**基于虚拟现实的心血管介入手术模拟系统的总结与感悟**

学生：徐鹏程 密西根学院

指导老师：顾力栩 生物医学工程学院

**个人承担工作：**

在此次大学生创新计划中，我主要担任的技术方面的工作有：初期的unity 的用户界面的编程部分和后期基于c++的程序的改进，例如在已有的显示模型的基础上使用openGL的库的功能增加一个可以移动的点用来表示传感器探针的位置。其中，c++的编程占据了工作比较重要的部分，经过学习OpenGL关于assmip的模型导入以及3d模型显示，回调函数，等知识，以及OpenCV中的aruco库的marker等相关知识，查阅相关文档，经历一段时间的试错之后，能够实现一个基本功能的demo。

在非技术方面，我也参与了用于参加比赛的宣传视频的拍摄和剪辑，同时也和组员们讨论润色演讲时使用的ppt，在最后的大学生创新计划研究论文中，我主要担任了：研究结果和讨论这两大部分的撰写，字数五千余字。

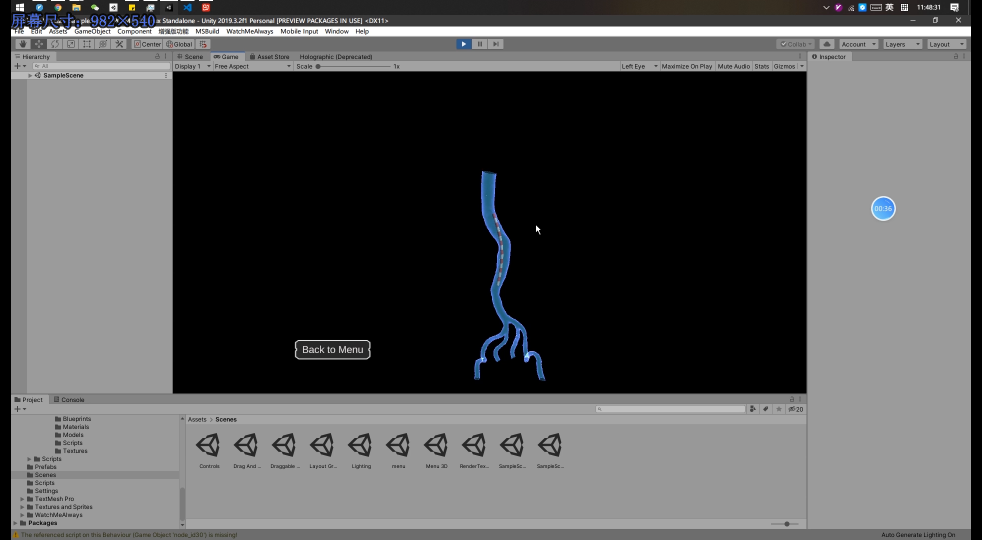
下面我将具体介绍我负责的技术和非技术方面的工作：

1. **基于Unity的血管建模**

**1.0职责介绍**

这部分中主要负责了用户界面的编程部分。

在血管三维显示部分，我们用unity平台进行软件开发，结合了hololens硬件的混合现实技术，来进行术前的手术规划图像展示，这样就可以让医生更清楚的了解位置信息，更精确的进行手术操作。

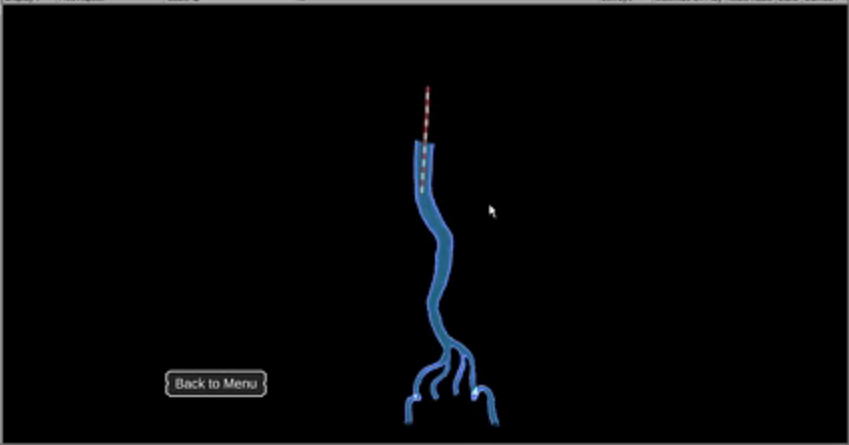


**1.1 血管的建模**

血管的建模来源于某开源网站的血管stl文件素材。此素材为心脏介入手术中主要涉及的血管。血管在此程序中为刚体。使其在坐标系中的位置不变，也无明显形变，便于后续操作。

**1.2 导管的模拟**

我们通过模拟导管的材质，使其硬度与现实中的导管一致，需要一定的力来使其扭转的变形，但不是刚体，具有一定的柔软程度。与血管材质不同。

**1.3 用户界面的逻辑**

**1.3.1 设置参数**

第一个选项是在电脑上进行的训练，第二个选项是结合增强现实器械的训练，第三个settings里有画面和声音的设置选项。我们打开第一个选项，画面中是一根仿真的导管和一个扫描得到的心血管模型，目前实现分别用六个按键控制导管在三个轴向上的位移，这是导管与血管碰撞的物理仿真结果，后续我们会加入旋转操作，并且和实际的外设配套，使他更接近现实中手术的使用。

**1.3.2 场景切换**

主要的场景切换包括开头的选项界面和之后的具体血管模型显示与介入手术训练界面。使用c#代码的场景切换来实现，场景切换的触发函数是按钮的点击onClickAction。在之后的设计中，我们将加入更多的实时反馈，例如方向引导，手术评价，来更好的起到手术的辅助与引导作用。

**2 基于C++中 OpenGL和OpenCV库实现软件图像空间坐标投映**

**2.0 职责介绍**

主要负责了传感器坐标的三维显示，支持传感器的旋转平移放缩在滑动条窗口调整。传感器的点被设置成了固定的id，在视频中出现对应marker的id时，就可以显示传感器的点的三维坐标，同时坐标可以随着时间变化而变化，这部分对于detectArucoMarkers(cv::Mat& image)的函数，在原有基础上增加了条件判断，用于改变marker的坐标，实现显示运动的传感器的效果。这一变化的点集主要由txt文件导入，这部分的point3d数据结构设计和导入文件存入数据结构的实现由本人完成，在之后也可支持实时传输点集，符合数据结构的point3d的定义即可。此外在探索时也走了一些弯路，某些尝试最后无法实现功能于是被舍弃，但也是探索项目的一部分，因此也在此介绍。在显示三维位置时，曾试图利用opengl内置的圆柱体函数，进行离散的点的首尾相接，来显示一个近似的三维血管图形，也支持了旋转，平移和放缩等操作，随后因为技术原因，renderscene函数中，试图在原函数基础上显示三维圆柱血管图像，但只能单独显示出其中一个，在权衡之下利用已有的marker的检测程序进行改进而最终得以通过marker的xyz向量的起点的移动来取代显示传感器的三维坐标变换，实现了功能需求。这一部分将在2.2.3介绍。

**2.1 软件介绍**

该软件主要的功能是输入一个视频和血管模型的stl文件以及传感器的实时移动路径的三维坐标，我们可以在视频中识别出aruco库对应的二维码，在视频的坐标系中以固定的的相对位移和相对旋转角度以及大小缩放，显示出整个血管的模型，同时，也在视频中显示出传感器在同一坐标系之下的实时运动。

**2.2 程序实现过程**

**2.2.1 类的声明**

|  |
| --- |
| struct MyModel |
| { |
| aiScene\* scene; // the global Assimp scene object for each model 每个模型的全局Assimp场景对象 |
| std::vector<cv::Mat> viewMatrix = { cv::Mat::zeros(4, 4, CV\_32F),cv::Mat::zeros(4, 4, CV\_32F),cv::Mat::zeros(4, 4, CV\_32F) }; |
| int seenlast = 0; |
| bool seennow = false; |
| Assimp::Importer importer; // Create an instance of the Importer class创建Importer类的实例 |
| std::vector<struct MyMesh> myMeshes; |
| float scaleFactor; // scale factor for the model to fit in the window 模型的比例因子以适合窗口 |
| int marker; //marker corresponding to this model 与此模型对应的标记 |
| }; |
|  |
| // Information to render each assimp node 渲染每个assimp节点的信息 |
| struct MyMesh |
| { |
| GLuint vao; |
| GLuint texIndex; |
| GLuint uniformBlockIndex; |
| int numFaces; |
| }; |
|  |
| // This is for a shader uniform block 这是用于着色器统一块 |
| struct MyMaterial |
| { |
| float diffuse[4]; |
| float ambient[4]; |
| float specular[4]; |
| float emissive[4]; |
| float shininess; |
| int texCount; |
| }; |
|  |
| class point3d { |
| public: |
| float x; |
| float y; |
| float z; |
| point3d() {} |
| void assignValue(float x, float y, float z); |
| ~point3d() {} |
| }; |
| //血管的点集 |
| point3d\* pt = new point3d[1400]; |

以上为主要的类的声明，包括：MyModel,MyMesh,MyMaterial,point3d。

MyModel主要储存模型的数据。

MyMesh主要渲染每个assimp节点的信息。

MyMaterial主要用于着色器统一块。

Point3d表示血管的点集。

**2.2.2 主函数逻辑**

接下来阐述主函数的逻辑。首先2452行的initsliderwindow()函数功能是加载滑动条的窗口，glutInit是OpenGL的GLUT初始化函数。2459~2481行正如注释所述，是加入的导入传感器坐标文件，把点的坐标赋值给pt[]这一变量，在之后的renderscene函数中将用到这一坐标来显示传感器的点的坐标。2483~2486行设置了glut的一些初始化参数。2489行glutInitWindowPosition()初始化了窗口的位置。2490~2493行windowWidth = cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH);分别根据视频的宽度和长度来设置窗口的长和宽。2494行为窗口命名。2500~2501行分别调用renderscene作为显示函数，以及调用changeSize作为改变大小的函数。GlutTimerFunc()按一定时间间隔触发的定时器回调函数。2505行std::thread t1(camTimer);调用多线程函数，回调函数为camTimer,此函数是递归函数，会不断地调用detectArucoMarkers(imageMat)来检查是否有二维码marker在视频中以及获取marker的相关位置、旋转等参数，而detectArucoMarkers(imageMat)的参数又会传给关于marker的全局变量，在renderscene中会不断地被调用来显示marker。2512~2513行glutMouseFunc(mouse); glutKeyboardFunc(processKeys);用于注册键盘和鼠标事件发生时的回调函数。2516行glutMotionFunc(motion);注册鼠标移动事件的回调函数，人机交互处理。2517行glutIdleFunc(&renderScene);在键盘按下时作某事。2522行初始化了glew，并在2522~2527行使用循环来检测是否支持OpenGLd 对应版本，若不支持则在命令行输出不支持的字样。

2529行的 loadModelFile((string)"modelToMarker.txt");的功能是根据某个txt文件来获取血管模型stl的文件名。这一函数通过文件流的方式循环地将文件中的marker序号保存进入vector markerIds[]中，而文件名字符串则保存在了modelMap[marker]这一std::map<int,string>类型的映射中。2533~2534行是当没有成功初始化模型时在命令行输出提示。

2536~2539行分别输出了Vender,Render,Version，GLSL等相应信息。2545行则初始化了着色器。在setupShaders()这一函数中，我们分别将p,v,f这三个Gluint类型的变量赋值给program,vertaxShader,fragmentShader。后两者为边着色器，片段着色器。

2549行是glut的主循环函数。在主循环函数之后的程序，都用于删除一些之前声明的数据结构。2551~2553行删除了pt[]，并关闭了fp文件指针，设置为空。2556行清楚了textureIDMap纹理序号的映射。而2558~2570利用循环，清楚了models这一模型文件中的边数组，纹理，缓冲等。最后2572删除缓冲。Exit(0)用于退出函数。函数如下所示。

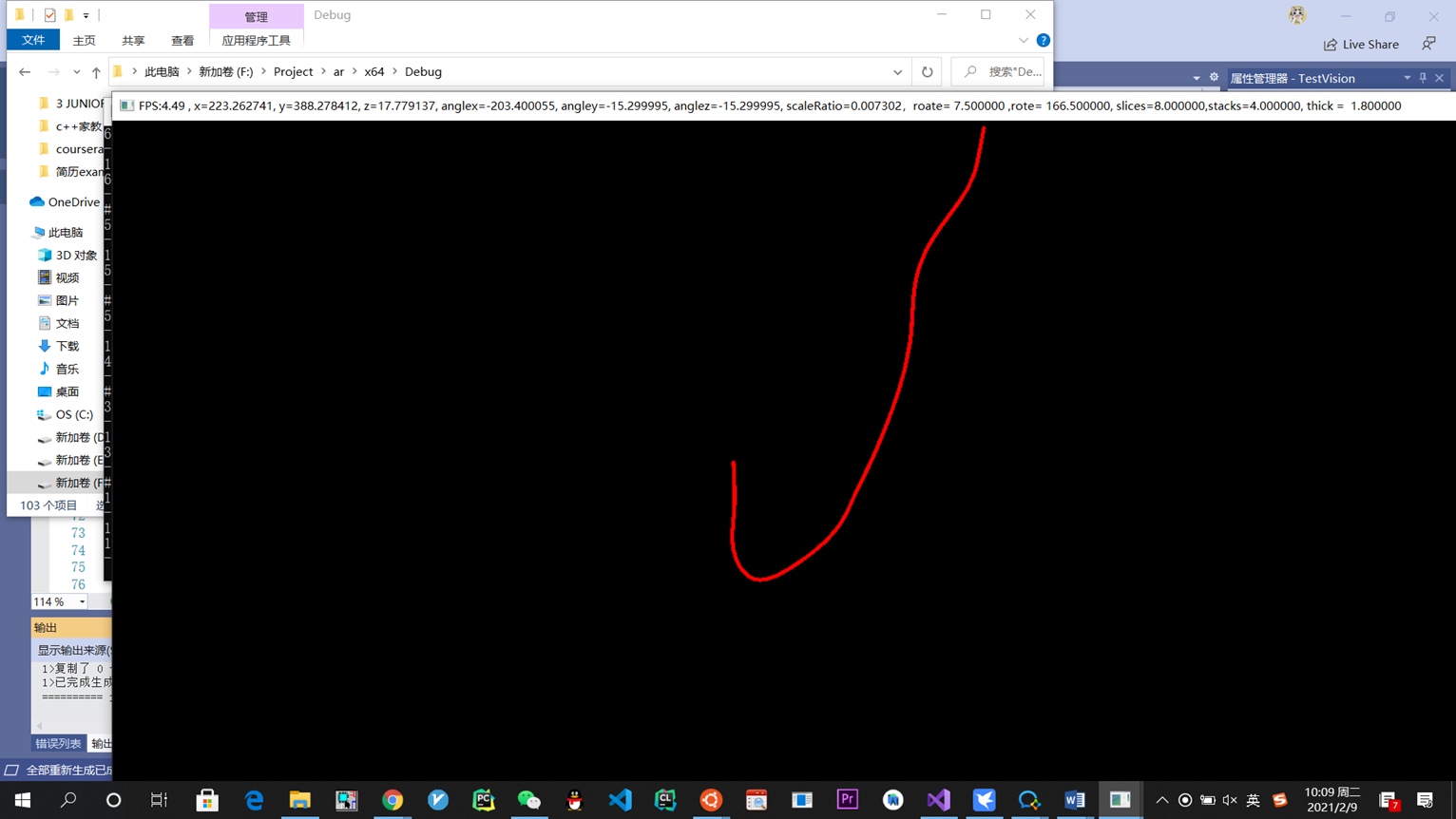
|  |
| --- |
| 1. int main(int argc, char\*\* argv){ |
|  |
|  |
| 1. initsliderwindow(); |
| 1. // GLUT initialization |
| 1. //init(); |
| 1. glutInit(&argc, argv); |
|  |
| 1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| 1. //导入文件 |
| 1. if ((fp = fopen("F:\\Project\\ar\\centerline.txt", "r")) == NULL) //文件路径需改为下面模拟数据所导入的文件路径 |
| 1. { |
| 1. printf("can't open this file"); |
| 1. exit(0); |
| 1. } |
| 1. //导入pt |
| 1. for (int count = 1; count < 1354; count++) { |
| 1. fscanf(fp, "%f%f%f", &pt[count].x, &pt[count].y, &pt[count].z); |
| 1. } |
| 1. //double k = 1.0; |
| 1. //给dx,dy,dz赋值 |
|  |
| 1. for (int i = 1; i < upperbound; i += di) { |
| 1. dx += pt[i].x; |
| 1. dy += pt[i].y; |
| 1. dz += pt[i].z; |
| 1. }dx = (double)dx / upperbound; |
| 1. dy = (double)dy / upperbound; |
| 1. dz = (double)dz / upperbound; |
|  |
| 1. dx = pt[630].x; |
| 1. dy = pt[630].y; |
| 1. dz = pt[630].z; |
| 1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| 1. glutInitDisplayMode(GLUT\_DEPTH | GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_MULTISAMPLE); |
|  |
| 1. glutInitContextVersion(3, 3); |
| 1. glutInitContextFlags(GLUT\_COMPATIBILITY\_PROFILE); |
|  |
|  |
| 1. glutInitWindowPosition(100, 100); |
| 1. windowWidth = cap.get(CV\_CAP\_PROP\_FRAME\_WIDTH); |
| 1. windowHeight = cap.get(CV\_CAP\_PROP\_FRAME\_HEIGHT); |
|  |
| 1. glutInitWindowSize(windowWidth, windowHeight); |
| 1. glutCreateWindow("Lighthouse3D - Assimp Demo"); |
|  |
|  |
| 1. // Callback Registration |
| 1. //glutDisplayFunc(renderScene); |
|  |
| 1. glutDisplayFunc(renderScene); |
| 1. glutReshapeFunc(changeSize); |
|  |
|  |
| 1. glutTimerFunc(1000.0f / 60.0f, myTimer, 0); |
| 1. std::thread t1(camTimer); |
| 1. //(1000.0f / 15.0f, camTimer, 0); |
|  |
| 1. //cap=cv::VideoCapture("D:/profiles/心血管比赛相关资料/video\_20201103\_132337.mp4"); |
| 1. //cap = cv::VideoCapture("F:/project/5-11-6.mp4"); |
| 1. // Mouse and Keyboard Callbacks |
| 1. glutMouseFunc(mouse);//用于注册键盘和鼠标事件发生时的回调函数 |
| 1. //glutKeyboardFunc(keyFunc); //血管部分的键盘函数 |
| 1. glutKeyboardFunc(processKeys); |
|  |
|  |
| 1. glutMotionFunc(motion);//注册鼠标移动事件的回调函数，人机交互处理 |
| 1. glutIdleFunc(&renderScene);//在键盘按下时作某事 |
|  |
| 1. // Init GLEW |
| 1. //glewExperimental = GL\_TRUE; |
| 1. glewInit(); |
| 1. if (glewIsSupported("GL\_VERSION\_3\_3")) |
| 1. printf("Ready for OpenGL 3.3\n"); |
| 1. else { |
| 1. printf("OpenGL 3.3 not supported\n"); |
| 1. return(1); |
| 1. } |
|  |
| 1. loadModelFile((string)"modelToMarker.txt"); |
|  |
|  |
| 1. // Init the app (load model and textures) and OpenGL |
| 1. if (!init()) |
| 1. printf("Could not Load the Model\n"); |
|  |
| 1. printf("Vendor: %s\n", glGetString(GL\_VENDOR)); |
| 1. printf("Renderer: %s\n", glGetString(GL\_RENDERER)); |
| 1. printf("Version: %s\n", glGetString(GL\_VERSION)); |
| 1. printf("GLSL: %s\n", glGetString(GL\_SHADING\_LANGUAGE\_VERSION)); |
|  |
|  |
| 1. // return from main loop |
| 1. //glutSetOption(GLUT\_ACTION\_ON\_WINDOW\_CLOSE, GLUT\_ACTION\_GLUTMAINLOOP\_RETURNS); |
|  |
| 1. program = setupShaders(); |
| 1. //setupShaders2D(); |
|  |
| 1. // GLUT main loop |
| 1. glutMainLoop(); |
|  |
| 1. delete[] pt; |
| 1. fclose(fp); |
| 1. fp = NULL; |
|  |
| 1. // cleaning up |
| 1. textureIdMap.clear(); |
|  |
| 1. // clear myMeshes stuff 清除myMeshes内容 |
| 1. map<int, MyModel>::iterator it; |
| 1. for (it = models.begin(); it != models.end(); it++) |
| 1. { |
| 1. MyModel currentModel = it->second; |
|  |
| 1. for (unsigned int j = 0; j < currentModel.myMeshes.size(); ++j) { |
| 1. glDeleteVertexArrays(1, &(currentModel.myMeshes[j].vao)); |
| 1. glDeleteTextures(1, &(currentModel.myMeshes[j].texIndex)); |
| 1. glDeleteBuffers(1, &(currentModel.myMeshes[j].uniformBlockIndex)); |
| 1. } |
| 1. } |
|  |
| 1. // delete buffers |
| 1. glDeleteBuffers(1, &matricesUniBuffer); |
|  |
| 1. exit(0); |
|  |
| 1. } |

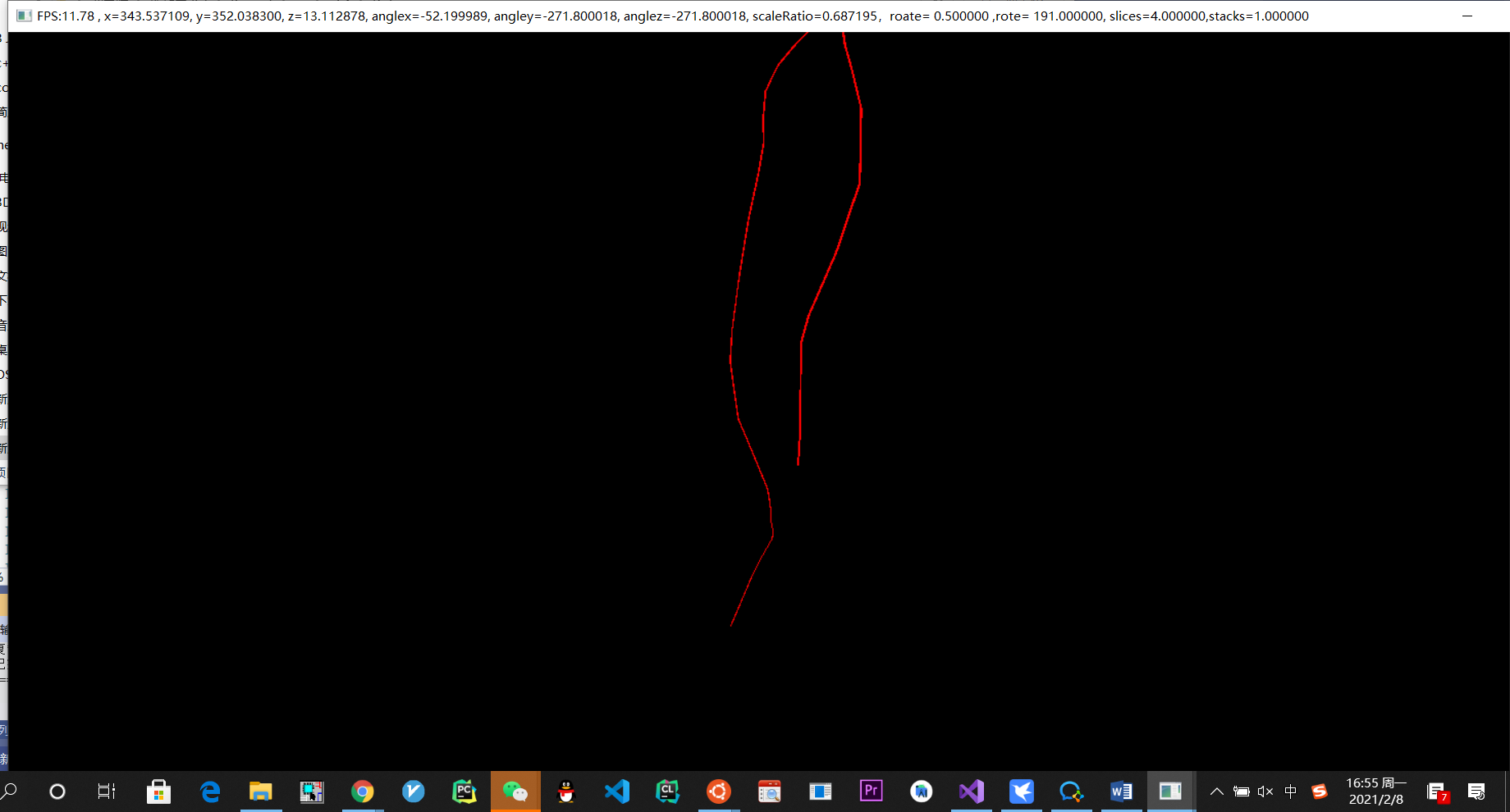
**2.2.3 renderbone和myDisplay 渲染场景函数实现**

|  |
| --- |
| void RenderBone(float x0, float y0, float z0, float x1, float y1, float z1, GLdouble radius, GLdouble slices, GLdouble stack) |
| { |
| GLdouble dir\_x = x1 - x0; |
| GLdouble dir\_y = y1 - y0; |
| GLdouble dir\_z = z1 - z0; |
| GLdouble bone\_length = sqrt(dir\_x \* dir\_x + dir\_y \* dir\_y + dir\_z \* dir\_z); |
| static GLUquadricObj\* quad\_obj = NULL; |
| if (quad\_obj == NULL) |
| quad\_obj = gluNewQuadric(); |
| gluQuadricDrawStyle(quad\_obj, GLU\_FILL); |
| gluQuadricNormals(quad\_obj, GLU\_SMOOTH); |
| glPushMatrix(); |
| // 平移到起始点 |
| glTranslated(x0, y0, z0); |
| // 计算长度 |
| double length; |
| length = sqrt(dir\_x \* dir\_x + dir\_y \* dir\_y + dir\_z \* dir\_z); |
| if (length < 0.0001) { |
| dir\_x = 0.0; dir\_y = 0.0; dir\_z = 1.0; length = 1.0; |
| } |
| dir\_x /= length; dir\_y /= length; dir\_z /= length; |
| GLdouble up\_x, up\_y, up\_z; |
| up\_x = 0.0; |
| up\_y = 1.0; |
| up\_z = 0.0; |
| double side\_x, side\_y, side\_z; |
| side\_x = up\_y \* dir\_z - up\_z \* dir\_y; |
| side\_y = up\_z \* dir\_x - up\_x \* dir\_z; |
| side\_z = up\_x \* dir\_y - up\_y \* dir\_x; |
| length = sqrt(side\_x \* side\_x + side\_y \* side\_y + side\_z \* side\_z); |
| if (length < 0.0001) { |
| side\_x = 1.0; side\_y = 0.0; side\_z = 0.0; length = 1.0; |
| } |
| side\_x /= length; side\_y /= length; side\_z /= length; |
| up\_x = dir\_y \* side\_z - dir\_z \* side\_y; |
| up\_y = dir\_z \* side\_x - dir\_x \* side\_z; |
| up\_z = dir\_x \* side\_y - dir\_y \* side\_x; |
| // 计算变换矩阵 |
| GLdouble m[16] = { side\_x, side\_y, side\_z, 0.0, |
| up\_x, up\_y, up\_z, 0.0, |
| dir\_x, dir\_y, dir\_z, 0.0, |
| 0.0, 0.0, 0.0, 1.0 }; |
| glMultMatrixd(m); |
| // 圆柱体参数 |
| /\*GLdouble radius = radius; // 半径 |
| GLdouble slices = slices; // 段数 |
| GLdouble stack = stack; // 递归次数\*/ |
| gluCylinder(quad\_obj, radius, radius, bone\_length, slices, stack); |
| glPopMatrix(); |
| } |

以上函数的作用是输入两个点的三维坐标，显示和一个圆柱体连接这两个点。那么如果我们将足够多的点首尾相接，那么就可以显示出一个近似的复杂管状物体，这适合用于血管的三维图像的显示。以下是函数在renderscene中的具体实现。

|  |
| --- |
| void myDisplay()//单独显示血管线 |
| { |
| glClear(GL\_COLOR\_BUFFER\_BIT); |
| glColor3f(1.0, 0.0, 0.0); //画笔红色 |
| glLoadIdentity(); //加载单位矩阵 |
| gluLookAt(0.0, 0.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); |
| glRotatef(rote, 0.0f, 1.0f, 0.0f);// |
| glRotatef(anglex, 1.0, 0.0, 0.0); |
| glRotatef(angley, 0.0, 1.0, 0.0); |
| glRotatef(anglez, 0.0, 0.0, 0.0); |
|  |
| point3d\* pt = new point3d[1400]; |
|  |
| FILE\* fp = nullptr; |
| if ((fp = fopen("F:\\Project\\ar\\centerline.txt", "r")) == NULL) //文件路径需改为下面模拟数据所导入的文件路径 |
| { |
| printf("can't open this file"); |
| exit(0); |
| } |
| int count = 1; |
| for (int count = 1; count < 1354; count++) { |
| fscanf(fp, "%f%f%f", &pt[count].x, &pt[count].y, &pt[count].z); |
| } |
| double k = 1.0; |
| int di = 2; |
|  |
| double dx = 0.0, dy = 0.0, dz = 0.0; |
| for (int i = 1; i < upperbound; i += di) { |
| dx += pt[i].x; |
| dy += pt[i].y; |
| dz += pt[i].z; |
| }dx = (double)dx / upperbound; |
| dy = (double)dy / upperbound; |
| dz = (double)dz / upperbound; |
|  |
| dx = pt[1].x; |
| dy = pt[1].y; |
| dz = pt[1].z; |
|  |
| for (int i = 1; i < upperbound; i += di) { |
| std::cout << "#" << i << "-th : " << pt[i].x << "\n" << pt[i].y << "\n" << pt[i].z << "\n" << pt[i + 1].x << "\n" << pt[i + 1].y << "\n" << pt[i + 1].z << "\n"; |
| RenderBone((pt[i].x - dx) \* ratio2, (pt[i].y - dy) \* ratio2, (pt[i].z - dz) \* ratio2, (pt[i + di].x - dx) \* ratio2, (pt[i + di].y - dy) \* ratio2, (pt[i + di].z - dz) \* ratio2, 0.3 \* ratio2, 4.0, 1.0); |
| } |
| glColor3f(0.0, 0.0, 1.0); //画笔红色 |
| rote += roate; |
| glutSwapBuffers();//交换缓冲区，使绘制的图形得以展示 |
| delete[] pt; |
| fclose(fp); |
| fp = NULL; |
| } |
| 以下是实验中，单独导入血管中心线的数据，显示出的血管三维图像： |





**2.2.4 renderscene 渲染场景函数实现**

1141~1144行更新timePin作为传感器坐标的时间戳。1146行给模型的ID赋值，这里0或1代表血管模型或者探针随后1146~1152行清除帧缓冲区（颜色和深度）。1155~1163行是对于2d的视频的显示模型。1169~1212行是显示模型的程序。首先清空gl\_depth\_buffer，然后利用循环遍历每个模型，每一步循环中利用setCamera(currentModel.viewMatrix[0]);设置摄像机的旋转矩阵。setIdentityMatrix(modelMatrix, 4);将模型矩阵设置为恒等矩阵。

scale((double)scalex[ModelNo] / 2000 \* currentModel.scaleFactor, (double)scaley[ModelNo] / 2000 \* currentModel.scaleFactor, (double)scalez[ModelNo] / 2000 \* currentModel.scaleFactor);这一步模型矩阵设置为比例矩阵，以便模型适合窗口。

而rotate( (double)angle[ModelNo] \* 1.0f , rotatex[ModelNo] \* 1.0f, rotatey[ModelNo] \* 1.0f, rotatez[ModelNo] \* 1.0f);这一函数的作用是旋转模型。

glUseProgram(program);使用我们的着色器。recursive\_render(currentModel.scene, currentModel.scene->mRootNode, currentModel.myMeshes);这一函数将递归地调用自己，作用是遍历每一个节点每一个mesh网络，最后实现整个模型的渲染。最后交换缓冲区，使绘制的图形得以展示。

**2.2.5 OpenGL导入assimp模型**

导入模型的函数主要是init()函数中的loadModels()函数和主函数中的loadModelFile()函数来实现。首先，我们的血管模型是基于三维打印的模型扫描之后的stl文件，保存的目录是209行的modelDir参数，模型如下图所示。



LoadModelsFile()函数的输入是字符串，字符串对应的是一个txt文件名，txt文件中储存了若干模型的obj文件名和对应的markerID如0,1等。这一程序将模型名和id这两个字符串存入了markerIds[]这一vector<string>类型，和modelMap[marker]这一映射。

**2.2.6 滑动条与模型旋转平移放缩变换的交互**



滑动条主要改变的是血管模型和传感器的平移、旋转和缩放的变换。

主要实现的函数是1630~1656行的void initsliderwindow() 涉及的参数有，tx[],ty[],tz[]表示平移变换，数组下标是模型序号。滑动条中显示的size of x/y/z 在程序中的变量为scalex[],scaley[],scalez[]，表示缩放的系数。Angle[]表示旋转的角度，区间在[0,360°]之间。而滑动条汇总的direction，有0,1,2三个可选值，表示朝向分别朝向x,y,z轴方向，在程序中的参数为rotatexyz[]和rotatex[], rotatey[], rotatez[]。

在这部分中，前期的实现主要由杨德坤同学实现，实现了单独控制血管模型的滑动条界面，我在他的基础上增加了对应的传感器的旋转和平移和缩放变换的参数，将tx, ty, tz, scalex, scaley, scalez, angle, rotatexyz, rotatex, rotatey, rotatez拓展成了数组，今后也有空间加入更多模型的变换。

**2.2.7 OpenCV的aruco库实现增强现实marker识别**





利用aruco库的marker识别是占据程序的重要部分。最核心的函数为void detectArucoMarkers(cv::Mat& image)这一函数。我们递归地探测视频，在视频中寻找和二维码匹配的图像区域，随后经过调用cv::aruco::detectMarkers(image, dictionary, markerCorners, markerIds, detectorParams, rejectedCandidates);这一函数我们把得到的marker的id和位置旋转等信息存储在markerIDs[]，markerCorners[]等变量中。如果我们markerIds的大小大于0，说明探测到了至少一个marker，随后我们调用cv::aruco::estimatePoseSingleMarkers()这一函数识别估计了单个的marker的位置。

这一程序主要来源于杨德坤同学和opencv的开源库和github和的贡献，我在此基础上增加了2.2.7的功能，使得特定的传感器可以根据时间来变换三维坐标。

**2.2.8 运动的传感器的坐标模拟**

以下的程序部分主要由本人实现。

传感器的实时坐标主要存储在了pt[]的point3d类型中，包括x,y,z的坐标成员变量。而在renderscene中，我们用timePin这一全局变量来表示时间，每次调用renderscene之后都会更新timePin变量，可作为pt[]的下标来调用，做到实时调用三维坐标的效果。

而pt[]的值主要用于marker的坐标的相对位移，用于void detectArucoMarkers(cv::Mat& image)这一函数中，从910行开始。主要更改的逻辑集中于961~965行，当markerIds[i] 等于我们设置的pinID时，说明这一marker是我们设置的传感器marker，于是我们在tvecs[pinID][0]/[1]/[2]这三个变量中加上相应的与(double)pt[timePin].x/y/z有关的偏移量，表示传感器marker会平移对应的xyz坐标值。这一坐标值的大小，偏移和旋转同样可以通过滑动条改变参数。

2.3 程序演示

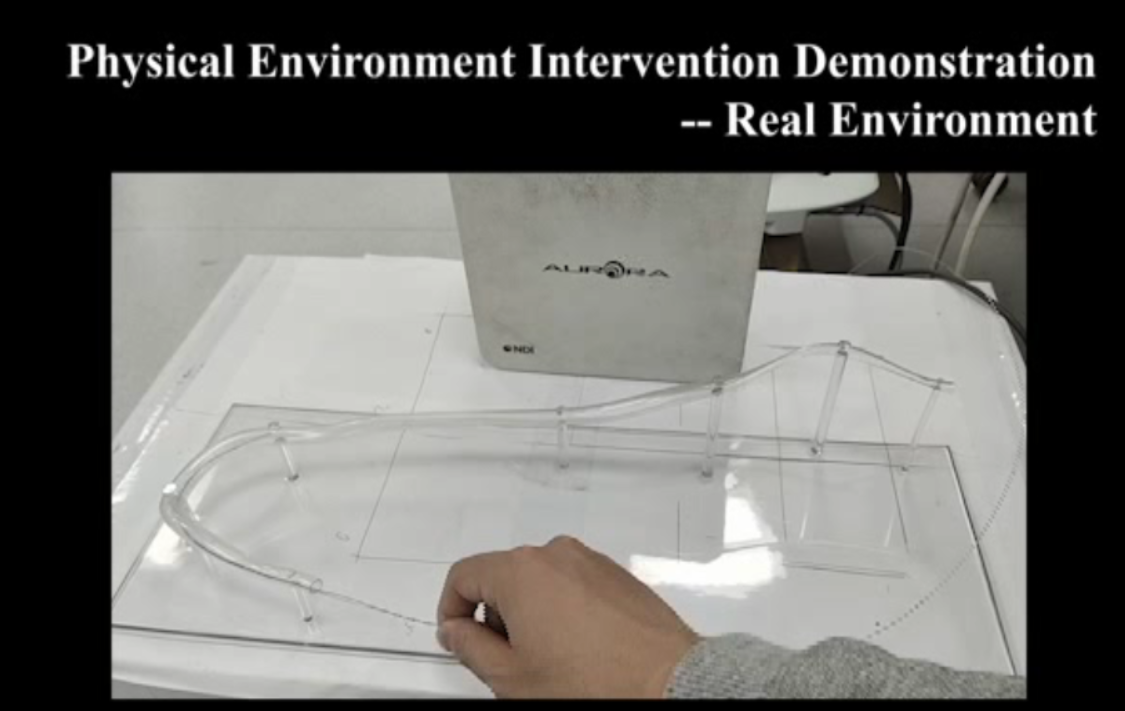


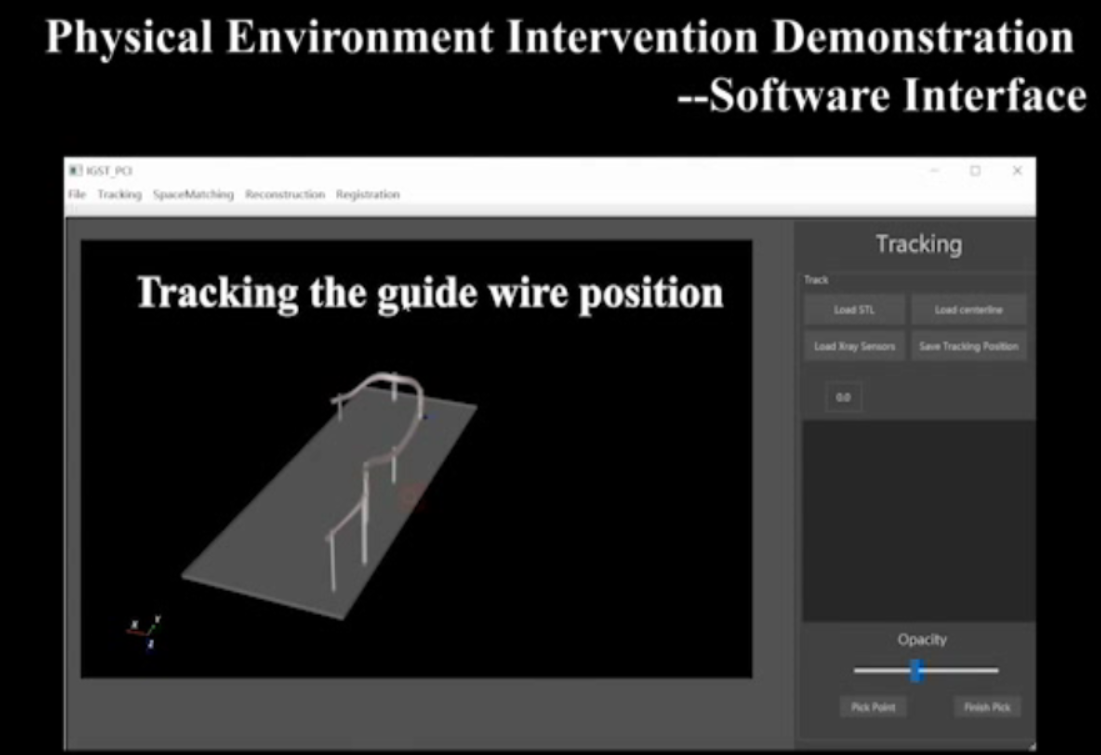
由此可见，这一程序同时显示了左边的滑动条窗口和右边的视频窗口。左边滑动条可以调整血管模型以及传感器的的旋转、平移和放缩的参数。而右边视频窗口首先显示了一个摄像头拍摄出的视频，在此为背景，我们识别出了类似二维码的marker标识。图片中包含两个marker标识，分别对应的id的0和1，id为0的marker用于标定显示血管模型，id为1的marker用于标定显示传感器的运动轨迹。

此部分最后的视频版本由本人试验拍摄并在软件中调用并录屏，（前期的视频由杨德坤参与拍摄）

**3.非技术贡献：**

视频剪辑：（部分页面）







**体验与收获：**

这次项目经历锻炼了我在一个科研项目中积极试错，找寻最优解，和解决意料之外问题的能力，对工程能力有了不小的提高，对编程技术方面也增加了经验。

关于积极试错的部分，很多工作并不是一蹴而就的，在探索时也走了一些弯路，某些尝试最后无法实现功能于是被舍弃，但也是探索项目的一部分。在显示三维位置时，曾试图利用opengl内置的圆柱体函数，进行离散的点的首尾相接，来显示一个近似的三维血管图形，也支持了旋转，平移和放缩等操作，随后因为技术原因，renderscene函数中，试图在原函数基础上显示三维圆柱血管图像，但只能单独显示出其中一个，在权衡之下利用已有的marker的检测程序进行改进而最终得以通过marker的xyz向量的起点的移动来取代显示传感器的三维坐标变换，实现了功能需求。这一试错的过程也并不是一无所获，至少已经积累了导入中心线的程序，这一部分的程序在之后也得到了应用。只是在试错中找到了一条难以实现的道路，这时转换思路，试图通过已有的知识体系和技术，查阅可靠的官方文档，可能可以找到更实际可行的路线，这一经历也给了我一些启发。

关于技术方面的解决问题的能力，我觉得可以用最后实现demo的例子来说明。最后拍摄视频时会经常遇到一些细小问题，例如光线问题导致marker的识别不准确，拍摄角度问题导致最后程序显示模型无法很好地贴合视频中的实际模型。我积极的查阅资料，不断的修改拍摄角度和光线原因，以及借助程序的某些算法上的调整，最后实现了一个满足要求的demo。可以说这一过程是曲折的但也是受益匪浅的。

关于团队合作的能力的培养。这是一个团队项目，从一开始我团队的三人以及带领我们的研究生杨智凯就经常开会讨论下一步的项目的方向以及改进空间，在观点的对撞中，我们收获了不少有价值的改进方向。同时，我们也参与了多次比赛，在共同合作向同一个目标迈进时，也收获了友谊和合作的满足感。

总之，这次的大学生创新实践项目，切实的让我有了一段长期科研的经历，培养了我科研方面的探索、积极试错能力，工程方面利用技术解决问题的能力，和团队合作的能力，让我收获颇丰，非常感谢导师和所有队友和研究生学长在项目期间的指导、帮助和支持，在我们共同努力下较顺利地完成了这一项目。