5371, 1651.0383, 1312.112878);

assignValue(pt[1301], 1643.5371, 1652.0383, 1313.112878);

assignValue(pt[1302], 1644.5371, 1653.0383, 1314.112878);

assignValue(pt[1303], 1645.5371, 1654.0383, 1315.112878);

assignValue(pt[1304], 1646.5371, 1655.0383, 1316.112878);

assignValue(pt[1305], 1647.5371, 1656.0383, 1317.112878);

assignValue(pt[1306], 1648.5371, 1657.0383, 1318.112878);

assignValue(pt[1307], 1649.5371, 1658.0383, 1319.112878);

assignValue(pt[1308], 1650.5371, 1659.0383, 1320.112878);

assignValue(pt[1309], 1651.5371, 1660.0383, 1321.112878);

assignValue(pt[1310], 1652.5371, 1661.0383, 1322.112878);

assignValue(pt[1311], 1653.5371, 1662.0383, 1323.112878);

assignValue(pt[1312], 1654.5371, 1663.0383, 1324.112878);

assignValue(pt[1313], 1655.5371, 1664.0383, 1325.112878);

assignValue(pt[1314], 1656.5371, 1665.0383, 1326.112878);

assignValue(pt[1315], 1657.5371, 1666.0383, 1327.112878);

assignValue(pt[1316], 1658.5371, 1667.0383, 1328.112878);

assignValue(pt[1317], 1659.5371, 1668.0383, 1329.112878);

assignValue(pt[1318], 1660.5371, 1669.0383, 1330.112878);

assignValue(pt[1319], 1661.5371, 1670.0383, 1331.112878);

assignValue(pt[1320], 1662.5371, 1671.0383, 1332.112878);

assignValue(pt[1321], 1663.5371, 1672.0383, 1333.112878);

assignValue(pt[1322], 1664.5371, 1673.0383, 1334.112878);

assignValue(pt[1323], 1665.5371, 1674.0383, 1335.112878);

assignValue(pt[1324], 1666.5371, 1675.0383, 1336.112878);

assignValue(pt[1325], 1667.5371, 1676.0383, 1337.112878);

assignValue(pt[1326], 1668.5371, 1677.0383, 1338.112878);

assignValue(pt[1327], 1669.5371, 1678.0383, 1339.112878);

assignValue(pt[1328], 1670.5371, 1679.0383, 1340.112878);

assignValue(pt[1329], 1671.5371, 1680.0383, 1341.112878);

assignValue(pt[1330], 1672.5371, 1681.0383, 1342.112878);

assignValue(pt[1331], 1673.5371, 1682.0383, 1343.112878);

assignValue(pt[1332], 1674.5371, 1683.0383, 1344.112878);

assignValue(pt[1333], 1675.5371, 1684.0383, 1345.112878);

assignValue(pt[1334], 1676.5371, 1685.0383, 1346.112878);

assignValue(pt[1335], 1677.5371, 1686.0383, 1347.112878);

assignValue(pt[1336], 1678.5371, 1687.0383, 1348.112878);

assignValue(pt[1337], 1679.5371, 1688.0383, 1349.112878);

assignValue(pt[1338], 1680.5371, 1689.0383, 1350.112878);

assignValue(pt[1339], 1681.5371, 1690.0383, 1351.112878);

assignValue(pt[1340], 1682.5371, 1691.0383, 1352.112878);

assignValue(pt[1341], 1683.5371, 1692.0383, 1353.112878);

assignValue(pt[1342], 1684.5371, 1693.0383, 1354.112878);

assignValue(pt[1343], 1685.5371, 1694.0383, 1355.112878);

assignValue(pt[1344], 1686.5371, 1695.0383, 1356.112878);

assignValue(pt[1345], 1687.5371, 1696.0383, 1357.112878);

assignValue(pt[1346], 1688.5371, 1697.0383, 1358.112878);

assignValue(pt[1347], 1689.5371, 1698.0383, 1359.112878);

assignValue(pt[1348], 1690.5371, 1699.0383, 1360.112878);

assignValue(pt[1349], 1691.5371, 1700.0383, 1361.112878);

assignValue(pt[1350], 1692.5371, 1701.0383, 1362.112878);

assignValue(pt[1351], 1693.5371, 1702.0383, 1363.112878);

assignValue(pt[1352], 1694.5371, 1703.0383, 1364.112878);

assignValue(pt[1353], 1695.5371, 1704.0383, 1365.112878);

assignValue(pt[1354], 1696.5371, 1705.0383, 1366.112878);

return pt;

}

void connectPoints() {

int ptsSize = 1354;

point3d\* pt = assignBatchPoint();

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

glBegin(GL\_LINE\_STRIP);

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

{

glVertex3f(pt[i].x, pt[i].y, pt[i].z);

}

glEnd();

glFlush();

}

//Window Default size

int windowWidth = 512, windowHeight = 512;

int tx=1000, ty=1000, tz=1000;//改变量

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

int rotatexyz = 0,rotatex,rotatey,rotatez;

int angle = 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/video\_20201103\_132337.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)

void get\_bounding\_box\_for\_node(const aiNode\* nd,

aiVector3D\* min,

aiVector3D\* max,

aiScene\* scene)

{

aiMatrix4x4 prev;

unsigned int n = 0, t;

for (; n < nd->mNumMeshes; ++n) {

const aiMesh\* mesh = scene->mMeshes[nd->mMeshes[n]];

for (t = 0; t < mesh->mNumVertices; ++t) {

aiVector3D tmp = mesh->mVertices[t];

min->x = aisgl\_min(min->x, tmp.x);

min->y = aisgl\_min(min->y, tmp.y);

min->z = aisgl\_min(min->z, tmp.z);

max->x = aisgl\_max(max->x, tmp.x);

max->y = aisgl\_max(max->y, tmp.y);

max->z = aisgl\_max(max->z, tmp.z);

}

}

for (n = 0; n < nd->mNumChildren; ++n) {

get\_bounding\_box\_for\_node(nd->mChildren[n], min, max, scene);

}

}

void get\_bounding\_box(aiVector3D\* min, aiVector3D\* max, aiScene\* scene)

{

min->x = min->y = min->z = 1e10f;

max->x = max->y = max->z = -1e10f;

get\_bounding\_box\_for\_node(scene->mRootNode, min, max, scene);

}

bool loadModelFile(string filename) {

filename = modelDir + filename;

//open file stream to the file

ifstream infile(filename);

//if opened successfully, read in the data

if (infile.is\_open()) {

int marker;

string file;

while (infile >> file >> marker)

{

modelMap[marker] = file;

}

}

else {

//if here, file was not opened correctly, notify user

printf("Error opening file,%s exiting", filename.c\_str());

exit(0);

}

return true;

}

bool Import3DFromFile(const std::string& pFile, aiScene\*& scene, Assimp::Importer& importer, float& scaleFactor)

{

std::string fileDir = modelDir + pFile;

//check if file exists

std::ifstream fin(fileDir.c\_str());

if (!fin.fail()) {

fin.close();

}

else {

printf("Couldn't open file: %s\n", fileDir.c\_str());

printf("%s\n", importer.GetErrorString());

return false;

}

scene = const\_cast<aiScene\*>(importer.ReadFile(fileDir, aiProcessPreset\_TargetRealtime\_Quality));

// If the import failed, report it

if (!scene)

{

printf("%s\n", importer.GetErrorString());

return false;

}

// Now we can access the file's contents.

printf("Import of scene %s succeeded.\n", fileDir.c\_str());

aiVector3D scene\_min, scene\_max, scene\_center;

get\_bounding\_box(&scene\_min, &scene\_max, scene);

float tmp;

tmp = scene\_max.x - scene\_min.x;

tmp = scene\_max.y - scene\_min.y > tmp ? scene\_max.y - scene\_min.y : tmp;

tmp = scene\_max.z - scene\_min.z > tmp ? scene\_max.z - scene\_min.z : tmp;

scaleFactor = 1.f / tmp;

// We're done. Everything will be cleaned up by the importer destructor

return true;

}

int LoadGLTextures(aiScene\* scene)

{

ILboolean success;

/\* initialization of DevIL \*/

ilInit();

/\* scan scene's materials for textures 扫描场景的材质以获取纹理\*/

for (unsigned int m = 0; m < scene->mNumMaterials; ++m)

{

int texIndex = 0;

aiString path; // filename

aiReturn texFound = scene->mMaterials[m]->GetTexture(aiTextureType\_DIFFUSE, texIndex, &path);

while (texFound == AI\_SUCCESS) {

//fill map with textures, OpenGL image ids set to 0

textureIdMap[path.data] = 0;

// more textures?

texIndex++;

texFound = scene->mMaterials[m]->GetTexture(aiTextureType\_DIFFUSE, texIndex, &path);

}

}

int numTextures = textureIdMap.size();

/\* create and fill array with DevIL texture ids 用DevIL纹理ID创建并填充数组\*/

ILuint\* imageIds = new ILuint[numTextures];

ilGenImages(numTextures, imageIds);

/\* create and fill array with GL texture ids \*/

GLuint\* textureIds = new GLuint[numTextures];

glGenTextures(numTextures, textureIds); /\* Texture name generation \*/

/\* get iterator \*/

std::map<std::string, GLuint>::iterator itr = textureIdMap.begin();

int i = 0;

for (; itr != textureIdMap.end(); ++i, ++itr)

{

//save IL image ID

std::string filename = (\*itr).first; // get filename

std::replace(filename.begin(), filename.end(), '\\', '/'); //Replace backslash with forward slash so linux can find the files

filename = modelDir + filename;

(\*itr).second = textureIds[i]; // save texture id for filename in map

ilBindImage(imageIds[i]); /\* Binding of DevIL image name \*/

ilEnable(IL\_ORIGIN\_SET);

ilOriginFunc(IL\_ORIGIN\_LOWER\_LEFT);

success = ilLoadImage((ILstring)filename.c\_str());

if (success) {

/\* Convert image to RGBA \*/

ilConvertImage(IL\_RGBA, IL\_UNSIGNED\_BYTE);

/\* Create and load textures to OpenGL \*/

glBindTexture(GL\_TEXTURE\_2D, textureIds[i]);

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

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

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, ilGetInteger(IL\_IMAGE\_WIDTH),

ilGetInteger(IL\_IMAGE\_HEIGHT), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE,

ilGetData());

}

else

printf("Couldn't load Image: %s\n", filename.c\_str());

}

/\* Because we have already copied image data into texture data

we can release memory used by image. \*/

//因为我们已经将图像数据复制到纹理数据中

// 我们可以释放图像使用的内存。

ilDeleteImages(numTextures, imageIds);

//Cleanup

delete[] imageIds;

delete[] textureIds;

//return success;

return true;

}

//// Can't send color down as a pointer to aiColor4D because AI colors are ABGR.

//void Color4f(const aiColor4D \*color)

//{

// glColor4f(color->r, color->g, color->b, color->a);

//}

void set\_float4(float f[4], float a, float b, float c, float d)

{

f[0] = a;

f[1] = b;

f[2] = c;

f[3] = d;

}

void color4\_to\_float4(const aiColor4D \*c, float f[4])

{

f[0] = c->r;

f[1] = c->g;

f[2] = c->b;

f[3] = c->a;

}

void genVAOsAndUniformBuffer(aiScene \*sc, std::vector<struct MyMesh> &myMeshes) {

struct MyMesh aMesh;

struct MyMaterial aMat;

GLuint buffer;

// For each mesh 对于每个啮合

for (unsigned int n = 0; n < sc->mNumMeshes; ++n)

{

const aiMesh\* mesh = sc->mMeshes[n];

// create array with faces

// have to convert from Assimp format to array

//创建带有面的数组

//必须从Assimp格式转换为数组

unsigned int \*faceArray;

faceArray = (unsigned int \*)malloc(sizeof(unsigned int) \* mesh->mNumFaces \* 3);

unsigned int faceIndex = 0;

for (unsigned int t = 0; t < mesh->mNumFaces; ++t) {

const aiFace\* face = &mesh->mFaces[t];

memcpy(&faceArray[faceIndex], face->mIndices, 3 \* sizeof(unsigned int));

faceIndex += 3;

}

aMesh.numFaces = sc->mMeshes[n]->mNumFaces;

// generate Vertex Array for mesh 生成网格的顶点数组

glGenVertexArrays(1, &(aMesh.vao));

glBindVertexArray(aMesh.vao);

// buffer for faces 面部缓冲

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(unsigned int) \* mesh->mNumFaces \* 3, faceArray, GL\_STATIC\_DRAW);

// buffer for vertex positions 顶点位置缓冲区

if (mesh->HasPositions()) {

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(float) \* 3 \* mesh->mNumVertices, mesh->mVertices, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(vertexLoc);

glVertexAttribPointer(vertexLoc, 3, GL\_FLOAT, 0, 0, 0);

}

// buffer for vertex normals 顶点法线的缓冲区

if (mesh->HasNormals()) {

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(float) \* 3 \* mesh->mNumVertices, mesh->mNormals, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(normalLoc);

glVertexAttribPointer(normalLoc, 3, GL\_FLOAT, 0, 0, 0);

}

// buffer for vertex texture coordinates 顶点纹理坐标的缓冲区

if (mesh->HasTextureCoords(0)) {

float \*texCoords = (float \*)malloc(sizeof(float) \* 2 \* mesh->mNumVertices);

for (unsigned int k = 0; k < mesh->mNumVertices; ++k) {

texCoords[k \* 2] = mesh->mTextureCoords[0][k].x;

texCoords[k \* 2 + 1] = mesh->mTextureCoords[0][k].y;

}

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(float) \* 2 \* mesh->mNumVertices, texCoords, GL\_STATIC\_DRAW);

glEnableVertexAttribArray(texCoordLoc);

glVertexAttribPointer(texCoordLoc, 2, GL\_FLOAT, 0, 0, 0);

}

// unbind buffers 解除绑定缓冲区

glBindVertexArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, 0);

// create material uniform buffer 创建材料统一缓冲区

aiMaterial \*mtl = sc->mMaterials[mesh->mMaterialIndex];

aiString texPath; //contains filename of texture

if (AI\_SUCCESS == mtl->GetTexture(aiTextureType\_DIFFUSE, 0, &texPath)) {

//bind texture

unsigned int texId = textureIdMap[texPath.data];

aMesh.texIndex = texId;

aMat.texCount = 1;

}

else

aMat.texCount = 0;

float c[4];

set\_float4(c, 0.8f, 0.8f, 0.8f, 1.0f);

aiColor4D diffuse;

if (AI\_SUCCESS == aiGetMaterialColor(mtl, AI\_MATKEY\_COLOR\_DIFFUSE, &diffuse))

color4\_to\_float4(&diffuse, c);

memcpy(aMat.diffuse, c, sizeof(c));

set\_float4(c, 0.2f, 0.2f, 0.2f, 1.0f);

aiColor4D ambient;

if (AI\_SUCCESS == aiGetMaterialColor(mtl, AI\_MATKEY\_COLOR\_AMBIENT, &ambient))

color4\_to\_float4(&ambient, c);

memcpy(aMat.ambient, c, sizeof(c));

set\_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);

aiColor4D specular;

if (AI\_SUCCESS == aiGetMaterialColor(mtl, AI\_MATKEY\_COLOR\_SPECULAR, &specular))

color4\_to\_float4(&specular, c);

memcpy(aMat.specular, c, sizeof(c));

set\_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);

aiColor4D emission;

if (AI\_SUCCESS == aiGetMaterialColor(mtl, AI\_MATKEY\_COLOR\_EMISSIVE, &emission))

color4\_to\_float4(&emission, c);

memcpy(aMat.emissive, c, sizeof(c));

float shininess = 0.0;

unsigned int max;

aiGetMaterialFloatArray(mtl, AI\_MATKEY\_SHININESS, &shininess, &max);

aMat.shininess = shininess;

glGenBuffers(1, &(aMesh.uniformBlockIndex));

glBindBuffer(GL\_UNIFORM\_BUFFER, aMesh.uniformBlockIndex);

glBufferData(GL\_UNIFORM\_BUFFER, sizeof(aMat), (void \*)(&aMat), GL\_STATIC\_DRAW);

myMeshes.push\_back(aMesh);

}

}

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

//

// Reshape Callback Function重塑回调函数

//

void changeSize(int w, int h) {

float ratio;

// Prevent a divide by zero, when window is too short 当窗口太短时防止被零除

// (you cant make a window of zero width).

if (h == 0)

h = 1;

windowWidth = w;

windowHeight = h;

// Set the viewport to be the entire window 将视口设置为整个窗口

glViewport(0, 0, w, h);

ratio = w / h;

buildProjectionMatrix(53.13f, ratio, 0.1f, 10.0f);

}

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

//

// Render stuff

//

// Render Assimp Model 渲染Assimp模型

void recursive\_render(aiScene \*sc, const aiNode\* nd, std::vector<struct MyMesh> &myMeshes)

{

// Get node transformation matrix 获取节点转换矩阵

aiMatrix4x4 m = nd->mTransformation;

// OpenGL matrices are column major

m.Transpose();

// save model matrix and apply node transformation 保存模型矩阵并应用节点变换

pushMatrix();

float aux[16];

memcpy(aux, &m, sizeof(float) \* 16);

multMatrix(modelMatrix, aux);

setModelMatrix();

// draw all meshes assigned to this node 绘制分配给该节点的所有网格

for (unsigned int n = 0; n < nd->mNumMeshes; ++n) {

// bind material uniform bind:绑定

glBindBufferRange(GL\_UNIFORM\_BUFFER, materialUniLoc, myMeshes[nd->mMeshes[n]].uniformBlockIndex, 0, sizeof(struct MyMaterial));

// bind texture

glBindTexture(GL\_TEXTURE\_2D, myMeshes[nd->mMeshes[n]].texIndex);

// bind VAO

glBindVertexArray(myMeshes[nd->mMeshes[n]].vao);

// draw

glDrawElements(GL\_TRIANGLES, myMeshes[nd->mMeshes[n]].numFaces \* 3, GL\_UNSIGNED\_INT, 0);

point3d \*pt = {};

//pt= assignBatchPoint();

int ptsSize = 1354;

}

// draw all children

for (unsigned int n = 0; n < nd->mNumChildren; ++n) {

recursive\_render(sc, nd->mChildren[n], myMeshes);

}

popMatrix();

}

// Rendering Callback Function 渲染回调功能

//!!!!!!!

void detectArucoMarkers(cv::Mat &image)

{

cv::aruco::detectMarkers(

image, // input image

dictionary, // type of markers that will be searched for

markerCorners, // output vector of marker corners

markerIds, // detected marker IDs

detectorParams, // algorithm parameters

rejectedCandidates);

map<int, MyModel>::iterator it;

for (it = models.begin(); it != models.end(); it++) {

it->second.seennow = false;

}

if (markerIds.size() > 0) {

// Draw all detected markers.

cv::aruco::drawDetectedMarkers(image, markerCorners, markerIds);

std::vector< cv::Vec3d > rvecs, tvecs;

cv::aruco::estimatePoseSingleMarkers(

markerCorners, // vector of already detected markers corners

markerLength, // length of the marker's side

K, // input 3x3 floating-point instrinsic camera matrix K

distCoeffs, // vector of distortion coefficients of 4, 5, 8 or 12 elements

rvecs, // array of output rotation vectors

tvecs); // array of output translation vectors

for (unsigned int i = 0; i < markerIds.size(); i++) {

cv::Mat viewMatrix = cv::Mat::zeros(4, 4, CV\_32F);;

cv::Mat viewMatrixavg = cv::Mat::zeros(4, 4, CV\_32F);

cv::Vec3d r = rvecs[i];

tvecs[i][0] = tvecs[i][0] + (double)tx / 100- (double)10;

tvecs[i][1] = tvecs[i][1] + (double)ty / 100- (double)20;

tvecs[i][2] = tvecs[i][2] + (double)tz / 100- (double)10;

cv::Vec3d t = tvecs[i];

cv::Mat rot;

Rodrigues(rvecs[i], rot);

for (unsigned int row = 0; row < 3; ++row)

{

for (unsigned int col = 0; col < 3; ++col)

{

viewMatrix.at<float>(row, col) = (float)rot.at<double>(row, col);

}

viewMatrix.at<float>(row, 3) = (float)tvecs[i][row] \* 0.1f;

}

viewMatrix.at<float>(3, 3) = 1.0f;

cv::Mat cvToGl = cv::Mat::zeros(4, 4, CV\_32F);

cvToGl.at<float>(0, 0) = 1.0f;

cvToGl.at<float>(1, 1) = -1.0f; // Invert the y axis

cvToGl.at<float>(2, 2) = -1.0f; // invert the z axis

cvToGl.at<float>(3, 3) = 1.0f;

viewMatrix = cvToGl \* viewMatrix;

cv::transpose(viewMatrix, viewMatrix);

bool flipped = false;

if (modelMap.count(markerIds[i])) {

models[markerIds[i]].seennow = true;

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

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

if (models[markerIds[i]].seenlast > 3 || models[markerIds[i]].seenlast < 0) {

if (std::abs(viewMatrix.at<float>(jj, yy) - models[markerIds[i]].viewMatrix[1].at<float>(jj, yy)) > .5)flipped = true;

}

viewMatrixavg.at<float>(jj, yy) = (viewMatrix.at<float>(jj, yy) + models[markerIds[i]].viewMatrix[1].at<float>(jj, yy) + models[markerIds[i]].viewMatrix[2].at<float>(jj, yy)) / 3;

}

}

if (models[markerIds[i]].seenlast < 0) {

models[markerIds[i]].seenlast = 0;

}

if (flipped) {

models[markerIds[i]].viewMatrix[0] = models[markerIds[i]].viewMatrix[1];

}

else {

models[markerIds[i]].viewMatrix[0] = viewMatrixavg;

models[markerIds[i]].viewMatrix[2] = models[markerIds[i]].viewMatrix[1];

models[markerIds[i]].viewMatrix[1] = viewMatrix;

}

}

// Draw coordinate axes.

cv::aruco::drawAxis(image,

K, distCoeffs, // camera parameters

r, t, // marker pose

0.5\*markerLength); // length of the axes to be drawn

// Draw a symbol in the upper right corner of the detected marker.

}

}

for (it = models.begin(); it != models.end(); it++) {

if (!it->second.seennow) {

if (it->second.seenlast == -10) {

it->second.viewMatrix[0] = cv::Mat::zeros(4, 4, CV\_32F);

it->second.seenlast = 0;

}

else if (it->second.seenlast < 0) {

it->second.seenlast = it->second.seenlast - 1;

}

else if (it->second.seenlast > 0) {

it->second.seenlast = -1;

}

}

else if (it->second.seenlast<100) {

it->second.seenlast++;

}

}

}

void prepareTexture(int w, int h, unsigned char\* data)

{

/\* Create and load texture to OpenGL \*/

glBindTexture(GL\_TEXTURE\_2D, textureID);

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA,

w, h,

0, GL\_RGB, GL\_UNSIGNED\_BYTE,

data);

glGenerateMipmap(GL\_TEXTURE\_2D);

}

void camTimer()

{

cap >> imageMat; // get image from camera

if (!imageMat.empty()) {

// Convert to RGB

cv::cvtColor(imageMat, imageMat, CV\_BGR2RGB);

if (flipped) {

cv::flip(imageMat, imageMat, 0);

cv::flip(imageMat, imageMat, 1);

}

detectArucoMarkers(imageMat);

imageMat.copyTo(imageMatGL);

camTimer();

}

}

void renderScene(void)

{

// Create Texture

prepareTexture(imageMatGL.cols, imageMatGL.rows, imageMatGL.data);

//gluBuild2DMipmaps(GL\_TEXTURE\_2D, GL\_RGB, imageMat.cols, imageMat.rows, GL\_RGB, GL\_UNSIGNED\_BYTE, imageMat.data);

// clear the framebuffer (color and depth) 清除帧缓冲区（颜色和深度）

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

//DRAW 2D VIDEO

// Use the program p

glUseProgram(p);

// Bind the vertex array object

glBindVertexArray(vao);

// Bind texture

glBindTexture(GL\_TEXTURE\_2D, textureID);

// draw the 6 vertices

glDrawArrays(GL\_TRIANGLES, 0, 6);

//DRAW 3D MODEL

glClear(GL\_DEPTH\_BUFFER\_BIT);

// set camera matrix

map<int, MyModel>::iterator it;

for (it = models.begin(); it != models.end(); it++)

{

MyModel currentModel = it->second;

setCamera(currentModel.viewMatrix[0]);

// set the model matrix to the identity Matrix 将模型矩阵设置为恒等矩阵

setIdentityMatrix(modelMatrix, 4);

// sets the model matrix to a scale matrix so that the model fits in the window

//模型矩阵设置为比例矩阵，以便模型适合窗口

scale((double)scalex/2000\*currentModel.scaleFactor, (double)scaley/2000\*currentModel.scaleFactor, (double)scalez/2000\*currentModel.scaleFactor);

rotatex = rotatey = rotatez = 0;

switch (rotatexyz)

{

case 0:rotatex=1; break;

case 1:rotatey = 1; break;

case 2:rotatez = 1; break;

}

// keep rotating the model 继续旋转模型

rotate((double)angle\*1.0f, rotatex\*1.0f, rotatey\*1.0f,rotatez\*1.0f);

// use our shadershader 使用我们的shadershader

glUseProgram(program);

// we are only going to use texture unit 0

// unfortunately samplers can't reside in uniform blocks

// so we have set this uniform separately

//我们将仅使用纹理单元0

//不幸的是，采样器不能驻留在统一的块中

//因此我们分别设置了此制服

glUniform1i(texUnit, 0);

//glLoadMatrixf((float\*)viewMatrix.data);

recursive\_render(currentModel.scene, currentModel.scene->mRootNode, currentModel.myMeshes);

}

// FPS computation and display FPS计算和显示

frame++;

timet = glutGet(GLUT\_ELAPSED\_TIME);

if (timet - timebase > 1000) {

sprintf(s, "FPS:%4.2f",

frame\*1000.0 / (timet - timebase));

timebase = timet;

frame = 0;

glutSetWindowTitle(s);

}

// swap buffers 交换缓冲区

glutSwapBuffers();

}

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

//

// Events from the Keyboard 键盘事件

//

void processKeys(unsigned char key, int xx, int yy)

{

switch (key) {

case 27:

glutLeaveMainLoop();

break;

case 'm': glEnable(GL\_MULTISAMPLE); break;

case 'n': glDisable(GL\_MULTISAMPLE); break;

case 'f': flipped = !flipped; break;

case '0': cap = cv::VideoCapture(0); break;

case '1': cap = cv::VideoCapture(1); break;

}

}

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

//

// Shader Stuff 材料着色器

//

void printShaderInfoLog(GLuint obj)

{

int infologLength = 0;

int charsWritten = 0;

char \*infoLog;

glGetShaderiv(obj, GL\_INFO\_LOG\_LENGTH, &infologLength);

if (infologLength > 0)

{

infoLog = (char \*)malloc(infologLength);

glGetShaderInfoLog(obj, infologLength, &charsWritten, infoLog);

printf("%s\n", infoLog);

free(infoLog);

}

}

void printProgramInfoLog(GLuint obj)

{

int infologLength = 0;

int charsWritten = 0;

char \*infoLog;

glGetProgramiv(obj, GL\_INFO\_LOG\_LENGTH, &infologLength);

if (infologLength > 0)

{

infoLog = (char \*)malloc(infologLength);

glGetProgramInfoLog(obj, infologLength, &charsWritten, infoLog);

printf("%s\n", infoLog);

free(infoLog);

}

}

GLuint setupShaders() {

char \*vs = NULL, \*fs = NULL;

GLuint p, v, f;

v = glCreateShader(GL\_VERTEX\_SHADER);

f = glCreateShader(GL\_FRAGMENT\_SHADER);

vs = textFileRead(vertexFileName);

fs = textFileRead(fragmentFileName);

const char \* vv = vs;

const char \* ff = fs;

glShaderSource(v, 1, &vv, NULL);

glShaderSource(f, 1, &ff, NULL);

free(vs); free(fs);

glCompileShader(v);

glCompileShader(f);

p = glCreateProgram();

glAttachShader(p, v);

glAttachShader(p, f);

glBindFragDataLocation(p, 0, "output");

glBindAttribLocation(p, vertexLoc, "position");

glBindAttribLocation(p, normalLoc, "normal");

glBindAttribLocation(p, texCoordLoc, "texCoord");

glLinkProgram(p);

glValidateProgram(p);

program = p;

vertexShader = v;

fragmentShader = f;

GLuint k = glGetUniformBlockIndex(p, "Matrices");

glUniformBlockBinding(p, k, matricesUniLoc);

glUniformBlockBinding(p, glGetUniformBlockIndex(p, "Material"), materialUniLoc);

texUnit = glGetUniformLocation(p, "texUnit");

return(p);

}

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

//

// Shader Stuff

//

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

void setupShaders2D() {

// variables to hold the shader's source code 用于保存着色器源代码的变量

char \*vs = NULL, \*fs = NULL;

// holders for the shader's ids 着色器ID的持有人

GLuint v, f;

// create the two shaders

v = glCreateShader(GL\_VERTEX\_SHADER);

f = glCreateShader(GL\_FRAGMENT\_SHADER);

// read the source code from file

vs = textFileRead((char \*)"texture.vert");

fs = textFileRead((char \*)"texture.frag");

// castings for calling the shader source function 用于调用着色器源函数的转换

const char \* vv = vs;

const char \* ff = fs;

// setting the source for each shader

glShaderSource(v, 1, &vv, NULL);

glShaderSource(f, 1, &ff, NULL);

// free the source strings

free(vs); free(fs);

// compile the sources

glCompileShader(v);

glCompileShader(f);

// create a program and attach the shaders

p = glCreateProgram();

glAttachShader(p, v);

glAttachShader(p, f);

// Bind the fragment data output variable location

// requires linking afterwards

glBindFragDataLocation(p, 0, "outputF");

// link the program

glLinkProgram(p);

GLint myLoc = glGetUniformLocation(p, "texUnit");

glProgramUniform1d(p, myLoc, 0);

}

int init2D() {

// Data for the two triangles

float position[] = {

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

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

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

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

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

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

};

float textureCoord[] = {

1.0f, 1.0f,

0.0f, 0.0f,

1.0f, 0.0f,

0.0f, 0.0f,

1.0f, 1.0f,

0.0f, 1.0f,

};

// variables to hold the shader's source code

char \*vs = NULL, \*fs = NULL;

// holders for the shader's ids

GLuint v, f;

// create the two shaders

v = glCreateShader(GL\_VERTEX\_SHADER);

f = glCreateShader(GL\_FRAGMENT\_SHADER);

// read the source code from file

vs = textFileRead((char \*)"texture.vert");

fs = textFileRead((char \*)"texture.frag");

// castings for calling the shader source function

const char \* vv = vs;

const char \* ff = fs;

// setting the source for each shader

glShaderSource(v, 1, &vv, NULL);

glShaderSource(f, 1, &ff, NULL);

// free the source strings

free(vs); free(fs);

// compile the sources

glCompileShader(v);

glCompileShader(f);

// create a program and attach the shaders

p = glCreateProgram();

glAttachShader(p, v);

glAttachShader(p, f);

// Bind the fragment data output variable location

// requires linking afterwards

glBindFragDataLocation(p, 0, "outputF");

// link the program

glLinkProgram(p);

GLint myLoc = glGetUniformLocation(p, "texUnit");

glProgramUniform1d(p, myLoc, 0);

GLuint vertexLoc, texCoordLoc;

// Get the locations of the attributes in the current program

vertexLoc = glGetAttribLocation(p, "position");

texCoordLoc = glGetAttribLocation(p, "texCoord");

// Generate and bind a Vertex Array Object

// this encapsulates the buffers used for drawing the triangle

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

// Generate two slots for the position and color buffers

GLuint buffers[2];

glGenBuffers(2, buffers);

// bind buffer for vertices and copy data into buffer

glBindBuffer(GL\_ARRAY\_BUFFER, buffers[0]);

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

glEnableVertexAttribArray(vertexLoc);

glVertexAttribPointer(vertexLoc, 4, GL\_FLOAT, 0, 0, 0);

// bind buffer for normals and copy data into buffer

glBindBuffer(GL\_ARRAY\_BUFFER, buffers[1]);

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

glEnableVertexAttribArray(texCoordLoc);

glVertexAttribPointer(texCoordLoc, 2, GL\_FLOAT, 0, 0, 0);

glGenTextures(1, &textureID); //Gen a new texture and store the handle in texname

//These settings stick with the texture that's bound. You only need to set them

//once.

glBindTexture(GL\_TEXTURE\_2D, textureID);

//allocate memory on the graphics card for the texture. It's fine if

//texture\_data doesn't have any data in it, the texture will just appear black

//until you update it.

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, 1, 1, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, {});

return true;

}

int loadModels() {

map<int, string>::iterator it;

for (it = modelMap.begin(); it != modelMap.end(); it++)

{

int markerNum = it->first;

string modelName = it->second;

models[markerNum].marker = markerNum;

if (!Import3DFromFile(modelName, models[markerNum].scene, models[markerNum].importer, models[markerNum].scaleFactor))

return(-1);

LoadGLTextures(models[markerNum].scene);

genVAOsAndUniformBuffer(models[markerNum].scene, models[markerNum].myMeshes);

}

return 0;

}

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

//

// Model loading and OpenGL setup 模型加载和OpenGL设置

//

int init()

{

glEnable(GL\_BLEND); // 打开混合

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

//glEnable(GL\_LIGHT1);

glColor4f(1, 1, 1, 0.5);

loadModels();

//glDisable(GL\_BLEND);

glGetUniformBlockIndex = (PFNGLGETUNIFORMBLOCKINDEXPROC)glutGetProcAddress("glGetUniformBlockIndex");

glUniformBlockBinding = (PFNGLUNIFORMBLOCKBINDINGPROC)glutGetProcAddress("glUniformBlockBinding");

glGenVertexArrays = (PFNGLGENVERTEXARRAYSPROC)glutGetProcAddress("glGenVertexArrays");

glBindVertexArray = (PFNGLBINDVERTEXARRAYPROC)glutGetProcAddress("glBindVertexArray");

glBindBufferRange = (PFNGLBINDBUFFERRANGEPROC)glutGetProcAddress("glBindBufferRange");

glDeleteVertexArrays = (PFNGLDELETEVERTEXARRAYSPROC)glutGetProcAddress("glDeleteVertexArrays");

glEnable(GL\_DEPTH\_TEST);

glClearColor(0, 0, 0, 0.0f);

//

// Uniform Block

//

glGenBuffers(1, &matricesUniBuffer);

glBindBuffer(GL\_UNIFORM\_BUFFER, matricesUniBuffer);

glBufferData(GL\_UNIFORM\_BUFFER, MatricesUniBufferSize, NULL, GL\_DYNAMIC\_DRAW);

glBindBufferRange(GL\_UNIFORM\_BUFFER, matricesUniLoc, matricesUniBuffer, 0, MatricesUniBufferSize); //setUniforms();

glBindBuffer(GL\_UNIFORM\_BUFFER, 0);

glEnable(GL\_MULTISAMPLE);

init2D();

return true;

}

void myTimer(int value) {

glutPostRedisplay();

glutTimerFunc(1000.0f / 60.0f, myTimer, 0);

}

void initsliderwindow() {

cv::namedWindow(WINDOW\_NAME, cv::WINDOW\_NORMAL);

cv::createTrackbar("tx：", WINDOW\_NAME, &tx, 2000, NULL);//滑动条创建

cv::createTrackbar("ty：", WINDOW\_NAME, &ty, 5000, NULL);

cv::createTrackbar("tz：", WINDOW\_NAME, &tz, 4000, NULL);

cv::createTrackbar("size of x：", WINDOW\_NAME, &scalex, 4000, NULL);//滑动条创建

cv::createTrackbar("size of y：", WINDOW\_NAME, &scaley, 4000, NULL);

cv::createTrackbar("size of z：", WINDOW\_NAME, &scalez, 4000, NULL);

cv::createTrackbar("Direction of rotation：", WINDOW\_NAME, &rotatexyz, 2, NULL);//滑动条创建

//cv::createTrackbar("rotate of y：", WINDOW\_NAME, &rotatey, 1,NULL);

//cv::createTrackbar("rotate of z：", WINDOW\_NAME, &rotatez, 1, NULL);

cv::createTrackbar("angle：", WINDOW\_NAME, &angle, 180, NULL);

}

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

//

// Main function

//

int main(int argc, char \*\*argv) {

initsliderwindow();

// GLUT initialization

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_MULTISAMPLE);

glutInitContextVersion(3, 3);

glutInitContextFlags(GLUT\_COMPATIBILITY\_PROFILE);

glutInitWindowPosition(100, 100);

windowWidth = cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH);

windowHeight = cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT);

glutInitWindowSize(windowWidth, windowHeight);

glutCreateWindow("Lighthouse3D - Assimp Demo");

// Callback Registration

//glutDisplayFunc(renderScene);

glutDisplayFunc(renderScene);

glutReshapeFunc(changeSize);

glutTimerFunc(1000.0f / 60.0f, myTimer, 0);

std::thread t1(camTimer);

//(1000.0f / 15.0f, camTimer, 0);

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

cap=cv::VideoCapture("F:/project/video\_20201103\_132337.mp4");

// Mouse and Keyboard Callbacks

glutKeyboardFunc(processKeys);

// Init GLEW

//glewExperimental = GL\_TRUE;

glewInit();

if (glewIsSupported("GL\_VERSION\_3\_3"))

printf("Ready for OpenGL 3.3\n");

else {

printf("OpenGL 3.3 not supported\n");

return(1);

}

loadModelFile((string)"modelToMarker.txt");

// Init the app (load model and textures) and OpenGL

if (!init())

printf("Could not Load the Model\n");

printf("Vendor: %s\n", glGetString(GL\_VENDOR));

printf("Renderer: %s\n", glGetString(GL\_RENDERER));

printf("Version: %s\n", glGetString(GL\_VERSION));

printf("GLSL: %s\n", glGetString(GL\_SHADING\_LANGUAGE\_VERSION));

// return from main loop

//glutSetOption(GLUT\_ACTION\_ON\_WINDOW\_CLOSE, GLUT\_ACTION\_GLUTMAINLOOP\_RETURNS);

program = setupShaders();

//setupShaders2D();

// GLUT main loop

glutMainLoop();

// cleaning up

textureIdMap.clear();

// clear myMeshes stuff 清除myMeshes内容

map<int, MyModel>::iterator it;

for (it = models.begin(); it != models.end(); it++)

{

MyModel currentModel = it->second;

for (unsigned int j = 0; j < currentModel.myMeshes.size(); ++j) {

glDeleteVertexArrays(1, &(currentModel.myMeshes[j].vao));

glDeleteTextures(1, &(currentModel.myMeshes[j].texIndex));

glDeleteBuffers(1, &(currentModel.myMeshes[j].uniformBlockIndex));

}

}

// delete buffers

glDeleteBuffers(1, &matricesUniBuffer);

exit(0);

};

/\*\*

int winWidth = 400, winHeight = 300;

//定义变量

int flag = 0;

int n = 0;

int tempX, tempY;

struct LineNode {

point3d point1;

point3d point2;

int x1;

int y1;

int z1;

int x2;

int y2;

int z2

}Line[1500];

void Initial(void) {

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

}

void ChangeSize(int w, int h) {

winWidth = w;

winHeight = h;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, winWidth, 0.0, winHeight);

}

void Display(void) {

glClear(GL\_COLOR\_BUFFER\_BIT);

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

int i = 0;

//绘制折线

for (i = 0; i < n; i++) {

glBegin(GL\_LINES);

glVertex3i(Line[i].x1, Line[i].y1,Line[i].z1);

glVertex3i(Line[i].x2, Line[i].y2, Line[i].z1);

glEnd();

}

if (flag == 1) {

glBegin(GL\_LINES);

glVertex2i(Line[i].x1, Line[i].y1);

//glVertex2i(tempX, tempY);

glVertex2i(Line[i].x2, Line[i].y2);

glEnd();

}

glutSwapBuffers();

}

//响应鼠标函数

void MousePlot(GLint button, GLint action, GLint xMouse, GLint yMouse) {

//左键点击

if (button == GLUT\_LEFT\_BUTTON && action == GLUT\_DOWN) {

if (flag == 0) {

flag = 1;

Line[n].x1 = xMouse;

Line[n].y1 = winHeight - yMouse;

//cout<<Line[n].x1<<" "<<Line[n].y1<<endl;

}

else {

//flag = 0;

Line[n].x2 = xMouse;

Line[n].y2 = winHeight - yMouse;

n++;

//

Line[n].x1 = Line[n - 1].x2;

Line[n].y1 = Line[n - 1].y2;

}

}

//右键点击

if (button == GLUT\_RIGHT\_BUTTON && action == GLUT\_DOWN) {

flag = 0;

n = 0;

glutPostRedisplay();

}

}

// 鼠标移动函数

void PassiveMouseMove(GLint xMouse, GLint yMouse) {

//

//if(flag == 1){

//tempX = xMouse;

//tempY = winHeight - yMouse;

Line[n].x2 = xMouse;

Line[n].y2 = winHeight - yMouse;

glutPostRedisplay();

//Display();

//}

}

//

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(400, 300);

glutInitWindowPosition(100, 100);

glutCreateWindow("drawline");

glutDisplayFunc(connectPoints);

glutReshapeFunc(ChangeSize);//回调函数

glutMouseFunc(MousePlot);//指定鼠标点击响应函数

glutPassiveMotionFunc(PassiveMouseMove);//指定鼠标移动响应函数

Initial();

glutMainLoop();

return 0;

}\*\*/

/\*\*

#include <GL/glut.h>

#include <cstdio>

#include <cmath>

const GLfloat Pi = 3.1415926536f;

//定义点集

struct data {

GLfloat x;

GLfloat y;

GLfloat z;

}Point[1500];

void init2() //初始化函数

{

glClearColor(1.0, 1.0, 1.0, 0.0); //设置背景颜色

//glMatrixMode(GL\_PROJECTION); // 设置投影参数

//gluOrtho2D(0.0, 50.0, 0.0, 50.0); // 设置场景的大小

}

void mydisplay()

{

int ptsSize = 1354;

point3d\* pt = assignBatchPoint();

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

glBegin(GL\_LINE\_STRIP);

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

{

glVertex3f(pt[i].x, pt[i].y, pt[i].z);

}

glEnd();

glFlush();

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0); //设置线条颜色

glPointSize(2); //设置点的大小

//glTranslatef(100.0f, 100.0f, 0.0f); //平移图形

glScalef(0.005f, 0.005f, 0.005f); //缩小图像0.5倍

//glRotatef(60.0f, 1.0f, 0.0f, 0.0f); //沿x轴旋转60度

glBegin(GL\_LINE\_STRIP);

for (int i = 1; i <= 1354; i++)

{

GLfloat t = i / 200.0;

pt[i].x = 50.0 \* cos(2.0 \* Pi \* t); //参数曲线

pt[i].y = 50.0 \* sin(2.0 \* Pi \* t); //参数曲线

pt[i].z = 100.0 \* t; //参数曲线

glVertex3i(pt[i].x, pt[i].y, pt[i].z); //绘制曲线

}

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowPosition(100, 100);

glutInitWindowSize(400, 400);

glutCreateWindow("绘制空间曲线");

init2();

glutDisplayFunc(&mydisplay);

glutMainLoop();

return 0;

}

\*\*/