

# 每天下班前手动备份！

7.27

main\_window.py

```
# 导入系统模块
import subprocess
import threading
import chardet
import json
import os
import re
import socket

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton, QMessageBox, QApplication

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

class MainWindow(QtWidgets.QMainWindow):
    """
    主窗口类，用于显示3D模型和提供相关功能。

    Attributes:
        actor_to_filename: dict, 保存每个actor和对应的STL文件名的映射。
        actor_to_color: dict, 保存每个actor和对应的颜色的映射。
        bom_data: list, BOM表数据。
        render: vtkRenderer对象，用于渲染3D模型。
        actors: list, 所有的vtkActor对象列表。
        is_hide: bool, 是否隐藏模型。
        is_middle_button_pressed: bool, 鼠标中键是否按下。

    Methods:
        __init__: 初始化方法，创建主窗口，读取STL文件，设置布局和功能按钮。
        create_layout: 创建布局。
        load_stl_files: 读取STL文件。
        create_buttons: 创建功能按钮。
        print_to_gui: 将文本显示到GUI中。
        reset: 重置模型和交互器状态。
        show_pdf_helper: 辅助函数，打开关联的PDF文件。
        show_pdf: 显示关联的PDF文件。
        show_current_selected_model: 显示当前选中的模型的父模型。
        middle_button_press: 模拟鼠标中键按下事件。
    """
```

```

"""

# 常量定义
STL_DIR = 'FDK' # .stl文件夹路径
BOM_FILE = '40881_40650_json.json' # bom表, 转成json格式, 让程序更好读取

def __init__(self):
    """
    初始化方法, 创建主窗口, 读取STL文件, 设置布局和功能按钮。
    """
    # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映
射:
    self.actor_to_filename = {}
    # 创建一个新的字典储存actor的颜色
    self.actor_to_color = {}
    # 调用父类的初始化方法
    super().__init__()

    # 获取当前路径
    current_dir = os.path.dirname(os.path.abspath(__file__))
    # 适配exe文件, 当使用代码调试时, 需要注释掉下面两行
    current_dir = os.path.dirname(current_dir)
    current_dir = os.path.dirname(current_dir)
    # 获取STL文件夹路径
    stl_dir = os.path.join(current_dir, self.STL_DIR)

    # 获取bom表编码信息
    with open(self.BOM_FILE, 'rb') as file:
        raw_data = file.read()
        bom_encoding = chardet.detect(raw_data)['encoding']
    # 获取模型对应BOM表信息
    with open(self.BOM_FILE, 'r', encoding=bom_encoding) as file:
        self.bom_data = json.load(file)

    # 设置窗口标题和大小
    self.setWindowTitle('FlightDeck 3D模型展示')
    # 设置窗口的固定大小
    self.resize(3840, 2160)

    # 创建布局
    self.current_content_layout, self.other_function_layout =
self.create_layouts()

    # 创建文本框
    self.create_text_box()

    # 创建 QVTKRenderWindowInteractor 实例
    self.vtkWidget = QVTKRenderWindowInteractor(self)

    # 创建VTK渲染器和窗口
    self.render = vtk.vtkRenderer()
    self.vtkWidget.GetRenderWindow().AddRenderer(self.render)
    self.render.SetBackground(1.0, 1.0, 1.0)

```

```
# 储存所有actor
self.actors = []

# 调用load_stl_files方法加载STL文件
self.load_stl_files(stl_dir)

# 初始化交互器
self.interactorStyle = MyInteractorStyle(
    self.render, self.actors, self)
self.vtkWidget.GetRenderWindow().GetInteractor(
).SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout (主布局) 中
self.current_content_layout.addWidget(self.vtkWidget)

# 调用 create_buttons 方法创建按钮
self.create_buttons(self.current_content_layout)
self.right_button_layout = self.create_inform_button()
self.other_function_layout.addLayout(self.right_button_layout)

# 设置变量记录是否隐藏模型，如果隐藏模型，记录当前父模型信息
self.is_hide = False
self.current_model = []

# 记录当前鼠标中键点击状态
self.is_middle_button_down = False

# 记录选择样式
self.select_type = False

# 记录右侧界面状态
self.is_chat = False

# 记录右侧初始文本框中的内容（用于切换后重置）
self.previous_text = ""

# 记录模型回答
self.answer = ""
self.source = []

def create_chat_part(self):
    # 创建label
    self.label1 = QtWidgets.QLabel("Response")
    self.label2 = QtWidgets.QLabel("Question")

    # 创建文本框
    self.response_box = QtWidgets.QTextEdit()
    self.question_box = QtWidgets.QTextEdit()
    self.response_box.setReadOnly(True)

def create_text_box(self):
    # 创建 QTextEdit 实例
```

```
self.text_edit = QTextEdit(self)
self.text_edit.setReadOnly(True)
self.other_function_layout.addWidget(self.text_edit)

def create_layouts(self):
    """
    创建主布局、当前内容布局和其他功能布局，并设置布局关系和占据比例。
    """
    # 创建主布局
    main_layout = QtWidgets.QHBoxLayout()
    main_widget = QtWidgets.QWidget(self)
    main_widget.setLayout(main_layout)
    self.setCentralWidget(main_widget)

    # 创建当前内容布局
    current_content_widget = QtWidgets.QWidget(self)
    current_content_layout = QtWidgets.QVBoxLayout()
    current_content_widget.setLayout(current_content_layout)
    main_layout.addWidget(current_content_widget, 3) # 可以按照要求调整比例

    # 创建其他功能布局
    other_function_widget = QtWidgets.QWidget(self)
    other_function_layout = QtWidgets.QVBoxLayout()
    other_function_widget.setLayout(other_function_layout)
    main_layout.addWidget(other_function_widget, 1)

    return current_content_layout, other_function_layout

def load_stl_files(self, stl_dir):
    """
    加载STL文件，并将模型添加到渲染器和本地的actor列表中。

    Args:
        stl_dir: STL文件夹路径。
    """
    # 创建进度条
    process = QMessageBox(self)
    process.setWindowTitle("Loading")

    # 计算总共需要加载的STL文件数量
    total_num = 0
    current_num = 0
    for file_name in os.listdir(stl_dir):
        if file_name.lower().endswith('.stl'):
            total_num += 1

    # 设置进度条
    process.setText("Loading STL"
files..." + str(current_num) + "/" + str(total_num))
    process.setStandardButtons(QtWidgets.QMessageBox.Cancel)
    process.show()
    for file_name in os.listdir(stl_dir):
        if file_name.lower().endswith('.stl'):
            current_num += 1
```

典

# 青灰色

# 钢蓝色

深紫色

```

file_path = os.path.join(stl_dir, file_name)
# print(file_name)

# 创建STL读取器, 读取STL文件
reader = vtk.vtkSTLReader()
reader.SetFileName(file_path)
reader.Update()

# 创建transform以保持模型原有位置 (相对于原点)
transform = vtk.vtkTransform()
transformFilter = vtk.vtkTransformPolyDataFilter()
transformFilter.SetInputConnection(reader.GetOutputPort())
transformFilter.SetTransform(transform)
transformFilter.Update()

# 将actor添加到渲染器和本地的actor列表中, 并保存到actor_to_filename字典

mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(transformFilter.GetOutputPort())
actor = vtk.vtkActor()
actor.SetMapper(mapper)
self.actors.append(actor)
# 设置模型颜色
if '40881' in file_name:
    actor.GetProperty().SetColor(1, 0.5, 0) # 橙色
elif '40650' in file_name:
    actor.GetProperty().SetColor(0, 1, 1) # 蓝色
elif '39000' in file_name:
    actor.GetProperty().SetColor(0.3, 1, 0.5) # 绿色
elif '41565' in file_name:
    actor.GetProperty().SetColor(51/255.0, 102/255.0, 102/255.0)
elif '19848' in file_name or "FDK" in file_name:
    actor.GetProperty().SetColor(51/255.0, 102/255.0, 153/255.0)
elif '42755' in file_name or "X球管" in file_name:
    actor.GetProperty().SetColor(102/255.0, 0/255.0, 102/255.0) #
else:
    actor.GetProperty().SetColor(153/255.0, 51/255.0, 102/255.0)
self.render.AddActor(actor)
self.actor_to_filename[actor] = file_name
self.actor_to_color[actor] = actor.GetProperty().GetColor()
# 更新进度条
process.setText("Loading STL
files..." + str(current_num) + "/" + str(total_num))
process.show()
# 使进度条可以正常显示
QApplication.processEvents()
process.close()

def create_chat_button(self):
    """

```

创建与模型交互模式的按钮并且返回布局。

```
"""
text_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
# 创建submit按钮
self.submit_button = QPushButton('Submit', self)
self.submit_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.submit_button.clicked.connect(self.submit)
text_button_layout.addWidget(self.submit_button)

# 创建update按钮
self.update_button = QPushButton('Update', self)
self.update_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.update_button.clicked.connect(self.updateResponse)
text_button_layout.addWidget(self.update_button)
self.update_button.setEnabled(False)

# 创建switch按钮
self.switch_button = QPushButton('Switch', self)
self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.switch_button.clicked.connect(self.switch)
text_button_layout.addWidget(self.switch_button)

return text_button_layout
```

```
def create_inform_button(self):
```

```
"""
```

创建点击获取信息模式的按钮并且返回布局。

```
"""
```

```
inform_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
# 创建 Show PDF 按钮
self.show_pdf_button = QPushButton('Show PDF', self)
self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
inform_button_layout.addWidget(self.show_pdf_button)
self.show_pdf_button.clicked.connect(self.show_pdf)
if self.interactorStyle.pdf_file is None:
    self.show_pdf_button.setEnabled(False)
else:
    self.show_pdf_button.setEnabled(True)

# 创建switch按钮
self.switch_button = QPushButton('Switch', self)
self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
inform_button_layout.addWidget(self.switch_button)
self.switch_button.clicked.connect(self.switch)

return inform_button_layout
```

```
def create_buttons(self, current_content_layout):
```

```
"""
```

创建"Reset"按钮,"Show Model"按钮,"Move"按钮和"Select"按钮并添加到布局中。

```
"""
```

```
self.button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
current_content_layout.addLayout(self.button_layout) # 将水平布局添加到当前内容布局中
```

前内容布局中

布局中

```
# 创建 Reset 按钮
self.reset_button = QPushButton('Reset', self)
self.reset_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.reset_button.clicked.connect(self.reset)
self.button_layout.addWidget(self.reset_button) # 将 Reset 按钮添加到水平

self.button_layout.addSpacing(20) # 添加间距

# 创建 show current selected model 按钮
self.show_current_selected_model_button = QPushButton(
    'Show Model', self)
self.show_current_selected_model_button.setFixedSize(
    80, 40) # 设置按钮的固定宽度和高度
self.show_current_selected_model_button.clicked.connect(
    self.show_current_selected_model)
self.button_layout.addWidget(self.show_current_selected_model_button)
self.show_current_selected_model_button.setEnabled(False)
self.button_layout.addSpacing(20) # 添加间距
```

和高度

```
# 创建消除单个模型的按钮
self.hide_model_button = QPushButton('Hide Model', self)
self.hide_model_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.hide_model_button.clicked.connect(self.hide_model)
self.button_layout.addWidget(self.hide_model_button)
self.button_layout.addSpacing(20) # 添加间距
self.hide_model_button.setEnabled(False)

# 创建对部分显示模型重置的按钮
self.reset_current_model_button = QPushButton('Reset Current', self)
self.reset_current_model_button.setFixedSize(100, 40) # 设置按钮的固定宽度

self.reset_current_model_button.clicked.connect(self.reset_current_model)
self.button_layout.addWidget(self.reset_current_model_button)
self.button_layout.addSpacing(20) # 添加间距
self.reset_current_model_button.setEnabled(False)
```

和高度

```
# 创建按钮模拟鼠标中键点击事件
self.middle_button_click_button = QPushButton('Move', self)
self.middle_button_click_button.setFixedSize(80, 40) # 设置按钮的固定宽度

self.middle_button_click_button.clicked.connect(
    self.middle_button_click)
self.button_layout.addWidget(self.middle_button_click_button)
self.button_layout.addSpacing(20) # 添加间距

# 创建按钮切换选择模式
self.selection_type_button = QPushButton('Select', self)
self.selection_type_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.selection_type_button.clicked.connect(self.selection_type)
self.button_layout.addWidget(self.selection_type_button)
```

```
def print_to_gui(self, text):
    """
```

将文本显示到GUI中。

Args:

text: 要显示的文本。

"""

# 在文本编辑区域中添加新的文本

if self.is\_chat == False:

self.text\_edit.append(text)

else:

if self.previous\_text == "":

self.previous\_text = text

else:

self.previous\_text += '\n' + text

def reset(self):

"""

重置模型和交互器状态。

"""

self.show\_current\_selected\_model\_button.setEnabled(False)

self.hide\_model\_button.setEnabled(False)

self.reset\_current\_model\_button.setEnabled(False)

self.current\_model = []

self.is\_hide = False

self.previous\_text = ""

if self.is\_chat == False:

self.text\_edit.clear()

self.show\_pdf\_button.setEnabled(False)

else:

if self.submit\_button.isEnabled() == True:

self.response\_box.clear()

self.question\_box.clear()

self.interactorStyle.pdf\_file = None

for actor in self.actors:

actor.GetProperty().SetColor(self.actor\_to\_color[actor])

actor.GetProperty().SetOpacity(1)

actor.SetPickable(True)

actor.SetVisibility(True)

self.render.ResetCamera()

self.vtkWidget.update()

def show\_pdf\_helper(self):

"""

辅助函数，打开关联的PDF文件。

"""

try:

subprocess.run(

['C:\Program Files\Wondershare\PDFelement\PDFelement.exe',

self.interactorStyle.pdf\_file])

except Exception as e:

print(e)

self.print\_to\_gui(

'打开PDF文件失败，请检查PDFelement是否安装，并且检查路径是否

为:C:\Program Files\Wondershare\PDFelement\PDFelement.exe')



```
def show_pdf(self):
    """
    显示关联的PDF文件。
    """
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

def switch(self):
    """
    切换与模型交互和点击获取信息之间的模式。
    """
    if self.is_chat == False:
        self.is_chat = True
        self.create_chat_part()
        # # 如果之前有文本，将其记录
        if self.text_edit.toPlainText() != "":
            self.previous_text = self.text_edit.toPlainText()
        # 删除之前的组件，避免因内存导致的重叠
        # 由于删除组件时会导致布局中的组件顺序发生变化，所以需要从后往前删除
        item_list = list(range(self.other_function_layout.count()))
        item_list.reverse()
        for i in item_list:
            item = self.other_function_layout.takeAt(i)
            self.other_function_layout.removeItem(item)
            # 如果是组件，将其从布局中删除
            if item.widget():
                item.widget().setParent(None)
            else:
                # 如果是布局，递归删除布局中的组件
                deleteItemOfLayout(item.layout())

        # 添加组件
        self.other_function_layout.addWidget(self.label1)
        self.other_function_layout.addWidget(self.response_box)
        self.other_function_layout.addWidget(self.label2)
        self.other_function_layout.addWidget(self.question_box)
        self.right_button_layout = self.create_chat_button()
        self.other_function_layout.addLayout(self.right_button_layout)

    else:
        # 恢复之前的组件之前先清空当前组件
        self.is_chat = False
        item_list = list(range(self.other_function_layout.count()))
        item_list.reverse()
        for i in item_list:
            item = self.other_function_layout.takeAt(i)
            self.other_function_layout.removeItem(item)
            if item.widget():
                item.widget().setParent(None)
            else:
                deleteItemOfLayout(item.layout())
        self.create_text_box()
        self.right_button_layout = self.create_inform_button()
        self.other_function_layout.addLayout(self.right_button_layout)
```

```

        # 如果之前有文本，将其显示
        if self.interactorStyle.pdf_file != "":
            self.text_edit.append(self.previous_text)
            self.previous_text = ""

    def submit(self):
        """
        获取模型回答，创建线程以避免阻塞主线程。使得模型展示界面可以在获取答案的同时进行交互。
        """
        # 如果回答框中有内容，清空回答框
        if self.response_box.toPlainText() != "":
            self.response_box.clear()
        # 创建线程来获取模型回答以避免阻塞主线程
        submit_thread = threading.Thread(target=self.submitHelper)
        submit_thread.start()

    def submitHelper(self):
        """
        辅助函数，用于在子线程中获取模型回答。
        """
        # 如果问题为空，直接返回
        if self.question_box.toPlainText() == "":
            return
        # 将submit按钮设置为不可用，避免重复点击
        # 将switch按钮设置为不可用，避免点击导致意外错误
        self.submit_button.setEnabled(False)
        self.switch_button.setEnabled(False)
        # 创建json数据用以发送给后端。根据需求更改knowledge_base_id,对于user1的模型，
        test all files是所有文件的知识库。
        # 在模型方可以根据需求细分数据库，如需修改可能需要根据具体需求对
        knowledge_base_id进行赋值方式的更改。
        request_data = {
            "knowledge_base_id": "test all files",
            "question": self.question_box.toPlainText(),
            "history": []
        }
        # 调用send_json_data函数获取模型回答
        self.answer, self.source = send_json_data(request_data)
        # 将update按钮设置为可用，以便用户更新模型回答（该功能为当前防止主线程被阻塞的必要功能，如果直接进行自动更新则会出现以下两种情况：
        # 1. 直接使用threading.Thread创建的线程会报错：cannot create children for a parent that is in a different thread
        # 2. 如果尝试使用自定义线程类：会出现QThread:Destroyed while thread is still running
        # 目前来说，并未找到方法解决这两个情况的方法，暂时使用让用户手动更新的方式以避免主线程被获取答案的函数阻塞)
        self.update_button.setEnabled(True)

    def updateResponse(self):
        # 将回答显示到回答框中
        self.response_box.append("Response: \n" + self.answer + "\n")
        # 由于source_documents是一个列表，需要将其转换为字符串
        source = ' '.join(map(str, self.source))

```

```

self.response_box.append("Source documents: \n" + source)
# 清空问题框
self.question_box.clear()
# 重置以下按钮与变量, 防止重复点击及错误显示
self.update_button.setEnabled(False)
self.submit_button.setEnabled(True)
self.switch_button.setEnabled(True)
self.answer = ""
self.source = []

def show_current_selected_model(self):
    """
    显示当前选中的模型所属的上级模型。
    """
    # 使部分模型的reset按钮可用
    self.reset_current_model_button.setEnabled(True)
    # 获取当前选中的模型的part number
    file_name = ""
    if self.interactorStyle.picked_actor:
        file_name = self.actor_to_filename[self.interactorStyle.picked_actor]
    elif self.interactorStyle.LastPickedActor:
        file_name =
self.actor_to_filename[self.interactorStyle.LastPickedActor]
    match = re.findall(r'\d{5}', file_name)
    # 如果没有匹配到part number, 说明是旋转平台或者FDK, 对此情况进行处理
    if len(match) == 0:
        for actor in self.actors:
            # 如果是包含FDK, 需要将19848和FDK都显示出来 (根据命名, 19848也是属于
            # FDK模型的一部分)
            if "FDK" in file_name:
                if '19848' not in self.actor_to_filename[actor] and "FDK" not
in self.actor_to_filename[actor]:
                    actor.SetVisibility(False)
                    actor.SetPickable(False)
                else:
                    # 将当前模型添加到current_model列表中, 以便后续重置, 提高速度
                    self.current_model.append(actor)
                    actor.SetVisibility(True)
                    actor.SetPickable(True)
            # 如果是包含X球管, 需要将42755和X球管都显示出来 (根据命名, 42755也是属
            # 于X球管模型的一部分)
            elif "X球管" in file_name:
                if '42755' not in self.actor_to_filename[actor] and "X球管"
not in self.actor_to_filename[actor]:
                    actor.SetVisibility(False)
                    actor.SetPickable(False)
                else:
                    # 将当前模型添加到current_model列表中, 以便后续重置, 提高速度
                    self.current_model.append(actor)
                    actor.SetVisibility(True)
                    actor.SetPickable(True)
            else:
                # 如果能匹配到part number, 在根据part number进行筛选之前, 先使用旋转平台,
                # fdk的筛选条件进行筛选以免遗漏

```

```

if "旋转平台" in file_name:
    for actor in self.actors:
        if "旋转平台" not in self.actor_to_filename[actor]:
            actor.SetVisibility(False)
            actor.SetPickable(False)
        else:
            self.current_model.append(actor)
            actor.SetVisibility(True)
            actor.SetPickable(True)
    elif "FDK" in file_name:
        for actor in self.actors:
            if '19848' not in self.actor_to_filename[actor] and "FDK" not
in self.actor_to_filename[actor]:
                actor.SetVisibility(False)
                actor.SetPickable(False)
            else:
                self.current_model.append(actor)
                actor.SetVisibility(True)
                actor.SetPickable(True)
        else:
            part_number = match[0]
            for actor in self.actors:
                if part_number == '19848':
                    if part_number not in self.actor_to_filename[actor] and
"FDK" not in self.actor_to_filename[actor]:
                        actor.SetVisibility(False)
                        actor.SetPickable(False)
                    else:
                        self.current_model.append(actor)
                        actor.SetVisibility(True)
                        actor.SetPickable(True)
                elif part_number == '42755':
                    if part_number not in self.actor_to_filename[actor] and "X
球管" not in self.actor_to_filename[actor]:
                        actor.SetVisibility(False)
                        actor.SetPickable(False)
                    else:
                        self.current_model.append(actor)
                        actor.SetVisibility(True)
                        actor.SetPickable(True)
                else:
                    # 根据part number进行筛选 (除19848和42755之外的模型命名都无需
考虑特殊情况)

                    if part_number not in self.actor_to_filename[actor]:
                        actor.SetVisibility(False)
                        actor.SetPickable(False)
                    else:
                        self.current_model.append(actor)
                        actor.SetVisibility(True)
                        actor.SetPickable(True)

# 更新渲染器
self.vtkWidget.update()
self.show_current_selected_model_button.setEnabled(False)
self.is_hide = True

```

```
def middle_button_click(self):
    """
    模拟鼠标中键点击事件。
    """
    self.is_middle_button_down = not self.is_middle_button_down

    if self.is_middle_button_down:
        self.interactorStyle.OnMiddleButtonDown()
    else:
        self.interactorStyle.OnMiddleButtonUp()

def selection_type(self):
    """
    切换选择模式。
    """
    self.select_type = not self.select_type

def hide_model(self):
    # 隐藏当前选中的模型
    self.interactorStyle.picked_actor.SetVisibility(False)
    self.interactorStyle.picked_actor.SetPickable(False)
    self.vtkWidget.update()
    # 重置交互器状态
    self.interactorStyle.reset_state()
    # 更改按钮状态
    self.hide_model_button.setEnabled(False)
    self.show_current_selected_model_button.setEnabled(False)
    # 根据情况清空对于的储存文本的变量
    if not self.is_chat:
        self.text_edit.clear()
    else:
        self.previous_text = ""
    self.show_pdf_button.setEnabled(False)

def reset_current_model(self):
    # 重置当前选中的模型
    for actor in self.current_model:
        actor.SetVisibility(True)
        actor.SetPickable(True)
        actor.GetProperty().SetColor(self.actor_to_color[actor])
        actor.GetProperty().SetOpacity(1)
    # 根据情况清空对于的储存文本的变量
    if not self.is_chat:
        self.text_edit.clear()
    else:
        self.previous_text = ""
    # 不掉用interactorStyle的reset_state函数, 避免重置所有模型
    self.show_pdf_button.setEnabled(False)
    self.interactorStyle.picked_actor = None
    self.interactorStyle.LastPickedActor = None
    self.interactorStyle.pdf_file = None
    self.hide_model_button.setEnabled(False)
    # 更新渲染器
```

```
self.vtkWidget.update()

# 链接模型获得答案的函数
def send_json_data(data):
    HOST = '10.1.4.12'
    PORT = 8082

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 设置尝试次数, 防止连接失败
    try:
        client_socket.connect((HOST, PORT))
    except Exception as e:
        index = 0
        print(e)
        while index < 5:
            try:
                client_socket.connect((HOST, PORT))
                break
            except Exception as e:
                print(e)
                index += 1
                if index == 5:
                    return 'Failed to get answer, please check status of GLM
server.', []
                continue

    count = 0
    response_data = None
    # 设置尝试次数, 防止获取答案失败
    while response_data is None:
        try:
            client_socket.sendall(json.dumps(data).encode('utf-8'))
            response = client_socket.recv(1048576).decode('utf-8')
        except Exception as e:
            print(e)
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect((HOST, PORT))
            continue
        response_data = decode_json(response)
        count += 1
        if count > 25:
            response_data = {
                'history': [['', 'Failed to get answer, please try later.']],
                'source_documents': []
            }
            break
    # 赋值并返回
    answer = response_data['history'][0][1]
    source_documents = response_data['source_documents']
    return answer, source_documents
```

# 因传输回来的数据中文会被转义为类似\\u6853的格式, 所以需要解码

# BUG: 传输json中的‘response’的值为中文，传输回来的编码与‘history’项中第二个元素完全一致，但是经过解码后变为乱码

```
def decode_json(json_str):
    # 将json字符串转换为字典
    try:
        data = json.loads(json_str)
    except Exception as e:
        data = None
        return data
    # 遍历字典，将字符串解码
    for key, value in data.items():
        if isinstance(value, str):
            value = value.encode('utf-8').decode('unicode_escape')
            data[key] = value
    return data

# 递归删除布局中的组件
def deleteItemOfLayout(layout):
    if layout is not None:
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            # 如果是组件，将其从布局中删除
            if widget is not None:
                widget.setParent(None)
            else:
                deleteItemOfLayout(item.layout())
```

#### my\_interactor\_style.py

```
# 导入系统模块
import re
import os

# 导入vtk
import vtk

# 常量定义
PDF_DIR = "source files" # 40881相关的所有pdf图纸所在的文件夹
PDF_EXTENSION1 = ".pdf" # pdf后缀
PDF_EXTENSION2 = ".PDF" # pdf后缀

class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    """
    自定义的VTK交互器，继承自vtkInteractorStyleTrackballCamera类。
    该交互器主要用于处理用户与3D模型交互的事件，包括鼠标左键按下事件。

    Attributes:
```

render: vtkRenderer对象, 用于渲染3D模型。  
 model\_actor: vtkActor对象的列表, 表示模型的可视化对象。  
 main\_window: 主窗口对象的引用。  
 picker: vtkCellPicker对象, 用于捕捉鼠标选中的模型对象。  
 LastPickedActor: 上一次选中的vtkActor对象。  
 LastPickedProperty: 上一次选中的vtkProperty对象。  
 pdf\_file: 当前选中模型关联的PDF文件名。

#### Methods:

\_\_init\_\_: 初始化方法, 设置渲染器、模型可视化对象、主窗口, 并添加鼠标左键按下事件  
的观察器。

on\_left\_button\_press: 鼠标左键按下事件处理方法, 处理模型选中逻辑。

handle\_selection: 处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM  
信息和PDF文件。

get\_bom\_info: 根据给定的零件号从BOM表中获取相关信息。

find\_pdf\_file\_by\_vendor\_part\_number: 根据供应商零件号查找关联的PDF文件。

reset\_state: 重置交互器的状态, 取消选中模型并清除关联的PDF文件。

"""

```
def __init__(self, render, model_actor, main_window):
```

"""

初始化方法。

#### Args:

render: vtkRenderer对象, 用于渲染3D模型。

model\_actor: vtkActor对象的列表, 表示模型的可视化对象。

main\_window: 主窗口对象的引用。

"""

```
self.render = render
```

```
self.model_actor = model_actor
```

```
self.main_window = main_window
```

```
self.picker = vtk.vtkCellPicker()
```

# 下面几个变量, 在初始化时没有特定的初始值, 因此不需要作为参数传入

# 它会在交互过程中根据用户的选择进行更新, 不会被其他类访问

```
self.LastPickedActor = None # 这个变量用于记录上一次选中的 actor
```

# 这个变量是一个 vtkProperty 对象, 用于记录上一次选中的 actor 的属性

```
self.LastPickedProperty = vtk.vtkProperty()
```

```
self.pdf_file = None # 这个变量用于记录选中的 actor 对应的 PDF 文件路径
```

```
self.AddObserver("LeftButtonPressEvent",
```

```
self.on_left_button_press) # 添加鼠标左键按下事件观察器
```

# 鼠标左键按下事件

```
def on_left_button_press(self, obj, event):
```

"""

鼠标左键按下事件。

#### Args:

obj: 事件对象。

event: 事件类型。

"""

# 将主窗口中的is\_middle\_button\_down更改为false



```

self.main_window.is_middle_button_down = False
if self.main_window.select_type:
    # 获取鼠标点击位置
    click_pos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(click_pos[0], click_pos[1], 0, self.render)
    self.picked_actor = self.picker.GetActor()

    if self.picked_actor:
        # 清空文本编辑框
        self.main_window.text_edit.clear()
        filename = self.main_window.actor_to_filename[self.picked_actor]

        # 检查是不是有已经选中的模型并分情况处理
        if self.LastPickedActor:
            if self.LastPickedActor == self.picked_actor:
                # 若选中相同的模型, 重置状态
                self.reset_state()
            else:
                # 处理选中的模型
                self.handle_selection(filename)
        else:
            # 处理选中的模型
            self.handle_selection(filename)

        # 检查处理之后是否还有选中的模型并且根据结果更新按钮状态
        if self.picked_actor or self.LastPickedActor:
            # 激活show current selected model按钮
            if self.main_window.is_hide != True:

self.main_window.show_current_selected_model_button.setEnabled(
    True)
            if self.picked_actor:
                self.main_window.hide_model_button.setEnabled(True)
            else:
                # 取消激活show current selected model按钮
                self.main_window.show_current_selected_model_button.setEnabled(
                    False)
                self.main_window.hide_model_button.setEnabled(False)

            # 根据是否关联PDF文件, 设置显示PDF按钮的状态
            if self.pdf_file is None:
                self.main_window.show_pdf_button.setEnabled(False)
            else:
                self.main_window.show_pdf_button.setEnabled(True)

self.OnLeftButtonDown()

def handle_selection(self, filename):
    """
    处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

    Args:
        filename: 模型文件名。
    """

```

```

# 从文件名中提取Part number
self.pdf_file = None
self.main_window.previous_text = ""
match = re.findall(r'\d{5}', filename)
if match == []:
    self.main_window.print_to_gui(f'{filename}')
else:
    part_number = match[-1]
    file_number = match[0]
    # 根据part number分为不同的情况处理
    if file_number == '19848' or file_number == '42755':
        # 对于19848和42755, 直接显示文件名, 没有BOM信息和供应商零件号
        self.main_window.print_to_gui(f'{filename}')
    else:
        self.main_window.print_to_gui(f'{filename}')
        # 对于40881和40650, 显示part number, 获取BOM信息和供应商零件号
        if file_number == '40881' or file_number == '40650':
            self.main_window.print_to_gui(
                f'Part number: {part_number}\n')
            if part_number != '40881' or part_number != '40650':
                # 若part number不是'40881', 获取BOM信息和供应商零件号
                bom_info, vendor_part_number = self.get_bom_info(
                    part_number)
                self.main_window.print_to_gui(bom_info)
                self.pdf_file = self.find_pdf_file_by_vendor_part_number(
                    vendor_part_number)
                if self.pdf_file is not None:
                    temp_path = self.pdf_file.replace("/", "\\")
                    self.main_window.print_to_gui(
                        f'PDF file: {temp_path}')
            else:
                # 对于39000和41565, 因为文件存储的结构不同, 需要分情况处理
                if file_number == '39000' or file_number == '41565':
                    # 39000中可能存储多个pdf文件显示信息需要单独处理
                    if file_number == '39000' and part_number != '39000':
                        self.main_window.print_to_gui(
                            f'Part number: {part_number}\n')
                        self.pdf_file = self.find_pdf_file_by_part_number(
                            file_number, part_number)
                        if self.pdf_file is not None:
                            temp_path = self.pdf_file.replace("/", "\\")
                            self.main_window.print_to_gui(
                                f'PDF file: {temp_path}\n\n It is possible
that there are multiple PDF files for this part number. Please check the PDF
folder.')
                    # 41565中存储方式为多层子文件夹, 需要逐层查找, 并且记录最后一层
                    的part number

                    elif file_number == '41565' and part_number != '41565':
                        index = 1
                        dir = PDF_DIR + '/' + file_number
                        have_next_level = False
                        last_part_number = part_number
                        # 逐层查找
                        while index < len(match):

```

```

        source_files_dir = os.path.join(
            os.path.dirname(os.path.abspath(__file__)),
            dir)

        # 适配exe文件，当使用代码调试时，需要注释掉下面这行代
        source_files_dir = os.path.join(os.path.dirname(
            os.path.dirname(os.path.abspath(__file__)))), dir)

        # 查找下一层的part number与更新路径
        part_number = match[index]
        for file_name in os.listdir(source_files_dir):
            if file_name == part_number:
                dir = dir + '/' + file_name
                have_next_level = True
                break
        # 如果还存在下一层，更新index和last_part_number
        if have_next_level:
            index += 1
            last_part_number = part_number

        # 如果不存在下一层或者文件名中以及达到最后一个part
        number，查找目录下的第一个PDF文件
        if not have_next_level or index == len(match):
            if index == len(match):
                source_files_dir = os.path.join(
                    os.path.dirname(os.path.abspath(__file__)), dir)
                # 适配exe文件，当使用代码调试时，需要注释掉下
                source_files_dir =
                    os.path.join(os.path.dirname(
                        os.path.dirname(os.path.abspath(__file__)))), dir)
                self.main_window.print_to_gui(
                    f'Part number: {last_part_number}\n')
                # 查找目录下的第一个PDF文件
                for file_name in os.listdir(source_files_dir):
                    if file_name.endswith(PDF_EXTENSION1) or
                    file_name.endswith(PDF_EXTENSION2):
                        self.pdf_file = os.path.join(
                            source_files_dir, file_name)
                        temp_path = self.pdf_file.replace(
                            "/", "\\")
                        self.main_window.print_to_gui(
                            f'PDF file: {temp_path}\n\n It is
possible that there are multiple PDF files for this part number. Please check the
PDF folder.')
                        break
                    break
                have_next_level = False

        for actor in self.model_actor:

```

```

        if actor != self.picked_actor:
            # 设置非选中模型的颜色和透明度
            actor.GetProperty().SetColor(1, 1, 1)
            actor.GetProperty().SetOpacity(0.1)

# 设置选中模型的颜色和透明度
self.picked_actor.GetProperty().SetColor(1, 0, 0)
self.picked_actor.GetProperty().SetOpacity(1)
self.LastPickedActor = self.picked_actor

def get_bom_info(self, part_number):
    """
    根据part number从BOM表中获取相关信息。

    Args:
        part_number: 零件号。

    Returns:
        tuple: 包含BOM信息和供应商零件号的元组。
    """
    bom_info = ""
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part Number']
            for key, value in item.items():
                # 将BOM信息格式化为字符串, 用于输出
                bom_info += f"{key}: {value}\n"
            break
    return bom_info, vendor_part_number

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    """
    根据vendor part number查找关联的PDF文件。

    Args:
        vendor_part_number: 供应商零件号。

    Returns:
        str: PDF文件的完整路径, 如果未找到匹配的文件则返回None。
    """
    source_files_dir = os.path.join(
        os.path.dirname(os.path.abspath(__file__)), PDF_DIR)

# 适配exe文件, 当使用代码调试时, 需要注释掉下面这行代码
source_files_dir = os.path.join(os.path.dirname(
    os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), PDF_DIR)
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(PDF_EXTENSION1) or
file_name.endswith(PDF_EXTENSION2):
            if vendor_part_number != "" and vendor_part_number in file_name:
                # 找到匹配的文件, 返回完整路径
                return os.path.join(source_files_dir, file_name)

```

```

# 未找到匹配的文件, 返回None
return None

def find_pdf_file_by_part_number(self, file_number, part_number):
    dir = PDF_DIR + '/' + file_number
    source_files_dir = os.path.join(
        os.path.dirname(os.path.abspath(__file__)), dir)

    # 适配exe文件, 当使用代码调试时, 需要注释掉下面这行代码
    source_files_dir = os.path.join(os.path.dirname(
        os.path.dirname(os.path.abspath(__file__)))), dir)

    for file_name in os.listdir(source_files_dir):
        if file_name == part_number:
            for inner_file_name in os.listdir(source_files_dir + '/' +
file_name):
                if inner_file_name.endswith(PDF_EXTENSION1) or
inner_file_name.endswith(PDF_EXTENSION2):
                    return os.path.join(source_files_dir + '/' + file_name,
inner_file_name)
            return None

def reset_state(self):
    """
    重置交互器的状态, 取消选中模型并清除关联的PDF文件。
    """
    self.picked_actor = None
    self.LastPickedActor = None
    self.pdf_file = None

    for actor in self.model_actor:
        # 重置模型的颜色和透明度为默认值
        actor.GetProperty().SetColor(
            self.main_window.actor_to_color[actor])
        actor.GetProperty().SetOpacity(1)

```

## app.py

```

import sys
from main_window import MainWindow, QtWidgets

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## setup.py

```
from cx_Freeze import setup, Executable

target_script = 'app.py'

build_options = {
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're',
'chardet', 'subprocess', 'threading', 'socket'],
    'include_files': ['FDK/', '40881_40650_json.json', 'my_interactor_style.py',
'main_window.py', 'source files/']
}

executable = Executable(
    script=target_script,
    base='Win32GUI'
)

setup(
    name='App',
    version='1.0',
    description='FlightDeck 3D模型展示',
    options={'build_exe': build_options},
    executables=[executable]
)
```

## 7.18

## main\_window.py

```
# 导入系统模块
import subprocess
import threading
import chardet
import json
import os
import re
import socket

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

class MainWindow(QtWidgets.QMainWindow):
```

```
"""
```

主窗口类，用于显示3D模型和提供相关功能。

Attributes:

actor\_to\_filename: dict, 保存每个actor和对应的STL文件名的映射。  
 actor\_to\_color: dict, 保存每个actor和对应的颜色的映射。  
 bom\_data: list, BOM表数据。  
 render: vtkRenderer对象, 用于渲染3D模型。  
 actors: list, 所有的vtkActor对象列表。  
 is\_hide: bool, 是否隐藏模型。  
 is\_middle\_button\_pressed: bool, 鼠标中键是否按下。

Methods:

\_\_init\_\_: 初始化方法, 创建主窗口, 读取STL文件, 设置布局和功能按钮。  
 create\_layout: 创建布局。  
 load\_stl\_files: 读取STL文件。  
 create\_buttons: 创建功能按钮。  
 print\_to\_gui: 将文本显示到GUI中。  
 reset: 重置模型和交互器状态。  
 show\_pdf\_helper: 辅助函数, 打开关联的PDF文件。  
 show\_pdf: 显示关联的PDF文件。  
 show\_current\_selected\_model: 显示当前选中的模型的父模型。  
 middle\_button\_press: 模拟鼠标中键按下事件。

```
"""
```

# 常量定义

STL\_DIR = '40881\_40650\_unlock' # .stl文件夹路径  
 BOM\_FILE = '40881\_40650\_json.json' # bom表, 转成json格式, 让程序更好读取

```
def __init__(self):
```

```
    """
```

初始化方法, 创建主窗口, 读取STL文件, 设置布局和功能按钮。

```
    """
```

# 创建一个新的字典self.actor\_to\_filename来存储每个actor和它对应的STL文件名的映

射:

```
self.actor_to_filename = {}
# 创建一个新的字典储存actor的颜色
self.actor_to_color = {}
# 调用父类的初始化方法
super().__init__()
```

# 获取当前路径

```
current_dir = os.path.dirname(os.path.abspath(__file__))
# 适配exe文件, 当使用代码调试时, 需要注释掉下面两行
current_dir = os.path.dirname(current_dir)
current_dir = os.path.dirname(current_dir)
# 获取STL文件夹路径
stl_dir = os.path.join(current_dir, self.STL_DIR)
```

# 获取bom表编码信息

```
with open(self.BOM_FILE, 'rb') as file:
    raw_data = file.read()
    bom_encoding = chardet.detect(raw_data)['encoding']
```

```
# 获取模型对应BOM表信息
with open(self.BOM_FILE, 'r', encoding=bom_encoding) as file:
    self.bom_data = json.load(file)

# 设置窗口标题和大小
self.setWindowTitle('FlightDeck 3D模型展示 (AA,RS部分) ')
# 设置窗口的固定大小
self.resize(3840, 2160)

# 创建布局
self.current_content_layout, self.other_function_layout =
self.create_layouts()

# 创建文本框
self.create_text_box()

# 创建 QVTKRenderWindowInteractor 实例
self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

# 储存所有actor
self.actors = []

# 调用load_stl_files方法加载STL文件
self.load_stl_files(stl_dir)

# 初始化交互器
self.interactorStyle = MyInteractorStyle(
    self.render, self.actors, self)
self.vtkWidget.GetRenderWindow().GetInteractor(
).SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout (主布局) 中
self.current_content_layout.addWidget(self.vtkWidget)

# 调用 create_buttons 方法创建按钮
self.create_buttons(self.current_content_layout)
self.right_button_layout = self.create_inform_button()
self.other_function_layout.addLayout(self.right_button_layout)

# 设置变量记录是否隐藏模型
self.is_hide = False

# 记录当前鼠标中键点击状态
self.is_middle_button_down = False

# 记录选择样式
self.select_type = False

# 记录右侧界面状态
```



```
self.is_chat = False

# 记录右侧初始文本框中的内容（用于切换后重置）
self.previous_text = ""

# 记录模型回答
self.answer = ""
self.source = []

def create_chat_part(self):
    # 创建label
    self.label1 = QtWidgets.QLabel("Response")
    self.label2 = QtWidgets.QLabel("Question")

    # 创建文本框
    self.response_box = QtWidgets.QTextEdit()
    self.question_box = QtWidgets.QTextEdit()
    self.response_box.setReadOnly(True)

def create_text_box(self):
    # 创建 QTextEdit 实例
    self.text_edit = QTextEdit(self)
    self.text_edit.setReadOnly(True)
    self.other_function_layout.addWidget(self.text_edit)

def create_layouts(self):
    """
    创建主布局、当前内容布局和其他功能布局，并设置布局关系和占据比例。
    """
    # 创建主布局
    main_layout = QtWidgets.QHBoxLayout()
    main_widget = QtWidgets.QWidget(self)
    main_widget.setLayout(main_layout)
    self.setCentralWidget(main_widget)

    # 创建当前内容布局
    current_content_widget = QtWidgets.QWidget(self)
    current_content_layout = QtWidgets.QVBoxLayout()
    current_content_widget.setLayout(current_content_layout)
    main_layout.addWidget(current_content_widget, 3) # 可以按照要求调整比例

    # 创建其他功能布局
    other_function_widget = QtWidgets.QWidget(self)
    other_function_layout = QtWidgets.QVBoxLayout()
    other_function_widget.setLayout(other_function_layout)
    main_layout.addWidget(other_function_widget, 1)

    return current_content_layout, other_function_layout

def load_stl_files(self, stl_dir):
    """
    加载STL文件，并将模型添加到渲染器和本地的actor列表中。
    """
```

```

Args:
    stl_dir: STL文件夹路径。
"""
for file_name in os.listdir(stl_dir):
    if file_name.lower().endswith('.stl'):
        file_path = os.path.join(stl_dir, file_name)
        print(file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到actor_to_filename字典

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        if '40881' in file_name:
            actor.GetProperty().SetColor(1, 0.5, 0)
        else:
            actor.GetProperty().SetColor(0, 1, 1)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name
        self.actor_to_color[actor] = actor.GetProperty().GetColor()

def create_chat_button(self):
    """
    创建与模型交互模式的按钮并且返回布局。
    """
    text_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
    # 创建submit按钮
    self.submit_button = QPushButton('Submit', self)
    self.submit_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    self.submit_button.clicked.connect(self.submit)
    text_button_layout.addWidget(self.submit_button)

    # 创建update按钮
    self.update_button = QPushButton('Update', self)
    self.update_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    self.update_button.clicked.connect(self.updateResponse)
    text_button_layout.addWidget(self.update_button)
    self.update_button.setEnabled(False)

    # 创建switch按钮

```

典

```

self.switch_button = QPushButton('Switch', self)
self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.switch_button.clicked.connect(self.switch)
text_button_layout.addWidget(self.switch_button)

```

```

return text_button_layout

```

```

def create_inform_button(self):

```

```

    """

```

```

    创建点击获取信息模式的按钮并且返回布局。

```

```

    """

```

```

    inform_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局

```

```

    # 创建 Show PDF 按钮

```

```

    self.show_pdf_button = QPushButton('Show PDF', self)

```

```

    self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度

```

```

    inform_button_layout.addWidget(self.show_pdf_button)

```

```

    self.show_pdf_button.clicked.connect(self.show_pdf)

```

```

    if self.interactorStyle.pdf_file is None:

```

```

        self.show_pdf_button.setEnabled(False)

```

```

    else:

```

```

        self.show_pdf_button.setEnabled(True)

```

```

    # 创建switch按钮

```

```

    self.switch_button = QPushButton('Switch', self)

```

```

    self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度

```

```

    inform_button_layout.addWidget(self.switch_button)

```

```

    self.switch_button.clicked.connect(self.switch)

```

```

    return inform_button_layout

```

```

def create_buttons(self, current_content_layout):

```

```

    """

```

```

    创建"Reset"按钮,"Show Model"按钮,"Move"按钮和"Select"按钮并添加到布局中。

```

```

    """

```

```

    self.button_layout = QtWidgets.QHBoxLayout() # 创建水平布局

```

```

    current_content_layout.addLayout(self.button_layout) # 将水平布局添加到当前内容布局中

```

前内容布局中

```

    # 创建 Reset 按钮

```

```

    self.reset_button = QPushButton('Reset', self)

```

```

    self.reset_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度

```

```

    self.reset_button.clicked.connect(self.reset)

```

```

    self.button_layout.addWidget(self.reset_button) # 将 Reset 按钮添加到水平

```

布局中

```

    self.button_layout.addSpacing(20) # 添加间距

```

```

    # 创建 show current selected model 按钮

```

```

    self.show_current_selected_model_button = QPushButton(
        'Show Model', self)

```

```

    self.show_current_selected_model_button.setFixedSize(
        80, 40) # 设置按钮的固定宽度和高度

```

```

    self.show_current_selected_model_button.clicked.connect(
        self.show_current_selected_model)

```

```

    self.button_layout.addWidget(self.show_current_selected_model_button)

```

和高度

```

self.show_current_selected_model_button.setEnabled(False)
self.button_layout.addSpacing(20) # 添加间距

# 创建按钮模拟鼠标中键点击事件
self.middle_button_click_button = QPushButton('Move', self)
self.middle_button_click_button.setFixedSize(80, 40) # 设置按钮的固定宽度

self.middle_button_click_button.clicked.connect(
    self.middle_button_click)
self.button_layout.addWidget(self.middle_button_click_button)
self.button_layout.addSpacing(20) # 添加间距

# 创建按钮切换选择模式
self.selection_type_button = QPushButton('Select', self)
self.selection_type_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.selection_type_button.clicked.connect(self.selection_type)
self.button_layout.addWidget(self.selection_type_button)

def print_to_gui(self, text):
    """
    将文本显示到GUI中。

    Args:
        text: 要显示的文本。
    """
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

def reset(self):
    """
    重置模型和交互器状态。
    """
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = False
    self.previous_text = ""
    if self.is_chat == False:
        self.text_edit.clear()
    else:
        if self.submit_button.isEnabled() == True:
            self.response_box.clear()
            self.question_box.clear()
    self.interactorStyle.pdf_file = None
    for actor in self.actors:
        # if '40881' in self.actor_to_filename[actor]:
        #     actor.GetProperty().SetColor(1, 0.5, 0)
        # else:
        #     actor.GetProperty().SetColor(0, 1, 1)
        actor.GetProperty().SetColor(self.actor_to_color[actor])
        actor.GetProperty().SetOpacity(1)
        actor.SetPickable(True)
        actor.SetVisibility(True)
    self.render.ResetCamera()
    self.vtkWidget.update()

```

```
def show_pdf_helper(self):
    """
    辅助函数，打开关联的PDF文件。
    """
    try:
        subprocess.run(
            ['C:\Program Files\Wondershare\PDFelement\PDFelement.exe',
self.interactorStyle.pdf_file])
    except Exception as e:
        print(e)
        self.print_to_gui(
            '打开PDF文件失败，请检查PDFelement是否安装，并且检查路径是否
为:C:\Program Files\Wondershare\PDFelement\PDFelement.exe')

def show_pdf(self):
    """
    显示关联的PDF文件。
    """
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

def switch(self):
    """
    切换与模型交互和点击获取信息之间的模式。
    """
    if self.is_chat == False:
        self.is_chat = True
        self.create_chat_part()
        # 如果之前有文本，将其记录
        if self.text_edit.toPlainText() != "":
            self.previous_text = self.text_edit.toPlainText()
        # 删除之前的组件，避免因内存导致的重叠
        # 由于删除组件时会导致布局中的组件顺序发生变化，所以需要从后往前删除
        item_list = list(range(self.other_function_layout.count()))
        item_list.reverse()
        for i in item_list:
            item = self.other_function_layout.takeAt(i)
            self.other_function_layout.removeItem(item)
            # 如果是组件，将其从布局中删除
            if item.widget():
                item.widget().setParent(None)
            else:
                # 如果是布局，递归删除布局中的组件
                deleteItemOfLayout(item.layout())

        # 添加组件
        self.other_function_layout.addWidget(self.label1)
        self.other_function_layout.addWidget(self.response_box)
        self.other_function_layout.addWidget(self.label2)
        self.other_function_layout.addWidget(self.question_box)
        # self.other_function_layout.addLayout(self.create_chat_button())
        self.right_button_layout = self.create_chat_button()
        self.other_function_layout.addLayout(self.right_button_layout)
```

```

else:
    self.is_chat = False
    item_list = list(range(self.other_function_layout.count()))
    item_list.reverse()
    for i in item_list:
        item = self.other_function_layout.takeAt(i)
        self.other_function_layout.removeItem(item)
        if item.widget():
            item.widget().setParent(None)
        else:
            deleteItemOfLayout(item.layout())

    self.create_text_box()
    # self.other_function_layout.addLayout(self.create_inform_button())
    self.right_button_layout = self.create_inform_button()
    self.other_function_layout.addLayout(self.right_button_layout)
    # 如果之前有文本, 将其显示
    if self.previous_text != "":
        self.text_edit.append(self.previous_text)

def submit(self):
    """
    获取模型回答, 创建线程以避免阻塞主线程。使得模型展示界面可以在获取答案的同时进行
    交互。
    """
    submit_thread = threading.Thread(target=self.submitHelper)
    submit_thread.start()

def submitHelper(self):
    """
    辅助函数, 用于在子线程中获取模型回答。
    """
    # 如果问题为空, 直接返回
    if self.question_box.toPlainText() == "":
        return
    # 将submit按钮设置为不可用, 避免重复点击
    # 将switch按钮设置为不可用, 避免点击导致意外错误
    self.submit_button.setEnabled(False)
    self.switch_button.setEnabled(False)
    # 创建json数据用以发送给后端。根据需求更改knowledge_base_id, 对于user1的模型,
    test all files是所有文件的知识库。
    # 在模型方可以根据需求细分数据库, 如需修改可能需要根据具体需求对
    knowledge_base_id进行赋值方式的更改。
    request_data = {
        "knowledge_base_id": "test all files",
        "question": self.question_box.toPlainText(),
        "history": []
    }
    # 调用send_json_data函数获取模型回答
    self.answer, self.source = send_json_data(request_data)
    # 将update按钮设置为可用, 以便用户更新模型回答 (该功能为当前防止主线程被阻塞的必要功能, 如果直接进行自动更新则会出现以下两种情况:

```

```

# 1. 直接使用threading.Thread创建的线程会报错: cannot create children for a
parent that is in a different thread
# 2. 如果尝试使用自定义线程类: 会出现QThread:Destroyed while thread is still
running
# 目前来说, 并未找到方法解决这两个情况的方法, 暂时使用让用户手动更新的方式以避免
主线程被获取答案的函数阻塞)
self.update_button.setEnabled(True)

def updateResponse(self):
    self.response_box.append("Response: \n" + self.answer + "\n")
    # 由于source_documents是一个列表, 需要将其转换为字符串
    source = ' '.join(map(str,self.source))
    self.response_box.append("Source documents: \n" + source)
    self.question_box.clear()
    self.update_button.setEnabled(False)
    self.submit_button.setEnabled(True)
    self.switch_button.setEnabled(True)
    self.answer = ""
    self.source = []

# def submit(self):
#     """
#     获取模型回答
#     """
#     if self.question_box.toPlainText() == "":
#         return
#     self.response_box.clear()
#     # 创建json数据用以发送给后端。根据需求更改knowledge_base_id,对于user1的模
#     型, test all files是所有文件的知识库。
#     # 在模型方可以根据需求细分数据库, 如需修改可能需要根据具体需求对
#     knowledge_base_id进行赋值方式的更改。
#     request_data = {
#         "knowledge_base_id": "test all files",
#         "question": self.question_box.toPlainText(),
#         "history": []
#     }
#     answer, source_documents = send_json_data(request_data)
#     self.response_box.append("Response: \n" + answer + "\n")
#     # 由于source_documents是一个列表, 需要将其转换为字符串
#     source = ' '.join(map(str,source_documents))
#     self.response_box.append("Source documents: \n" + source)
#     self.question_box.clear()

def show_current_selected_model(self):
    """
    显示当前选中的模型所属的上级模型。
    """
    file_name = ""
    if self.interactorStyle.picked_actor:
        file_name = self.actor_to_filename[self.interactorStyle.picked_actor]
    elif self.interactorStyle.LastPickedActor:
        file_name =

```

```
self.actor_to_filename[self.interactorStyle.LastPickedActor]
    match = re.findall(r'\d{5}', file_name)
    part_number = match[0]
    for actor in self.actors:
        if part_number not in self.actor_to_filename[actor]:
            actor.SetVisibility(False)
            actor.SetPickable(False)
    self.vtkWidget.update()
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = True

def middle_button_click(self):
    """
    模拟鼠标中键点击事件。
    """
    self.is_middle_button_down = not self.is_middle_button_down

    if self.is_middle_button_down:
        self.interactorStyle.OnMiddleButtonDown()
    else:
        self.interactorStyle.OnMiddleButtonUp()

def selection_type(self):
    """
    切换选择模式。
    """
    self.select_type = not self.select_type

# 链接模型获得答案的函数
def send_json_data(data):
    HOST = '10.1.4.12'
    PORT = 8082

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((HOST, PORT))
    except Exception as e:
        index = 0
        print(e)
        while index < 5:
            try:
                client_socket.connect((HOST, PORT))
                break
            except Exception as e:
                print(e)
                index += 1
            if index == 5:
                return 'Failed to get answer, please check status of GLM
server.', []
            continue

    count = 0
```



```

response_data = None
while response_data is None:
    try:
        client_socket.sendall(json.dumps(data).encode('utf-8'))
        response = client_socket.recv(12288).decode('utf-8')
    except Exception as e:
        print(e)
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((HOST, PORT))
        continue
    response_data = decode_json(response)
    count += 1
    if count > 5:
        response_data['history'][0][1] = 'Failed to get answer, please try
later.'
        response_data['source_documents'] = []
        break
# print(response_data)
answer = response_data['history'][0][1]
source_documents = response_data['source_documents']
# print(answer)
# print(source_documents)
return answer, source_documents

# 因传输回来的数据中文会被转义为类似\\u6853的格式，所以需要解码
# BUG: 传输json中的‘response’的值为中文，传输回来的编码与‘history’项中第二个元素完全一
致，但是经过解码后变为乱码
def decode_json(json_str):
    try:
        data = json.loads(json_str)
    except Exception as e:
        data = None
        return data
    for key, value in data.items():
        if isinstance(value, str):
            value = value.encode('utf-8').decode('unicode_escape')
            data[key] = value
    return data

# 递归删除布局中的组件
def deleteItemOfLayout(layout):
    if layout is not None:
        while layout.count():
            item = layout.takeAt(0)
            widget = item.widget()
            # 如果是组件，将其从布局中删除
            if widget is not None:
                widget.setParent(None)
            else:
                deleteItemOfLayout(item.layout())

```

```

# 导入系统模块
import re
import os

# 导入vtk
import vtk

# 常量定义
PART_NUMBER_LENGTH = 5 # 我们的bom表和.stl文件的part number长度都是5
BOM_FILE = "40881 json.json" # bom表, 转成json格式, 让程序更好读取
PDF_DIR = "source files" # 40881相关的所有pdf图纸所在的文件夹
PDF_EXTENSION1 = ".pdf" # pdf后缀
PDF_EXTENSION2 = ".PDF" # pdf后缀

class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    """
    自定义的VTK交互器, 继承自vtkInteractorStyleTrackballCamera类。
    该交互器主要用于处理用户与3D模型交互的事件, 包括鼠标左键按下事件。

    Attributes:
        render: vtkRenderer对象, 用于渲染3D模型。
        model_actor: vtkActor对象的列表, 表示模型的可视化对象。
        main_window: 主窗口对象的引用。
        picker: vtkCellPicker对象, 用于捕捉鼠标选中的模型对象。
        LastPickedActor: 上一次选中的vtkActor对象。
        LastPickedProperty: 上一次选中的vtkProperty对象。
        pdf_file: 当前选中模型关联的PDF文件名。

    Methods:
        __init__: 初始化方法, 设置渲染器、模型可视化对象、主窗口, 并添加鼠标左键按下事件的观察器。
        on_left_button_press: 鼠标左键按下事件处理方法, 处理模型选中逻辑。
        handle_selection: 处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。
        get_bom_info: 根据给定的零件号从BOM表中获取相关信息。
        find_pdf_file_by_vendor_part_number: 根据供应商零件号查找关联的PDF文件。
        reset_state: 重置交互器的状态, 取消选中模型并清除关联的PDF文件。
    """
    def __init__(self, render, model_actor, main_window):
        """
        初始化方法。

        Args:
            render: vtkRenderer对象, 用于渲染3D模型。
            model_actor: vtkActor对象的列表, 表示模型的可视化对象。
            main_window: 主窗口对象的引用。
        """
        self.render = render
        self.model_actor = model_actor
        self.main_window = main_window
        self.picker = vtk.vtkCellPicker()

```

```
# 下面几个变量，在初始化时没有特定的初始值，因此不需要作为参数传入
# 它会在交互过程中根据用户的选择进行更新，不会被其他类访问

self.LastPickedActor = None # 这个变量用于记录上一次选中的 actor
self.LastPickedProperty = vtk.vtkProperty() # 这个变量是一个 vtkProperty 对象，用于记录上一次选中的 actor 的属性
self.pdf_file = None # 这个变量用于记录选中的 actor 对应的 PDF 文件路径

self.AddObserver("LeftButtonPressEvent", self.on_left_button_press) # 添加鼠标左键按下事件观察器

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    """
    鼠标左键按下事件。

    Args:
        obj: 事件对象。
        event: 事件类型。
    """
    # 将主窗口中的is_middle_button_down更改为false
    self.main_window.is_middle_button_down = False
    if self.main_window.select_type:
        # 获取鼠标点击位置
        click_pos = self.GetInteractor().GetEventPosition()
        self.picker.Pick(click_pos[0], click_pos[1], 0, self.render)
        self.picked_actor = self.picker.GetActor()

    if self.picked_actor:
        # 清空文本编辑框
        self.main_window.text_edit.clear()
        filename = self.main_window.actor_to_filename[self.picked_actor]

        # 检查是不是有已经选中的模型并分情况处理
        if self.LastPickedActor:
            if self.LastPickedActor == self.picked_actor:
                # 若选中相同的模型，重置状态
                self.reset_state()
            else:
                # 处理选中的模型
                self.handle_selection(filename)
        else:
            # 处理选中的模型
            self.handle_selection(filename)

        # 检查处理之后是否还有选中的模型并且根据结果更新按钮状态
        if self.picked_actor or self.LastPickedActor:
            # 激活show current selected model按钮
            if self.main_window.is_hide != True:
                self.main_window.show_current_selected_model_button.setEnabled(True)
            else:
                # 取消激活show current selected model按钮
```

```
self.main_window.show_current_selected_model_button.setEnabled(False)

# 根据是否关联PDF文件, 设置显示PDF按钮的状态
if self.pdf_file is None:
    self.main_window.show_pdf_button.setEnabled(False)
else:
    self.main_window.show_pdf_button.setEnabled(True)

self.OnLeftButtonDown()

def handle_selection(self, filename):
    """
    处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

    Args:
        filename: 模型文件名。
    """
    # 从文件名中提取Part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'{filename}')
    self.main_window.print_to_gui(f'Part number: {part_number}')

    if part_number != '40881' or part_number != '40650':
        # 若part number不是'40881', 获取BOM信息和供应商零件号
        bom_info, vendor_part_number = self.get_bom_info(part_number)
        self.main_window.print_to_gui(bom_info)
        self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if self.pdf_file is not None:
            self.main_window.print_to_gui(f'PDF file: {self.pdf_file}')

    for actor in self.model_actor:
        if actor != self.picked_actor:
            # 设置非选中模型的颜色和透明度
            actor.GetProperty().SetColor(1, 1, 1)
            actor.GetProperty().SetOpacity(0.1)

    # 设置选中模型的颜色和透明度
    self.picked_actor.GetProperty().SetColor(1, 0, 0)
    self.picked_actor.GetProperty().SetOpacity(1)
    self.LastPickedActor = self.picked_actor

def get_bom_info(self, part_number):
    """
    根据part number从BOM表中获取相关信息。

    Args:
        part_number: 零件号。

    Returns:
```

```

        tuple: 包含BOM信息和供应商零件号的元组。
    """
    bom_info = ""
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part Number']
            for key, value in item.items():
                # 将BOM信息格式化为字符串, 用于输出
                bom_info += f"{key}: {value}\n"
            break
    return bom_info, vendor_part_number

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    """
    根据vendor part number查找关联的PDF文件。

    Args:
        vendor_part_number: 供应商零件号。

    Returns:
        str: PDF文件的完整路径, 如果未找到匹配的文件则返回None。
    """
    source_files_dir =
os.path.join(os.path.dirname(os.path.abspath(__file__)), PDF_DIR)

    # 适配exe文件, 当使用代码调试时, 需要注释掉下面这行代码
    source_files_dir =
os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), PDF_DIR)
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(PDF_EXTENSION1) or
file_name.endswith(PDF_EXTENSION2):
            if vendor_part_number != "" and vendor_part_number in file_name:
                # 找到匹配的文件, 返回完整路径
                return os.path.join(source_files_dir, file_name)

    # 未找到匹配的文件, 返回None
    return None

def reset_state(self):
    """
    重置交互器的状态, 取消选中模型并清除关联的PDF文件。
    """
    self.picked_actor = None
    self.LastPickedActor = None
    self.pdf_file = None

    for actor in self.model_actor:
        # 重置模型的颜色和透明度为默认值
        if '40881' in self.main_window.actor_to_filename[actor]:
            actor.GetProperty().SetColor(1, 0.5, 0)

```

```
else:
    actor.GetProperty().SetColor(0, 1, 1)
    actor.GetProperty().SetOpacity(1)
```

#### app.py

```
import sys
from main_window import MainWindow, QtWidgets

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

#### setup.py

```
import sys
from cx_Freeze import setup, Executable

target_script = 'app.py'

build_options = {
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're',
    'chardet', 'subprocess', 'threading', 'socket'],
    'include_files': ['40881_40650_unlock/', '40881_40650_json.json',
    'my_interactor_style.py', 'main_window.py', 'source files/']
}

executable = Executable(
    script=target_script,
    base='Win32GUI'
)

setup(
    name='App',
    version='1.0',
    description='testPYQT',
    options={'build_exe': build_options},
    executables=[executable]
)
```

## 7.12

#### main\_window.py

```
# 导入系统模块
import subprocess
import threading
import chardet
import json
import os
import re
import socket

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

class MainWindow(QtWidgets.QMainWindow):
    """
    主窗口类，用于显示3D模型和提供相关功能。

    Attributes:
        actor_to_filename: dict, 保存每个actor和对应的STL文件名的映射。
        actor_to_color: dict, 保存每个actor和对应的颜色的映射。
        bom_data: list, BOM表数据。
        render: vtkRenderer对象，用于渲染3D模型。
        actors: list, 所有的vtkActor对象列表。
        is_hide: bool, 是否隐藏模型。
        is_middle_button_pressed: bool, 鼠标中键是否按下。

    Methods:
        __init__: 初始化方法，创建主窗口，读取STL文件，设置布局和功能按钮。
        create_layout: 创建布局。
        load_stl_files: 读取STL文件。
        create_buttons: 创建功能按钮。
        print_to_gui: 将文本显示到GUI中。
        reset: 重置模型和交互器状态。
        show_pdf_helper: 辅助函数，打开关联的PDF文件。
        show_pdf: 显示关联的PDF文件。
        show_current_selected_model: 显示当前选中的模型的父模型。
        middle_button_press: 模拟鼠标中键按下事件。

    """

    # 常量定义
    STL_DIR = '40881_40650_unlock' # .stl文件夹路径
    BOM_FILE = '40881_40650_json.json' # bom表，转成json格式，让程序更好读取

    def __init__(self):
        """
```

初始化方法，创建主窗口，读取STL文件，设置布局和功能按钮。

"""

射：  
# 创建一个新的字典self.actor\_to\_filename来存储每个actor和它对应的STL文件名的映

self.actor\_to\_filename = {}

# 创建一个新的字典储存actor的颜色

self.actor\_to\_color = {}

# 调用父类的初始化方法

super().\_\_init\_\_()

# 获取当前路径

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

# 适配exe文件，当使用代码调试时，需要注释掉下面两行

current\_dir = os.path.dirname(current\_dir)

current\_dir = os.path.dirname(current\_dir)

# 获取STL文件夹路径

stl\_dir = os.path.join(current\_dir, self.STL\_DIR)

# 获取bom表编码信息

with open(self.BOM\_FILE, 'rb') as file:

raw\_data = file.read()

bom\_encoding = chardet.detect(raw\_data)['encoding']

# 获取模型对应BOM表信息

with open(self.BOM\_FILE, 'r', encoding=bom\_encoding) as file:

self.bom\_data = json.load(file)

# 设置窗口标题和大小

self.setWindowTitle('FlightDeck 3D模型展示 (AA,RS部分)')

# 设置窗口的固定大小

self.resize(3840, 2160)

# 创建布局

self.current\_content\_layout, self.other\_function\_layout =

self.create\_layouts()

# 创建文本框

self.create\_text\_box()

# 创建 QVTKRenderWindowInteractor 实例

self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口

self.render = vtk.vtkRenderer()

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

# 储存所有actor

self.actors = []

# 调用load\_stl\_files方法加载STL文件

self.load\_stl\_files(stl\_dir)

# 初始化交互器

self.interactorStyle = MyInteractorStyle(

self.render, self.actors, self)



```

self.vtkWidget.GetRenderWindow().GetInteractor(
).SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout (主布局) 中
self.current_content_layout.addWidget(self.vtkWidget)

# 调用 create_buttons 方法创建按钮
self.create_buttons(self.current_content_layout)
self.other_function_layout.addLayout(self.create_inform_button())

# 设置变量记录是否隐藏模型
self.is_hide = False

# 记录当前鼠标中键点击状态
self.is_middle_button_down = False

# 记录选择样式
self.select_type = False

# 记录右侧界面状态
self.is_chat = False

# 记录右侧初始文本框中的内容 (用于切换后重置)
self.previous_text = ""

def create_chat_part(self):
    # 创建label
    self.label1 = QtWidgets.QLabel("Response")
    self.label2 = QtWidgets.QLabel("Question")

    # 创建文本框
    self.response_box = QtWidgets.QTextEdit()
    self.question_box = QtWidgets.QTextEdit()
    self.response_box.setReadOnly(True)

def create_text_box(self):
    # 创建 QTextEdit 实例
    self.text_edit = QTextEdit(self)
    self.text_edit.setReadOnly(True)
    self.other_function_layout.addWidget(self.text_edit)

def create_layouts(self):
    """
    创建主布局、当前内容布局和其他功能布局，并设置布局关系和占据比例。
    """
    # 创建主布局
    main_layout = QtWidgets.QHBoxLayout()
    main_widget = QtWidgets.QWidget(self)
    main_widget.setLayout(main_layout)
    self.setCentralWidget(main_widget)

    # 创建当前内容布局
    current_content_widget = QtWidgets.QWidget(self)

```

```

current_content_layout = QtWidgets.QVBoxLayout()
current_content_widget.setLayout(current_content_layout)
main_layout.addWidget(current_content_widget, 3) # 可以按照要求调整比例

# 创建其他功能布局
other_function_widget = QtWidgets.QWidget(self)
other_function_layout = QtWidgets.QVBoxLayout()
other_function_widget.setLayout(other_function_layout)
main_layout.addWidget(other_function_widget, 1)

return current_content_layout, other_function_layout

def load_stl_files(self, stl_dir):
    """
    加载STL文件，并将模型添加到渲染器和本地的actor列表中。

    Args:
        stl_dir: STL文件夹路径。
    """
    for file_name in os.listdir(stl_dir):
        if file_name.lower().endswith('.stl'):
            file_path = os.path.join(stl_dir, file_name)
            print(file_name)

            # 创建STL读取器，读取STL文件
            reader = vtk.vtkSTLReader()
            reader.SetFileName(file_path)
            reader.Update()

            # 创建transform以保持模型原有位置（相对于原点）
            transform = vtk.vtkTransform()
            transformFilter = vtk.vtkTransformPolyDataFilter()
            transformFilter.SetInputConnection(reader.GetOutputPort())
            transformFilter.SetTransform(transform)
            transformFilter.Update()

            # 将actor添加到渲染器和本地的actor列表中，并保存到actor_to_filename字典

            mapper = vtk.vtkPolyDataMapper()
            mapper.SetInputConnection(transformFilter.GetOutputPort())
            actor = vtk.vtkActor()
            actor.SetMapper(mapper)
            self.actors.append(actor)
            if '40881' in file_name:
                actor.GetProperty().SetColor(1, 0.5, 0)
            else:
                actor.GetProperty().SetColor(0, 1, 1)
            self.render.AddActor(actor)
            self.actor_to_filename[actor] = file_name
            self.actor_to_color[actor] = actor.GetProperty().GetColor()

def create_chat_button(self):
    """
    创建与模型交互模式的按钮并且返回布局。

```

典

```

"""
text_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
# 创建submit按钮
self.submit_button = QPushButton('Submit', self)
self.submit_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.submit_button.clicked.connect(self.submit)
text_button_layout.addWidget(self.submit_button)

# 创建switch按钮
self.switch_button = QPushButton('Switch', self)
self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.switch_button.clicked.connect(self.switch)
text_button_layout.addWidget(self.switch_button)

return text_button_layout

def create_inform_button(self):
    """
    创建点击获取信息模式的按钮并且返回布局。
    """
    inform_button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
    # 创建 Show PDF 按钮
    self.show_pdf_button = QPushButton('Show PDF', self)
    self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    inform_button_layout.addWidget(self.show_pdf_button)
    self.show_pdf_button.clicked.connect(self.show_pdf)
    if self.interactorStyle.pdf_file is None:
        self.show_pdf_button.setEnabled(False)
    else:
        self.show_pdf_button.setEnabled(True)

    # 创建switch按钮
    self.switch_button = QPushButton('Switch', self)
    self.switch_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    inform_button_layout.addWidget(self.switch_button)
    self.switch_button.clicked.connect(self.switch)

    return inform_button_layout

def create_buttons(self, current_content_layout):
    """
    创建"Reset"按钮,"Show Model"按钮,"Move"按钮和"Select"按钮并添加到布局中。
    """
    self.button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
    current_content_layout.addLayout(self.button_layout) # 将水平布局添加到当前内容布局中

    # 创建 Reset 按钮
    self.reset_button = QPushButton('Reset', self)
    self.reset_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    self.reset_button.clicked.connect(self.reset)
    self.button_layout.addWidget(self.reset_button) # 将 Reset 按钮添加到水平布局中

    self.button_layout.addSpacing(20) # 添加间距

```

和高度

```

# 创建 show current selected model 按钮
self.show_current_selected_model_button = QPushButton(
    'Show Model', self)
self.show_current_selected_model_button.setFixedSize(
    80, 40) # 设置按钮的固定宽度和高度
self.show_current_selected_model_button.clicked.connect(
    self.show_current_selected_model)
self.button_layout.addWidget(self.show_current_selected_model_button)
self.show_current_selected_model_button.setEnabled(False)
self.button_layout.addSpacing(20) # 添加间距

# 创建按钮模拟鼠标中键点击事件
self.middle_button_click_button = QPushButton('Move', self)
self.middle_button_click_button.setFixedSize(80, 40) # 设置按钮的固定宽度

self.middle_button_click_button.clicked.connect(
    self.middle_button_click)
self.button_layout.addWidget(self.middle_button_click_button)
self.button_layout.addSpacing(20) # 添加间距

# 创建按钮切换选择模式
self.selection_type_button = QPushButton('Select', self)
self.selection_type_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.selection_type_button.clicked.connect(self.selection_type)
self.button_layout.addWidget(self.selection_type_button)

def print_to_gui(self, text):
    """
    将文本显示到GUI中。

    Args:
        text: 要显示的文本。
    """
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

def reset(self):
    """
    重置模型和交互器状态。
    """
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = False
    self.previous_text = ""
    if self.is_chat == False:
        self.text_edit.clear()
    else:
        self.response_box.clear()
        self.question_box.clear()
    self.interactorStyle.pdf_file = None
    for actor in self.actors:
        # if '40881' in self.actor_to_filename[actor]:
        #     actor.GetProperty().SetColor(1, 0.5, 0)
        # else:

```

```

        # actor.GetProperty().SetColor(0, 1, 1)
        actor.GetProperty().SetColor(self.actor_to_color[actor])
        actor.GetProperty().SetOpacity(1)
        actor.SetPickable(True)
        actor.SetVisibility(True)
        self.render.ResetCamera()
        self.vtkWidget.update()

def show_pdf_helper(self):
    """
    辅助函数，打开关联的PDF文件。
    """
    try:
        subprocess.run(
            ['C:\Program Files\Wondershare\PDFelement\PDFelement.exe',
self.interactorStyle.pdf_file])
    except Exception as e:
        print(e)
        self.print_to_gui(
            '打开PDF文件失败，请检查PDFelement是否安装，并且检查路径是否
为C:\Program Files\Wondershare\PDFelement\PDFelement.exe')

def show_pdf(self):
    """
    显示关联的PDF文件。
    """
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

def switch(self):
    """
    切换与模型交互和点击获取信息之间的模式。
    """
    if self.is_chat == False:
        self.is_chat = True
        self.create_chat_part()
        # 如果之前有文本，将其记录
        if self.text_edit.toPlainText() != "":
            self.previous_text = self.text_edit.toPlainText()
        # 删除之前的组件，避免因内存导致的重叠
        while self.other_function_layout.count():
            item = self.other_function_layout.takeAt(0)
            if item.widget() is not None:
                item.widget().deleteLater()
        # 添加组件
        self.other_function_layout.addWidget(self.label1)
        self.other_function_layout.addWidget(self.response_box)
        self.other_function_layout.addWidget(self.label2)
        self.other_function_layout.addWidget(self.question_box)
        self.other_function_layout.addLayout(self.create_chat_button())
    else:
        self.is_chat = False
        while self.other_function_layout.count():
            item = self.other_function_layout.takeAt(0)

```

```

        if item.widget() is not None:
            item.widget().deleteLater()
        self.create_text_box()
        self.other_function_layout.addLayout(self.create_inform_button())
        # 如果之前有文本, 将其显示
        if self.previous_text != "":
            self.text_edit.append(self.previous_text)

def submit(self):
    """
    获取模型回答
    """
    if self.question_box.toPlainText() == "":
        return
    self.response_box.clear()
    # 创建json数据用以发送给后端。根据需求更改knowledge_base_id,对于user1的模型,
    test all files是所有文件的知识库。
    # 在模型方可以根据需求细分数据库, 如需修改可能需要根据具体需求对
    knowledge_base_id进行赋值方式的更改。
    request_data = {
        "knowledge_base_id": "test all files",
        "question": self.question_box.toPlainText(),
        "history": []
    }
    answer, source_documents = send_json_data(self, request_data)
    self.response_box.append("Response: \n" + answer + "\n")
    # 由于source_documents是一个列表, 需要将其转换为字符串
    source = ' '.join(map(str, source_documents))
    self.response_box.append("Source documents: \n" + source)
    self.question_box.clear()

def show_current_selected_model(self):
    """
    显示当前选中的模型所属的上级模型。
    """
    file_name = ""
    if self.interactorStyle.picked_actor:
        file_name = self.actor_to_filename[self.interactorStyle.picked_actor]
    elif self.interactorStyle.LastPickedActor:
        file_name =
self.actor_to_filename[self.interactorStyle.LastPickedActor]
    match = re.findall(r'\d{5}', file_name)
    part_number = match[0]
    for actor in self.actors:
        if part_number not in self.actor_to_filename[actor]:
            actor.SetVisibility(False)
            actor.SetPickable(False)
    self.vtkWidget.update()
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = True

def middle_button_click(self):
    """

```

```

        模拟鼠标中键点击事件。
        """
        self.is_middle_button_down = not self.is_middle_button_down

        if self.is_middle_button_down:
            self.interactorStyle.OnMiddleButtonDown()
        else:
            self.interactorStyle.OnMiddleButtonUp()

    def selection_type(self):
        """
        切换选择模式。
        """
        self.select_type = not self.select_type

# 链接模型获得答案的函数
def send_json_data(mainwindow: MainWindow, data):
    HOST = '10.1.4.12'
    PORT = 8082

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        client_socket.connect((HOST, PORT))
    except Exception as e:
        index = 0
        print(e)
        while index < 5:
            try:
                client_socket.connect((HOST, PORT))
                break
            except Exception as e:
                print(e)
                index += 1
                if index == 5:
                    return 'Failed to get answer, please check status of GLM
server.', []
                continue

    count = 0
    response_data = None
    while response_data is None:
        try:
            client_socket.sendall(json.dumps(data).encode('utf-8'))
            response = client_socket.recv(12288).decode('utf-8')
        except Exception as e:
            print(e)
            client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client_socket.connect((HOST, PORT))
            continue
        response_data = decode_json(response)
        count += 1
        if count > 5:
            response_data['history'][0][1] = 'Failed to get answer, please try

```

```

later.'
        response_data['source_documents'] = []
        break
    # print(response_data)
    answer = response_data['history'][0][1]
    source_documents = response_data['source_documents']
    # print(answer)
    # print(source_documents)
    return answer, source_documents

# 因传输回来的数据中文会被转义为类似\\u6853的格式，所以需要解码
# BUG：传输json中的‘response’的值为中文，传输回来的编码与‘history’项中第二个元素完全一致，但是经过解码后变为乱码
def decode_json(json_str):
    try:
        data = json.loads(json_str)
    except Exception as e:
        data = None
        return data
    for key, value in data.items():
        if isinstance(value, str):
            value = value.encode('utf-8').decode('unicode_escape')
            data[key] = value
    return data

```

#### my\_interactor\_style.py

```

# 导入系统模块
import re
import os

# 导入vtk
import vtk

# 常量定义
PART_NUMBER_LENGTH = 5 # 我们的bom表和.stl文件的part number长度都是5
BOM_FILE = "40881 json.json" # bom表，转成json格式，让程序更好读取
PDF_DIR = "source files" # 40881相关的所有pdf图纸所在的文件夹
PDF_EXTENSION1 = ".pdf" # pdf后缀
PDF_EXTENSION2 = ".PDF" # pdf后缀

class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    """
    自定义的VTK交互器，继承自vtkInteractorStyleTrackballCamera类。
    该交互器主要用于处理用户与3D模型交互的事件，包括鼠标左键按下事件。

    Attributes:
        render: vtkRenderer对象，用于渲染3D模型。
        model_actor: vtkActor对象的列表，表示模型的可视化对象。
        main_window: 主窗口对象的引用。
    """

```



picker: vtkCellPicker对象, 用于捕捉鼠标选中的模型对象。  
 LastPickedActor: 上一次选中的vtkActor对象。  
 LastPickedProperty: 上一次选中的vtkProperty对象。  
 pdf\_file: 当前选中模型关联的PDF文件名。

#### Methods:

\_\_init\_\_: 初始化方法, 设置渲染器、模型可视化对象、主窗口, 并添加鼠标左键按下事件的观察器。

on\_left\_button\_press: 鼠标左键按下事件处理方法, 处理模型选中逻辑。

handle\_selection: 处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

get\_bom\_info: 根据给定的零件号从BOM表中获取相关信息。

find\_pdf\_file\_by\_vendor\_part\_number: 根据供应商零件号查找关联的PDF文件。

reset\_state: 重置交互器的状态, 取消选中模型并清除关联的PDF文件。

"""

```
def __init__(self, render, model_actor, main_window):
```

"""

初始化方法。

#### Args:

render: vtkRenderer对象, 用于渲染3D模型。

model\_actor: vtkActor对象的列表, 表示模型的可视化对象。

main\_window: 主窗口对象的引用。

"""

```
self.render = render
```

```
self.model_actor = model_actor
```

```
self.main_window = main_window
```

```
self.picker = vtk.vtkCellPicker()
```

```
# 下面几个变量, 在初始化时没有特定的初始值, 因此不需要作为参数传入
```

```
# 它会在交互过程中根据用户的选择进行更新, 不会被其他类访问
```

```
self.LastPickedActor = None # 这个变量用于记录上一次选中的 actor
```

```
self.LastPickedProperty = vtk.vtkProperty() # 这个变量是一个 vtkProperty 对象, 用于记录上一次选中的 actor 的属性
```

```
self.pdf_file = None # 这个变量用于记录选中的 actor 对应的 PDF 文件路径
```

```
self.AddObserver("LeftButtonPressEvent", self.on_left_button_press) # 添加鼠标左键按下事件观察器
```

```
# 鼠标左键按下事件
```

```
def on_left_button_press(self, obj, event):
```

"""

鼠标左键按下事件。

#### Args:

obj: 事件对象。

event: 事件类型。

"""

```
# 将主窗口中的is_middle_button_down更改为false
```

```
self.main_window.is_middle_button_down = False
```

```
if self.main_window.select_type:
```

```
# 获取鼠标点击位置
```

```
click_pos = self.GetInteractor().GetEventPosition()
```

```

self.picker.Pick(click_pos[0], click_pos[1], 0, self.render)
self.picked_actor = self.picker.GetActor()

if self.picked_actor:
    # 清空文本编辑框
    self.main_window.text_edit.clear()
    filename = self.main_window.actor_to_filename[self.picked_actor]

    # 检查是不是有已经选中的模型并分情况处理
    if self.LastPickedActor:
        if self.LastPickedActor == self.picked_actor:
            # 若选中相同的模型, 重置状态
            self.reset_state()
        else:
            # 处理选中的模型
            self.handle_selection(filename)
    else:
        # 处理选中的模型
        self.handle_selection(filename)

    # 检查处理之后是否还有选中的模型并且根据结果更新按钮状态
    if self.picked_actor or self.LastPickedActor:
        # 激活show current selected model按钮
        if self.main_window.is_hide != True:

self.main_window.show_current_selected_model_button.setEnabled(True)
    else:
        # 取消激活show current selected model按钮

self.main_window.show_current_selected_model_button.setEnabled(False)

    # 根据是否关联PDF文件, 设置显示PDF按钮的状态
    if self.pdf_file is None:
        self.main_window.show_pdf_button.setEnabled(False)
    else:
        self.main_window.show_pdf_button.setEnabled(True)

self.OnLeftButtonDown()

def handle_selection(self, filename):
    """
    处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

    Args:
        filename: 模型文件名。
    """
    # 从文件名中提取Part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'{filename}')
    self.main_window.print_to_gui(f'Part number: {part_number}')

    if part_number != '40881' or part_number != '40650':
        # 若part number不是'40881', 获取BOM信息和供应商零件号

```

```

        bom_info, vendor_part_number = self.get_bom_info(part_number)
        self.main_window.print_to_gui(bom_info)
        self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if self.pdf_file is not None:
            self.main_window.print_to_gui(f'PDF file: {self.pdf_file}')

    for actor in self.model_actor:
        if actor != self.picked_actor:
            # 设置非选中模型的颜色和透明度
            actor.GetProperty().SetColor(1, 1, 1)
            actor.GetProperty().SetOpacity(0.1)

# 设置选中模型的颜色和透明度
self.picked_actor.GetProperty().SetColor(1, 0, 0)
self.picked_actor.GetProperty().SetOpacity(1)
self.LastPickedActor = self.picked_actor

def get_bom_info(self, part_number):
    """
    根据part number从BOM表中获取相关信息。

    Args:
        part_number: 零件号。

    Returns:
        tuple: 包含BOM信息和供应商零件号的元组。
    """
    bom_info = ""
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part Number']
            for key, value in item.items():
                # 将BOM信息格式化为字符串, 用于输出
                bom_info += f"{key}: {value}\n"
            break
    return bom_info, vendor_part_number

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    """
    根据vendor part number查找关联的PDF文件。

    Args:
        vendor_part_number: 供应商零件号。

    Returns:
        str: PDF文件的完整路径, 如果未找到匹配的文件则返回None。
    """
    source_files_dir =

```

```

os.path.join(os.path.dirname(os.path.abspath(__file__)), PDF_DIR)

    # 适配exe文件, 当使用代码调试时, 需要注释掉下面这行代码
    # source_files_dir =
os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), PDF_DIR)
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(PDF_EXTENSION1) or
file_name.endswith(PDF_EXTENSION2):
            if vendor_part_number != "" and vendor_part_number in file_name:
                # 找到匹配的文件, 返回完整路径
                return os.path.join(source_files_dir, file_name)

    # 未找到匹配的文件, 返回None
    return None

def reset_state(self):
    """
    重置交互器的状态, 取消选中模型并清除关联的PDF文件。
    """
    self.picked_actor = None
    self.LastPickedActor = None
    self.pdf_file = None

    for actor in self.model_actor:
        # 重置模型的颜色和透明度为默认值
        if '40881' in self.main_window.actor_to_filename[actor]:
            actor.GetProperty().SetColor(1, 0.5, 0)
        else:
            actor.GetProperty().SetColor(0, 1, 1)
            actor.GetProperty().SetOpacity(1)

```

## app.py

```

import sys
from main_window import MainWindow, QtWidgets

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## setup.py

```

import sys
from cx_Freeze import setup, Executable

target_script = 'app.py'

build_options = {
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're',
    'chardet', 'subprocess', 'threading', 'socket'],
    'include_files': ['40881_40650_unlock/', '40881_40650_json.json',
    'my_interactor_style.py', 'main_window.py', 'source files/']
}

executable = Executable(
    script=target_script,
    base='Win32GUI'
)

setup(
    name='App',
    version='1.0',
    description='testPYQT',
    options={'build_exe': build_options},
    executables=[executable]
)

```

## 6.28

### main\_window.py

```

# 导入系统模块
import subprocess
import threading
import chardet
import json
import os
import re

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

class MainWindow(QtWidgets.QMainWindow):
    """
    主窗口类，用于显示3D模型和提供相关功能。
    """

```

**Attributes:**

actor\_to\_filename: dict, 保存每个actor和对应的STL文件名的映射。  
 bom\_data: list, BOM表数据。  
 render: vtkRenderer对象, 用于渲染3D模型。  
 actors: list, 所有的vtkActor对象列表。

**Methods:**

\_\_init\_\_: 初始化方法, 创建主窗口, 读取STL文件, 设置布局和功能按钮。  
 print\_to\_gui: 将文本显示到GUI中。  
 reset: 重置模型和交互器状态。  
 show\_pdf\_helper: 辅助函数, 打开关联的PDF文件。  
 show\_pdf: 显示关联的PDF文件。

"""

## # 常量定义

STL\_DIR = '40881\_40650\_unlock' # .stl文件夹路径  
 BOM\_FILE = '40881\_40650\_json.json' # bom表, 转成json格式, 让程序更好读取

def \_\_init\_\_(self):

"""

初始化方法, 创建主窗口, 读取STL文件, 设置布局和功能按钮。

"""

# 创建一个新的字典self.actor\_to\_filename来存储每个actor和它对应的STL文件名的映

射:

self.actor\_to\_filename = {}

# 调用父类的初始化方法

super().\_\_init\_\_()

# 获取当前路径

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

# 适配exe文件, 当使用代码调试时, 需要注释掉下面两行

# current\_dir = os.path.dirname(current\_dir)

# current\_dir = os.path.dirname(current\_dir)

# 获取STL文件夹路径

stl\_dir = os.path.join(current\_dir, self.STL\_DIR)

# 获取bom表编码信息

with open(self.BOM\_FILE, 'rb') as file:

raw\_data = file.read()

bom\_encoding = chardet.detect(raw\_data)['encoding']

# 获取模型对应BOM表信息

with open(self.BOM\_FILE, 'r', encoding=bom\_encoding) as file:

self.bom\_data = json.load(file)

# 设置窗口标题和大小

self.setWindowTitle('FlightDeck 3D模型展示 (AA,RS部分)')

# 设置窗口的固定大小

self.resize(3840, 2160)

# 创建布局

current\_content\_layout, other\_function\_layout = self.create\_layouts()

# 创建 QTextEdit 实例

```
self.text_edit = QTextEdit(self)
self.text_edit.setReadOnly(True)
other_function_layout.addWidget(self.text_edit)

# 创建 QVTKRenderWindowInteractor 实例
self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

# 储存所有actor
self.actors = []

# 调用load_stl_files方法加载STL文件
self.load_stl_files(stl_dir)

# 初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout (主布局) 中
current_content_layout.addWidget(self.vtkWidget)

# 调用 create_buttons 方法创建按钮
self.create_buttons(other_function_layout, current_content_layout)

# 设置变量记录是否隐藏模型
self.is_hide = False

# 记录当前鼠标中键点击状态
self.is_middle_button_down = False

def create_layouts(self):
    """
    创建主布局、当前内容布局和其他功能布局，并设置布局关系和占据比例。
    """
    # 创建主布局
    main_layout = QtWidgets.QHBoxLayout()
    main_widget = QtWidgets.QWidget(self)
    main_widget.setLayout(main_layout)
    self.setCentralWidget(main_widget)

    # 创建当前内容布局
    current_content_widget = QtWidgets.QWidget(self)
    current_content_layout = QtWidgets.QVBoxLayout()
    current_content_widget.setLayout(current_content_layout)
    main_layout.addWidget(current_content_widget, 3)# 可以按照要求调整比例

    # 创建其他功能布局
    other_function_widget = QtWidgets.QWidget(self)
```

```

other_function_layout = QtWidgets.QVBoxLayout()
other_function_widget.setLayout(other_function_layout)
main_layout.addWidget(other_function_widget, 1)

return current_content_layout, other_function_layout

```

```
def load_stl_files(self, stl_dir):
```

```
    """
```

```
    加载STL文件，并将模型添加到渲染器和本地的actor列表中。
```

```
    Args:
```

```
        stl_dir: STL文件夹路径。
```

```
    """
```

```
    for file_name in os.listdir(stl_dir):
```

```
        if file_name.lower().endswith('.stl'):
```

```
            file_path = os.path.join(stl_dir, file_name)
```

```
            print(file_name)
```

```
            # 创建STL读取器，读取STL文件
```

```
            reader = vtk.vtkSTLReader()
```

```
            reader.SetFileName(file_path)
```

```
            reader.Update()
```

```
            # 创建transform以保持模型原有位置（相对于原点）
```

```
            transform = vtk.vtkTransform()
```

```
            transformFilter = vtk.vtkTransformPolyDataFilter()
```

```
            transformFilter.SetInputConnection(reader.GetOutputPort())
```

```
            transformFilter.SetTransform(transform)
```

```
            transformFilter.Update()
```

```
            # 将actor添加到渲染器和本地的actor列表中，并保存到actor_to_filename字
```

```
            mapper = vtk.vtkPolyDataMapper()
```

```
            mapper.SetInputConnection(transformFilter.GetOutputPort())
```

```
            actor = vtk.vtkActor()
```

```
            actor.SetMapper(mapper)
```

```
            self.actors.append(actor)
```

```
            if '40881' in file_name:
```

```
                actor.GetProperty().SetColor(1, 0.5, 0)
```

```
            else:
```

```
                actor.GetProperty().SetColor(0, 1, 1)
```

```
            self.render.AddActor(actor)
```

```
            self.actor_to_filename[actor] = file_name
```

```
def create_buttons(self, other_function_layout, current_content_layout):
```

```
    """
```

```
    创建"Show PDF"按钮和"Reset"按钮，并添加到布局中。
```

```
    """
```

```
    button_layout = QtWidgets.QHBoxLayout() # 创建水平布局
```

```
    current_content_layout.addLayout(button_layout) # 将水平布局添加到当前内容
```

```
    # 创建 Show PDF 按钮
```

```
    self.show_pdf_button = QPushButton('Show PDF', self)
```

典

布局中



```

self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
other_function_layout.addWidget(self.show_pdf_button)
self.show_pdf_button.clicked.connect(self.show_pdf)
self.show_pdf_button.setEnabled(False)

# 创建 Reset 按钮
self.reset_button = QPushButton('Reset', self)
self.reset_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.reset_button.clicked.connect(self.reset)
button_layout.addWidget(self.reset_button) # 将 Reset 按钮添加到水平布局中

# 创建 show current selected model 按钮
self.show_current_selected_model_button = QPushButton('Show Model', self)
self.show_current_selected_model_button.setFixedSize(80, 40) # 设置按钮的
固定宽度和高度

self.show_current_selected_model_button.clicked.connect(self.show_current_selected
_model)
button_layout.addWidget(self.show_current_selected_model_button)
self.show_current_selected_model_button.setEnabled(False)

# 创建按钮模拟鼠标中键点击事件
self.middle_button_click_button = QPushButton('Middle Button Click', self)
self.middle_button_click_button.setFixedSize(80, 40) # 设置按钮的固定宽度
和高度

self.middle_button_click_button.clicked.connect(self.middle_button_click)
button_layout.addWidget(self.middle_button_click_button)

def print_to_gui(self, text):
    """
    将文本显示到GUI中。

    Args:
        text: 要显示的文本。
    """
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

def reset(self):
    """
    重置模型和交互器状态。
    """
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = False
    self.text_edit.clear()
    self.interactorStyle.pdf_file = None
    for actor in self.actors:
        if '40881' in self.actor_to_filename[actor]:
            actor.GetProperty().SetColor(1, 0.5, 0)
        else:

```

```

        actor.GetProperty().SetColor(0, 1, 1)
        actor.GetProperty().SetOpacity(1)
        actor.SetPickable(True)
        actor.SetVisibility(True)
        self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

self.vtkWidget.Initialize()
self.vtkWidget.update()

def show_pdf_helper(self):
    """
    辅助函数，打开关联的PDF文件。
    """
    try:
        subprocess.run(['C:\Program
Files\Wondershare\PDFelement\PDFelement.exe', self.interactorStyle.pdf_file])
    except Exception as e:
        print(e)
        self.print_to_gui('打开PDF文件失败，请检查PDFelement是否安装，并且检查路
径是否为:C:\Program Files\Wondershare\PDFelement\PDFelement.exe')

def show_pdf(self):
    """
    显示关联的PDF文件。
    """
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

def show_current_selected_model(self):
    """
    显示当前选中的模型所属的上级模型。
    """
    file_name = ""
    if self.interactorStyle.picked_actor:
        file_name = self.actor_to_filename[self.interactorStyle.picked_actor]
    elif self.interactorStyle.LastPickedActor:
        file_name =
self.actor_to_filename[self.interactorStyle.LastPickedActor]
    match = re.findall(r'\d{5}', file_name)
    part_number = match[0]
    for actor in self.actors:
        if part_number not in self.actor_to_filename[actor]:
            actor.SetVisibility(False)
            actor.SetPickable(False)
    self.vtkWidget.update()
    self.show_current_selected_model_button.setEnabled(False)
    self.is_hide = True

def middle_button_click(self):
    """
    模拟鼠标中键点击事件。
    """

```

```

self.is_middle_button_down = not self.is_middle_button_down

if self.is_middle_button_down:
    self.interactorStyle.OnMiddleButtonDown()
else:
    self.interactorStyle.OnMiddleButtonUp()

```

## my\_interactor\_style.py

```

# 导入系统模块
import re
import os

# 导入vtk
import vtk

# 常量定义
PART_NUMBER_LENGTH = 5 # 我们的bom表和.stl文件的part number长度都是5
BOM_FILE = "40881.json.json" # bom表, 转成json格式, 让程序更好读取
PDF_DIR = "source files" # 40881相关的所有pdf图纸所在的文件夹
PDF_EXTENSION1 = ".pdf" # pdf后缀
PDF_EXTENSION2 = ".PDF" # pdf后缀

class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    """
    自定义的VTK交互器, 继承自vtkInteractorStyleTrackballCamera类。
    该交互器主要用于处理用户与3D模型交互的事件, 包括鼠标左键按下事件。

    Attributes:
        render: vtkRenderer对象, 用于渲染3D模型。
        model_actor: vtkActor对象的列表, 表示模型的可视化对象。
        main_window: 主窗口对象的引用。
        picker: vtkCellPicker对象, 用于捕捉鼠标选中的模型对象。
        LastPickedActor: 上一次选中的vtkActor对象。
        LastPickedProperty: 上一次选中的vtkProperty对象。
        pdf_file: 当前选中模型关联的PDF文件名。

    Methods:
        __init__: 初始化方法, 设置渲染器、模型可视化对象、主窗口, 并添加鼠标左键按下事件
        的观察器。
        on_left_button_press: 鼠标左键按下事件处理方法, 处理模型选中逻辑。
        handle_selection: 处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM
        信息和PDF文件。
        get_bom_info: 根据给定的零件号从BOM表中获取相关信息。
        find_pdf_file_by_vendor_part_number: 根据供应商零件号查找关联的PDF文件。
        reset_state: 重置交互器的状态, 取消选中模型并清除关联的PDF文件。
    """
    def __init__(self, render, model_actor, main_window):
        """
        初始化方法。

```

```

    Args:
        render: vtkRenderer对象, 用于渲染3D模型。
        model_actor: vtkActor对象的列表, 表示模型的可视化对象。
        main_window: 主窗口对象的引用。
    """
    self.render = render
    self.model_actor = model_actor
    self.main_window = main_window
    self.picker = vtk.vtkCellPicker()

    # 下面几个变量, 在初始化时没有特定的初始值, 因此不需要作为参数传入
    # 它会在交互过程中根据用户的选择进行更新, 不会被其他类访问

    self.LastPickedActor = None # 这个变量用于记录上一次选中的 actor
    self.LastPickedProperty = vtk.vtkProperty() # 这个变量是一个 vtkProperty 对
    象, 用于记录上一次选中的 actor 的属性
    self.pdf_file = None # 这个变量用于记录选中的 actor 对应的 PDF 文件路径

    self.AddObserver("LeftButtonPressEvent", self.on_left_button_press) # 添加
    鼠标左键按下事件观察器

    # 鼠标左键按下事件
    def on_left_button_press(self, obj, event):
        """
        鼠标左键按下事件。

        Args:
            obj: 事件对象。
            event: 事件类型。
        """
        # 将主窗口中的is_middle_button_down更改为false
        self.main_window.is_middle_button_down = False

        # 获取鼠标点击位置
        click_pos = self.GetInteractor().GetEventPosition()
        self.picker.Pick(click_pos[0], click_pos[1], 0, self.render)
        self.picked_actor = self.picker.GetActor()

        if self.picked_actor:
            # 清空文本编辑框
            self.main_window.text_edit.clear()
            filename = self.main_window.actor_to_filename[self.picked_actor]

            # 检查是不是有已经选中的模型并分情况处理
            if self.LastPickedActor:
                if self.LastPickedActor == self.picked_actor:
                    # 若选中相同的模型, 重置状态
                    self.reset_state()
                else:
                    # 处理选中的模型
                    self.handle_selection(filename)
            else:
                # 处理选中的模型

```

```

        self.handle_selection(filename)

# 检查处理之后是否还有选中的模型并且根据结果更新按钮状态
if self.picked_actor or self.LastPickedActor:
    # 激活show current selected model按钮
    if self.main_window.is_hide != True:

self.main_window.show_current_selected_model_button.setEnabled(True)
    else:
        # 取消激活show current selected model按钮
        self.main_window.show_current_selected_model_button.setEnabled(False)

# 根据是否关联PDF文件, 设置显示PDF按钮的状态
if self.pdf_file is None:
    self.main_window.show_pdf_button.setEnabled(False)
else:
    self.main_window.show_pdf_button.setEnabled(True)

self.OnLeftButtonDown()

def handle_selection(self, filename):
    """
    处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

    Args:
        filename: 模型文件名。
    """
    # 从文件名中提取Part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'{filename}')
    self.main_window.print_to_gui(f'Part number: {part_number}')

    if part_number != '40881' or part_number != '40650':
        # 若part number不是'40881', 获取BOM信息和供应商零件号
        bom_info, vendor_part_number = self.get_bom_info(part_number)
        self.main_window.print_to_gui(bom_info)
        self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if self.pdf_file is not None:
            self.main_window.print_to_gui(f'PDF file: {self.pdf_file}')

    for actor in self.model_actor:
        if actor != self.picked_actor:
            # 设置非选中模型的颜色和透明度
            actor.GetProperty().SetColor(1, 1, 1)
            actor.GetProperty().SetOpacity(0.1)

# 设置选中模型的颜色和透明度
self.picked_actor.GetProperty().SetColor(1, 0, 0)
self.picked_actor.GetProperty().SetOpacity(1)
self.LastPickedActor = self.picked_actor

```

```

def get_bom_info(self, part_number):
    """
    根据part number从BOM表中获取相关信息。

    Args:
        part_number: 零件号。

    Returns:
        tuple: 包含BOM信息和供应商零件号的元组。
    """
    bom_info = ""
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part Number']
            for key, value in item.items():
                # 将BOM信息格式化为字符串, 用于输出
                bom_info += f"{key}: {value}\n"
            break
    return bom_info, vendor_part_number

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    """
    根据vendor part number查找关联的PDF文件。

    Args:
        vendor_part_number: 供应商零件号。

    Returns:
        str: PDF文件的完整路径, 如果未找到匹配的文件则返回None。
    """
    source_files_dir =
os.path.join(os.path.dirname(os.path.abspath(__file__)), PDF_DIR)

    # 适配exe文件, 当使用代码调试时, 需要注释掉下面这行代码
    # source_files_dir =
os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), PDF_DIR)
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(PDF_EXTENSION1) or
file_name.endswith(PDF_EXTENSION2):
            if vendor_part_number != "" and vendor_part_number in file_name:
                # 找到匹配的文件, 返回完整路径
                return os.path.join(source_files_dir, file_name)

    # 未找到匹配的文件, 返回None
    return None

def reset_state(self):
    """

```

```

重置交互器的状态，取消选中模型并清除关联的PDF文件。
"""
self.picked_actor = None
self.LastPickedActor = None
self.pdf_file = None

for actor in self.model_actor:
    # 重置模型的颜色和透明度为默认值
    if '40881' in self.main_window.actor_to_filename[actor]:
        actor.GetProperty().SetColor(1, 0.5, 0)
    else:
        actor.GetProperty().SetColor(0, 1, 1)
        actor.GetProperty().SetOpacity(1)

```

### app.py

```

import sys
from main_window import MainWindow, QtWidgets

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

### setup.py

```

import sys
from cx_Freeze import setup, Executable

target_script = 'app.py'

build_options = {
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're',
    'chardet', 'subprocess', 'threading'],
    'include_files': ['40881_40650_unlock/', '40881_40650_json.json',
    'my_interactor_style.py', 'main_window.py', 'source files/']
}

executable = Executable(
    script=target_script,
    base='Win32GUI'
)

setup(
    name='App',

```

```

    version='1.0',
    description='testPYQT',
    options={'build_exe': build_options},
    executables=[executable]
)

```

## 6.25 morning

main\_window.py

```

# 导入系统模块
import subprocess
import threading
import chardet
import json
import os

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

class MainWindow(QtWidgets.QMainWindow):
    """
    主窗口类，用于显示3D模型和提供相关功能。

    Attributes:
        actor_to_filename: dict, 保存每个actor和对应的STL文件名的映射。
        bom_data: list, BOM表数据。
        render: vtkRenderer对象，用于渲染3D模型。
        actors: list, 所有的vtkActor对象列表。

    Methods:
        __init__: 初始化方法，创建主窗口，读取STL文件，设置布局和功能按钮。
        print_to_gui: 将文本显示到GUI中。
        reset: 重置模型和交互器状态。
        show_pdf_helper: 辅助函数，打开关联的PDF文件。
        show_pdf: 显示关联的PDF文件。
    """

    # 常量定义
    STL_DIR = '40881-STL' # .stl文件夹路径
    BOM_FILE = '40881 json.json' # bom表，转成json格式，让程序更好读取

    def __init__(self):
        """

```



初始化方法，创建主窗口，读取STL文件，设置布局和功能按钮。

"""

射：  
# 创建一个新的字典self.actor\_to\_filename来存储每个actor和它对应的STL文件名的映

self.actor\_to\_filename = {}

# 调用父类的初始化方法

super().\_\_init\_\_()

# 获取当前路径

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

# 适配exe文件

current\_dir = os.path.dirname(current\_dir)

current\_dir = os.path.dirname(current\_dir)

# 获取STL文件夹路径

stl\_dir = os.path.join(current\_dir, self.STL\_DIR)

# 获取bom表编码信息

with open(self.BOM\_FILE, 'rb') as file:

raw\_data = file.read()

bom\_encoding = chardet.detect(raw\_data)['encoding']

# 获取模型对应BOM表信息

with open(self.BOM\_FILE, 'r', encoding=bom\_encoding) as file:

self.bom\_data = json.load(file)

# 设置窗口标题和大小

self.setWindowTitle('FlightDeck 3D模型展示 (AA部分)')

# 设置窗口的固定大小

self.resize(3840, 2160)

# 创建布局

current\_content\_layout, other\_function\_layout = self.create\_layouts()

# 创建 QTextEdit 实例

self.text\_edit = QTextEdit(self)

self.text\_edit.setReadOnly(True)

other\_function\_layout.addWidget(self.text\_edit)

# 创建 QVTKRenderWindowInteractor 实例

self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口

self.render = vtk.vtkRenderer()

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

# 储存所有actor

self.actors = []

# 调用load\_stl\_files方法加载STL文件

self.load\_stl\_files(stl\_dir)

# 初始化交互器

self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interacto

```
rStyle)
    self.vtkWidget.Initialize()

    # 将当前内容部分放入 current_content_layout (主布局) 中
    current_content_layout.addWidget(self.vtkWidget)

    # 调用 create_buttons 方法创建按钮
    self.create_buttons(other_function_layout, current_content_layout)

def create_layouts(self):
    """
    创建主布局、当前内容布局和其他功能布局，并设置布局关系和占据比例。
    """
    # 创建主布局
    main_layout = QtWidgets.QHBoxLayout()
    main_widget = QtWidgets.QWidget(self)
    main_widget.setLayout(main_layout)
    self.setCentralWidget(main_widget)

    # 创建当前内容布局
    current_content_widget = QtWidgets.QWidget(self)
    current_content_layout = QtWidgets.QVBoxLayout()
    current_content_widget.setLayout(current_content_layout)
    main_layout.addWidget(current_content_widget, 3)# 可以按照要求调整比例

    # 创建其他功能布局
    other_function_widget = QtWidgets.QWidget(self)
    other_function_layout = QtWidgets.QVBoxLayout()
    other_function_widget.setLayout(other_function_layout)
    main_layout.addWidget(other_function_widget, 1)

    return current_content_layout, other_function_layout

def load_stl_files(self, stl_dir):
    """
    加载STL文件，并将模型添加到渲染器和本地的actor列表中。

    Args:
        stl_dir: STL文件夹路径。
    """
    for file_name in os.listdir(stl_dir):
        if file_name.lower().endswith('.stl'):
            file_path = os.path.join(stl_dir, file_name)

            # 创建STL读取器，读取STL文件
            reader = vtk.vtkSTLReader()
            reader.SetFileName(file_path)
            reader.Update()

            # 创建transform以保持模型原有位置（相对于原点）
            transform = vtk.vtkTransform()
            transformFilter = vtk.vtkTransformPolyDataFilter()
            transformFilter.SetInputConnection(reader.GetOutputPort())
```

典

```

        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到actor_to_filename字典

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name

def create_buttons(self, other_function_layout, current_content_layout):
    """
    创建"Show PDF"按钮和"Reset"按钮, 并添加到布局中。
    """
    # 创建 Show PDF 按钮
    self.show_pdf_button = QPushButton('Show PDF', self)
    self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    other_function_layout.addWidget(self.show_pdf_button)

    self.show_pdf_button.clicked.connect(self.show_pdf)

    # 创建 Reset 按钮
    self.reset_button = QPushButton('Reset', self)
    self.reset_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    self.reset_button.clicked.connect(self.reset)
    current_content_layout.addWidget(self.reset_button)

def print_to_gui(self, text):
    """
    将文本显示到GUI中。

    Args:
        text: 要显示的文本。
    """
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

def reset(self):
    """
    重置模型和交互器状态。
    """
    self.text_edit.clear()
    self.interactorStyle.pdf_file = None
    for actor in self.actors:
        actor.GetProperty().SetColor(1, 1, 1)
        actor.GetProperty().SetOpacity(1)
    self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

```

```

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
    self.vtkWidget.Initialize()
    self.vtkWidget.update()

def show_pdf_helper(self):
    """
    辅助函数，打开关联的PDF文件。
    """
    try:
        subprocess.run(['C:\Program
Files\Wondershare\PDFelement\PDFelement.exe', self.interactorStyle.pdf_file])
    except Exception as e:
        print(e)
        self.print_to_gui('打开PDF文件失败，请检查PDFelement是否安装，并且检查路
径是否为:C:\Program Files\Wondershare\PDFelement\PDFelement.exe')

def show_pdf(self):
    """
    显示关联的PDF文件。
    """
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

```

## my\_interactor\_style.py

```

# 导入系统模块
import re
import os
import sys

# 导入vtk
import vtk

# 常量定义
PART_NUMBER_LENGTH = 5 # 我们的bom表和.stl文件的part number长度都是5
BOM_FILE = "40881 json.json" # bom表，转成json格式，让程序更好读取
PDF_DIR = "source files" # 40881相关的所有pdf图纸所在的文件夹
PDF_EXTENSION = ".pdf" # pdf后缀

class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    """
    自定义的VTK交互器，继承自vtkInteractorStyleTrackballCamera类。
    该交互器主要用于处理用户与3D模型交互的事件，包括鼠标左键按下事件。

    Attributes:
        render: vtkRenderer对象，用于渲染3D模型。
        model_actor: vtkActor对象的列表，表示模型的可视化对象。
        main_window: 主窗口对象的引用。
        picker: vtkCellPicker对象，用于捕捉鼠标选中的模型对象。
    """

```

LastPickedActor: 上一次选中的vtkActor对象。  
 LastPickedProperty: 上一次选中的vtkProperty对象。  
 pdf\_file: 当前选中模型关联的PDF文件名。

Methods:

\_\_init\_\_: 初始化方法, 设置渲染器、模型可视化对象、主窗口, 并添加鼠标左键按下事件的观察器。

on\_left\_button\_press: 鼠标左键按下事件处理方法, 处理模型选中逻辑。

handle\_selection: 处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

get\_bom\_info: 根据给定的零件号从BOM表中获取相关信息。

find\_pdf\_file\_by\_vendor\_part\_number: 根据供应商零件号查找关联的PDF文件。

reset\_state: 重置交互器的状态, 取消选中模型并清除关联的PDF文件。

"""

```
def __init__(self, render, model_actor, main_window):
```

"""

初始化方法。

Args:

render: vtkRenderer对象, 用于渲染3D模型。

model\_actor: vtkActor对象的列表, 表示模型的可视化对象。

main\_window: 主窗口对象的引用。

"""

```
self.render = render
```

```
self.model_actor = model_actor
```

```
self.main_window = main_window
```

```
self.picker = vtk.vtkCellPicker()
```

# 下面几个变量, 在初始化时没有特定的初始值, 因此不需要作为参数传入

# 它会在交互过程中根据用户的选择进行更新, 不会被其他类访问

```
self.LastPickedActor = None # 这个变量用于记录上一次选中的 actor
```

```
self.LastPickedProperty = vtk.vtkProperty() # 这个变量是一个 vtkProperty 对象, 用于记录上一次选中的 actor 的属性
```

```
self.pdf_file = None # 这个变量用于记录选中的 actor 对应的 PDF 文件路径
```

```
self.AddObserver("LeftButtonPressEvent", self.on_left_button_press) # 添加鼠标左键按下事件观察器
```

# 鼠标左键按下事件

```
def on_left_button_press(self, obj, event):
```

"""

鼠标左键按下事件。

Args:

obj: 事件对象。

event: 事件类型。

"""

# 获取鼠标点击位置

```
click_pos = self.GetInteractor().GetEventPosition()
```

```
self.picker.Pick(click_pos[0], click_pos[1], 0, self.render)
```

```
self.picked_actor = self.picker.GetActor()
```

```
if self.picked_actor:
```

```

# 清空文本编辑框
self.main_window.text_edit.clear()
filename = self.main_window.actor_to_filename[self.picked_actor]

# 检查是不是有已经选中的模型并分情况处理
if self.LastPickedActor:
    if self.LastPickedActor == self.picked_actor:
        # 若选中相同的模型, 重置状态
        self.reset_state()
    else:
        # 处理选中的模型
        self.handle_selection(filename)
else:
    # 处理选中的模型
    self.handle_selection(filename)

# 根据是否关联PDF文件, 设置显示PDF按钮的状态
if self.pdf_file is None:
    self.main_window.show_pdf_button.setEnabled(False)
else:
    self.main_window.show_pdf_button.setEnabled(True)

self.OnLeftButtonDown()

def handle_selection(self, filename):
    """
    处理模型选中的具体逻辑, 显示模型相关信息, 查找并显示关联的BOM信息和PDF文件。

    Args:
        filename: 模型文件名。
    """
    # 从文件名中提取Part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'{filename}')
    self.main_window.print_to_gui(f'Part number: {part_number}')

    if part_number != '40881':
        # 若part number不是'40881', 获取BOM信息和供应商零件号
        bom_info, vendor_part_number = self.get_bom_info(part_number)
        self.main_window.print_to_gui(bom_info)
        self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if self.pdf_file is not None:
            self.main_window.print_to_gui(f'PDF file: {self.pdf_file}')

    for actor in self.model_actor:
        if actor != self.picked_actor:
            # 设置非选中模型的颜色和透明度
            actor.GetProperty().SetColor(1, 1, 1)
            actor.GetProperty().SetOpacity(0.1)

# 设置选中模型的颜色和透明度
self.picked_actor.GetProperty().SetColor(1, 0, 0)

```

```
self.picked_actor.GetProperty().SetOpacity(1)
self.LastPickedActor = self.picked_actor

def get_bom_info(self, part_number):
    """
    根据part number从BOM表中获取相关信息。

    Args:
        part_number: 零件号。

    Returns:
        tuple: 包含BOM信息和供应商零件号的元组。
    """
    bom_info = ""
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part Number']
            for key, value in item.items():
                # 将BOM信息格式化为字符串, 用于输出
                bom_info += f"{key}: {value}\n"
            break
    return bom_info, vendor_part_number

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    """
    根据vendor part number查找关联的PDF文件。

    Args:
        vendor_part_number: 供应商零件号。

    Returns:
        str: PDF文件的完整路径, 如果未找到匹配的文件则返回None。
    """
    #source_files_dir =
    os.path.join(os.path.dirname(os.path.abspath(__file__)), PDF_DIR)

    # 适配exe文件
    source_files_dir =
    os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), PDF_DIR)
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(PDF_EXTENSION):
            # 提取文件名中的零件号 (去除扩展名)
            match = re.search(r'(.*)\{}'.format(PDF_EXTENSION), file_name)
            if match.group(1) == vendor_part_number:
                # 找到匹配的文件, 返回完整路径
                return os.path.join(source_files_dir, file_name)

    # 未找到匹配的文件, 返回None
    return None
```

```
def reset_state(self):  
    """  
    重置交互器的状态，取消选中模型并清除关联的PDF文件。  
    """  
    self.LastPickedActor = None  
    self.pdf_file = None  
  
    for actor in self.model_actor:  
        # 重置模型的颜色和透明度为默认值  
        actor.GetProperty().SetColor(1, 1, 1)  
        actor.GetProperty().SetOpacity(1)
```

## app.py

```
import sys  
from main_window import MainWindow, QtWidgets  
  
def main():  
    app = QtWidgets.QApplication(sys.argv)  
    window = MainWindow()  
    window.show()  
    sys.exit(app.exec_())  
  
if __name__ == "__main__":  
    main()
```

## setup.py

```
import sys  
from cx_Freeze import setup, Executable  
  
target_script = 'app.py'  
  
build_options = {  
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're',  
    'chardet', 'subprocess', 'threading'],  
    'include_files': ['40881-STL/', '40881 json.json', 'my_interactor_style.py',  
    'main_window.py', 'source files/']  
}  
  
executable = Executable(  
    script=target_script,  
    base='Win32GUI'  
)  
  
setup(  

```



```
name='App',
version='1.0',
description='testPYQT',
options={'build_exe': build_options},
executables=[executable]
)
```

## 6.21 14:00

app.py

```
import sys
from main_window import MainWindow, QtWidgets

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

my\_interactor\_style.py

```
# 导入vtk
import vtk

# 导入系统模块
import re
import os
import sys

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):
        self.render = render
        self.modle_actor = modle_actor
        self.main_window = main_window # 将 main_window 作为实例变量保存
        self.picker = vtk.vtkCellPicker()
        # 记录上一次选中的actor
        self.LastPickedActor = None
        self.LastPickedProperty = vtk.vtkProperty()
        self.pdf_file = None
        # 添加鼠标左键按下事件
        self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
```

```

def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 清空文本框
        self.main_window.text_edit.clear()
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
                self.pdf_file = None
            else:
                # 获得part number
                match = re.findall(r'\d{5}', filename)
                part_number = match[-1]
                self.main_window.print_to_gui(f'{filename}')
                self.main_window.print_to_gui(f'Part Number: {part_number}')
                # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
                if part_number != '40881':
                    # 从 bom 表中获取 part number 对应的信息
                    bom_info = ""
                    # 这个变量用作去json中找Vendor Part Number
                    vendor_part_number = ""
                    for item in self.main_window.bom_data:
                        if item['Part Number'] == part_number:
                            vendor_part_number = item['Vendor Part Number']
                            for key, value in item.items():
                                bom_info += f"{key}: {value}\n"
                            break
                    # 使用 print_to_gui 方法来显示信息
                    self.main_window.print_to_gui(bom_info)
                    self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
                    if self.pdf_file is not None:
                        self.main_window.print_to_gui(f'PDF File:
{self.pdf_file}')
                # 将actor的颜色设置为红色
                # 将其他actor的颜色设置为白色
                # 将actor的透明度设置为1
                # 将其他actor的透明度设置为0.1
                for actor1 in self.modle_actor:
                    if actor1 != self.picked_actor:

```

```

        actor1.GetProperty().SetColor(1,1,1)
        actor1.GetProperty().SetOpacity(0.1)
        self.picked_actor.GetProperty().SetColor(1,0,0)
        self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
# 如果上一次选中的actor不存在
else:
    # 使用 print_to_gui 方法来显示信息
    self.main_window.print_to_gui(f'{filename}')
    # 获得part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'Part Number: {part_number}')
    # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
    if part_number != '40881':
        # 从 bom 表中获取 part number 对应的信息
        bom_info = ""
        # 用来从我们的json中找Vendor Part Number
        vendor_part_number = ""
        for item in self.main_window.bom_data:
            if item['Part Number'] == part_number:
                vendor_part_number = item['Vendor Part Number']
                for key, value in item.items():
                    bom_info += f"{key}: {value}\n"
                break
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(bom_info)
        self.pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if self.pdf_file is not None:
            self.main_window.print_to_gui(f'PDF File:
{self.pdf_file}')
        # 将actor的颜色设置为红色
        # 将其他actor的颜色设置为白色
        # 将actor的透明度设置为1
        # 将其他actor的透明度设置为0.1
        for actor1 in self.modle_actor:
            if actor1 != self.picked_actor:
                actor1.GetProperty().SetColor(1,1,1)
                actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
    if self.pdf_file is None:
        self.main_window.show_pdf_button.setEnabled(False)
    else:
        self.main_window.show_pdf_button.setEnabled(True)
# 执行默认操作
self.OnLeftButtonDown()

def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    # source_files_dir =

```

```

os.path.join(os.path.dirname(os.path.abspath(__file__)), 'source files')
    # 适配exe文件
    source_files_dir =
os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.path.abspath(__file__)))), 'source files')
    # 遍历source files文件夹中的文件
    for file_name in os.listdir(source_files_dir):
        if file_name.endswith(".pdf"):
            # 从PDF文件名中提取Vendor Part Number
            match = re.search(r'(.*)\.pdf', file_name)
            if match.group(1) == vendor_part_number:
                # 返回找到的PDF文件路径
                return os.path.join(source_files_dir, file_name)
    # 如果未找到对应的PDF文件, 返回None
    return None

```

## main\_window.py

```

# 导入PyQt5模块
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QTextEdit, QPushButton

# 导入自定义模块
from my_interactor_style import MyInteractorStyle, vtk

# 导入vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

# 导入系统模块
import subprocess
import threading
import chardet
import json
import os

class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        # 获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        # 适配exe文件
        current_dir = os.path.dirname(current_dir)
        current_dir = os.path.dirname(current_dir)
        # 获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

```

```
# 获取bom表编码信息
with open("40881 json.json", "rb") as file:
    raw_data = file.read()
    bom_encoding = chardet.detect(raw_data)['encoding']
# 获取模型对应BOM表信息
with open("40881 json.json", "r", encoding=bom_encoding) as file:
    self.bom_data = json.load(file)

# 设置窗口标题和大小
self.setWindowTitle("VTK Test")
self.resize(3840, 2160)

# 创建主布局
main_layout = QtWidgets.QHBoxLayout()
main_widget = QtWidgets.QWidget(self)
main_widget.setLayout(main_layout)
self.setCentralWidget(main_widget)
# 创建当前内容部分
current_content_widget = QtWidgets.QWidget(self)
current_content_layout = QtWidgets.QVBoxLayout()
current_content_widget.setLayout(current_content_layout)
main_layout.addWidget(current_content_widget, 3)

# 创建其他功能部分
other_function_widget = QtWidgets.QWidget(self)
other_function_layout = QtWidgets.QVBoxLayout()
other_function_widget.setLayout(other_function_layout)
main_layout.addWidget(other_function_widget, 1)

# 创建 QTextEdit 实例
self.text_edit = QTextEdit(self)
self.text_edit.setReadOnly(True)
other_function_layout.addWidget(self.text_edit)

# 创建 Show PDF 按钮
self.show_pdf_button = QPushButton("Show PDF", self)
self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
other_function_layout.addWidget(self.show_pdf_button)

self.show_pdf_button.clicked.connect(self.show_pdf)

# 创建 QVTKRenderWindowInteractor 实例
self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

#储存所有actor
self.actors = []
```

```

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename
        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

```

字典

映射

```

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中
current_content_layout.addWidget(self.vtkWidget)

# 创建按钮
self.button = QPushButton("Reset", self)
self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.button.clicked.connect(self.reset)
current_content_layout.addWidget(self.button)

# 创建其他功能部件并放入 other_function_layout 中
# 例如: other_button = QPushButton("Other Function", self)

def print_to_gui(self, text):
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

```

#重置

```

def reset(self):
    print("reset")
    self.text_edit.clear()
    self.interactorStyle.pdf_file = None
    for actor in self.actors:
        actor.GetProperty().SetColor(1,1,1)
        actor.GetProperty().SetOpacity(1)
    self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

self.vtkWidget.Initialize()
self.vtkWidget.update()

# 显示PDF
def show_pdf_helper(self):
    try:
        subprocess.run(['C:\Program
Files\Wondershare\PDFelement\PDFelement.exe', self.interactorStyle.pdf_file])
    except Exception as e:
        print(e)
        self.print_to_gui("打开PDF文件失败，请检查PDFelement是否安装，并且检查路径是否为:C:\Program Files\Wondershare\PDFelement\PDFelement.exe")

def show_pdf(self):
    thread = threading.Thread(target=self.show_pdf_helper)
    thread.start()

```

## setup.py

```

import sys
from cx_Freeze import setup, Executable

target_script = 'testPYQT.py'

build_options = {
    'packages': ['vtkmodules', 'vtk', 'PyQt5', 'numpy', 'os', 'json', 're', 'chardet'],
    'include_files': ['40881-STL/', '40881 json.json']
}

executable = Executable(
    script=target_script,
    base='Win32GUI'
)

setup(
    name='testPYQT',
    version='1.0',
    description='testPYQT',
    options={'build_exe': build_options},

```

```
        executables=[executable]
    )
```

## 6.21 morning

```
#导入系统模块
import sys
import os
import json
import re
import chardet

#导入PyQt5模块
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication,
QTextEdit

#导入VTK模块
import vtk
from vtkmodules.qt.QVTKRenderWindowInteractor import
QVTKRenderWindowInteractor

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):
        self.render = render
        self.modle_actor = modle_actor
        self.main_window = main_window # 将 main_window 作为实例变量保存
        self.picker = vtk.vtkCellPicker()
        # 记录上一次选中的actor
        self.LastPickedActor = None
        self.LastPickedProperty = vtk.vtkProperty()
        # 添加鼠标左键按下事件
        self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 清空文本框
        self.main_window.text_edit.clear()
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
```



```

if self.LastPickedActor:
    # 如果这次选中的actor和上次选中的actor相同
    if self.LastPickedActor == self.picked_actor:
        # 将actor恢复到上次选中时的状态
        for actor1 in self.modle_actor:
            actor1.GetProperty().SetColor(1,1,1)
            actor1.GetProperty().SetOpacity(1)
        self.LastPickedActor = None
        self.picked_actor = None
    else:
        # 获得part number
        match = re.findall(r'\d{5}', filename)
        part_number = match[-1]
        self.main_window.print_to_gui(f'{filename}')
        self.main_window.print_to_gui(f'Part Number:
{part_number}')

# 如果获取到的数字是40881, 说明该模型文件没有在bom表中
if part_number != '40881':
    # 从 bom 表中获取 part number 对应的信息
    bom_info = ""
    # 这个变量用作去json中找Vendor Part Number
    vendor_part_number = ""
    for item in self.main_window.bom_data:
        if item['Part Number'] == part_number:
            vendor_part_number = item['Vendor Part
Number']

            for key, value in item.items():
                bom_info += f"{key}: {value}\n"
            break
    # 使用 print_to_gui 方法来显示信息
    self.main_window.print_to_gui(bom_info)
# 将actor的颜色设置为红色
# 将其他actor的颜色设置为白色
# 将actor的透明度设置为1
# 将其他actor的透明度设置为0.1
for actor1 in self.modle_actor:
    if actor1 != self.picked_actor:
        actor1.GetProperty().SetColor(1,1,1)
        actor1.GetProperty().SetOpacity(0.1)
    self.picked_actor.GetProperty().SetColor(1,0,0)
    self.picked_actor.GetProperty().SetOpacity(1)
# 记录这次选中的actor
self.LastPickedActor = self.picked_actor
# 如果上一次选中的actor不存在
else:
    # 使用 print_to_gui 方法来显示信息
    self.main_window.print_to_gui(f'{filename}')
    # 获得part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'Part Number: {part_number}')
    # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
    if part_number != '40881':
        # 从 bom 表中获取 part number 对应的信息

```

```

        bom_info = ""
        # 用来从我们的json中找Vendor Part Number
        vendor_part_number = ""
        for item in self.main_window.bom_data:
            if item['Part Number'] == part_number:
                vendor_part_number = item['Vendor Part Number']
                for key, value in item.items():
                    bom_info += f"{key}: {value}\n"
                break
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(bom_info)
        pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if pdf_file is not None:
            self.main_window.print_to_gui(f'PDF File: {pdf_file}')
        # 将actor的颜色设置为红色
        # 将其他actor的颜色设置为白色
        # 将actor的透明度设置为1
        # 将其他actor的透明度设置为0.1
        for actor1 in self.modle_actor:
            if actor1 != self.picked_actor:
                actor1.GetProperty().SetColor(1,1,1)
                actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
        # 执行默认操作
        self.OnLeftButtonDown()
    def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
        source_files_dir =
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'source files')

        # 遍历source files文件夹中的文件
        for file_name in os.listdir(source_files_dir):
            if file_name.endswith(".pdf"):
                # 从PDF文件名中提取Vendor Part Number
                match = re.search(r'(.*)\.pdf', file_name)
                if match.group(1) == vendor_part_number:
                    # 返回找到的PDF文件路径
                    return os.path.join(source_files_dir, file_name)

        # 如果未找到对应的PDF文件, 返回None
        return None

class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件
        # 名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法

```

```
super().__init__()

#获取当前路径
current_dir = os.path.dirname(os.path.abspath(__file__))
#获取STL文件夹路径
stl_dir = os.path.join(current_dir, '40881-STL')

# 获取bom表编码信息
with open("40881 json.json", "rb") as file:
    raw_data = file.read()
    bom_encoding = chardet.detect(raw_data)['encoding']
# 获取模型对应BOM表信息
with open("40881 json.json", "r", encoding=bom_encoding) as file:
    self.bom_data = json.load(file)

# 设置窗口标题和大小
self.setWindowTitle("VTK Test")
self.resize(3840, 2160)

# 创建主布局
main_layout = QtWidgets.QHBoxLayout()
main_widget = QtWidgets.QWidget(self)
main_widget.setLayout(main_layout)
self.setCentralWidget(main_widget)
# 创建当前内容部分
current_content_widget = QtWidgets.QWidget(self)
current_content_layout = QtWidgets.QVBoxLayout()
current_content_widget.setLayout(current_content_layout)
main_layout.addWidget(current_content_widget, 3)

# 创建其他功能部分
other_function_widget = QtWidgets.QWidget(self)
other_function_layout = QtWidgets.QVBoxLayout()
other_function_widget.setLayout(other_function_layout)
main_layout.addWidget(other_function_widget, 1)

# 创建 QTextEdit 实例
self.text_edit = QTextEdit(self)
self.text_edit.setReadOnly(True)
other_function_layout.addWidget(self.text_edit)

self.show_pdf_button = QPushButton("Show PDF", self)
self.show_pdf_button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
other_function_layout.addWidget(self.show_pdf_button)

#self.show_pdf_button.clicked.connect(self.show_pdf)

# 创建 QVTKRenderWindowInteractor 实例
self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
```

```

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

#储存所有actor
self.actors = []

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到
actor_to_filename 字典
        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件
名的映射

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors,
self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interacto
rStyle)

self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中
current_content_layout.addWidget(self.vtkWidget)

# 创建按钮
self.button = QPushButton("Reset", self)
self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.button.clicked.connect(self.reset)
current_content_layout.addWidget(self.button)

# 创建其他功能部件并放入 other_function_layout 中

```

```

        # 例如: other_button = QPushButton("Other Function", self)

    def print_to_gui(self, text):
        # 在文本编辑区域中添加新的文本
        self.text_edit.append(text)
    #重置
    def reset(self):
        print("reset")
        self.text_edit.clear()
        for actor in self.actors:
            actor.GetProperty().SetColor(1,1,1)
            actor.GetProperty().SetOpacity(1)
        self.interactorStyle = MyInteractorStyle(self.render, self.actors,
self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interacto
rStyle)

        self.vtkWidget.Initialize()
        self.vtkWidget.update()


def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

6.20 16: 20

```

import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication,
QTextEdit
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
import vtk
import os
import numpy as np
import json
import re
import chardet

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):

```

```

self.render = render
self.modle_actor = modle_actor
self.main_window = main_window # 将 main_window 作为实例变量保存
self.picker = vtk.vtkCellPicker()
# 记录上一次选中的actor
self.LastPickedActor = None
self.LastPickedProperty = vtk.vtkProperty()
# 添加鼠标左键按下事件
self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 清空文本框
        self.main_window.text_edit.clear()
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
            else:
                # 获得part number
                match = re.findall(r'\d{5}', filename)
                part_number = match[-1]
                self.main_window.print_to_gui(f'{filename}')
                self.main_window.print_to_gui(f'Part Number: {part_number}')
                # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
                if part_number != '40881':
                    # 从 bom 表中获取 part number 对应的信息
                    bom_info = ""
                    # 这个变量用作去json中找Vendor Part Number
                    vendor_part_number = ""
                    for item in self.main_window.bom_data:
                        if item['Part Number'] == part_number:
                            vendor_part_number = item['Vendor Part Number']
                            for key, value in item.items():
                                bom_info += f"{key}: {value}\n"
                            break
                    # 使用 print_to_gui 方法来显示信息
                    self.main_window.print_to_gui(bom_info)
                # 将actor的颜色设置为红色

```

```

        # 将其他actor的颜色设置为白色
        # 将actor的透明度设置为1
        # 将其他actor的透明度设置为0.1
        for actor1 in self.modle_actor:
            if actor1 != self.picked_actor:
                actor1.GetProperty().SetColor(1,1,1)
                actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
            # 记录这次选中的actor
            self.LastPickedActor = self.picked_actor
# 如果上一次选中的actor不存在
else:
    # 使用 print_to_gui 方法来显示信息
    self.main_window.print_to_gui(f'{filename}')
    # 获得part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'Part Number: {part_number}')
    # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
    if part_number != '40881':
        # 从 bom 表中获取 part number 对应的信息
        bom_info = ""
        # 用来从我们的json中找Vendor Part Number
        vendor_part_number = ""
        for item in self.main_window.bom_data:
            if item['Part Number'] == part_number:
                vendor_part_number = item['Vendor Part Number']
                for key, value in item.items():
                    bom_info += f"{key}: {value}\n"
                break
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(bom_info)
        pdf_file =
self.find_pdf_file_by_vendor_part_number(vendor_part_number)
        if pdf_file is not None:
            self.main_window.print_to_gui(f'PDF File: {pdf_file}')
        # 将actor的颜色设置为红色
        # 将其他actor的颜色设置为白色
        # 将actor的透明度设置为1
        # 将其他actor的透明度设置为0.1
        for actor1 in self.modle_actor:
            if actor1 != self.picked_actor:
                actor1.GetProperty().SetColor(1,1,1)
                actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
            # 记录这次选中的actor
            self.LastPickedActor = self.picked_actor
# 执行默认操作
self.OnLeftButtonDown()
def find_pdf_file_by_vendor_part_number(self, vendor_part_number):
    source_files_dir =
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'source files')

```

```

# 遍历source_files文件夹中的文件
for file_name in os.listdir(source_files_dir):
    if file_name.endswith(".pdf"):
        # 从PDF文件名中提取Vendor Part Number
        match = re.search(r'(.*)\.pdf', file_name)
        if match.group(1) == vendor_part_number:
            # 返回找到的PDF文件路径
            return os.path.join(source_files_dir, file_name)

# 如果未找到对应的PDF文件, 返回None
return None

class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        # 获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        # 获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

        # 获取bom表编码信息
        with open("40881 json.json", "rb") as file:
            raw_data = file.read()
            bom_encoding = chardet.detect(raw_data)['encoding']
        # 获取模型对应BOM表信息
        with open("40881 json.json", "r", encoding=bom_encoding) as file:
            self.bom_data = json.load(file)

        # 设置窗口标题和大小
        self.setWindowTitle("VTK Test")
        self.resize(3840, 2160)

        # 创建主布局
        main_layout = QtWidgets.QHBoxLayout()
        main_widget = QtWidgets.QWidget(self)
        main_widget.setLayout(main_layout)
        self.setCentralWidget(main_widget)
        # 创建当前内容部分
        current_content_widget = QtWidgets.QWidget(self)
        current_content_layout = QtWidgets.QVBoxLayout()
        current_content_widget.setLayout(current_content_layout)
        main_layout.addWidget(current_content_widget, 3)

        # 创建其他功能部分
        other_function_widget = QtWidgets.QWidget(self)

```



```

other_function_layout = QtWidgets.QVBoxLayout()
other_function_widget.setLayout(other_function_layout)
main_layout.addWidget(other_function_widget, 1)

# 创建 QTextEdit 实例
self.text_edit = QTextEdit(self)
self.text_edit.setReadOnly(True)
other_function_layout.addWidget(self.text_edit)

# 创建 QVTKRenderWindowInteractor 实例
self.vtkWidget = QVTKRenderWindowInteractor(self)

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

#储存所有actor
self.actors = []

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interacto

```

字典

映射

```

rStyle)
    self.vtkWidget.Initialize()

    # 将当前内容部分放入 current_content_layout 中
    current_content_layout.addWidget(self.vtkWidget)

    # 创建按钮
    self.button = QPushButton("Reset", self)
    self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
    self.button.clicked.connect(self.reset)
    current_content_layout.addWidget(self.button)

    # 创建其他功能部件并放入 other_function_layout 中
    # 例如: other_button = QPushButton("Other Function", self)

def print_to_gui(self, text):
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

#重置
def reset(self):
    print("reset")
    self.text_edit.clear()
    for actor in self.actors:
        actor.GetProperty().SetColor(1,1,1)
        actor.GetProperty().SetOpacity(1)
    self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interacto
rStyle)
    self.vtkWidget.Initialize()
    self.vtkWidget.update()

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## 6.20

```

import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication,
QTextEdit

```

```
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
import vtk
import os
import numpy as np
import json
import re
import chardet

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):
        self.render = render
        self.modle_actor = modle_actor
        self.main_window = main_window # 将 main_window 作为实例变量保存
        self.picker = vtk.vtkCellPicker()
        # 记录上一次选中的actor
        self.LastPickedActor = None
        self.LastPickedProperty = vtk.vtkProperty()
        # 添加鼠标左键按下事件
        self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 清空文本框
        self.main_window.text_edit.clear()
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
            else:
                # 获得part number
                match = re.findall(r'\d{5}', filename)
                part_number = match[-1]
                self.main_window.print_to_gui(f'{filename}')
                self.main_window.print_to_gui(f'Part Number: {part_number}')
                # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
                if part_number != '40881':
                    # 从 bom 表中获取 part number 对应的信息
```

```

        bom_info = ""
        for item in self.main_window.bom_data:
            if item['Part Number'] == part_number:
                for key, value in item.items():
                    bom_info += f"{key}: {value}\n"
                break
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(bom_info)
    # 将actor的颜色设置为红色
    # 将其他actor的颜色设置为白色
    # 将actor的透明度设置为1
    # 将其他actor的透明度设置为0.1
    for actor1 in self.modle_actor:
        if actor1 != self.picked_actor:
            actor1.GetProperty().SetColor(1,1,1)
            actor1.GetProperty().SetOpacity(0.1)
    self.picked_actor.GetProperty().SetColor(1,0,0)
    self.picked_actor.GetProperty().SetOpacity(1)
    # 记录这次选中的actor
    self.LastPickedActor = self.picked_actor
# 如果上一次选中的actor不存在
else:
    # 使用 print_to_gui 方法来显示信息
    self.main_window.print_to_gui(f'{filename}')
    # 获得part number
    match = re.findall(r'\d{5}', filename)
    part_number = match[-1]
    self.main_window.print_to_gui(f'Part Number: {part_number}')
    # 如果获取到的数字是40881, 说明该模型文件没有在bom表中
    if part_number != '40881':
        # 从 bom 表中获取 part number 对应的信息
        bom_info = ""
        for item in self.main_window.bom_data:
            if item['Part Number'] == part_number:
                for key, value in item.items():
                    bom_info += f"{key}: {value}\n"
                break
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(bom_info)
    # 将actor的颜色设置为红色
    # 将其他actor的颜色设置为白色
    # 将actor的透明度设置为1
    # 将其他actor的透明度设置为0.1
    for actor1 in self.modle_actor:
        if actor1 != self.picked_actor:
            actor1.GetProperty().SetColor(1,1,1)
            actor1.GetProperty().SetOpacity(0.1)
    self.picked_actor.GetProperty().SetColor(1,0,0)
    self.picked_actor.GetProperty().SetOpacity(1)
    # 记录这次选中的actor
    self.LastPickedActor = self.picked_actor
# 执行默认操作
self.OnLeftButtonDown()

```

```
class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        #获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        #获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

        # 获取bom表编码信息
        with open("40881.json.json", "rb") as file:
            raw_data = file.read()
            bom_encoding = chardet.detect(raw_data)['encoding']
        # 获取模型对应BOM表信息
        with open("40881.json.json", "r", encoding=bom_encoding) as file:
            self.bom_data = json.load(file)

        # 设置窗口标题和大小
        self.setWindowTitle("VTK Test")
        self.resize(3840, 2160)

        # 创建主布局
        main_layout = QtWidgets.QHBoxLayout()
        main_widget = QtWidgets.QWidget(self)
        main_widget.setLayout(main_layout)
        self.setCentralWidget(main_widget)
        # 创建当前内容部分
        current_content_widget = QtWidgets.QWidget(self)
        current_content_layout = QtWidgets.QVBoxLayout()
        current_content_widget.setLayout(current_content_layout)
        main_layout.addWidget(current_content_widget, 3)

        # 创建其他功能部分
        other_function_widget = QtWidgets.QWidget(self)
        other_function_layout = QtWidgets.QVBoxLayout()
        other_function_widget.setLayout(other_function_layout)
        main_layout.addWidget(other_function_widget, 1)

        # 创建 QTextEdit 实例
        self.text_edit = QTextEdit(self)
        self.text_edit.setReadOnly(True)
        other_function_layout.addWidget(self.text_edit)

        # 创建 QVTKRenderWindowInteractor 实例
        self.vtkWidget = QVTKRenderWindowInteractor(self)

        # 创建VTK渲染器和窗口
```

```

self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

#储存所有actor
self.actors = []

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中
current_content_layout.addWidget(self.vtkWidget)

# 创建按钮
self.button = QPushButton("Reset", self)
self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.button.clicked.connect(self.reset)
current_content_layout.addWidget(self.button)

# 创建其他功能部件并放入 other_function_layout 中

```

字典

映射

```

        # 例如: other_button = QPushButton("Other Function", self)

    def print_to_gui(self, text):
        # 在文本编辑区域中添加新的文本
        self.text_edit.append(text)

    #重置
    def reset(self):
        print("reset")
        self.text_edit.clear()
        for actor in self.actors:
            actor.GetProperty().SetColor(1,1,1)
            actor.GetProperty().SetOpacity(1)
        self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

self.vtkWidget.Initialize()
self.vtkWidget.update()

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## 6.19

8:58

```

import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication, QTextEdit
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
import vtk
import os
import numpy as np

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):
        self.render = render
        self.modle_actor = modle_actor
        self.main_window = main_window # 将 main_window 作为实例变量保存
        self.picker = vtk.vtkCellPicker()

```

```
# 记录上一次选中的actor
self.LastPickedActor = None
self.LastPickedProperty = vtk.vtkProperty()
# 添加鼠标左键按下事件
self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 清空文本框
        self.main_window.text_edit.clear()
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
            else:
                # 使用 print_to_gui 方法来显示信息
                self.main_window.print_to_gui(f'{filename}')
                # 将actor的颜色设置为红色
                # 将其他actor的颜色设置为白色
                # 将actor的透明度设置为1
                # 将其他actor的透明度设置为0.1
                for actor1 in self.modle_actor:
                    if actor1 != self.picked_actor:
                        actor1.GetProperty().SetColor(1,1,1)
                        actor1.GetProperty().SetOpacity(0.1)
                self.picked_actor.GetProperty().SetColor(1,0,0)
                self.picked_actor.GetProperty().SetOpacity(1)
                # 记录这次选中的actor
                self.LastPickedActor = self.picked_actor
        # 如果上一次选中的actor不存在
        else:
            # 使用 print_to_gui 方法来显示信息
            self.main_window.print_to_gui(f'{filename}')
            # 将actor的颜色设置为红色
            # 将其他actor的颜色设置为白色
            # 将actor的透明度设置为1
            # 将其他actor的透明度设置为0.1
            for actor1 in self.modle_actor:
                if actor1 != self.picked_actor:
```



```

        actor1.GetProperty().SetColor(1,1,1)
        actor1.GetProperty().SetOpacity(0.1)
        self.picked_actor.GetProperty().SetColor(1,0,0)
        self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
# 执行默认操作
self.OnLeftButtonDown()

class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        #获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        #获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

        # 设置窗口标题和大小
        self.setWindowTitle("VTK Test")
        self.resize(3840, 2160)

        # 创建主布局
        main_layout = QtWidgets.QHBoxLayout()
        main_widget = QtWidgets.QWidget(self)
        main_widget.setLayout(main_layout)
        self.setCentralWidget(main_widget)
        # 创建当前内容部分
        current_content_widget = QtWidgets.QWidget(self)
        current_content_layout = QtWidgets.QVBoxLayout()
        current_content_widget.setLayout(current_content_layout)
        main_layout.addWidget(current_content_widget, 3)

        # 创建其他功能部分
        other_function_widget = QtWidgets.QWidget(self)
        other_function_layout = QtWidgets.QVBoxLayout()
        other_function_widget.setLayout(other_function_layout)
        main_layout.addWidget(other_function_widget, 1)

        # 创建 QTextEdit 实例
        self.text_edit = QTextEdit(self)
        self.text_edit.setReadOnly(True)
        other_function_layout.addWidget(self.text_edit)

        # 创建 QVTKRenderWindowInteractor 实例
        self.vtkWidget = QVTKRenderWindowInteractor(self)

        # 创建VTK渲染器和窗口
        self.render = vtk.vtkRenderer()

```

```

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

#储存所有actor
self.actors = []
self.visibleActors = []

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # # 创建mapper和actor, 将actor添加到渲染器中和本地的actor列表中
        # mapper = vtk.vtkPolyDataMapper()
        # mapper.SetInputConnection(transformFilter.GetOutputPort())
        # actor = vtk.vtkActor()
        # actor.SetMapper(mapper)
        # self.actors.append(actor)
        # self.render.AddActor(actor)

        # 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename

字典
        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

映射

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中

```

```

        current_content_layout.addWidget(self.vtkWidget)

        # 创建按钮
        self.button = QPushButton("Reset", self)
        self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
        self.button.clicked.connect(self.reset)
        current_content_layout.addWidget(self.button)

        # 创建其他功能部件并放入 other_function_layout 中
        # 例如: other_button = QPushButton("Other Function", self)
        #
    def print_to_gui(self, text):
        # 在文本编辑区域中添加新的文本
        self.text_edit.append(text)

    #重置
    def reset(self):
        print("reset")
        for actor in self.actors:
            actor.GetProperty().SetColor(1,1,1)
            actor.GetProperty().SetOpacity(1)
        self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

self.vtkWidget.Initialize()
self.vtkWidget.update()

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

8: 52

```

import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication,
QTextEdit
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
import vtk
import os
import numpy as np

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):

```

```
# 初始化
def __init__(self, render, modle_actor, main_window):
    self.render = render
    self.modle_actor = modle_actor
    self.main_window = main_window # 将 main_window 作为实例变量保存
    self.picker = vtk.vtkCellPicker()
    # 记录上一次选中的actor
    self.LastPickedActor = None
    self.LastPickedProperty = vtk.vtkProperty()
    # 添加鼠标左键按下事件
    self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        # 获取选中的actor对应的STL文件名
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
            else:
                # 使用 print_to_gui 方法来显示信息
                self.main_window.print_to_gui(f'{filename}')
                # 将actor的颜色设置为红色
                # 将其他actor的颜色设置为白色
                # 将actor的透明度设置为1
                # 将其他actor的透明度设置为0.1
                for actor1 in self.modle_actor:
                    if actor1 != self.picked_actor:
                        actor1.GetProperty().SetColor(1,1,1)
                        actor1.GetProperty().SetOpacity(0.1)
                self.picked_actor.GetProperty().SetColor(1,0,0)
                self.picked_actor.GetProperty().SetOpacity(1)
                # 记录这次选中的actor
                self.LastPickedActor = self.picked_actor
        # 如果上一次选中的actor不存在
        else:
            # 使用 print_to_gui 方法来显示信息
            self.main_window.print_to_gui(f'{filename}')
            # 将actor的颜色设置为红色
            # 将其他actor的颜色设置为白色
```

```

        # 将actor的透明度设置为1
        # 将其他actor的透明度设置为0.1
        for actor1 in self.modle_actor:
            if actor1 != self.picked_actor:
                actor1.GetProperty().SetColor(1,1,1)
                actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
    # 执行默认操作
    self.OnLeftButtonDown()

class MainWindow(QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映
        # 射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        #获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        #获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

        # 设置窗口标题和大小
        self.setWindowTitle("VTK Test")
        self.resize(3840, 2160)

        # 创建主布局
        main_layout = QtWidgets.QHBoxLayout()
        main_widget = QtWidgets.QWidget(self)
        main_widget.setLayout(main_layout)
        self.setCentralWidget(main_widget)
        # 创建当前内容部分
        current_content_widget = QtWidgets.QWidget(self)
        current_content_layout = QtWidgets.QVBoxLayout()
        current_content_widget.setLayout(current_content_layout)
        main_layout.addWidget(current_content_widget, 3)

        # 创建其他功能部分
        other_function_widget = QtWidgets.QWidget(self)
        other_function_layout = QtWidgets.QVBoxLayout()
        other_function_widget.setLayout(other_function_layout)
        main_layout.addWidget(other_function_widget, 1)

        # 创建 QTextEdit 实例
        self.text_edit = QTextEdit(self)
        other_function_layout.addWidget(self.text_edit)

        # 创建 QVTKRenderWindowInteractor 实例
        self.vtkWidget = QVTKRenderWindowInteractor(self)

```

```

# 创建VTK渲染器和窗口
self.render = vtk.vtkRenderer()
self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

#储存所有actor
self.actors = []
self.visibleActors = []

#批量读取STL文件
for file_name in os.listdir(stl_dir):
    if file_name.endswith(".STL") or file_name.endswith(".stl"):
        # 获得文件路径
        file_path = os.path.join(stl_dir, file_name)

        # 创建STL读取器, 读取STL文件
        reader = vtk.vtkSTLReader()
        reader.SetFileName(file_path)
        reader.Update()

        # 创建transform以保持模型原有位置 (相对于原点)
        transform = vtk.vtkTransform()
        transformFilter = vtk.vtkTransformPolyDataFilter()
        transformFilter.SetInputConnection(reader.GetOutputPort())
        transformFilter.SetTransform(transform)
        transformFilter.Update()

        # # 创建mapper和actor, 将actor添加到渲染器中和本地的actor列表中
        # mapper = vtk.vtkPolyDataMapper()
        # mapper.SetInputConnection(transformFilter.GetOutputPort())
        # actor = vtk.vtkActor()
        # actor.SetMapper(mapper)
        # self.actors.append(actor)
        # self.render.AddActor(actor)

        # 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(transformFilter.GetOutputPort())
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        self.actors.append(actor)
        self.render.AddActor(actor)
        self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

```

字典

映射

```

self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中
current_content_layout.addWidget(self.vtkWidget)

# 创建按钮
self.button = QPushButton("Reset", self)
self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.button.clicked.connect(self.reset)
current_content_layout.addWidget(self.button)

# 创建其他功能部件并放入 other_function_layout 中
# 例如: other_button = QPushButton("Other Function", self)
#
def print_to_gui(self, text):
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

#重置
def reset(self):
    print("reset")
    for actor in self.actors:
        actor.GetProperty().SetColor(1,1,1)
        actor.GetProperty().SetOpacity(1)
    self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)

self.vtkWidget.Initialize()
self.vtkWidget.update()

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

## 6.15

```

import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QPushButton, QApplication,
QTextEdit
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor
import vtk
import os
import numpy as np

```

```

# 自定义交互器
class MyInteractorStyle(vtk.vtkInteractorStyleTrackballCamera):
    # 初始化
    def __init__(self, render, modle_actor, main_window):
        self.render = render
        self.modle_actor = modle_actor
        self.main_window = main_window # 将 main_window 作为实例变量保存
        self.picker = vtk.vtkCellPicker()
        # 记录上一次选中的actor
        self.LastPickedActor = None
        self.LastPickedProperty = vtk.vtkProperty()
        # 添加鼠标左键按下事件
        self.AddObserver("LeftButtonPressEvent", self.on_left_button_press)

# 鼠标左键按下事件
def on_left_button_press(self, obj, event):
    # 获取点击位置
    clickPos = self.GetInteractor().GetEventPosition()
    self.picker.Pick(clickPos[0], clickPos[1], 0, self.render)
    # 获取选中的actor
    self.picked_actor = self.picker.GetActor()
    # 如果有选中的actor
    if self.picked_actor:
        filename = self.main_window.actor_to_filename[self.picked_actor]
        # 使用 print_to_gui 方法来显示信息
        self.main_window.print_to_gui(f'{filename}')
        # 如果上一次选中的actor存在
        if self.LastPickedActor:
            # 如果这次选中的actor和上次选中的actor相同
            if self.LastPickedActor == self.picked_actor:
                # 将actor恢复到上次选中时的状态
                for actor1 in self.modle_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(1)
                self.LastPickedActor = None
                self.picked_actor = None
            # 如果上一次选中的actor不存在
        else:
            # 将actor的颜色设置为红色
            # 将其他actor的颜色设置为白色
            # 将actor的透明度设置为1
            # 将其他actor的透明度设置为0.1
            for actor1 in self.modle_actor:
                if actor1 != self.picked_actor:
                    actor1.GetProperty().SetColor(1,1,1)
                    actor1.GetProperty().SetOpacity(0.1)
            self.picked_actor.GetProperty().SetColor(1,0,0)
            self.picked_actor.GetProperty().SetOpacity(1)
        # 记录这次选中的actor
        self.LastPickedActor = self.picked_actor
    # 执行默认操作
    self.OnLeftButtonDown()

```



```

class MainWindow(QtWidgets.QMainWindow):
    # 初始化窗口类
    def __init__(self):
        # 创建一个新的字典self.actor_to_filename来存储每个actor和它对应的STL文件名的映射:
        self.actor_to_filename = {}
        # 调用父类的初始化方法
        super().__init__()

        #获取当前路径
        current_dir = os.path.dirname(os.path.abspath(__file__))
        #获取STL文件夹路径
        stl_dir = os.path.join(current_dir, '40881-STL')

        # 设置窗口标题和大小
        self.setWindowTitle("VTK Test")
        self.resize(3840, 2160)

        # 创建主布局
        main_layout = QtWidgets.QHBoxLayout()
        main_widget = QtWidgets.QWidget(self)
        main_widget.setLayout(main_layout)
        self.setCentralWidget(main_widget)
        # 创建当前内容部分
        current_content_widget = QtWidgets.QWidget(self)
        current_content_layout = QtWidgets.QVBoxLayout()
        current_content_widget.setLayout(current_content_layout)
        main_layout.addWidget(current_content_widget, 3)

        # 创建其他功能部分
        other_function_widget = QtWidgets.QWidget(self)
        other_function_layout = QtWidgets.QVBoxLayout()
        other_function_widget.setLayout(other_function_layout)
        main_layout.addWidget(other_function_widget, 1)

        # 创建 QTextEdit 实例
        self.text_edit = QTextEdit(self)
        other_function_layout.addWidget(self.text_edit)

        # 创建 QVTKRenderWindowInteractor 实例
        self.vtkWidget = QVTKRenderWindowInteractor(self)

        # 创建VTK渲染器和窗口
        self.render = vtk.vtkRenderer()
        self.vtkWidget.GetRenderWindow().AddRenderer(self.render)

        #储存所有actor
        self.actors = []
        self.visibleActors = []

        #批量读取STL文件
        for file_name in os.listdir(stl_dir):
            if file_name.endswith(".STL") or file_name.endswith(".stl"):

```

```

# 获得文件路径
file_path = os.path.join(stl_dir, file_name)

# 创建STL读取器, 读取STL文件
reader = vtk.vtkSTLReader()
reader.SetFileName(file_path)
reader.Update()

# 创建transform以保持模型原有位置 (相对于原点)
transform = vtk.vtkTransform()
transformFilter = vtk.vtkTransformPolyDataFilter()
transformFilter.SetInputConnection(reader.GetOutputPort())
transformFilter.SetTransform(transform)
transformFilter.Update()

# # 创建mapper和actor, 将actor添加到渲染器中和本地的actor列表中
# mapper = vtk.vtkPolyDataMapper()
# mapper.SetInputConnection(transformFilter.GetOutputPort())
# actor = vtk.vtkActor()
# actor.SetMapper(mapper)
# self.actors.append(actor)
# self.render.AddActor(actor)

# 将actor添加到渲染器和本地的actor列表中, 并保存到 actor_to_filename

mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(transformFilter.GetOutputPort())
actor = vtk.vtkActor()
actor.SetMapper(mapper)
self.actors.append(actor)
self.render.AddActor(actor)
self.actor_to_filename[actor] = file_name # 保存 actor 到文件名的

#初始化交互器
self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()

# 将当前内容部分放入 current_content_layout 中
current_content_layout.addWidget(self.vtkWidget)

# 创建按钮
self.button = QPushButton("Reset", self)
self.button.setFixedSize(80, 40) # 设置按钮的固定宽度和高度
self.button.clicked.connect(self.reset)
current_content_layout.addWidget(self.button)

# 创建其他功能部件并放入 other_function_layout 中
# 例如: other_button = QPushButton("Other Function", self)

```

字典

映射

```
#
def print_to_gui(self, text):
    # 在文本编辑区域中添加新的文本
    self.text_edit.append(text)

#重置
def reset(self):
    print("reset")
    for actor in self.actors:
        actor.GetProperty().SetColor(1,1,1)
        actor.GetProperty().SetOpacity(1)
    self.interactorStyle = MyInteractorStyle(self.render, self.actors, self)

self.vtkWidget.GetRenderWindow().GetInteractor().SetInteractorStyle(self.interactorStyle)
self.vtkWidget.Initialize()
self.vtkWidget.update()

def main():
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```