**基于草图特征的3D物体表面交互建模v1.0**

# 代码

## 1 GUI设计逻辑 GUI\_QT.py

# 界面主程序

import os

import sys

import cv2

import configparser

import numpy as np

import vtk

from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

import open3d as o3d

import pyqtgraph.opengl as gl

from PyQt5.QtWidgets import QApplication,QMainWindow, QWidget,QDesktopWidget,QFrame, QPushButton,QLabel ,QFileDialog, QColorDialog, QSlider, QToolTip, QLCDNumber

from PyQt5.QtGui import QPainter, QPixmap ,QPen, QColor, QFont

from PyQt5.QtCore import Qt, QPoint

from PyQt5 import QtWidgets, QtGui

from qt\_material import apply\_stylesheet

from pyqtgraph.opengl import GLViewWidget

from tools import read10pics,parseObj,GLPainterItem

from model import do\_predict

conf = configparser.ConfigParser()

conf.read('config.ini', encoding="utf-8-sig")

class OpencvMethods():

    def \_\_init\_\_(self,pix\_width,pix\_height):

        self.points = []

        self.height=pix\_height

        self.width=pix\_width

        self.dms = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # 2dmask

        self.ctr = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # clmask轮廓线

        self.ds  = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # ds深度信息

        self.fm  = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # 锐利掩码

        self.cur = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # 曲率提示

        self.inf = np.zeros((pix\_width, pix\_height, 1), dtype=np.float32)  # 内部特征线

    # 灰度图转二值图

    def gray2Blackwhite(self,img):

        #二值化

        img1=img.copy()

        thresh = 1

        img1[img1 > thresh] = 255

        img1[img1 <= thresh] = 0

        return img1

    # 背景像素灰度由change\_gray替换为target\_gray

    def clearcurvMagBKG2gray(self,img,target\_gray=126,change\_gray=0):

        img1=img.copy()

        img1[img1 == change\_gray] = target\_gray

        return img1

    #连线算法：将图片上点连接起来

    #points: 点集合 img: 图片  isClosed:是否闭合 0 不闭合 1 闭合 color：颜色

    def point2line(self,points,img, isClosed=0,color=255,thickness=1):

        a=np.array(points)

        cv2.polylines(img,  np.int32([a]), isClosed, color, thickness)

        return img

    # 清空变量及初始化显示图片

    def clearAllPictures(self):

        # 成员变量清空

        self.dms = np.zeros((self.width, self.height, 1), dtype=np.float32)  # 2dmask

        self.ctr = np.zeros((self.width, self.height, 1), dtype=np.float32)  # clmask轮廓线

        self.ds  = np.zeros((self.width, self.height, 1), dtype=np.float32)  # ds深度信息

        self.fm  = np.zeros((self.width, self.height, 1), dtype=np.float32)  # 锐利掩码

        self.cur = np.zeros((self.width, self.height, 1), dtype=np.float32)  # 曲率提示

        self.inf = np.zeros((self.width, self.height, 1), dtype=np.float32)  # 内部特征线

        # 图片初始化

        black = np.zeros((self.width, self.height, 1), dtype=np.float32)

        white = np.full ((self.width, self.height, 1), 255, dtype=np.float32)

        grayd = np.full ((self.width, self.height, 1), 126, dtype=np.float32)

        # 偷懒了 反正又看不出来 干嘛写规则

        cv2.imwrite("input\\2dMask.png",black)

        cv2.imwrite("input\\contour.png",white)

        cv2.imwrite("input\\shapeMask.png",black)

        cv2.imwrite('input\\ds.png', black)

        cv2.imwrite('input\\dsMask.png', black)

        cv2.imwrite('input\\fLMask.png', black)

        cv2.imwrite('input\\fLInvMask.png', white)

        cv2.imwrite('input\\curvMag.png',grayd)

        cv2.imwrite('input\\sLMask.png', black)

        cv2.imwrite('input\\infetr.png',white)

        cv2.imwrite('input\\npr.png',white)

        cv2.imwrite('input\\clIMask.png',white)

    # 画dmask&shapemask&contour

    def generateDSC(self,points,dmask,contour):

        a=np.array(points)

        cv2.fillPoly(dmask, np.int32([a]) ,color=255)                              # 黑色背景 用白色填充

        cv2.imwrite("input\\2dMask.png",dmask)

        cv2.polylines(contour,  np.int32([a]), isClosed=0, color=255, thickness=4) # 黑底白线 宽度为4

        contour=255-contour                                                        # 白底黑线

        cv2.imwrite("input\\contour.png",contour)

        shapemask=dmask-(255-contour)                                              # 黑色背景 用白色填充

        cv2.imwrite("input\\shapeMask.png",shapemask)

    # 画npr&clmask

    def generateNC(self):

        # npr 白底灰线

        # 先把曲率图由灰底灰线转换为白底灰1线 用255减之成为黑底灰2线(灰1+灰2线等于255) 黑底灰2线与其他黑底白线相加 最后再被255减掉 这样曲率图中灰度提示和npr中的就能保持一致了

        cv2.imwrite('input\\npr.png',  255-((self.gray2Blackwhite(self.inf))+      # 黑底白线

                                                (self.gray2Blackwhite(self.ctr))+  # 黑底白线

                                                (self.gray2Blackwhite(self.fm))+   # 黑底白线

                                                (255-self.clearcurvMagBKG2gray(self.cur,target\_gray=255,change\_gray=0)))) #黑底灰线

        # clmask 白底黑线

        cv2.imwrite('input\\clIMask.png',  255-((self.gray2Blackwhite(self.inf))+

                                                   (self.gray2Blackwhite(self.ctr))))

    # 画其他线

    def generateOtherLines(self,mode,gray,thickness):

        # dmask&shapemask&contour

        if(mode==1):

            self.generateDSC(self.points,self.dms,self.ctr)

        # ds&dsmask

        elif(mode==2):

            self.point2line(self.points,self.ds,0,gray,thickness)

            cv2.imwrite('input\\ds.png', self.ds)                                    # 黑底灰线

            cv2.imwrite('input\\dsMask.png', self.gray2Blackwhite(self.ds))          # 黑底白线

        # fLInvMask&fLMask

        elif(mode==3):

            self.point2line(self.points,self.fm,0,gray,thickness)

            cv2.imwrite('input\\fLMask.png', self.gray2Blackwhite(self.fm))          # 黑底白线

            cv2.imwrite('input\\fLInvMask.png', 255-self.gray2Blackwhite(self.fm))   # 白底黑线

        # curvMag&sLMask

        elif(mode==4):

            self.point2line(self.points,self.cur,0,gray,thickness)

            cv2.imwrite('input\\curvMag.png', self.clearcurvMagBKG2gray(self.cur))    # 灰底(126)灰线 灰度值低于126为负曲率 纯黑色曲率值=-2；灰度值高于126为正曲率 纯白色曲率值=2

            cv2.imwrite('input\\sLMask.png', self.gray2Blackwhite(self.cur))          # 黑底白线

        # infetr

        elif(mode==5):

            self.point2line(self.points,self.inf,0,gray,thickness)

            cv2.imwrite('input\\infetr.png',255-self.gray2Blackwhite(self.inf))       # 白底黑线

class Winform(QMainWindow):

    def \_\_init\_\_(self, parent=None):

        super(Winform, self).\_\_init\_\_(parent)

        #QSS显示模式

        apply\_stylesheet(app, theme=conf['GUI\_QT']['QSS\_THEME'])

        #设置标题

        self.setWindowTitle("3D交互建模")

        #实例化QPixmap类

        self.pix = QPixmap()

        #起点，终点

        self.lastPoint = QPoint()

        self.endPoint = QPoint()

        self.win\_width  =   1900    # 窗口长宽

        self.win\_height =   900

        self.pix\_width  =   900     # 交互窗口长宽(相等)

        self.pix\_height =   900

        self.color      =   QColor(255,0,0)  # 默认画笔颜色及粗细

        self.thickness  =   4

        self.show\_mode  =   int(conf['GUI\_QT']['3D\_SHOW\_STYLE'])       # 最终3D图形显示模式 1--gl散点图 2--open3d 3--gl网格图 4--vtk图

        self.mode=0                 # （debugger用）当前绘画模式 0(无 默认) 1(2dmask shapeMask 轮廓线contour) 2(ds dsMask) 3(fLInvMask fLMask) 4(curvMag sLMask) 5(infetr)

        self.opencv=OpencvMethods(self.pix\_width,self.pix\_height)

        # 点云显示控件

        self.graphicsView = GLViewWidget(self)

        self.graphicsView.setCameraPosition(distance=0.5,azimuth=100)#初始视距 方位角

        self.graphicsView.setBackgroundColor([0,0,0])#设置背景色

        # 网格基准线

        self.grid = gl.GLGridItem()

        self.grid.scale(0.05,0.05,1)

        # 视角信息显示

        self.paintitem = GLPainterItem()

        # 字体显示

        self.txt = gl.GLTextItem()

        # vtk显示控件

        self.vtkWidget = QVTKRenderWindowInteractor(self)

        self.render = vtk.vtkRenderer()

        self.render.SetBackground(0.01, 0.2, 0.01)# 绿色显示背景

        #self.render.SetBackground(1.0,1.0,1.0)# 黑色背景

        self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

        self.iren = self.vtkWidget.GetRenderWindow().GetInteractor()

        #初始化

        self.initUi()

    def initUi(self):

        self.setFixedSize(self.win\_width, self.win\_height)  # 固定窗口大小，不可缩放

        self.center()                                       # 适应不同电脑屏幕，程序初始化主窗口居中

        # 画布大小，背景为白色

        self.pix = QPixmap(self.pix\_width, self.pix\_height)

        self.pix.fill(Qt.white)

        # 偏移量，保证鼠标的位置和画的线点是重合的

        self.offset = QPoint(self.width() - self.pix.width(), self.height() - self.pix.height())

        QToolTip.setFont(QFont('SansSerif', 5))# 按钮消息提示框

        self.statusBar().showMessage('就绪')   # 状态栏，在窗口左下角输出信息

        btn\_clear = QPushButton(self)

        btn\_clear.setText("模板")

        btn\_clear.setToolTip('点击以导入模板')

        btn\_clear.resize(80, 30)

        btn\_clear.move(10, 30)

        btn\_clear.clicked.connect(self.clearStokes)

        sld = QSlider(Qt.Horizontal, self)

        lcd = QLCDNumber(self)

        sld.setToolTip('滑动改变画笔粗细')

        lcd.setToolTip('画笔粗细显示器')

        sld.setRange(1,10)

        sld.setFocusPolicy(Qt.NoFocus)

        sld.resize(80, 30)

        sld.move(10, 80)

        lcd.resize(80, 30)

        lcd.move(10, 130)

        sld.valueChanged[int].connect(self.changeThickness)

        sld.valueChanged.connect(lcd.display)

        btn\_color = QPushButton(self)

        btn\_color.setText("颜色")

        btn\_color.setToolTip('点击改变画笔颜色')

        btn\_color.resize(80, 30)

        btn\_color.move(10, 180)

        btn\_color.clicked.connect(self.changePenColor)

        # 当前颜色灰度

        self.lblpen = QLabel(self)

        self.lblpen.move(0,220)

        gray = self.getGray(self.color)

        self.lblpen.setText('颜色灰度:'+str(gray))

        # 当前画笔颜色显示框

        lbl\_color = QLabel(self)

        lbl\_color.setText("当前画笔颜色")

        lbl\_color.move(10,260)

        self.color\_label = QLabel(self)

        self.color\_label .resize(50,50)

        self.color\_label.move(30,300)

        styleSheet = 'QLabel{background-color: ' + self.color.name()+ '; border:1px solid 0; font: 16px;color: black;}'

        self.color\_label.setStyleSheet(styleSheet)

        btn\_save1 = QPushButton(self)

        btn\_save1.setText("DSC")

        btn\_save1.setObjectName("开始绘制3图");

        btn\_save1.setToolTip('点击以绘制2dmask shapeMask 轮廓线contour(clmask第一部分) 松开左键绘制完成')

        btn\_save1.resize(80, 30)

        btn\_save1.move(10, 370)

        btn\_save1.clicked.connect(self.buttonEvent)

        btn\_save2 = QPushButton(self)

        btn\_save2.setText("DDM")

        btn\_save2.setObjectName("保存2");

        btn\_save2.setToolTip('先点击一次，绘制完成后，再点击以生成ds dsMask')

        btn\_save2.resize(80, 30)

        btn\_save2.move(10, 420)

        btn\_save2.clicked.connect(self.buttonEvent)

        btn\_save3 = QPushButton(self)

        btn\_save3.setText("FFL")

        btn\_save3.setObjectName("保存3");

        btn\_save3.setToolTip('先点击一次，绘制完成后，再点击以生成fLInvMask fLMask')

        btn\_save3.resize(80, 30)

        btn\_save3.move(10, 470)

        btn\_save3.clicked.connect(self.buttonEvent)

        btn\_save4 = QPushButton(self)

        btn\_save4.setText("CS")

        btn\_save4.setObjectName("保存4");

        btn\_save4.setToolTip('先点击一次，绘制完成后，再点击以生成curvMag sLMask')

        btn\_save4.resize(80, 30)

        btn\_save4.move(10, 520)

        btn\_save4.clicked.connect(self.buttonEvent)

        btn\_save5 = QPushButton(self)

        btn\_save5.setText("infetr")

        btn\_save5.setObjectName("保存5");

        btn\_save5.setToolTip('先点击一次，绘制完成后，再点击以生成infetr(内部特征线 clmask第二部分)')

        btn\_save5.resize(80, 30)

        btn\_save5.move(10, 570)

        btn\_save5.clicked.connect(self.buttonEvent)

        btn\_save6 = QPushButton(self)

        btn\_save6.setText("CPR")

        btn\_save6.setObjectName("保存6");

        btn\_save6.setToolTip('融合clmask npr')

        btn\_save6.resize(80, 30)

        btn\_save6.move(10, 620)

        btn\_save6.clicked.connect(self.buttonEvent)

        btn\_save6 = QPushButton(self)

        btn\_save6.setText("Finish")

        btn\_save6.setToolTip('点击以生成3D网格')

        btn\_save6.resize(80, 30)

        btn\_save6.move(10, 670)

        btn\_save6.clicked.connect(self.doPredict)

        # gl图形显示窗口

        if(self.show\_mode==1 or self.show\_mode==3):

            self.graphicsView.setGeometry(100, 0, 900, 600)

        elif(self.show\_mode==4):

            self.vtkWidget.setGeometry(100, 0 , 900, 600)

        # 图片显示窗口

        self.lbl1 = QLabel(self)

        self.lbl1.resize(256,256)

        self.lbl1.move(100,644)

        self.lbl2 = QLabel(self)

        self.lbl2.resize(256,256)

        self.lbl2.move(422,644)

        self.lbl3 = QLabel(self)

        self.lbl3.resize(256,256)

        self.lbl3.move(744,644)

        lbl1 = QLabel(self)

        lbl1.setText("置信度图")

        lbl1.move(190,622)

        lbl2 = QLabel(self)

        lbl2.setText("深度图")

        lbl2.move(512,622)

        lbl3 = QLabel(self)

        lbl3.setText("法线图")

        lbl3.move(834,622)

    # 主窗口居中函数

    def center(self):

        qr = self.frameGeometry()

        cp = QDesktopWidget().availableGeometry().center()

        qr.moveCenter(cp)

        self.move(qr.topLeft())

    # 按钮模式

    def buttonEvent(self):

        button=self.sender()

        if("开始绘制3图" == button.objectName()):

            self.mode=1

        elif("保存2" == button.objectName()):

            self.mode=2

        elif("保存3" == button.objectName()):

            self.mode=3

        elif("保存4" == button.objectName()):

            self.mode=4

        elif("保存5" == button.objectName()):

            self.mode=5

        elif("保存6" == button.objectName()):

            self.mode=6

            self.opencv.generateNC()

        self.saveStokes()

    # 清空并导入屏幕上的所有笔划

    def clearStokes(self):

        self.pix.fill(Qt.white)

        self.openPicture()

        self.opencv.clearAllPictures()

        self.update()

    # 存储图片

    def saveStokes(self):

        print('当前的mode为：', self.mode)

        self.statusBar().showMessage('当前的mode为：', self.mode)

        self.pix.save('input\\draw.png')

    #获取rgb颜色灰度值

    def getGray(self,penColor):

        grayRGB = penColor.getRgb()

        gray = grayRGB[0]\*0.299 + grayRGB[1]\*0.587 + grayRGB[2]\*0.114

        return gray

    #改变画笔颜色

    def changePenColor(self):

        color = QColorDialog.getColor()

        if color.isValid():

            self.color=color

            styleSheet = 'QLabel{background-color: ' + self.color.name()+ '; border:1px solid 0; font: 16px;color: black;}'

            self.color\_label.setStyleSheet(styleSheet)

            gray = self.getGray(self.color)

            self.lblpen.setText('颜色灰度:'+str(gray))

    #改变画笔粗细

    def changeThickness(self,sld\_value):

        self.thickness=sld\_value

    # 打开背景模板图片

    def openPicture(self):

        imgName, imgType = QFileDialog.getOpenFileName(self, "打开图片", "", "\*.jpg;;\*.png;;All Files(\*)")

        bgd = QPixmap(imgName).scaled(self.pix.width(), self.pix.height())

        if(bgd.isNull()!=True):

            self.pix = bgd

    # 绘画器

    def paintEvent(self, event):

        pp = QPainter(self.pix)

        pp.setPen(QPen(self.color, self.thickness))

        pp.drawLine(self.lastPoint, self.endPoint)

        self.lastPoint = self.endPoint

        painter = QPainter(self)

        #绘制画布到窗口指定位置处

        painter.drawPixmap(self.win\_width-self.win\_height, 0, self.pix)

    # 鼠标按下事件

    def mousePressEvent(self, event):

        if event.button() == Qt.LeftButton:

            self.lastPoint = event.pos() - self.offset

            self.endPoint = self.lastPoint

    # 鼠标移动事件

    def mouseMoveEvent(self, event):

        if event.buttons() and Qt.LeftButton:

            self.endPoint = event.pos() - self.offset

            tmp=(int(self.endPoint.x()),int(self.endPoint.y()))

            self.opencv.points.append(tmp)

            self.update()

    # 鼠标释放事件

    def mouseReleaseEvent(self, event):

        if event.button() == Qt.LeftButton:

            self.endPoint = event.pos() - self.offset

            if(self.mode!=6):

                self.opencv.generateOtherLines(self.mode,self.getGray(self.color),self.thickness)

            self.opencv.points.clear()

            self.update()

    # 开始预测

    def doPredict(self):

        print("OK.Ready to push lines to network.\n")

        self.statusBar().showMessage('正在读取输入图片')

        npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur=read10pics("input\\")

        self.statusBar().showMessage('正在进行模型推理')

        do\_predict(npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur)

        self.statusBar().showMessage('输出图像至屏幕')

        self.showFinalPitures()

        self.statusBar().showMessage('3D视图展示')

        self.finalView()

    # 后处理 3D视图显示

    def finalView(self):

        ply\_fileName=os.path.join(conf['PATH']['PROJECT\_PATH'],'output',conf['GUI\_QT']['PLY\_NAME'])

        obj\_fileName=os.path.join(conf['PATH']['PROJECT\_PATH'],'output',conf['GUI\_QT']['OBJ\_NAME'])

        mes\_fileName=os.path.join(conf['PATH']['PROJECT\_PATH'],'output',conf['GUI\_QT']['MES\_NAME'])

        # 获取显示模式

        if(self.show\_mode==1 or self.show\_mode==2):

            print("开始读取ply文件")

            if ply\_fileName != '':

                # 读取点云

                pcd = o3d.io.read\_point\_cloud(ply\_fileName)

                if(self.show\_mode==1):

                    # 显示方式1：gl散点图

                    np\_points = np.asarray(pcd.points)

                    plot = gl.GLScatterPlotItem()# 会ply转vertex-face-color格式的obj之后，这个plot换成GLMeshItem 怎么呈现参考：https://stackoom.com/question/3xakt

                    plot.setData(pos=np\_points, color=(float(conf['GUI\_QT']['PLY\_COLOR\_R']), float(conf['GUI\_QT']['PLY\_COLOR\_G']), float(conf['GUI\_QT']['PLY\_COLOR\_B']), float(conf['GUI\_QT']['PLY\_COLOR\_A'])), size=float(conf['GUI\_QT']['PLY\_POINT\_SIZE']), pxMode=False) #散点图颜色及密集程度

                    self.txt.setData(pos=(0.5, -0.5, -0.5), color=(127, 255, 127, 255), text='点云图')

                    self.graphicsView.addItem(self.txt)         #字体显示

                    self.graphicsView.addItem(self.grid)        #添加网格基准线

                    self.graphicsView.addItem(self.paintitem)   #左上角信息窗口

                    self.graphicsView.addItem(plot)

                elif(self.show\_mode==2):

                    # 显示方式2：open3d

                    o3d.visualization.draw\_geometries([pcd])

            else:

                print("未找到ply文件")

        elif(self.show\_mode==3 or self.show\_mode==4):

            print("开始读取obj文件")

            if obj\_fileName  != '':

                 if(self.show\_mode==3):

                    # 显示方式：gl网格图

                    self.grid.scale(2,2,1)

                    self.graphicsView.setCameraPosition(distance=150)

                    # 解析obj 获取顶点与面

                    verts,faces = parseObj(obj\_fileName)

                    # 加载控件并显示

                    # shader ： balloon normalColor

                    mesh = gl.GLMeshItem(vertexes=verts,

                        faces=faces,

                        smooth  =   bool(conf['GUI\_QT']['OBJ\_SMOOTH']),                # 是否需要平滑

                        shader  =   conf['GUI\_QT']['OBJ\_SHADER'],       # 着色器(需要smooth=True)balloon normalColor viewNormalColor shaded edgeHilight heightColor

                        drawFaces=  bool(conf['GUI\_QT']['OBJ\_DRAW\_FACES']),             # 是否画面

                        color   =   (float(conf['GUI\_QT']['OBJ\_FACES\_COLOR\_R']), float(conf['GUI\_QT']['OBJ\_FACES\_COLOR\_G']), float(conf['GUI\_QT']['OBJ\_FACES\_COLOR\_B']), float(conf['GUI\_QT']['OBJ\_FACES\_COLOR\_A'])),       # 面的颜色(需要drawFaces=True)

                        drawEdges=  bool(conf['GUI\_QT']['OBJ\_DRAW\_EDGES']),             # 是否画面的边沿

                        edgeColor=  (float(conf['GUI\_QT']['OBJ\_EDGES\_COLOR\_R']), float(conf['GUI\_QT']['OBJ\_EDGES\_COLOR\_G']), float(conf['GUI\_QT']['OBJ\_EDGES\_COLOR\_B']), float(conf['GUI\_QT']['OBJ\_EDGES\_COLOR\_A'])),       # 边的颜色(需要drawEdges=True)

                        glOptions=   conf['GUI\_QT']['OBJ\_GLOPTIONS'])       # additive opaque

                    self.txt.setData(pos=(10.0, -10.0, -10.0), color=(127, 255, 127, 255), text='网格图')

                    self.graphicsView.addItem(self.txt)

                    self.graphicsView.addItem(self.paintitem)

                    self.graphicsView.addItem(self.grid)

                    self.graphicsView.addItem(mesh)

                    np.seterr(divide='ignore',invalid='ignore')# 忽略除法警告

                 elif(self.show\_mode==4):

                    # 显示方式：vtk图

                    filename = mes\_fileName

                    # Create source

                    reader = vtk.vtkOBJReader()

                    reader.SetFileName(filename)

                    reader.Update()

                    # Create a mapper

                    mapper = vtk.vtkPolyDataMapper()

                    mapper.SetInputConnection(reader.GetOutputPort())

                    # Create an actor

                    actor = vtk.vtkActor()

                    actor.SetMapper(mapper)

                    #actor.GetProperty().SetEdgeColor(1, 0.706, 0)# wireframe颜色

                    actor.GetProperty().SetEdgeColor(0, 0, 0)# wireframe颜色

                    actor.GetProperty().SetColor(1, 0.706, 0)# fill颜色

                    actor.GetProperty().SetEdgeVisibility(1)

                    self.render.AddActor(actor)

                    self.render.ResetCamera()

                    self.iren.Initialize()

            else:

                print("未找到obj文件")

    # 显示最终置信度图 深度图 法线图

    def showFinalPitures(self):

        conp = QPixmap(os.path.join(conf['PATH']['PROJECT\_PATH'], "output\\model\_output\_img\\fwd\_conf\_map\_0.jpg"))

        self.lbl1.setPixmap(conp)

        dept = QPixmap(os.path.join(conf['PATH']['PROJECT\_PATH'], "output\\model\_output\_img\\fwd\_depth\_0.jpg"))

        self.lbl2.setPixmap(dept)

        norm = QPixmap(os.path.join(conf['PATH']['PROJECT\_PATH'], "output\\model\_output\_img\\fwd\_normal\_0.jpg"))

        self.lbl3.setPixmap(norm)

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    form = Winform()

    form.show()

    sys.exit(app.exec\_())

## 2 调用模型并进行推理 model.py

# 模型预测

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import unicode\_literals

import os

import logging

import argparse

import configparser

import tensorflow.compat.v1 as tf

from network import SKETCHNET

from tools import slice\_tensor, make\_dir, dump\_params,create\_obj,point\_cloud\_generator,ply\_pcd\_to\_obj\_mesh\_byo3d\_possion,ply\_pcd\_to\_obj\_mesh\_byo3d\_bpa

import cv2

import numpy as np

hyper\_params = {

    'inpDir': '',

    'outDir': '',

    'device': '0',

    'rootFt': 32,

    'cktDir': '',

    'nbThreads': 1,

    'dsWeight': 5.0,

    'regWeight': 2.0,

    'dlossScale': 920.0,

    'nlossScale': 430.0,

    'graphName': '',

}

out\_output\_img\_dir=''

out\_input\_img\_dir=''

conf = configparser.ConfigParser()

conf.read('config.ini', encoding="utf-8-sig")

tf.disable\_eager\_execution()

nprLine\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='npr\_input')

ds\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='ds\_input')

fm\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='fLMask\_input')

fmInv\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='fLInvMask\_input')

gtNormal\_input = tf.placeholder(tf.float32, [None, None, None, 3], name='gtN\_input')

gtDepth\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='gtD\_input')

gtField\_input = tf.placeholder(tf.float32, [None, None, None, 4], name='gtField\_input')

clineInvMask\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='clIMask\_input')

maskShape\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='shapeMask\_input')

maskDs\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='dsMask\_input')

mask2D\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='2dMask\_input')

selLineMask\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='sLMask\_input')

vdotnScalar\_input = tf.placeholder(tf.float32, [None, None, None, 1], name='curvMag\_input')

def reg\_loss(logit\_n, logit\_d, shape\_mask, cl\_mask\_inverse, fl\_mask\_inv, scope='reg\_loss'):

    with tf.name\_scope(scope) as \_:

        # 将正常信号转换回 [-1, 1]

        converted\_n = (logit\_n \* 2.0) - 1.0

        img\_shape = tf.shape(logit\_d)

        N = img\_shape[0]

        H = img\_shape[1]

        W = img\_shape[2]

        K = 0.007843137254902

        shape\_mask\_crop = slice\_tensor(shape\_mask, logit\_d)

        l\_mask\_crop = slice\_tensor(cl\_mask\_inverse, logit\_d)

        fl\_mask\_inv\_crop = slice\_tensor(fl\_mask\_inv, logit\_d)

        combined\_mask = shape\_mask\_crop \* l\_mask\_crop \* fl\_mask\_inv\_crop

        mask\_shift\_x = tf.slice(combined\_mask, [0, 0, 0, 0], [-1, -1, W - 1, -1])

        mask\_shift\_y = tf.slice(combined\_mask, [0, 0, 0, 0], [-1, H - 1, -1, -1])

        c0 = tf.fill([N, H, W - 1, 1], K)

        c1 = tf.zeros(shape=[N, H, W - 1, 1])

        cx = logit\_d[:, :, 1:, :] - logit\_d[:, :, :-1, :]

        t\_x = tf.concat([c0, c1, cx], axis=3)

        # 近似归一化

        t\_x /= K

        c2 = tf.zeros(shape=[N, H - 1, W, 1])

        c3 = tf.fill([N, H - 1, W, 1], K)

        cy = logit\_d[:, 1:, :, :] - logit\_d[:, :-1, :, :]

        t\_y = tf.concat([c2, c3, cy], axis=3)

        # 近似归一化

        t\_y /= K

        normal\_shift\_x = tf.slice(converted\_n, [0, 0, 0, 0], [-1, -1, W - 1, -1])

        normal\_shift\_y = tf.slice(converted\_n, [0, 0, 0, 0], [-1, H - 1, -1, -1])

        reg\_loss1\_diff = tf.reduce\_sum(t\_x \* normal\_shift\_x, 3)

        reg\_loss1 = tf.losses.mean\_squared\_error(tf.zeros(shape=[N, H, W - 1]), reg\_loss1\_diff,

                                                 weights=tf.squeeze(mask\_shift\_x, [3]))

        reg\_loss2\_diff = tf.reduce\_sum(t\_y \* normal\_shift\_y, 3)

        reg\_loss2 = tf.losses.mean\_squared\_error(tf.zeros(shape=[N, H - 1, W]), reg\_loss2\_diff,

                                                 weights=tf.squeeze(mask\_shift\_y, [3]))

        return reg\_loss1 + reg\_loss2

# 总损失

def loss(logit\_d, logit\_n, logit\_c, normal, depth, shape\_mask, ds\_mask, cl\_mask\_inverse,

         gt\_ds, npr, logit\_f, gt\_f, fl\_mask\_inv):

    img\_shape = tf.shape(logit\_d)

    N = img\_shape[0]

    H = img\_shape[1]

    W = img\_shape[2]

    mask\_crop = slice\_tensor(shape\_mask, logit\_n)

    mask\_crop3 = tf.tile(mask\_crop, [1, 1, 1, 3])

    zero\_tensor = tf.zeros(shape=[N, H, W, 1])

    zero\_tensor3 = tf.zeros(shape=[N, H, W, 3])

    logit\_c3 = tf.tile(logit\_c, [1, 1, 1, 3])

    # normal loss (l2)

    gt\_normal = slice\_tensor(normal, logit\_n)

    n\_loss = tf.losses.mean\_squared\_error(zero\_tensor3, logit\_c3 \* (gt\_normal - logit\_n),

                                          weights=mask\_crop3)

    real\_n\_loss = tf.losses.absolute\_difference(gt\_normal, logit\_n, weights=mask\_crop3)

    # depth loss (l2)

    gt\_depth = slice\_tensor(depth, logit\_n)

    d\_loss = tf.losses.mean\_squared\_error(zero\_tensor, logit\_c \* (gt\_depth - logit\_d),

                                          weights=mask\_crop)

    real\_d\_loss = tf.losses.absolute\_difference(gt\_depth, logit\_d, weights=mask\_crop)

    # omega\_loss (l2)

    omega\_loss = tf.losses.mean\_squared\_error(zero\_tensor, logit\_c - 1.0, weights=mask\_crop)

    # depth sample loss (l2)

    d\_mask\_crop = slice\_tensor(ds\_mask, logit\_n)

    ds\_loss = tf.losses.mean\_squared\_error(gt\_depth, logit\_d, weights=d\_mask\_crop)

    # regularization loss (l2)

    r\_loss = reg\_loss(logit\_n, logit\_d, shape\_mask, cl\_mask\_inverse, fl\_mask\_inv)

    total\_loss = hyper\_params['dlossScale'] \* d\_loss + hyper\_params['nlossScale'] \* n\_loss + omega\_loss + \

                 hyper\_params['dsWeight'] \* ds\_loss + hyper\_params['regWeight'] \* r\_loss

    shape\_mask\_crop = slice\_tensor(shape\_mask, logit\_n)

    shape\_mask\_crop3 = tf.tile(shape\_mask\_crop, [1, 1, 1, 3])

    shape\_mask\_crop4 = tf.tile(shape\_mask\_crop, [1, 1, 1, 4])

    cl\_mask\_inverse4 = tf.tile(cl\_mask\_inverse, [1, 1, 1, 4])

    gt\_f = slice\_tensor(gt\_f, logit\_n) \* cl\_mask\_inverse4

    logit\_f = logit\_f \* shape\_mask\_crop4 \* cl\_mask\_inverse4

    cur\_shape = tf.shape(logit\_n)

    lc = tf.zeros([cur\_shape[0], cur\_shape[1], cur\_shape[2], 1], tf.float32)

    gt\_coeff\_a = tf.concat([tf.slice(gt\_f, [0, 0, 0, 0], [-1, -1, -1, 2]), lc], axis=3)

    gt\_coeff\_b = tf.concat([tf.slice(gt\_f, [0, 0, 0, 2], [-1, -1, -1, 2]), lc], axis=3)

    f\_coeff\_a = tf.concat([tf.slice(logit\_f, [0, 0, 0, 0], [-1, -1, -1, 2]), lc], axis=3)

    f\_coeff\_b = tf.concat([tf.slice(logit\_f, [0, 0, 0, 2], [-1, -1, -1, 2]), lc], axis=3)

    return total\_loss, d\_loss, n\_loss, ds\_loss, r\_loss, real\_d\_loss, real\_n\_loss, omega\_loss, \

           gt\_normal \* shape\_mask\_crop3, logit\_n \* shape\_mask\_crop3, gt\_depth \* shape\_mask\_crop, \

           logit\_d \* shape\_mask\_crop, gt\_ds \* shape\_mask\_crop, npr, \

           slice\_tensor(cl\_mask\_inverse, logit\_n) \* mask\_crop, \

           logit\_c \* shape\_mask\_crop, gt\_coeff\_a, gt\_coeff\_b, f\_coeff\_a, f\_coeff\_b

# 测试过程

def test\_procedure(net):

    # 前向网络

    logit\_f, \_ = net.load\_field\_net(nprLine\_input,

                                    mask2D\_input,

                                    ds\_input,

                                    fm\_input,

                                    selLineMask\_input,

                                    vdotnScalar\_input,

                                    hyper\_params['rootFt'],

                                    is\_training=False)

    logit\_d, logit\_n, logit\_c, \_ = net.load\_GeomNet(nprLine\_input,

                                                    ds\_input,

                                                    mask2D\_input,

                                                    fm\_input,

                                                    selLineMask\_input,

                                                    vdotnScalar\_input,

                                                    logit\_f,

                                                    clineInvMask\_input,

                                                    hyper\_params['rootFt'],

                                                    is\_training=False)

    # 测试误差

    test\_loss, test\_d\_loss, test\_n\_loss, test\_ds\_loss, test\_reg\_loss, test\_real\_dloss, \

    test\_real\_nloss, test\_omega\_loss, out\_gt\_normal, out\_f\_normal, out\_gt\_depth, out\_f\_depth, out\_gt\_ds, gt\_lines, \

    reg\_mask, out\_cf\_map, test\_gt\_a, test\_gt\_b, test\_f\_a, test\_f\_b \

        = loss(logit\_d,

               logit\_n,

               logit\_c,

               gtNormal\_input,

               gtDepth\_input,

               maskShape\_input,

               maskDs\_input,

               clineInvMask\_input,

               ds\_input,

               nprLine\_input,

               logit\_f,

               gtField\_input,

               fmInv\_input)

    return test\_loss, test\_d\_loss, test\_n\_loss, test\_ds\_loss, test\_reg\_loss, test\_real\_dloss, \

           test\_real\_nloss, test\_omega\_loss, out\_gt\_normal, out\_f\_normal, out\_gt\_depth, \

           out\_f\_depth, out\_gt\_ds, gt\_lines, reg\_mask, out\_cf\_map, test\_gt\_a, test\_gt\_b, test\_f\_a, test\_f\_b

def test\_net(npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur):

    # 设定日志

    test\_logger = logging.getLogger('main.testing')

    test\_logger.info('---Begin testing: ---')

    # 加载网络

    net = SKETCHNET()

    test\_loss, test\_d\_loss, test\_n\_loss, test\_ds\_loss, test\_r\_loss, test\_real\_dloss, \

    test\_real\_nloss, test\_omega\_loss, test\_gt\_normal, test\_f\_normal, test\_gt\_depth, test\_f\_depth, test\_gt\_ds, \

    test\_gt\_lines, test\_reg\_mask, test\_f\_cfmap, test\_gt\_a, test\_gt\_b, test\_f\_a, test\_f\_b \

        = test\_procedure(net)

    # Saver

    tf\_saver = tf.train.Saver()

    config = tf.ConfigProto()

    config.gpu\_options.allow\_growth = True

    config.allow\_soft\_placement = True

    with tf.Session(config=config) as sess:

        # initialize

        init\_op = tf.group(tf.global\_variables\_initializer(), tf.local\_variables\_initializer())

        sess.run(init\_op)  # 对变量进行初始化，变量运行前必须做初始化操作

        # Restore model

        ckpt = tf.train.latest\_checkpoint(hyper\_params['cktDir'])

        if ckpt:

            tf\_saver.restore(sess, ckpt)

            test\_logger.info('restore from the checkpoint {}'.format(ckpt))

        try:

            titr = 0

            avg\_loss = 0.0

            # Load data

            all0tensor1 = tf.zeros([1, 256, 256, 1])

            all0tensor2 = tf.zeros([1, 256, 256, 3])

            all0tensor3 = tf.zeros([1, 256, 256, 4])

            all0tensor = sess.run(all0tensor1)

            all0tensor\_2 = sess.run(all0tensor2)

            all0tensor\_3 = sess.run(all0tensor3)

            npr\_1, ds\_1, fm\_1, fmi\_1, cm\_1, sm\_1, dsm\_1, msk\_1, slm\_1, cur\_1 = npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur

            npr = sess.run(npr\_1)

            ds = sess.run(ds\_1)

            fm = sess.run(fm\_1)

            fmi = sess.run(fmi\_1)

            cm = sess.run(cm\_1)

            sm = sess.run(sm\_1)

            dsm = sess.run(dsm\_1)

            msk = sess.run(msk\_1)

            slm = sess.run(slm\_1)

            cur = sess.run(cur\_1)

            t\_loss, t\_d\_loss, t\_n\_loss, t\_ds\_loss, t\_r\_loss, t\_real\_dloss, t\_real\_nloss, \

            t\_omega\_loss, t\_gt\_normal, t\_f\_normal, t\_gt\_depth, t\_f\_depth, t\_gt\_ds, t\_gt\_lines, t\_reg\_mask, \

            t\_f\_cfmap, t\_gt\_a, t\_gt\_b, t\_f\_a, t\_f\_b \

                = sess.run([test\_loss, test\_d\_loss, test\_n\_loss, test\_ds\_loss, test\_r\_loss,

                            test\_real\_dloss, test\_real\_nloss, test\_omega\_loss, test\_gt\_normal, test\_f\_normal,

                            test\_gt\_depth, test\_f\_depth, test\_gt\_ds, test\_gt\_lines, test\_reg\_mask, test\_f\_cfmap,

                            test\_gt\_a, test\_gt\_b, test\_f\_a, test\_f\_b],

                           feed\_dict={'npr\_input:0': npr,

                                      'ds\_input:0': ds,

                                      'fLMask\_input:0': fm,

                                      'fLInvMask\_input:0': fmi,

                                      'gtN\_input:0': all0tensor\_2,

                                      'gtD\_input:0': all0tensor,

                                      'gtField\_input:0': all0tensor\_3,

                                      'clIMask\_input:0': cm,

                                      'shapeMask\_input:0': sm,

                                      'dsMask\_input:0': dsm,

                                      '2dMask\_input:0': msk,

                                      'sLMask\_input:0': slm,

                                      'curvMag\_input:0': cur

                                      })

            # Record loss

            avg\_loss += t\_loss

            test\_logger.info(

                'Test case {}, loss: {}, {}, {}, {}, {}, {}, {}, 0.0, {}'.format(titr, t\_loss, t\_real\_dloss,

                                                                                 t\_real\_nloss, t\_d\_loss,

                                                                                 t\_n\_loss,

                                                                                 t\_ds\_loss, t\_r\_loss,

                                                                                 t\_omega\_loss))

            # 3D输出

            fn5 = os.path.join(out\_output\_img\_dir, 'fwd\_conf\_map\_' + str(titr) + '.jpg')

            fn6 = os.path.join(out\_output\_img\_dir, 'fwd\_depth\_' + str(titr) + '.jpg')

            fn7 = os.path.join(out\_output\_img\_dir, 'fwd\_normal\_' + str(titr) + '.jpg')

            fn8 = os.path.join(out\_output\_img\_dir, 'fwd\_field\_a\_' + str(titr) + '.jpg')

            fn9 = os.path.join(out\_output\_img\_dir, 'fwd\_field\_b\_' + str(titr) + '.jpg')

            # 预测的置信度图 [0,1] 连续

            out\_f\_cfmap = t\_f\_cfmap[0, :, :, :]

            out\_f\_cfmap.astype(np.float32)

            out\_f\_cfmap = out\_f\_cfmap \* 255

            #out\_f\_cfmap = np.flip(out\_f\_cfmap, 0)

            cv2.imwrite(fn5, out\_f\_cfmap)

            # 预测的深度图 [0,2] 连续

            out\_f\_d = t\_f\_depth[0, :, :, :]

            out\_f\_d.astype(np.float32)

            out\_f\_d = out\_f\_d \* 127          # 必须\*127 如果\*256可能会使fwd\_depth\_0丢失深度信息

            #out\_f\_d = np.flip(out\_f\_d, 0)

            cv2.imwrite(fn6, out\_f\_d)

            # 预测的法线图 [0,1] 连续

            out\_f\_normal = t\_f\_normal[0, :, :, :]

            out\_f\_normal = out\_f\_normal[:, :, [2, 1, 0]]

            out\_f\_normal.astype(np.float32)

            out\_f\_normal = out\_f\_normal \* 255

            #out\_f\_normal = np.flip(out\_f\_normal, 0)

            cv2.imwrite(fn7, out\_f\_normal)

            # 预测流场a [-2,2] 连续

            out\_f\_a = t\_f\_a[0, :, :, :]

            out\_f\_a = out\_f\_a[:, :, [2, 1, 0]]

            out\_f\_a.astype(np.float32)

            #out\_f\_a = out\_f\_a \* 255

            out\_f\_a = (out\_f\_a + 2)\* 63

            #out\_f\_a = np.flip(out\_f\_a, 0)

            cv2.imwrite(fn8, out\_f\_a)

            # 预测流场b [-1,1] 连续

            out\_f\_b = t\_f\_b[0, :, :, :]

            out\_f\_b = out\_f\_b[:, :, [2, 1, 0]]

            out\_f\_b.astype(np.float32)

            #out\_f\_b = out\_f\_b \* 255

            out\_f\_b = (out\_f\_b + 1) \* 127

            #out\_f\_b = np.flip(out\_f\_b, 0)

            cv2.imwrite(fn9, out\_f\_b)

            # 2D输入

            fna = os.path.join(out\_input\_img\_dir, 'npr\_' + str(titr) + '.jpg')

            fnb = os.path.join(out\_input\_img\_dir, 'ds\_' + str(titr) + '.jpg')

            fnc = os.path.join(out\_input\_img\_dir, 'fLMask\_' + str(titr) + '.jpg')

            fnd = os.path.join(out\_input\_img\_dir, 'fLInvMask\_' + str(titr) + '.jpg')

            fne = os.path.join(out\_input\_img\_dir, 'clIMask\_' + str(titr) + '.jpg')

            fnf = os.path.join(out\_input\_img\_dir, 'shapeMask\_' + str(titr) + '.jpg')

            fng = os.path.join(out\_input\_img\_dir, 'dsMask\_' + str(titr) + '.jpg')

            fnh = os.path.join(out\_input\_img\_dir, '2dMask\_' + str(titr) + '.jpg')

            fni = os.path.join(out\_input\_img\_dir, 'sLMask\_' + str(titr) + '.jpg')

            fnj = os.path.join(out\_input\_img\_dir, 'curvMag\_' + str(titr) + '.jpg')

            # npr [0,1] 连续

            npr\_line = npr[0, :, :, :]

            npr\_line.astype(np.float32)

            npr\_line = npr\_line \* 255

            #npr\_line = np.flip(npr\_line, 0)

            cv2.imwrite(fna, npr\_line)

            # 深度ds [0,2] 连续

            ds = ds[0, :, :, :]

            ds.astype(np.float32)

            ds = ds \* 127   # 将[0,2]的深度信息区间映射到[0,255]的图像灰度区间

            #ds = np.flip(ds, 0)

            cv2.imwrite(fnb, ds)

            # 深度掩码dsMask [0,1] 离散

            dsmask = dsm[0, :, :, :]

            dsmask.astype(np.float32)

            dsmask = dsmask \* 255

            #dsmask = np.flip(dsmask, 0)

            cv2.imwrite(fng, dsmask)

            # 尖锐信息fLMask [0,1] 离散

            fm = fm[0, :, :, :]

            fm.astype(np.float32)

            fm = fm \* 255

            #fm = np.flip(fm, 0)

            cv2.imwrite(fnc, fm)

            # 尖锐信息掩码fLInvMask [0,1] 离散

            fmi = fmi[0, :, :, :]

            fmi.astype(np.float32)

            fmi = fmi \* 255

            #fmi = np.flip(fmi, 0)

            cv2.imwrite(fnd, fmi)

            # 轮廓模板2dMask [0,1] 离散

            mask2d = msk[0, :, :, :]

            mask2d.astype(np.float32)

            mask2d = mask2d \* 255

            #mask2d = np.flip(mask2d, 0)

            cv2.imwrite(fnh, mask2d)

            # 形状掩码shapeMask\_input [0,1] 离散

            shapemask\_input = sm[0, :, :, :]

            shapemask\_input.astype(np.float32)

            shapemask\_input = shapemask\_input \* 255

            #shapemask\_input = np.flip(shapemask\_input, 0)

            cv2.imwrite(fnf, shapemask\_input)

            # 轮廓及内部特征线clIMask [0,1] 离散

            mask\_cline\_inv = cm[0, :, :, :]

            mask\_cline\_inv.astype(np.float32)

            mask\_cline\_inv = mask\_cline\_inv \* 255

            #mask\_cline\_inv = np.flip(mask\_cline\_inv, 0)

            cv2.imwrite(fne, mask\_cline\_inv)

            # 曲率信息curvMag [-2,2] 连续

            ndotv = cur[0, :, :, :]

            ndotv.astype(np.float32)

            ndotv = (ndotv+2) \* 63

            #ndotv = ndotv \* 255

            #ndotv = np.flip(ndotv, 0)

            cv2.imwrite(fnj, ndotv)

            # 曲率掩码sLMask [0,1] 离散

            selm = slm[0, :, :, :]

            selm.astype(np.float32)

            selm = selm \* 255

            #selm = np.flip(selm, 0)

            cv2.imwrite(fni, selm)

            test\_logger.info('Finish test model, average loss is: {}'.format(avg\_loss))

        except tf.errors.OutOfRangeError:

            print('Test Done.')

def write\_ply(rgb\_file, depth\_file, pc\_file, focal\_length, scalingfactor):

    print("开始写入ply文件")

    a = point\_cloud\_generator(rgb\_file, depth\_file, pc\_file, focal\_length, scalingfactor)

    a.calculate()

    a.write\_ply()

def write\_obj(depthPath, objPath):

    print("开始写入obj文件")

    create\_obj(depthPath, objPath)

def do\_predict(npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur):

    hyper\_params['cktDir'] = ".\\savedModel"

    hyper\_params['outDir'] = ".\\output"

    hyper\_params['device'] = conf['MODEL']['CPU\_INDEX']

    # Set GPU

    os.environ['CUDA\_VISIBLE\_DEVICES'] = "hyper\_params['device']"

    # out img dir

    global out\_output\_img\_dir,out\_input\_img\_dir

    out\_output\_img\_dir = os.path.join(hyper\_params['outDir'], 'model\_output\_img')

    make\_dir(out\_output\_img\_dir)

    out\_input\_img\_dir = os.path.join(hyper\_params['outDir'], 'model\_input\_img')

    make\_dir(out\_input\_img\_dir)

    # Set logger

    logger = logging.getLogger('main')

    logger.setLevel(logging.DEBUG)

    fh = logging.FileHandler(os.path.join(hyper\_params['outDir'], 'log.txt'))

    fh.setLevel(logging.DEBUG)

    ch = logging.StreamHandler()

    ch.setLevel(logging.INFO)

    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')

    fh.setFormatter(formatter)

    ch.setFormatter(formatter)

    logger.addHandler(fh)

    logger.addHandler(ch)

    logger.info('---Test preparation: ---')

    # Begin testing

    test\_net(npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur)

    # Write ply file

    write\_ply(

        os.path.join(out\_output\_img\_dir, 'fwd\_normal\_0.jpg'),

        os.path.join(out\_output\_img\_dir, 'fwd\_depth\_0.jpg'),

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['PLY\_NAME']),

        focal\_length=50,

        scalingfactor=1000)

    # Write obj file

    write\_obj(

        os.path.join(out\_output\_img\_dir, 'fwd\_depth\_0.jpg'),

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['OBJ\_NAME']))

    # 将ply点云转化为obj网格 0-泊松重建 1-BPA重建

    if(int(conf['MODEL']['RESTRUCT\_MODE']) == 0):

        ply\_pcd\_to\_obj\_mesh\_byo3d\_possion(

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['PLY\_NAME']),

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['MES\_NAME']))

    elif(int(conf['MODEL']['RESTRUCT\_MODE']) == 1):

        ply\_pcd\_to\_obj\_mesh\_byo3d\_bpa(

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['PLY\_NAME']),

        os.path.join(hyper\_params['outDir'], conf['GUI\_QT']['MES\_NAME']))

## 3 模型的网络结构定义 network.py

# 模型网络结构

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import unicode\_literals

import tensorflow as tf

from   tools import cropconcat\_layer

import tf\_slim as slim

import logging

net\_logger = logging.getLogger('main.network')

class SKETCHNET(object):

    @staticmethod

    def cook\_raw\_inputs(raw\_input):

        with tf.compat.v1.name\_scope("cook\_raw\_input") as \_:

            input\_data, label\_data = raw\_input

            npr\_line = tf.slice(input\_data, [0, 0, 0, 0], [-1, -1, -1, 1])

            depth\_sample = tf.slice(input\_data, [0, 0, 0, 3], [-1, -1, -1, 1])

            distance\_field = tf.slice(input\_data, [0, 0, 0, 1], [-1, -1, -1, 2])

            feature\_mask = tf.slice(input\_data, [0, 0, 0, 4], [-1, -1, -1, 1])

            feature\_mask\_inv = tf.slice(input\_data, [0, 0, 0, 5], [-1, -1, -1, 1])

            selLine\_mask = tf.slice(label\_data, [0, 0, 0, 15], [-1, -1, -1, 1])

            vdotn\_scalar = tf.slice(label\_data, [0, 0, 0, 16], [-1, -1, -1, 1])

            label\_normal = tf.slice(label\_data, [0, 0, 0, 0], [-1, -1, -1, 3])

            label\_depth = tf.slice(label\_data, [0, 0, 0, 3], [-1, -1, -1, 1])

            mask\_shape = tf.slice(label\_data, [0, 0, 0, 8], [-1, -1, -1, 1])

            mask\_depth\_sample = tf.slice(label\_data, [0, 0, 0, 9], [-1, -1, -1, 1])

            label\_field = tf.slice(label\_data, [0, 0, 0, 4], [-1, -1, -1, 4])

            cmask\_line = tf.slice(label\_data, [0, 0, 0, 10], [-1, -1, -1, 1])

            mask\_line\_inv = tf.slice(label\_data, [0, 0, 0, 12], [-1, -1, -1, 1])

            mask\_line = tf.slice(label\_data, [0, 0, 0, 11], [-1, -1, -1, 1])

            mask2d = tf.slice(label\_data, [0, 0, 0, 13], [-1, -1, -1, 1])

            return npr\_line, depth\_sample, distance\_field, feature\_mask, feature\_mask\_inv, label\_normal, label\_depth, \

                   label\_field, cmask\_line, mask\_shape, mask\_depth\_sample, mask\_line, mask\_line\_inv, mask2d, \

                   selLine\_mask, vdotn\_scalar

    @staticmethod

    def load\_field\_net(lines, mask2d, ds, fm, selm, vdotn, root\_feature=32, is\_training=True, padding='SAME',

                       reuse=None, d\_rate=1,

                       l2\_reg=0.0005):

        with tf.compat.v1.variable\_scope('SASFieldNet', reuse=tf.compat.v1.AUTO\_REUSE) as f\_vs:

            with slim.arg\_scope([slim.conv2d], kernel\_size=[3, 3], rate=d\_rate,

                                padding=padding, activation\_fn=tf.nn.relu,

                                normalizer\_fn=slim.batch\_norm,

                                normalizer\_params={'is\_training': is\_training, 'decay': 0.95, 'fused': True},

                                weights\_initializer=tf.compat.v1.truncated\_normal\_initializer(stddev=0.01),

                                weights\_regularizer=tf.keras.regularizers.l2(0.5 \* (l2\_reg))

                                ):

                # encoder

                reg\_f\_input = tf.concat([lines, mask2d, ds, fm, selm, vdotn], axis=3, name='concat\_field\_input')

                f\_input = tf.identity(reg\_f\_input, name='reg\_f\_input')

                conv1 = slim.conv2d(f\_input, root\_feature, scope='f\_conv1')

                conv2 = slim.conv2d(conv1, root\_feature, scope='f\_conv2')

                pool1 = slim.max\_pool2d(conv2, [2, 2], scope='f\_pool1')

                conv3 = slim.conv2d(pool1, root\_feature \* 2, scope='f\_conv3')

                conv4 = slim.conv2d(conv3, root\_feature \* 2, scope='f\_conv4')

                pool2 = slim.max\_pool2d(conv4, [2, 2], scope='f\_pool2')

                conv5 = slim.conv2d(pool2, root\_feature \* 4, scope='f\_conv5')

                conv6 = slim.conv2d(conv5, root\_feature \* 4, scope='f\_conv6')

                # decoder

                with slim.arg\_scope([slim.conv2d\_transpose], kernel\_size=[2, 2], stride=2,

                                    padding=padding, activation\_fn=None,

                                    weights\_initializer=tf.compat.v1.truncated\_normal\_initializer(stddev=0.01),

                                    weights\_regularizer=tf.keras.regularizers.l2(0.5 \* (l2\_reg))):

                    f\_deconv3 = slim.conv2d\_transpose(conv6, root\_feature \* 2, scope='f\_deconv3\_1')

                    f\_concat3 = cropconcat\_layer(conv4, f\_deconv3, 3, name='f\_concat3')

                    f\_deconv3\_2 = slim.conv2d(f\_concat3, root\_feature \* 2, scope='f\_deconv3\_2')

                    f\_deconv3\_3 = slim.conv2d(f\_deconv3\_2, root\_feature \* 2, scope='f\_deconv3\_3')

                    f\_deconv4 = slim.conv2d\_transpose(f\_deconv3\_3, root\_feature, scope='f\_deconv4\_1')

                    f\_concat4 = cropconcat\_layer(conv2, f\_deconv4, 3, name='f\_concat4')

                    f\_deconv4\_2 = slim.conv2d(f\_concat4, root\_feature, scope='f\_deconv4\_2')

                    f\_deconv4\_3 = slim.conv2d(f\_deconv4\_2, root\_feature, scope='f\_deconv4\_3')

                    f\_res = slim.conv2d(f\_deconv4\_3, 4, kernel\_size=[1, 1], activation\_fn=None,

                                        scope='f\_output')  # direction field

                    logit\_f = tf.identity(f\_res, name='output\_f')

        #f\_net\_variables = tf.contrib.framework.get\_variables(f\_vs)

        f\_net\_variables = slim.get\_variables(f\_vs)

        return logit\_f, f\_net\_variables

    @staticmethod

    def load\_GeomNet(lines, ds, mask2d, fm, selm, vdotn, field, cl\_mask, root\_feature=32,

                     is\_training=True, padding='SAME', reuse=None, d\_rate=1, l2\_reg=0.0005):

        with tf.compat.v1.variable\_scope('SASMFGeoNet', reuse=tf.compat.v1.AUTO\_REUSE) as g\_vs:

            with slim.arg\_scope([slim.conv2d], kernel\_size=[3, 3], rate=d\_rate,

                                padding=padding, activation\_fn=tf.nn.relu,

                                normalizer\_fn=slim.batch\_norm,

                                normalizer\_params={'is\_training': is\_training, 'decay': 0.95, 'fused': True},

                                weights\_initializer=tf.compat.v1.truncated\_normal\_initializer(stddev=0.01),

                                weights\_regularizer=tf.keras.regularizers.l2(0.5 \* (l2\_reg))

                                ):

                # encoder

                full\_mask = tf.tile(mask2d \* cl\_mask, [1, 1, 1, 4])

                field = field \* full\_mask

                reg\_geo\_input = tf.concat([lines, ds, mask2d, fm, selm, vdotn, field], axis=3, name='concat\_geo\_input')

                conv1 = slim.conv2d(reg\_geo\_input, root\_feature, scope='geo\_conv1')

                conv2 = slim.conv2d(conv1, root\_feature, scope='geo\_conv2')

                pool1 = slim.max\_pool2d(conv2, [2, 2], scope='geo\_pool1')

                conv3 = slim.conv2d(pool1, root\_feature \* 2, scope='geo\_conv3')

                conv4 = slim.conv2d(conv3, root\_feature \* 2, scope='geo\_conv4')

                pool2 = slim.max\_pool2d(conv4, [2, 2], scope='geo\_pool2')

                conv5 = slim.conv2d(pool2, root\_feature \* 4, scope='geo\_onv5')

                conv6 = slim.conv2d(conv5, root\_feature \* 4, scope='geo\_conv6')

                pool3 = slim.max\_pool2d(conv6, [2, 2], scope='geo\_pool3')

                conv7 = slim.conv2d(pool3, root\_feature \* 8, scope='geo\_conv7')

                conv8 = slim.conv2d(conv7, root\_feature \* 8, scope='geo\_conv8')

                pool4 = slim.max\_pool2d(conv8, [2, 2], scope='geo\_pool4')

                conv9 = slim.conv2d(pool4, root\_feature \* 16, scope='geo\_conv9')

                conv10 = slim.conv2d(conv9, root\_feature \* 16, scope='geo\_conv10')

                # decoder

                with slim.arg\_scope([slim.conv2d\_transpose], kernel\_size=[2, 2], stride=2,

                                    padding=padding, activation\_fn=None,

                                    weights\_initializer=tf.compat.v1.truncated\_normal\_initializer(stddev=0.01),

                                    weights\_regularizer=tf.keras.regularizers.l2(0.5 \* (l2\_reg))):

                    d\_deconv1 = slim.conv2d\_transpose(conv10, root\_feature \* 8, scope='d\_geo\_deconv1\_1')

                    n\_deconv1 = slim.conv2d\_transpose(conv10, root\_feature \* 8, scope='n\_geo\_deconv1\_1')

                    c\_deconv1 = slim.conv2d\_transpose(conv10, root\_feature \* 8, scope='c\_geo\_deconv1\_1')

                    d\_concat1 = cropconcat\_layer(conv8, d\_deconv1, 3, name='d\_geo\_concat1')

                    n\_concat1 = cropconcat\_layer(conv8, n\_deconv1, 3, name='n\_geo\_concat1')

                    c\_concat1 = cropconcat\_layer(conv8, c\_deconv1, 3, name='c\_geo\_concat1')

                    d\_deconv1\_2 = slim.conv2d(d\_concat1, root\_feature \* 8, scope='d\_geo\_deconv1\_2')

                    n\_deconv1\_2 = slim.conv2d(n\_concat1, root\_feature \* 8, scope='n\_geo\_deconv1\_2')

                    c\_deconv1\_2 = slim.conv2d(c\_concat1, root\_feature \* 8, scope='c\_geo\_deconv1\_2')

                    d\_deconv1\_3 = slim.conv2d(d\_deconv1\_2, root\_feature \* 8, scope='d\_geo\_deconv1\_3')

                    n\_deconv1\_3 = slim.conv2d(n\_deconv1\_2, root\_feature \* 8, scope='n\_geo\_deconv1\_3')

                    c\_deconv1\_3 = slim.conv2d(c\_deconv1\_2, root\_feature \* 8, scope='c\_geo\_deconv1\_3')

                    d\_deconv2 = slim.conv2d\_transpose(d\_deconv1\_3, root\_feature \* 4, scope='d\_geo\_deconv2\_1')

                    n\_deconv2 = slim.conv2d\_transpose(n\_deconv1\_3, root\_feature \* 4, scope='n\_geo\_deconv2\_1')

                    c\_deconv2 = slim.conv2d\_transpose(c\_deconv1\_3, root\_feature \* 4, scope='c\_geo\_deconv2\_1')

                    d\_concat2 = cropconcat\_layer(conv6, d\_deconv2, 3, name='d\_geo\_concat2')

                    n\_concat2 = cropconcat\_layer(conv6, n\_deconv2, 3, name='n\_geo\_concat2')

                    c\_concat2 = cropconcat\_layer(conv6, c\_deconv2, 3, name='c\_geo\_concat2')

                    d\_deconv2\_2 = slim.conv2d(d\_concat2, root\_feature \* 4, scope='d\_geo\_deonv2\_2')

                    n\_deconv2\_2 = slim.conv2d(n\_concat2, root\_feature \* 4, scope='n\_geo\_deonv2\_2')

                    c\_deconv2\_2 = slim.conv2d(c\_concat2, root\_feature \* 4, scope='c\_geo\_deonv2\_2')

                    d\_deconv2\_3 = slim.conv2d(d\_deconv2\_2, root\_feature \* 4, scope='d\_geo\_deconv2\_3')

                    n\_deconv2\_3 = slim.conv2d(n\_deconv2\_2, root\_feature \* 4, scope='n\_geo\_deconv2\_3')

                    c\_deconv2\_3 = slim.conv2d(c\_deconv2\_2, root\_feature \* 4, scope='c\_geo\_deconv2\_3')

                    d\_deconv3 = slim.conv2d\_transpose(d\_deconv2\_3, root\_feature \* 2, scope='d\_geo\_deconv3\_1')

                    n\_deconv3 = slim.conv2d\_transpose(n\_deconv2\_3, root\_feature \* 2, scope='n\_geo\_deconv3\_1')

                    c\_deconv3 = slim.conv2d\_transpose(c\_deconv2\_3, root\_feature \* 2, scope='c\_geo\_deconv3\_1')

                    d\_concat3 = cropconcat\_layer(conv4, d\_deconv3, 3, name='d\_geo\_concat3')

                    n\_concat3 = cropconcat\_layer(conv4, n\_deconv3, 3, name='n\_geo\_concat3')

                    c\_concat3 = cropconcat\_layer(conv4, c\_deconv3, 3, name='c\_geo\_concat3')

                    d\_deconv3\_2 = slim.conv2d(d\_concat3, root\_feature \* 2, scope='d\_geo\_deconv3\_2')

                    n\_deconv3\_2 = slim.conv2d(n\_concat3, root\_feature \* 2, scope='n\_geo\_deconv3\_2')

                    c\_deconv3\_2 = slim.conv2d(c\_concat3, root\_feature \* 2, scope='c\_geo\_deconv3\_2')

                    d\_deconv3\_3 = slim.conv2d(d\_deconv3\_2, root\_feature \* 2, scope='d\_geo\_deconv3\_3')

                    n\_deconv3\_3 = slim.conv2d(n\_deconv3\_2, root\_feature \* 2, scope='n\_geo\_deconv3\_3')

                    c\_deconv3\_3 = slim.conv2d(c\_deconv3\_2, root\_feature \* 2, scope='c\_geo\_deconv3\_3')

                    d\_deconv4 = slim.conv2d\_transpose(d\_deconv3\_3, root\_feature, scope='d\_geo\_deconv4\_1')

                    n\_deconv4 = slim.conv2d\_transpose(n\_deconv3\_3, root\_feature, scope='n\_geo\_deconv4\_1')

                    c\_deconv4 = slim.conv2d\_transpose(c\_deconv3\_3, root\_feature, scope='c\_geo\_deconv4\_1')

                    d\_concat4 = cropconcat\_layer(conv2, d\_deconv4, 3, name='d\_geo\_concat4')

                    n\_concat4 = cropconcat\_layer(conv2, n\_deconv4, 3, name='n\_geo\_concat4')

                    c\_concat4 = cropconcat\_layer(conv2, c\_deconv4, 3, name='c\_geo\_concat4')

                    d\_deconv4\_2 = slim.conv2d(d\_concat4, root\_feature, scope='d\_geo\_deconv4\_2')

                    n\_deconv4\_2 = slim.conv2d(n\_concat4, root\_feature, scope='n\_geo\_deconv4\_2')

                    c\_deconv4\_2 = slim.conv2d(c\_concat4, root\_feature, scope='c\_geo\_deconv4\_2')

                    d\_deconv4\_3 = slim.conv2d(d\_deconv4\_2, root\_feature, scope='d\_geo\_deconv4\_3')

                    n\_deconv4\_3 = slim.conv2d(n\_deconv4\_2, root\_feature, scope='n\_geo\_deconv4\_3')

                    c\_deconv4\_3 = slim.conv2d(c\_deconv4\_2, root\_feature, scope='c\_geo\_deconv4\_3')

                    res\_d = slim.conv2d(d\_deconv4\_3, 1, kernel\_size=[1, 1], scope='d\_geo\_output')  # depth

                    res\_n = slim.conv2d(n\_deconv4\_3, 3, kernel\_size=[1, 1], scope='n\_geo\_output')  # normal

                    res\_c = slim.conv2d(c\_deconv4\_3, 1, kernel\_size=[1, 1], activation\_fn=tf.nn.sigmoid,

                                        scope='c\_geo\_output')  # confidence map

                    logit\_d = tf.identity(res\_d, name='output\_d')

                    logit\_n = tf.identity(res\_n, name='output\_n')

                    logit\_c = tf.identity(res\_c, name='output\_c')

        #geom\_net\_variables = tf.contrib.framework.get\_variables(g\_vs)

        geom\_net\_variables = slim.get\_variables(g\_vs)

        return logit\_d, logit\_n, logit\_c, geom\_net\_variables

## 4 用到的所有自定义工具类函数 tools.py

# 工具箱

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import unicode\_literals

import os

import cv2

import time

import math

import logging

import open3d as o3d

import numpy as np

import OpenGL.GL as GL

import pyqtgraph as pg

import tensorflow as tf

import pyqtgraph.opengl as gl

from PIL import Image

from PyQt5 import QtCore, QtGui

from PyQt5.QtWidgets import QApplication

util\_logger = logging.getLogger('main.utils')

def slice\_tensor(tensor1, tensor2):

    with tf.compat.v1.name\_scope("slice\_tenosr") as \_:

        t1\_shape = tf.shape(input=tensor1)

        t2\_shape = tf.shape(input=tensor2)

        offsets = [0, (t1\_shape[1] - t2\_shape[1]) // 2, (t1\_shape[2] - t2\_shape[2]) // 2, 0]

        size = [-1, t2\_shape[1], t2\_shape[2], -1]

        return tf.slice(tensor1, offsets, size)

def make\_dir(folder\_fn):

    if tf.io.gfile.exists(folder\_fn):

        tf.io.gfile.rmtree(folder\_fn)

    tf.io.gfile.makedirs(folder\_fn)

def dump\_params(path, params):

    util\_logger.info('Training settings:')

    with open(path + r'/params.txt', 'w') as f:

        for param in params:

            f.write('{}: {}\n'.format(param, params[param]))

            util\_logger.info('{}: {}'.format(param, params[param]))

def cropconcat\_layer(tensor1, tensor2, concat\_dim=1, name=None):

    with tf.compat.v1.name\_scope(name) as \_:

        t1\_shape = tensor1.get\_shape().as\_list()

        t2\_shape = tensor2.get\_shape().as\_list()

        if t1\_shape[1] != t2\_shape[1] and t1\_shape[2] != t2\_shape[2]:

            offsets = [0, (t1\_shape[1] - t2\_shape[1]) // 2, (t1\_shape[2] - t2\_shape[2]) // 2, 0]

            size = [-1, t2\_shape[1], t2\_shape[2], -1]

            t1\_crop = tf.slice(tensor1, offsets, size)

            output = tf.concat([t1\_crop, tensor2], concat\_dim)

        else:

            output = tf.concat([tensor1, tensor2], concat\_dim)

        return output

# 预处理：用opencv读取10张单通道输入图 并转换为(1,256,256,1)的张量

def read10pics(imgdir):

    fna = os.path.join(imgdir, 'npr.png')

    fnb = os.path.join(imgdir, 'ds.png')

    fnc = os.path.join(imgdir, 'fLMask.png')

    fnd = os.path.join(imgdir, 'fLInvMask.png')

    fne = os.path.join(imgdir, 'clIMask.png')

    fnf = os.path.join(imgdir, 'shapeMask.png')

    fng = os.path.join(imgdir, 'dsMask.png')

    fnh = os.path.join(imgdir, '2dMask.png')

    fni = os.path.join(imgdir, 'sLMask.png')

    fnj = os.path.join(imgdir, 'curvMag.png')

    npr = cv2.imread(fna, 0)

    npr = cv2.resize(npr, (256, 256))

    npr = npr[np.newaxis, :, :, np.newaxis]

    npr.astype(np.float32)

    npr = npr / 255

    npr = tf.convert\_to\_tensor(npr)

    ds = cv2.imread(fnb, 0)

    ds = cv2.resize(ds, (256, 256))

    ds = ds[np.newaxis, :, :, np.newaxis]

    ds.astype(np.float32)

    ds = ds / 127                 #注意这里 网络输入的ds是[0,2] 图像灰度读取是[0,255] 所以/127就是从[0,255]到[0,2]的区域映射

    ds = tf.convert\_to\_tensor(ds)

    fm = cv2.imread(fnc, 0)

    fm = cv2.resize(fm, (256, 256))

    fm = fm[np.newaxis, :, :, np.newaxis]

    fm.astype(np.float32)

    fm = fm / 255

    fm = tf.convert\_to\_tensor(fm)

    fmi = cv2.imread(fnd, 0)

    fmi = cv2.resize(fmi, (256, 256))

    fmi = fmi[np.newaxis, :, :, np.newaxis]

    fmi.astype(np.float32)

    fmi = fmi / 255

    fmi = tf.convert\_to\_tensor(fmi)

    cm = cv2.imread(fne, 0)

    cm = cv2.resize(cm, (256, 256))

    cm = cm[np.newaxis, :, :, np.newaxis]

    cm.astype(np.float32)

    cm = cm / 255

    cm = tf.convert\_to\_tensor(cm)

    sm = cv2.imread(fnf, 0)

    sm = cv2.resize(sm, (256, 256))

    sm = sm[np.newaxis, :, :, np.newaxis]

    sm.astype(np.float32)

    sm = sm / 255

    sm = tf.convert\_to\_tensor(sm)

    dsm = cv2.imread(fng, 0)

    dsm = cv2.resize(dsm, (256, 256))

    dsm = dsm[np.newaxis, :, :, np.newaxis]

    dsm.astype(np.float32)

    dsm = dsm / 255

    dsm = tf.convert\_to\_tensor(dsm)

    msk = cv2.imread(fnh, 0)

    msk = cv2.resize(msk, (256, 256))

    msk = msk[np.newaxis, :, :, np.newaxis]

    msk.astype(np.float32)

    msk = msk / 255

    msk = tf.convert\_to\_tensor(msk)

    slm = cv2.imread(fni, 0)

    slm = cv2.resize(slm, (256, 256))

    slm = slm[np.newaxis, :, :, np.newaxis]

    slm.astype(np.float32)

    slm = slm / 255

    slm = tf.convert\_to\_tensor(slm)

    cur = cv2.imread(fnj, 0)

    cur = cv2.resize(cur, (256, 256))

    cur = cur[np.newaxis, :, :, np.newaxis]

    cur.astype(np.float32)

    cur = (cur / 63)-2

    cur = tf.convert\_to\_tensor(cur)

    return npr, ds, fm, fmi, cm, sm, dsm, msk, slm, cur

# 后处理：将深度图+法线图(当做rgb图)转换为ply点云文件(但是点云输出并没有用到rgb图 我把他删了。。。)

class point\_cloud\_generator():

    def \_\_init\_\_(self, rgb\_file, depth\_file, pc\_file, focal\_length, scalingfactor):

        self.rgb\_file = rgb\_file

        self.depth\_file = depth\_file

        self.pc\_file = pc\_file

        self.focal\_length = focal\_length

        self.scalingfactor = scalingfactor

        self.rgb = Image.open(rgb\_file)

        self.depth = Image.open(depth\_file).convert('I')

        self.width = self.rgb.size[0]

        self.height = self.rgb.size[1]

    def calculate(self):

        t1=time.time()

        depth = np.asarray(self.depth).T

        self.Z = depth / self.scalingfactor

        X = np.zeros((self.width, self.height))

        Y = np.zeros((self.width, self.height))

        for i in range(self.width):

            X[i, :] = np.full(X.shape[1], i)

        self.X = ((X - self.width / 2) \* self.Z) / self.focal\_length

        for i in range(self.height):

            Y[:, i] = np.full(Y.shape[0], i)

        self.Y = ((Y - self.height / 2) \* self.Z) / self.focal\_length

        df=np.zeros((6,self.width\*self.height))

        df[0] = self.X.T.reshape(-1)

        df[1] = -self.Y.T.reshape(-1)

        df[2] = -self.Z.T.reshape(-1)

        img = np.array(self.rgb)#(256,256)

        df[3] = img[ :, :, 0:1].reshape(-1)

        df[4] = img[ :, :, 1:2].reshape(-1)

        df[5] = img[ :, :, 2:3].reshape(-1)

        self.df=df

        t2=time.time()

        print('calcualte 3d point cloud Done.',t2-t1)

    def write\_ply(self):

        t1=time.time()

        float\_formatter = lambda x: "%.4f" % x

        points =[]

        for i in self.df.T:

            # 点云降噪：要求写入ply的点坐标在半径在0.02的球体外边 也就是挖掉中心那一簇密集点

            if((abs(float(float\_formatter(i[0]))) >= 0.02) or

               (abs(float(float\_formatter(i[1]))) >= 0.02) or

               (abs(float(float\_formatter(i[2]))) >= 0.02) ):

                points.append("{} {} {}\n".format(float\_formatter(i[0]), float\_formatter(i[1]), float\_formatter(i[2])))

        file = open(self.pc\_file, "w")

        file.write('''ply

        format ascii 1.0

        element vertex %d

        property float x

        property float y

        property float z

        end\_header

        %s

        ''' % (len(points), "".join(points)))

        file.close()

        t2=time.time()

        print("Write into .ply file Done.",t2-t1)

# 后处理：用open3d直接把ply点云文件输出为obj网格文件

# 泊松重建

def ply\_pcd\_to\_obj\_mesh\_byo3d\_possion(pcd\_file,mesh\_file):

    print('将ply点云泊松重建为obj网格')

    # 导入点云

    pcd = o3d.io.read\_point\_cloud(pcd\_file)

    # pcd.paint\_uniform\_color([1, 0.706, 0])#指定点云显示为金色

    # 计算法向量

    radius = 0.01  # 搜索半径

    max\_nn = 30  # 邻域内用于估算法线的最大点数

    pcd.estimate\_normals(search\_param=o3d.geometry.KDTreeSearchParamHybrid(radius, max\_nn))

    # 泊松重建三角形

    poisson\_mesh = o3d.geometry.TriangleMesh.create\_from\_point\_cloud\_poisson(pcd, depth=8, width=0, scale=1.1, linear\_fit=False)[0]

    # 按照bbox去除伪影

    bbox = pcd.get\_axis\_aligned\_bounding\_box() # 包围盒

    p\_mesh\_crop = poisson\_mesh.crop(bbox)

    # 导出为100000个三角形组成的网格模型

    p\_mesh\_crop\_lod = p\_mesh\_crop.simplify\_quadric\_decimation(100000)

    o3d.io.write\_triangle\_mesh(mesh\_file, p\_mesh\_crop\_lod,write\_ascii=True)

# BPA重建

def ply\_pcd\_to\_obj\_mesh\_byo3d\_bpa(pcd\_file,mesh\_file):

    print('将ply点云BPA重建为obj网格')

    # 导入点云

    pcd = o3d.io.read\_point\_cloud(pcd\_file)

    # 计算法向量

    radius = 0.01  # 搜索半径

    max\_nn = 30  # 邻域内用于估算法线的最大点数

    pcd.estimate\_normals(search\_param=o3d.geometry.KDTreeSearchParamHybrid(radius, max\_nn))

    # BPA重建三角形

    radii = [0.005, 0.01, 0.02, 0.04]

    bpa\_mesh = o3d.geometry.TriangleMesh.create\_from\_point\_cloud\_ball\_pivoting(pcd,o3d.utility.DoubleVector(radii))

    # 重建为100000个三角形组成的网格模型

    b\_mesh\_crop\_lod = bpa\_mesh.simplify\_quadric\_decimation(100000)

    # 去除伪影

    b\_mesh\_crop\_lod.remove\_degenerate\_triangles()

    b\_mesh\_crop\_lod.remove\_duplicated\_triangles()

    b\_mesh\_crop\_lod.remove\_duplicated\_vertices()

    b\_mesh\_crop\_lod.remove\_non\_manifold\_edges()

    o3d.io.write\_triangle\_mesh(mesh\_file, b\_mesh\_crop\_lod,write\_ascii=True)

# 后处理：将深度图转换为obj文件

def vete(v, vt):

    return str(v)+"/"+str(vt)

def create\_obj(depthPath, objPath):

    print("文件名",depthPath,objPath)

    img = cv2.imread(depthPath, -1).astype(np.float32) / 1000.0

    w = img.shape[1]

    h = img.shape[0]

    FOV = math.pi/4

    D = (img.shape[0]/2)/math.tan(FOV/2)

    with open(objPath,"w") as f:

        ids = np.zeros((img.shape[1], img.shape[0]), int)

        vid = 1

        for u in range(0, w):

            for v in range(h-1, -1, -1):

                d = img[v, u]

                ids[u,v] = vid

                if d == 0.0:

                    ids[u,v] = 0

                vid += 1

                x = u - w/2

                y = v - h/2

                z = -D

                norm = 1 / math.sqrt(x\*x + y\*y + z\*z)

                t = d/(z\*norm)

                x = -t\*x\*norm

                y = t\*y\*norm

                z = -t\*z\*norm

                f.write("v " + str(x) + " " + str(y) + " " + str(z) + "\n")

        for u in range(0, img.shape[1]):

            for v in range(0, img.shape[0]):

                f.write("vt " + str(u/img.shape[1]) + " " + str(v/img.shape[0]) + "\n")

        for u in range(0, img.shape[1]-1):

            for v in range(0, img.shape[0]-1):

                v1 = ids[u,v]; v2 = ids[u+1,v]; v3 = ids[u,v+1]; v4 = ids[u+1,v+1];

                if v1 == 0 or v2 == 0 or v3 == 0 or v4 == 0:

                    continue

                f.write("f " + vete(v1,v1) + " " + vete(v2,v2) + " " + vete(v3,v3) + "\n")

                f.write("f " + vete(v3,v3) + " " + vete(v2,v2) + " " + vete(v4,v4) + "\n")

    print("Write into .obj file Done")

# 后处理：解析obj中的顶点和面

def parseObj(obj\_filename):

    with open(obj\_filename) as file:

        verts = []

        faces = []

        while 1:

            line = file.readline()

            if not line:

                break

            strs = line.split(" ")

            if strs[0] == "v":

                # 方法一：试图在读取的时候直接舍弃全0顶点

                #if((float(strs[1])!=-0.0 and float(strs[2])!=-0.0 and float(strs[3])!=-0.0)):

                verts.append((float(strs[1])\*500, float(strs[2])\*500, float(strs[3])\*500))

            elif strs[0] == 'f':

                face = []

                for v in strs[1:]:

                    a = v.split('/')

                    face.append(float(a[0]))

                faces.append(face)

            # points由列表转变为矩阵

        verts = np.array(verts).astype('int64')

        #方法2：试图在顶点数组中去除全0顶点 方法1 2都失败了 仅仅删除顶点是没有用的 还需要删除面 但是肯定不是删除全0面 因为压根没有全0面

        #not\_row = verts[[not np.all(verts[i] == 0) for i in range(verts.shape[0])], :]

        #print(np.shape(not\_row))

        faces = np.array(faces).astype('int64')

    return verts,faces

# 后处理：视图右上角数据显示

class GLPainterItem(gl.GLGraphicsItem.GLGraphicsItem):

    def \_\_init\_\_(self, \*\*kwds):

        super().\_\_init\_\_()

        glopts = kwds.pop('glOptions', 'additive')

        self.setGLOptions(glopts)

    def compute\_projection(self):

        modelview = GL.glGetDoublev(GL.GL\_MODELVIEW\_MATRIX)

        projection = GL.glGetDoublev(GL.GL\_PROJECTION\_MATRIX)

        mvp = projection.T @ modelview.T

        mvp = QtGui.QMatrix4x4(mvp.ravel().tolist())

        # note that QRectF.bottom() != QRect.bottom()

        rect = QtCore.QRectF(self.view().rect())

        ndc\_to\_viewport = QtGui.QMatrix4x4()

        ndc\_to\_viewport.viewport(rect.left(), rect.bottom(), rect.width(), -rect.height())

        return ndc\_to\_viewport \* mvp

    def paint(self):

        self.setupGLState()

        painter = QtGui.QPainter(self.view())

        self.draw(painter)

        painter.end()

    def draw(self, painter):

        painter.setPen(QtCore.Qt.GlobalColor.white)

        painter.setRenderHints(QtGui.QPainter.RenderHint.Antialiasing | QtGui.QPainter.RenderHint.TextAntialiasing)

        rect = self.view().rect()

        af = QtCore.Qt.AlignmentFlag

        painter.drawText(rect, af.AlignTop | af.AlignRight, 'TR')

        painter.drawText(rect, af.AlignBottom | af.AlignLeft, 'BL')

        painter.drawText(rect, af.AlignBottom | af.AlignRight, 'BR')

        opts = self.view().cameraParams()

        lines = []

        center = opts['center']

        lines.append(f"center : ({center.x():.1f}, {center.y():.1f}, {center.z():.1f})")

        for key in ['distance', 'fov', 'elevation', 'azimuth']:

            lines.append(f"{key} : {opts[key]:.1f}")

        xyz = self.view().cameraPosition()

        lines.append(f"xyz : ({xyz.x():.1f}, {xyz.y():.1f}, {xyz.z():.1f})")

        info = "\n".join(lines)

        painter.drawText(rect, af.AlignTop | af.AlignLeft, info)

        project = self.compute\_projection()

        hsize = 32 // 2

        for xi in range(-hsize, hsize+1):

            for yi in range(-hsize, hsize+1):

                if xi == -hsize and yi == -hsize:

                    # skip one corner for visual orientation

                    continue

                vec3 = QtGui.QVector3D(xi, yi, 0)

                pos = project.map(vec3).toPointF()

                painter.drawEllipse(pos, 1, 1)