浙江大学

面向对象程序设计

大 程 序 报 告

大程名称：GeometryTool

姓名 ： xxx

学号： xxxxxxxxxx

电话： xxxxxxxxxx

指导老师： 李际军

2021~2022 春夏学期 2022 年 6 月 25 日

目 录

GeometryTool 大程序设计

1 大程序简介

1.1 选题背景及意义

计算机图形学的核心目标在于创建有效的视觉交流。在科学领域，图形学可以将科学成果通过可视化的方式展示给公众；在娱乐领域，如在 PC 游戏、手机游戏、3D 电影与电影特效中，计算机图形学发挥着越来越重要的作用；在创意或艺术创作、商业广告、产品设计等行业，图形学也起着重要的基础作用。而在科学领域中，这一点是在 1987 年关于科学计算可视报告中才被重点提出。该报告引用了 Richard Hamming 在 1962 年的经典论断：“计算的目的是洞察事物的本质，而不是获得数字。”报告中提到了计算机图形学在帮助人脑从图形图像的角度理解事物本质的重要作用，因为图形图像比单纯数字具有更强的洞察力。

在计算机图形学的基础上，最直接的就是对于 2D 平面上图形的绘制内容。为此，我们构建一个 GeometryTool（“几何画板”）。GeometryTool 是一个作图的辅助软件，适用于数学、平面几何、物理的矢量分析、作图，函数作图的动态几何工具。

1.2 目标要求

- 实现一个基本的二维坐标轴，基于坐标轴实现任意点的二维坐标表示。并且支持拖动、缩放等功能。
- 通过特定输入限制实现点的插入
- 通过特定输入限制实现线的插入
- 通过特定输入限制实现三角形的插入
- 通过特定输入限制实现复杂图形的插入
- 对于上述物体，实时显示各个物体的具体位置信息
- 通过鼠标互动，实现上述物体的移动、缩放
- 在鼠标互动及任意时刻，实现上述物体信息的实时更新
- 实现上述物体的删除
- 设计用户引导说明
- 设计窗口和各个窗体模块设计
- 设计良好的用户接口和简单有效的图标设计

1.3 开发工具介绍

本程序使用 C++语言开发，其中调用到的一些开发工具如下

1.3.1 Qt

Qt 是一个 1991 年由 Qt Company 开发的跨平台 C++图形用户界面应用程序开发框架。它既可以开发 GUI 程序，也可用于开发非 GUI 程序，比如控制台工具和服务器。Qt 是面向对象的框架，使用特殊的代码生成扩展（称为元对象编译器(Meta Object Compiler)）以及一些宏，Qt 很容易扩展，并且允许真正地组件编程。

Qt 的良好封装机制使得 Qt 的模块化程度非常高，可重用性较好，对于用户开发来说是非常方便的。Qt 提供了一种称为 signals/slots 的安全类型来替代 callback，这使得各个元件之间的协同工作变得十分简单。

Qt Designer 被称为 Qt 设计师，用于设计和构建图形用户界面(Qt Widgets)。你可以组合和自定义窗口或对话框（所见即所得），并使用不同的风格和分辨率进行测试。用 Qt Designer 创建的窗口部件和表格无缝集成编程代码，采用 Qt 信号和槽机制，这样就可以轻松地分配图形元素的行为。在 Qt 设计师中设置的所有属性可以动态地在代码中进行更改。此外，类似插件推广和自定义插件功能，可以使用自己的组件来使用 Qt Designer

1.3.2 QtCreator

Qt Creator 是跨平台的 Qt IDE， Qt Creator 是 Qt 被 Nokia 收购后推出的一款新的轻量级集成开发环境（IDE）。此 IDE 能够跨平台运行，支持的系统包括 Linux（32 位及 64 位）、Mac OS X 以及 Windows。根据官方描述，Qt Creator 的设计目标是使开发人员能够利用 Qt 这个应用程序框架更加快速及轻易的完成开发任务。

1.3.3 OpenGL

OpenGL（英语：Open Graphics Library，译名：开放图形库或者“开放式图形库”）是用于渲染 2D、3D 矢量图形的跨语言、跨平台的应用程序编程接口（API）。这个接口由近 350 个不同的函数调用组成，用来绘制从简单的图形比特到复杂的三维景象。而另一种程序接口系统是仅用于 Microsoft Windows 上的 Direct3D。

OpenGL 常用于 CAD、虚拟现实、科学可视化程序和电子游戏开发。OpenGL 的高效实现（利用了图形加速硬件）存在于 Windows，部分 UNIX 平台和 Mac OS。这些实现一般由显示设备厂商提供，而且非常依赖于该厂商提供的硬件。

当今，OpenGL 是视频行业领域中用于处理 2D/3D 图形的最为广泛接纳的 API，在此基础上，为了用于计算机视觉技术的研究，从而催生了各种计算机平台上的应用功能以及设备上的许多应用程序。其是独立于视窗操作系统以及操作系统平台，可以进行多种不同邻域的开发和内容创作，简而言之，其帮助研发人员能够实现 PC、工作站、超级计算机以及各种工控机等硬件设备上实现高性能、对于视觉要求极高的高视觉图形处理软件的开发。

1.3.4 Qt & OpenGL

OpenGL 只处理与 3D 图形的绘制，基本上不提供创建用户界面的功能，所以为 OpenGL 应用程序创建用户界面必须使用其它的图形工其包（如 Motif、MFC 等）。而 Qt 的 OpenGL 模块很好的解决了这个问题，它提供了一个继承自 Qwidget 的 OpenGL 部件类 QGLWidget，使得该部件类能够像 Qt 其它部件那样使用，还可以在绘制窗口部件时直接使用 OpenGL 的 API 接口。在 Qt 中为 OpenGL 提供支持的类主要有以下几个：

1. QGLWidget：用于渲染 OpenGL 场景的易于使用的 Qt 部件。
2. QGLColormap：用于在 QGLWidget 中安装用户自定义的颜色图。
3. QGLContext：封装了用于 OpenGL 渲染的场景。
4. QGLFormat：指定 OpenGL 演染场景的显示模式。
5. QGLFrameBufferObject 和 QGLPixelBuffer 分别提供了对 GL 帧缓冲对象和 GL 像素缓冲的支持。
6. QGLPaintEngine：QPaintEngine 的派生类，为 QPainter 提供了 OpenGL 绘图引擎。

实际上，在进行二维图形的绘制中，完全可以不使用 OpenGL 的原生接口而使用更为方便的 QPainter。QPainter 只是个 API，实际的渲染是由 QPaintEngine 来执行的。Qt 自带有 QGLPaintEngine，为 QPainter 提供了 OpenGL 后台。在 QGLWidget 基础上创建的 QPainter，自动将 QGLPaintEngine 作为绘画引擎。这样，任何使用此 QPainter 类型绘制的 2D 图形都会自动解析至 OpenGL，间接以 GL 的原生接口来实现二维图形的绘制。使用 QPainter 而不是 OpenGL 命令的优势是可轻易将渲染在平台默认引擎和 OpenGL 之间切换。这样不仅可以充分利用二者各自的优势，同时还可以直接使用 Qt 二维绘图系统所提供的众多高级功能。

Qt 提供了 QtOpenGL 模块来支持三维图形功能的 OpenGL 模块。在使用 QtOpengl 模块时，必须在对应的工程文件（*.pro）中加入:

QT += opengl

来声明使用 QtOpengl 模块，以便 qmake 在利用*.pro 生成编译器所需的 makefile 时添加所需的 opengl 库。

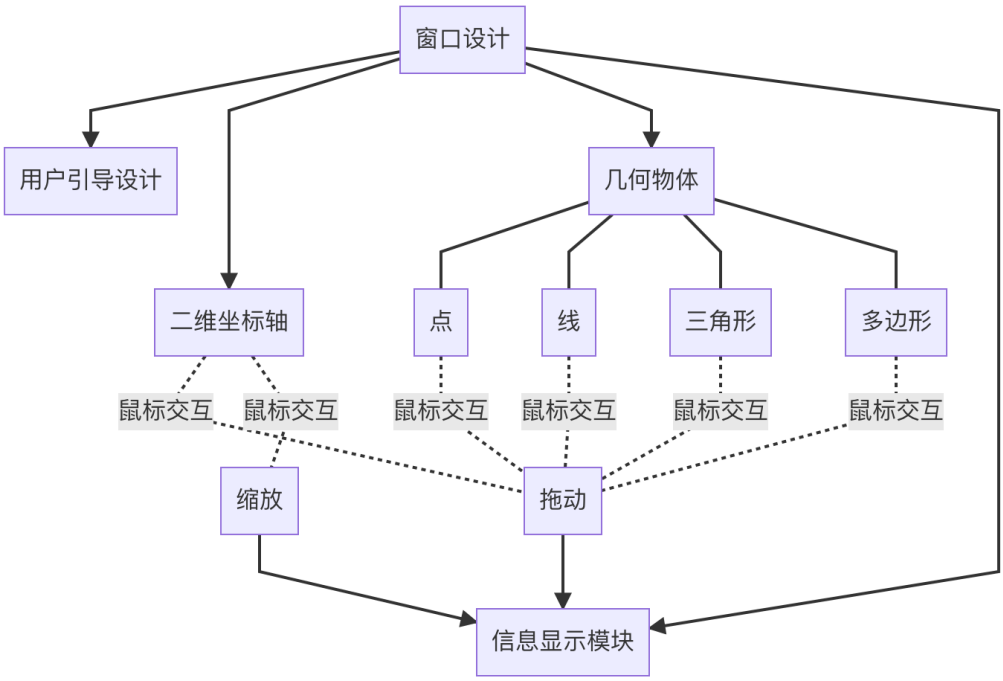
2 需求分析

2.1 业务需求

本程序目标构建一个简单的几何画板类软件，满足基本的画图功能，支持用户自定义增加几何图形、删除几何图形、修改几何图形、查看几何图形信息等等。用户界面有一定的美观性和易用性，用户交互设计合理有效，程序运行正常。

2.2 功能需求

程序整体功能模块分析如下：



- 实现一个基本的二维坐标轴，基于坐标轴实现任意点的二维坐标表示。并且支持拖动、缩放等功能。
- 通过特定输入限制实现点的插入
- 通过特定输入限制实现线的插入
- 通过特定输入限制实现三角形的插入
- 通过特定输入限制实现复杂图形的插入
- 对于上述物体，实时显示各个物体的具体位置信息
- 通过鼠标互动，实现上述物体的移动、缩放

- 在鼠标互动及任意时刻，实现上述物体信息的实时更新
- 实现上述物体的删除
- 设计用户引导说明
- 设计窗口和各个窗体模块设计
- 设计良好的用户接口和简单有效的图标设计

2.3 数据需求

本大程使用 c++构建，应当充分利用文件程序、类体系设计（含抽象、封装、继承、多态等特点，具有较好的代码复用性、代码简洁且清晰，类的定义明确。

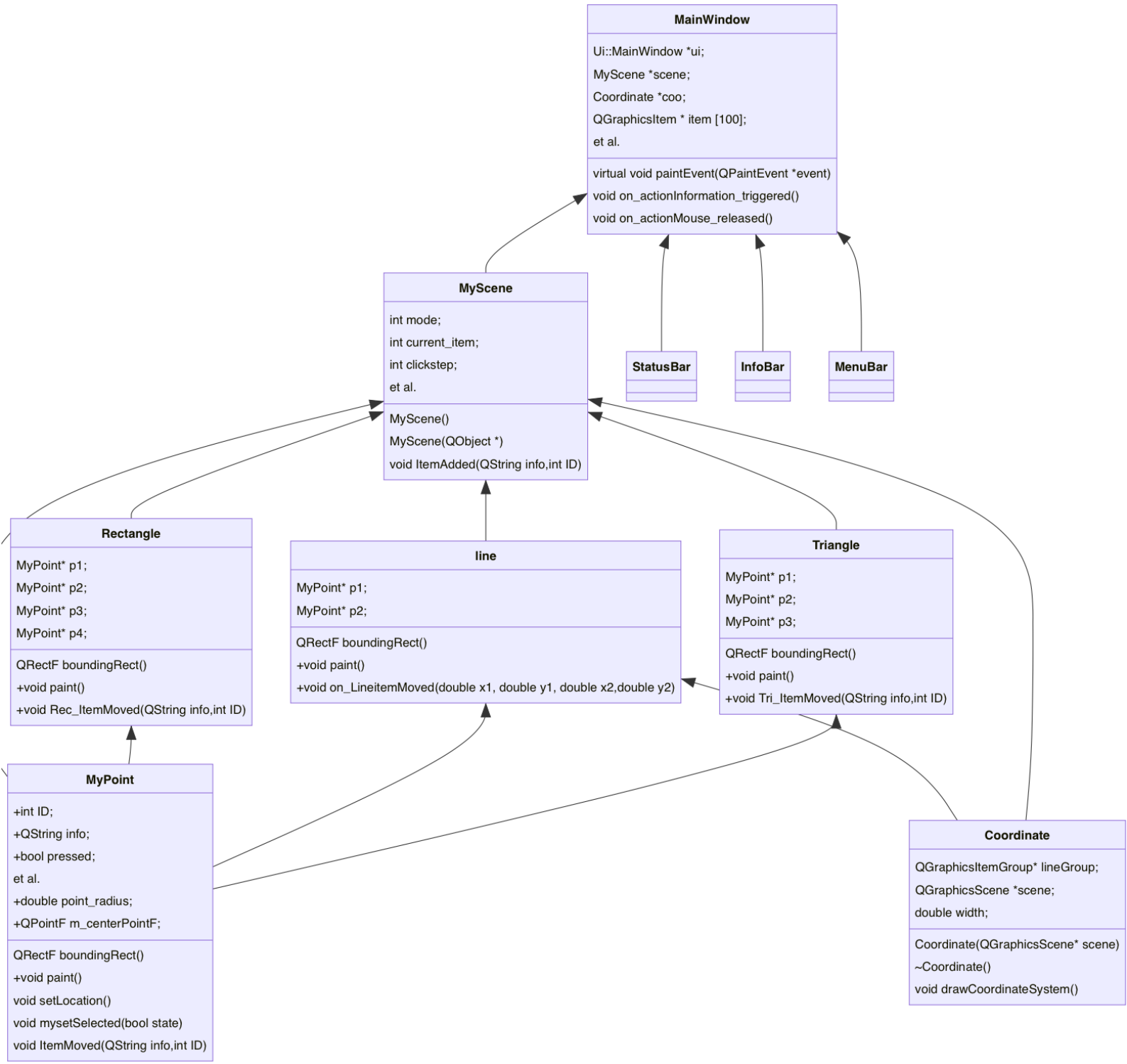
2.4 性能需求

在实现上，要求程序同时支持大量几何物体的显示，同时支持线性时间内的物体增删改查功能。程序具有较好的鲁棒性，在完整的正常使用过程中不会崩溃。

3 类库功能分析

3.1 总体架构设计

关于工程的总体架构设计功能，先给出设计总体架构图，以便理解：



3.2 基于模版的类的设计

3.2.1 MainWindow 类

程序的整体框架囊括在 MainWindow 类的实现中。入口程序 main 函数调用 MainWindow 设计。

```
1. #include "mainwindow.h"
2.
3. #include <QApplication>
4.
5. int main(int argc, char *argv[])
6. {
7.     QApplication a(argc, argv);
8.     MainWindow w;
9.     w.show();
10.    return a.exec();
11.}
```

其中，MainWindow 类的模版由 Qt 给出，继承了 QMainWindow 类，重要的模块包括：

- ui：基于 QtDesginer 设计模块的 xml 语法文件，规定了主窗口的各个模块的图形位置
- scene：一个自定义的画板实例(MyScene 类)
- coo：一个自定义的坐标轴实例(Coordinate 类)
- item：一个自定义的图形实例的数组，上限支持同时存在 100 个物体

其中，当收到画板类传来的消息的时候，需要调用一下函数来完成右侧信息栏的更新

```
1.     void on_actionAddObject(QString info,int ID);
2.     void on_actionMoveObject(QString info,int ID);
```

完整类的定义如下：

```
1. class MainWindow : public QMainWindow
2. {
3.     Q_OBJECT
4.
5. public:
6.     MainWindow(QWidget *parent = nullptr);
7.     ~MainWindow();
8.     virtual void paintEvent(QPaintEvent *event);
9.     void writeInfo();
```

```
10.
11.private slots:
12.    void on_actionInformation_triggered();
13.    void on_actionMouse_released();
14.    void on_actionPoint_triggered();
15.    void on_actionSelect_triggered();
16.    void on_actionLine_triggered();
17.    void on_actionTriangle_triggered();
18.    void on_actionCircle_triggered();
19.    void on_actionRectangle_triggered();
20.    void on_actionAddObject(QString info,int ID);
21.    void on_actionMoveObject(QString info,int ID);
22.    void on_actionclear_triggered();
23.
24.
25.private:
26.    Ui::MainWindow *ui;
27.    MyScene *scene;
28.    Coordinate *coo;
29.    QGraphicsItem * item [100];
30.
31.};
```

其中，`ui`的 xml 语句规定如下，

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <ui version="4.0">
3.  <class>MainWindow</class>
4.  <widget class="QMainWindow" name="MainWindow">
5.   <property name="geometry">
6.    <rect>
7.     <x>0</x>
8.     <y>0</y>
9.     <width>1000</width>
10.    <height>1000</height>
11.   </rect>
12.  </property>
13.  <property name="windowTitle">
14.   <string>MainWindow</string>
15.  </property>
16.  <widget class="QWidget" name="centralwidget">
17.   <layout class="QGridLayout" name="gridLayout">
18.    <item row="0" column="0">
19.     <layout class="QHBoxLayout" name="horizontalLayout" st
```

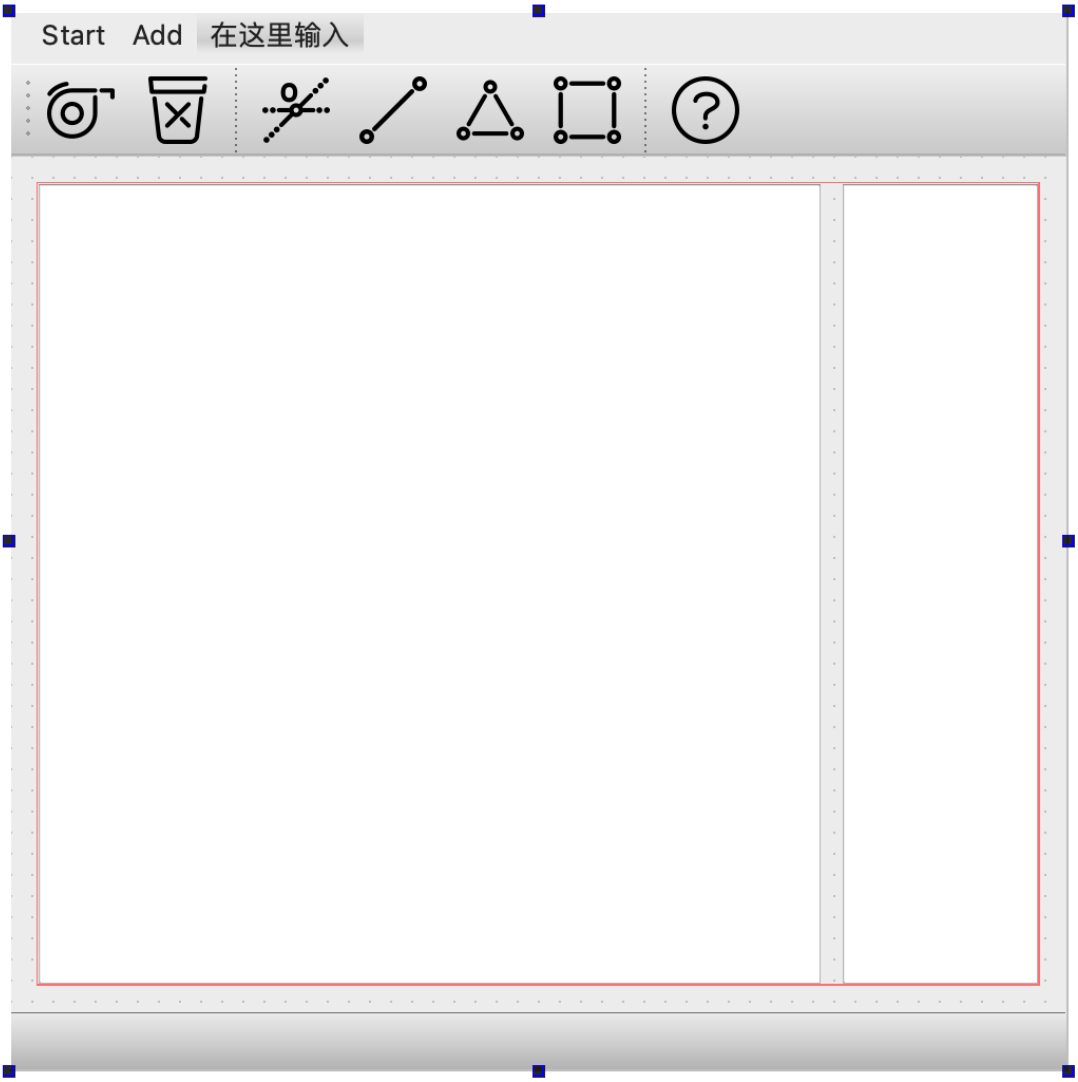
```
   retch="8,2">
20.       <item>
21.        <widget class="QGraphicsView" name="graphicsView"/>
22.       </item>
23.       <item>
24.        <widget class="QListWidget" name="listWidget"/>
25.       </item>
26.      </layout>
27.     </item>
28.    </layout>
29.   </widget>
30.   <widget class="QMenuBar" name="menubar">
31.    <property name="geometry">
32.     <rect>
33.      <x>0</x>
34.      <y>0</y>
35.      <width>1000</width>
36.      <height>24</height>
37.     </rect>
38.    </property>
39.    <widget class="QMenu" name="menuSelect">
40.     <property name="title">
41.      <string>Start</string>
42.     </property>
43.     <addaction name="actionSelect"/>
44.     <addaction name="actionclear"/>
45.     <addaction name="actionInformation"/>
46.    </widget>
47.    <widget class="QMenu" name="menuPoint">
48.     <property name="title">
49.      <string>Add</string>
50.     </property>
51.     <addaction name="actionPoint"/>
52.     <addaction name="actionLine"/>
53.     <addaction name="actionTriangle"/>
54.     <addaction name="actionRectangle"/>
55.    </widget>
56.    <addaction name="menuSelect"/>
57.    <addaction name="menuPoint"/>
58.   </widget>
59.   <widget class="QStatusBar" name="statusbar"/>
60.   <widget class="QToolBar" name="toolBar">
61.    <property name="windowTitle">
62.     <string>toolBar</string>
```

```
63.    </property>
64.    <attribute name="toolBarArea">
65.     <enum>TopToolBarArea</enum>
66.    </attribute>
67.    <attribute name="toolBarBreak">
68.     <bool>false</bool>
69.    </attribute>
70.    <addaction name="actionSelect"/>
71.    <addaction name="actionclear"/>
72.    <addaction name="separator"/>
73.    <addaction name="actionPoint"/>
74.    <addaction name="actionLine"/>
75.    <addaction name="actionTriangle"/>
76.    <addaction name="actionRectangle"/>
77.    <addaction name="separator"/>
78.    <addaction name="actionInformation"/>
79.   </widget>
80.   <action name="actionSelect">
81.    <property name="icon">
82.     <iconset resource="icon.qrc">
83.      <normaloff>:/png/010-measuring-tape.png</normaloff>:/p
   ng/010-measuring-tape.png</iconset>
84.    </property>
85.    <property name="text">
86.     <string>Select</string>
87.    </property>
88.    <property name="shortcut">
89.     <string>Meta+S</string>
90.    </property>
91.   </action>
92.   <action name="actionPoint">
93.    <property name="icon">
94.     <iconset resource="icon.qrc">
95.      <normaloff>:/png/intersection.png</normaloff>:/png/int
   ersection.png</iconset>
96.    </property>
97.    <property name="text">
98.     <string>Point</string>
99.    </property>
100.     <property name="shortcut">
101.      <string>Meta+P</string>
102.     </property>
103.    </action>
104.    <action name="actionLine">
```

```
105.     <property name="icon">
106.      <iconset resource="icon.qrc">
107.       <normaloff>:/png/005-segment.png</normaloff>:/png/00
   5-segment.png</iconset>
108.     </property>
109.     <property name="text">
110.      <string>Line</string>
111.     </property>
112.     <property name="shortcut">
113.      <string>Meta+L</string>
114.     </property>
115.    </action>
116.    <action name="actionTriangle">
117.     <property name="icon">
118.      <iconset resource="icon.qrc">
119.       <normaloff>:/png/007-triangle-1.png</normaloff>:/png
   /007-triangle-1.png</iconset>
120.     </property>
121.     <property name="text">
122.      <string>Triangle</string>
123.     </property>
124.     <property name="shortcut">
125.      <string>Meta+T</string>
126.     </property>
127.    </action>
128.    <action name="actionRectangle">
129.     <property name="icon">
130.      <iconset resource="icon.qrc">
131.       <normaloff>:/png/006-square-1.png</normaloff>:/png/0
   06-square-1.png</iconset>
132.     </property>
133.     <property name="text">
134.      <string>Rectangle</string>
135.     </property>
136.     <property name="shortcut">
137.      <string>Meta+R</string>
138.     </property>
139.    </action>
140.    <action name="actionInformation">
141.     <property name="icon">
142.      <iconset resource="icon.qrc">
143.       <normaloff>:/png/008-question.png</normaloff>:/png/0
   08-question.png</iconset>
144.     </property>
```

```
145.     <property name="text">
146.      <string>Information</string>
147.     </property>
148.     <property name="shortcut">
149.      <string>Meta+I</string>
150.     </property>
151.    </action>
152.    <action name="actionclear">
153.     <property name="icon">
154.      <iconset resource="icon.qrc">
155.       <normaloff>:/png/trash.png</normaloff>:/png/trash.pn
   g</iconset>
156.     </property>
157.     <property name="text">
158.      <string>clear</string>
159.     </property>
160.     <property name="shortcut">
161.      <string>Meta+D</string>
162.     </property>
163.    </action>
164.   </widget>
165.   <resources>
166.    <include location="icon.qrc"/>
167.   </resources>
168.   <connections/>
169.  </ui>
```

上述语句实际构建的主窗口形态示意图



3.3 更新类设计

3.3.1 MyScene 类

`MyScene` 继承了 `QGraphicsScene` 类，提供了一些基本的画图板操作功能。其中重要的说明如下：

`mode` :标识了程序目前所处的状态，比如 0 是选择模式，1 是点模式，2 是线模式，3 是三角形模式，4 是多边形模式等等。

`current_item` :目前画板中所有的图形的个数

`clickstep` ： 一个辅助变量。在用户点击的时候，有时候需要连续若干次点击才算是一次完整的步骤，这个时候就需要这个变量来记录用户点击的顺序次数。

用户交互的重载函数：

```
1.      void mousePressEvent(QGraphicsSceneMouseEvent *event);
```

```
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event);
    // Backspace 键移除 item
    void keyPressEvent(QKeyEvent *event);
```

如果用户输入了点击、移动、按键等输入信号，就会调用上述重载的函数完成操作。

在上述三类函数中，很明显与每一个画板中的物体进行交互后，需要给出信息，才能满足物体信息随着用户的输入进行更新，因此在上述三个函数中会给出以下的一些信号，接受者是 MainWindow 类

```
   void ItemAdded(QString info,int ID);
    void Scene_ItemMoved(QString info,int ID);
    void Scene_ItemDeleted(int ID);
```

在每一个物体进行了交互之后，也需要发出信号让 MyScene 的实例进行更新，因此有接受槽

```
private slots:
    void on_itemMoved(QString info,int ID);
```

完整的类的定义如下：

```
class MyScene : public QGraphicsScene
{
    Q_OBJECT
public:
    MyScene();
    MyScene(QObject *);
    int mode;
    int current_item;
    int clickstep;
    /*
     * mode = 0 选择
     * mode = 1 加点
     * mode = 2 加线
     * 等等
     *
     */

signals:
    void ItemAdded(QString info,int ID);
    void Scene_ItemMoved(QString info,int ID);
    void Scene_ItemDeleted(int ID);

protected:
    // 左键：添加item  右键：移除item
    void mousePressEvent(QGraphicsSceneMouseEvent *event);
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event);
    // Backspace 键移除 item
    void keyPressEvent(QKeyEvent *event);
```

```
29.     void paintSelected();
30.
31.private slots:
32.     void on_itemMoved(QString info,int ID);
33.};
34.
```

3.3.2 Coordinate 类

Coordinate 类的设计没有继承，也比较简单。在程序初始化的时候画出一个坐标轴即可。包括：

- `drawCoordinateSystem()`函数，画出坐标轴
- `lineGroup`，实质是多个线的组合，注意！这里不只是 x 轴和 y 轴的两条线，也包括了刻度标注的小线！
- `scene`，指向存在于中的画板
- `width`， 定义了坐标轴的宽度

```
1. class Coordinate
2. {
3. public:
4.     Coordinate(QGraphicsScene* scene);
5.     ~Coordinate();
6.
7.
8. private:
9.
10.     void drawCoordinateSystem();
11.     QGraphicsItemGroup* lineGroup;
12.
13.     QGraphicsScene *scene;
14.     double width;
15.
16.};
```

3.4 基于 QGraphicsItem 的类的设计

3.4.1 MyPoint 类

继承了 `QObject` 和 `QGraphicsItem`，以方便定义和不同窗口的信息交互
一些重要的成员定义如下：

- `m_centerPointF`： 标识了点的中心具体位置信息
- `ID`：在整个画板的物体序列中的位置
- `point_radius` ： 画点的半径
- `pressed` ： 是否正在被按下

重载了以下函数：

```
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;
    void keyPressEvent(QKeyEvent *event) override;
```

用来实现用户的鼠标介入和交互

整体类的定义如下

```
class MyPoint : public QObject, public QGraphicsItem
{
    Q_OBJECT
public:
    MyPoint();
    ~MyPoint();

    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget = nullptr) override;
    void setLocation();
    double location_x;
    double location_y;
    void mysetSelected(bool state);
    //StatusWidget * infoWidget;
    //QWidget * infoWidget;
    int ID;
    QString info;
    bool pressed;
    int in_line;
    QColor color;

    // 0 -> single point
    // 1 -> in line;
```

```
    // 2 -> in triangle
    QGraphicsEllipseItem point_circle;

    line* line_parent;
    Triangle * tri_parent;
    Rectangle * rec_parent;

signals:
    void ItemMoved(QString info,int ID);
    void LineItemMoved(double x1, double y1, double x2,double y2);

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;
    void keyPressEvent(QKeyEvent *event) override;

private:

    bool m_bResizing;
    double point_radius;
    QPointF m_centerPointF;
};
```

3.4.2 line 类

同上，在 line 的设计中包括了两个上述的 MyPoint 的实例，其他具体内容同理

```
class line : public QObject, public QGraphicsItem{
    Q_OBJECT
public:
    MyPoint* p1;
    MyPoint* p2;
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget = nullptr) override;

```

```
    double x1;
    double y1;
    double x2;
    double y2;

    int ID;
    QString info;
    bool pressed;

signals:
    void Line_ItemMoved(QString info,int ID);


protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;


private slots:
    void on_LineitemMoved(double x1, double y1, double x2,double y2);
};
```

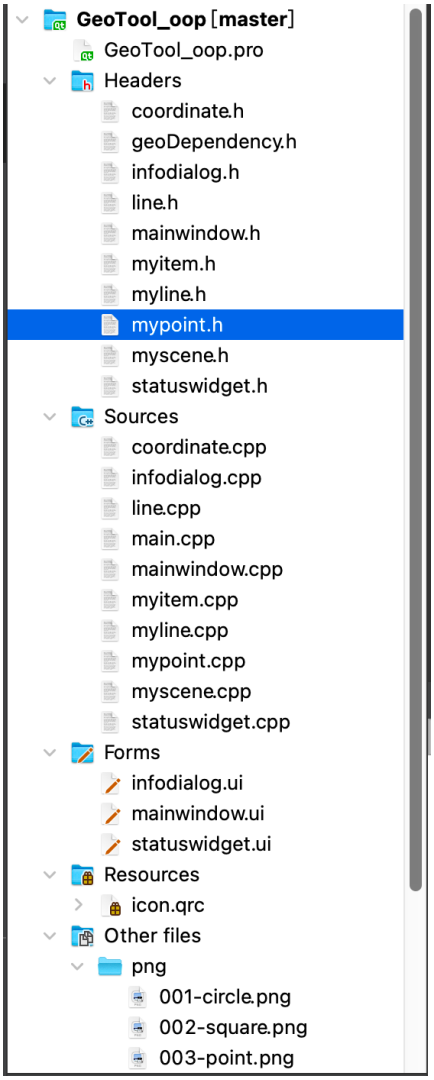
3.4.3 Triangle 类，Rectangle 类等

因为这几个类实现机制和 line 类相同，在此就不再赘述。仅展示 Triangle 类的设计

```
class Triangle : public QObject, public QGraphicsItem{
    Q_OBJECT
public:
    MyPoint* p1;
    MyPoint* p2;
    MyPoint* p3;
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget = nullptr) override;

```

```
    double x1;
    double y1;
    double x2;
    double y2;
    double x3;
    double y3;

    int ID;
    QString info;
    bool pressed;

signals:
    void Tri_ItemMoved(QString info,int ID);


protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;

};
```

3.5 源代码文件组织设计



3.5.1 文件结构

在根目录文件中包含的文件如下

- /Headers ： 包含了所有头文件内容 （.h）
- /Sources ： 包含了所有实际的代码实现内容(.cpp)
- GeoTool_oop.pro : Qt Creator 工程文件
- /Forms : xml 语言文件，包含了窗体的设计
- /Resources ： 所有属性 icon 的设计文件管理信息
- /Other\ files/png: 所有属性 icon 的实际内容位置

3.5.2 多文件构成机制

以 MyScene.h 设计为例，MyScene 中包含了 class MyScene 的定义，而在 MyScene 的头尾包括了：

```
#ifndef MYSCENE_H
#define MYSCENE_H

#include "geoDependency.h"

/* …… */


#endif // MYSCENE_H
```

为了避免每一个头文件的 include 定义太过冗余，在设计的时候专门开辟了一个 geoDependency.h 文件，用来包括 include 所有的内容，这样就可以节省一些代码空间

```
#ifndef GEODEPENDENCY_H
#define GEODEPENDENCY_H

#include <QPainter>
#include <QTextDocument>
#include <QHBoxLayout>
#include <QDialog>
#include <QtCore>
#include <QtGui>
#include <QGraphicsScene>
#include <QGraphicsItem>
#include <QGraphicsItemGroup>
#include <QDebug>
#include "myitem.h"
#include "coordinate.h"
#include "infodialog.h"
#include "mypoint.h"
#include <QGraphicsSceneMouseEvent>
#include "myscene.h"
#include "statuswidget.h"
//#include "mainwindow.h"
#include <QListWidgetItem>
#include "myline.h"
//#include "line.h"
#include <QLabel>
```

```
26.
27.#endif // GEODEPENDENCY_H
```

3.6 关键函数设计描述

鉴于本程序设计了函数很多，仅挑选几个具有代表性的作为展示：

3.6.1 实例 1:MainWindow 类的构造函数和析构函数

函数原型：

```
1. MainWindow::MainWindow(QWidget *parent);
```

功能描述：MainWindow 类的构造函数

函数内容：新建了界面的主体框架，初始化了窗体内容，并且利用 Qt 的 connect 机制将 scene 中的信号和信息栏中的信息变化联系在了一起。

函数代码：

```
1. MainWindow::MainWindow(QWidget *parent)
2.     : QMainWindow(parent)
3.     , ui(new Ui::MainWindow)
4. {
5.     ui->setupUi(this);
6.     //ui->horizontalLayout
7.     setWindowTitle("Geometry Tool");
8.
9.     ui->menubar->setNativeMenuBar(false);
10.
11.
12.    scene = new MyScene(this);
13.    ui->graphicsView->setScene(scene);
14.
15.    coo = new Coordinate(scene);
16.    scene->mode = 0;
17.
18.    ui->statusbar->showMessage(tr("current mode is : select
   "));
19.
20.
```

```
21.     QLabel *permanent = new QLabel(this);
22.     permanent->setFrameStyle(QFrame::Box | QFrame::Sunken);
23.     permanent->setText(
24.       tr("<a href=\"https://www.math10.com/en/geometry/geogebra/geogebra.html\">geogebra</a>"));
25.     permanent->setTextFormat(Qt::RichText);
26.     permanent->setOpenExternalLinks(true);
27.     ui->statusbar->addPermanentWidget(permanent);
28.
29.     //获取系统现在的时间
30.//     QDateTime time = QDateTime::currentDateTime();
31.//     //设置系统时间显示格式
32.//     QString str = time.toString("yyyy-MM-dd hh:mm:ss dddd");
33.//     //在标签上显示时间
34.//     QLabel *timelabel = new QLabel(this);
35.//     ui->statusbar->showMessage(str);
36.
37.     //ui->scrollArea->setWidget(x);
38.     //ui->scrollArea->addScrollBarWidget(x,Qt::AlignTop);
39.
40.     //item[0] = new MyItem();
41.     //item[1] = new MyPoint();
42.
43.
44.
45.     //scene->addItem(item[0]);
46.     //scene->addItem(item[1]);
47.
48.     QBrush redBrush(Qt::red);
49.     QBrush blueBrush(Qt::blue);
50.     QPen blackpen(Qt::black);
51.
52.     blackpen.setWidth(6);
53.
54.     //ellpise = scene->addEllipse(10,10,100,100,blackpen,redBrush);
55.     //rect = scene->addRect(-100,-100,50,50,blackpen,blueBrush);
56.
57.     //rect->setFlag(QGraphicsItem::ItemIsMovable);
58.
59.     scene->clickstep = 0;
60.
```

```
61.     connect(scene,SIGNAL(ItemAdded(QString,int)),this,SLOT(
    on_actionAddObject(QString,int)));
62.     connect(scene,SIGNAL(Scene_ItemMoved(QString,int)),this
    ,SLOT(on_actionMoveObject(QString,int)));
63.
64.}
65.
66.MainWindow::~MainWindow()
67.{
68.     delete ui;
69.}
```

3.6.2 实例 2:MainWindow 类点击菜单选择 icon 响应函数

函数原型：

```
void MainWindow::on_actionSelect_triggered();
```

功能描述：MainWindow 类点击菜单选择 icon 响应函数

函数内容： 首先将全局变量模式 mode 设为选择（0）， 并且在状态栏上更新内容。同时在 debug console 输出信息以便纠错。响应的机制通过在 MainWindow 类中的 connect 将信号和槽连接在了一起

函数代码：

```
1. void MainWindow::on_actionSelect_triggered()
2. {
3.     ui->statusbar->showMessage(tr("current mode is : select
    "));
4.     scene->mode = 0;
5.     qDebug()<<"scene mode selection ";
6. }
```

3.6.3 实例 3:MainWindow 类的添加物体更新信息栏响应函数

函数原型：

```
void MainWindow::on_actionAddObject(QString info,int ID);
```

功能描述：MainWindow 类的添加物体更新信息栏响应函数

函数内容：

输入信息： 物体状态，物体编号

在 debug console 输出信息以便纠错。

在右侧信息栏的 QListWidgetItem 中按照一定格式将物体信息展示

函数代码：

```
1. void MainWindow::on_actionAddObject(QString info,int ID)
2. {
3.
4.     qDebug()<<"received item Added, ID = " + QString::numbe
   r(ID) +" current row = " + QString::number(ui->listWidget->
   currentRow());;
5.     QListWidgetItem* infoItem = new QListWidgetItem;
6.
7.     //StatusWidget* infoWidget = new StatusWidget();
8.
9.     infoItem->setText(info);
10.
11.    ui->listWidget->insertItem(ID+1, infoItem);
12.    //ui->listWidget->addScrollBarWidget(infoWidget,Qt::Ali
   gnTop);
13.    ui->listWidget->setVisible(true);
14.
15.}
```

3.6.4 实例 4:Coordinate 类的绘制坐标轴函数

函数原型：

```
void Coordinate::drawCoordinateSystem();
```

功能描述：Coordinate 类的绘制坐标轴函数

函数内容：

将 x 轴、y 轴两条线以及轴上的坐标标轴同时加到图形组合之中，在分别显示各个内容。

函数代码：

```
1. void Coordinate::drawCoordinateSystem()
2. {
3.
4.     lineGroup = new QGraphicsItemGroup();
5.
6.     QGraphicsLineItem* oy = new QGraphicsLineItem(-width/2,
   0,width/2,0);
7.     QGraphicsLineItem* ox = new QGraphicsLineItem(0,-width/
   2,0,width/2);
8.
9.     QPen pen(Qt::black);
10.    ox->setPen(pen);
11.    ox->setZValue(-2000);
12.
13.    oy->setPen(pen);
14.    oy->setZValue(-2000);
15.
16.
17.    for(int x=-width/2;x<=width/2;x+=width/40)
18.    {
19.        QGraphicsLineItem* item = new QGraphicsLineItem(x,-
   2,x,+2);
20.        item->setPen(pen);
21.        item->setZValue(-2000);
22.        lineGroup->addToGroup(item);
23.    }
24.    for(int y=-width/2;y<=width/2;y+=width/40)
25.    {
26.        QGraphicsLineItem* item = new QGraphicsLineItem(-2,
   y,+2,y);
27.        item->setPen(pen);
28.        item->setZValue(-2000);
29.        lineGroup->addToGroup(item);
30.    }
31.
32.    lineGroup->addToGroup(ox);
33.    lineGroup->addToGroup(oy);
34.
35.    lineGroup->setVisible(true);
36.    lineGroup->setFlag(QGraphicsItem::ItemIgnoresTransforma
   tions);
37.    this->scene->addItem(lineGroup);
```

```
38.}
```

3.6.5 实例 5:MyScene 类的鼠标点击响应函数

函数原型：

```
void MyScene::mousePressEvent(QGraphicsSceneMouseEvent *event)
```

功能描述：MyScene 类的鼠标点击相应函数

函数内容：
这个函数大概是我写的最久，可能也是这个工程里面最长的一个函数，因为我对每一个不同的点的所属类别内容都做了区分。
当选择不同的模式的时候，函数体对于同样的鼠标点击都需要做出不同的反应，同时还需要加上区分鼠标是否是点在点所认为的范围内，或者是在点的判定范围之外等等，因此比较复杂。

函数代码：

```
1. void MyScene::mousePressEvent(QGraphicsSceneMouseEvent *event)
2. {
3.
4.     qDebug() << "Custom scene clicked.";
5.     QGraphicsScene::mousePressEvent(event);
6.          if (!event->isAccepted()) {
7.              if(mode == 1){ // is add point mode
8.                  if (event->button() == Qt::LeftButton) {
9.                      // 在 Scene 上添加一个自定义 item
10.                       QPointF point = event->scenePos();
11.                       MyPoint *item = new MyPoint();
12.                       item->location_x = point.x();
13.                       item->location_y = point.y();
14.                       item->setLocation();
15.                       item->in_line = 0;
16.                       addItem(item);
17.                       item->info = "point: (" + QString::number(item->location_
    x) + "," + QString::number(-item->location_y) +  ")";
18.
19.                       item->ID = current_item++;
```

```
20.
21.                     emit ItemAdded(item->info,item->ID);
22.                     connect(item,SIGNAL(ItemMoved(QString,int)),this,SLOT(on_
    itemMoved(QString,int)));
23.
24.                     /*else if (event->button() == Qt::RightButton) {
25.                             // 检测光标下是否有 item
26.                             QGraphicsItem *itemToRemove = NULL;
27.                             foreach (QGraphicsItem *item, items(event->scenePo
    s())) {
28.                                 if (item->type() == QGraphicsItem::UserType+1)
     {
29.                                     itemToRemove = item;
30.                                     break;
31.                                 }
32.                             }
33.                             // 从 Scene 上移除 item
34.                             if (itemToRemove != NULL)
35.                                 removeItem(itemToRemove);
36.                         }*/
37.                 }
38.             }
39.             else if(mode == 2){ // is add line mode
40.                     if (event->button() == Qt::LeftButton) {
41.                         //qDebug()<<"click step is "<<QString::number(clickst
    ep);
42.                         if(clickstep == 0){
43.                             qDebug()<<"first click";
44.                             // 在 Scene 上添加一个自定义 item
45.                             QPointF point = event->scenePos();
46.                             MyPoint* p1 = new MyPoint();
47.                             p1->location_x = point.x();
48.                             p1->location_y = point.y();
49.                             temppoint = p1;
50. //                            temppoint = new MyPoint();
51. //                            temppoint->location_x = point.x();
52. //                            temppoint->location_y = point.y();
53.
54.                             qDebug()<<"add line point 1";
55.
56.                             p1->in_line = 1;
57.                             p1->ID = current_item ++;
58.                             p1->info = "line: point: (" + QString::number(p1-
    >location_x) + "," + QString::number(-p1->location_y) +  ")";
```

```
59.     //                          item->location_x = point.x();
60.     //                          item->location_y = point.y();
61.     //                          item->setLocation();
62.                             addItem(p1);
63.
64.                             clickstep = 1;
65.
66.                         }else if(clickstep){
67.                             qDebug()<<"second click";
68.                             //qDebug()<<"click step is "+QString::number(clic
    kstep);
69.                             QPointF point = event->scenePos();
70.
71.
72.                             MyPoint * p2 = new MyPoint();
73.                             p2->location_x = point.x();
74.                             p2->location_y = point.y();
75.                             p2->in_line = 1;
76.
77.                             p2->ID = current_item ++;
78.                             p2->info = "line: point: (" + QString::number(p2-
    >location_x) + "," + QString::number(-p2->location_y) +  ")";
79.
80.                             //item->p2 = *temppoint;
81.
82.                             addItem(p2);
83.                             line* lineitem = new line();
84.                             lineitem->p1 = temppoint;
85.                             lineitem->p2 = p2;
86.                             lineitem->setZValue(-10000);
87.                             addItem(lineitem);
88.
89.                             lineitem->p1->line_parent = lineitem;
90.                             lineitem->p2->line_parent = lineitem;
91.
92.                             lineitem->x1 = lineitem->p1->location_x;
93.                             lineitem->y1 = lineitem->p1->location_y;
94.                             lineitem->x2 = lineitem->p2->location_x;
95.                             lineitem->y2 = lineitem->p2->location_y;
96.
97.
98.                             //lineitem->info1 = "line: point1(" + QString::nu
    mber(lineitem->p1->location_x) + "," + QString::number(-lineitem->p1->location
    _y) +  ")";
```

```
99.
100.
101.//                              lineitem->ID = current_item++;
102.
103.//                              lineitem->p1->ID = lineitem->ID;
104.//                              lineitem->p2->ID = lineitem->ID;
105.                                emit ItemAdded(lineitem->p1->info,lineitem->p1->I
    D);
106.                                emit ItemAdded(lineitem->p2->info,lineitem->p2->I
    D);
107.
108.                                //emit ItemAdded(lineitem->info,lineitem->ID);
109.
110.
111.                                //connect(lineitem->p1,SIGNAL(LineItemMoved(doubl
    e, double, int)),lineitem,SLOT(on_LineitemMoved(double, double, int)));
112.                                //connect(lineitem->p2,SIGNAL(LineItemMoved(doubl
    e, double, int)),lineitem,SLOT(on_LineitemMoved(double, double, int)));
113.                                //connect(lineitem,SIGNAL(Line_ItemMoved(QString,
    int)),this,SLOT(on_itemMoved(QString,int)));
114.                                //connect(lineitem->p1,ToLineItemMoved(MyPoint*),
    lineitem,SLOT(line_on_pointMoved(MyPoint* point, int pointNum)));
115.                                //connect(lineitem->p2,ToLineItemMoved(MyPoint*),
    lineitem,SLOT(line_on_pointMoved(MyPoint* point, int pointNum)));
116.                                connect(lineitem->p1,SIGNAL(ItemMoved(QString,int
    )),this,SLOT(on_itemMoved(QString,int)));
117.                                connect(lineitem->p2,SIGNAL(ItemMoved(QString,int
    )),this,SLOT(on_itemMoved(QString,int)));
118.
119.                                clickstep = 0;
120.                            }
121.
122.
123.                        }
124.
125.
126.
127.                }
128.               else if(mode == 4){ // is add rec mode
129.                   if (event->button() == Qt::LeftButton) {
130.                       //qDebug()<<"click step is "<<QString::number(clickstep);
131.                       if(clickstep == 0){
132.                           qDebug()<<"first click";
133.                           // 在 Scene 上添加一个自定义 item
```

```
134.                         QPointF point = event->scenePos();
135.                         MyPoint* p1 = new MyPoint();
136.                         p1->location_x = point.x();
137.                         p1->location_y = point.y();
138.                         temppoint = p1;
139.                         qDebug()<<"add line point 1";
140.                         p1->in_line = 3;
141.                         p1->ID = current_item ++;
142.                         p1->info = "rectangle: point: (" + QString::number(p1
    ->location_x) + "," + QString::number(-p1->location_y) +  ")";
143.                         addItem(p1);
144.                         clickstep = 1;
145.
146.                     }else if(clickstep == 1){
147.                         qDebug()<<"second click";
148.                         // 在 Scene 上添加一个自定义 item
149.                         QPointF point = event->scenePos();
150.                         MyPoint* p1 = new MyPoint();
151.                         p1->location_x = point.x();
152.                         p1->location_y = point.y();
153.                         temppoint2 = p1;
154.                         qDebug()<<"add rec point 1";
155.
156.                         p1->in_line = 3;
157.                         p1->ID = current_item ++;
158.                         p1->info = "rectangle: point: (" + QString::number(p1
    ->location_x) + "," + QString::number(-p1->location_y) +  ")";
159.                         addItem(p1);
160.                         clickstep = 2;
161.
162.                     }else if(clickstep == 2){
163.                         qDebug()<<"second click";
164.                         // 在 Scene 上添加一个自定义 item
165.                         QPointF point = event->scenePos();
166.                         MyPoint* p1 = new MyPoint();
167.                         p1->location_x = point.x();
168.                         p1->location_y = point.y();
169.                         temppoint3 = p1;
170.                         qDebug()<<"add rec point 1";
171.
172.                         p1->in_line = 3;
173.                         p1->ID = current_item ++;
174.                         p1->info = "rectangle: point: (" + QString::number(p1
    ->location_x) + "," + QString::number(-p1->location_y) +  ")";
```

```
175.                         addItem(p1);
176.                         clickstep = 3;
177.                     }else if(clickstep == 3){
178.                         qDebug()<<"third click";
179.                         //qDebug()<<"click step is "+QString::number(clickstep);
180.                         QPointF point = event->scenePos();
181.                         MyPoint * p2 = new MyPoint();
182.                         p2->location_x = point.x();
183.                         p2->location_y = point.y();
184.                         p2->in_line = 3;
185.                         p2->ID = current_item ++;
186.                         p2->info = "rectangle: point: (" + QString::number(p2->location_x) + "," + QString::number(-p2->location_y) +  ")";
187.
188.                         //item->p2 = *temppoint;
189.
190.                         addItem(p2);
191.                         Rectangle* recitem = new Rectangle();
192.                         recitem->p1 = temppoint;
193.                         recitem->p2 = temppoint2;
194.                         recitem->p3 = temppoint3;
195.                         recitem->p4 = p2;
196.
197.
198.                         recitem->setZValue(-10000);
199.                         addItem(recitem);
200.
201.
202.                         recitem->p1->rec_parent = recitem;
203.                         recitem->p2->rec_parent = recitem;
204.                         recitem->p3->rec_parent = recitem;
205.                         recitem->p4->rec_parent = recitem;
206.
207.
208.                         recitem->x1 = recitem->p1->location_x;
209.                         recitem->y1 = recitem->p1->location_y;
210.                         recitem->x2 = recitem->p2->location_x;
211.                         recitem->y2 = recitem->p2->location_y;
212.                         recitem->x3 = recitem->p3->location_x;
213.                         recitem->y3 = recitem->p3->location_y;
214.                         recitem->x4 = recitem->p4->location_x;
215.                         recitem->y4 = recitem->p4->location_y;
216.
```

```
217.
218.
219.//                              lineitem->p2->ID = lineitem->ID;
220.                            emit ItemAdded(recitem->p1->info,recitem->p1->ID);
221.                            emit ItemAdded(recitem->p2->info,recitem->p2->ID);
222.                            emit ItemAdded(recitem->p3->info,recitem->p3->ID);
223.                            emit ItemAdded(recitem->p4->info,recitem->p4->ID);
224.
225.
226.                            //connect(lineitem->p1,SIGNAL(ItemMoved(QString,int))
    ,this,SLOT(on_itemMoved(QString,int)));
227.                            //connect(lineitem->p2,SIGNAL(ItemMoved(QString,int))
    ,this,SLOT(on_itemMoved(QString,int)));
228.                            connect(recitem->p1,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
229.                            connect(recitem->p2,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
230.                            connect(recitem->p3,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
231.                            connect(recitem->p4,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
232.
233.                            clickstep = 0;
234.                        }
235.                }
236.
237.                }
238.                else if(mode == 3 ){ // is add tri mode
239.                    if (event->button() == Qt::LeftButton) {
240.                        //qDebug()<<"click step is "<<QString::number(clickstep);
241.                        if(clickstep == 0){
242.                            qDebug()<<"first click";
243.                            // 在 Scene 上添加一个自定义 item
244.                            QPointF point = event->scenePos();
245.                            MyPoint* p1 = new MyPoint();
246.                            p1->location_x = point.x();
247.                            p1->location_y = point.y();
248.                            temppoint = p1;
249.                            qDebug()<<"add line point 1";
250.                            p1->in_line = 2;
251.                            p1->ID = current_item ++;
252.                            p1->info = "triangle: point: (" + QString::number(p1-
    >location_x) + "," + QString::number(-p1->location_y) +  ")";
253.                            addItem(p1);
```

```
254.                         clickstep = 1;
255.
256.                     }else if(clickstep == 1){
257.                         qDebug()<<"second click";
258.                         // 在 Scene 上添加一个自定义 item
259.                         QPointF point = event->scenePos();
260.                         MyPoint* p1 = new MyPoint();
261.                         p1->location_x = point.x();
262.                         p1->location_y = point.y();
263.                         temppoint2 = p1;
264.                         qDebug()<<"add tri point 1";
265.
266.                         p1->in_line = 2;
267.                         p1->ID = current_item ++;
268.                         p1->info = "triangle: point: (" + QString::number(p1-
    >location_x) + "," + QString::number(-p1->location_y) +  ")";
269.                         addItem(p1);
270.                         clickstep = 2;
271.
272.                     }else if(clickstep == 2){
273.                         qDebug()<<"second click";
274.                         //qDebug()<<"click step is "+QString::number(clickste
    p);
275.                         QPointF point = event->scenePos();
276.                         MyPoint * p2 = new MyPoint();
277.                         p2->location_x = point.x();
278.                         p2->location_y = point.y();
279.                         p2->in_line = 2;
280.                         p2->ID = current_item ++;
281.                         p2->info = "triangle: point: (" + QString::number(p2-
    >location_x) + "," + QString::number(-p2->location_y) +  ")";
282.
283.                         //item->p2 = *temppoint;
284.
285.                         addItem(p2);
286.                         Triangle* triitem = new Triangle();
287.                         triitem->p1 = temppoint;
288.                         triitem->p2 = temppoint2;
289.                         triitem->p3 = p2;
290.                         triitem->setZValue(-10000);
291.                         addItem(triitem);
292.
293.                         triitem->p1->tri_parent = triitem;
294.                         triitem->p2->tri_parent = triitem;
```

```
295.                          triitem->p3->tri_parent = triitem;
296.
297.                          triitem->x1 = triitem->p1->location_x;
298.                          triitem->y1 = triitem->p1->location_y;
299.                          triitem->x2 = triitem->p2->location_x;
300.                          triitem->y2 = triitem->p2->location_y;
301.                          triitem->x3 = triitem->p3->location_x;
302.                          triitem->y3 = triitem->p3->location_y;
303.
304.
305.//                              lineitem->p2->ID = lineitem->ID;
306.                          emit ItemAdded(triitem->p1->info,triitem->p1->ID);
307.                          emit ItemAdded(triitem->p2->info,triitem->p2->ID);
308.                          emit ItemAdded(triitem->p3->info,triitem->p3->ID);
309.
310.                          //connect(lineitem->p1,SIGNAL(ItemMoved(QString,int))
    ,this,SLOT(on_itemMoved(QString,int)));
311.                          //connect(lineitem->p2,SIGNAL(ItemMoved(QString,int))
    ,this,SLOT(on_itemMoved(QString,int)));
312.                          connect(triitem->p1,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
313.                          connect(triitem->p2,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
314.
315.                          connect(triitem->p3,SIGNAL(ItemMoved(QString,int)),th
    is,SLOT(on_itemMoved(QString,int)));
316.                          clickstep = 0;
317.                      }
318.             }
319.             }
320.         }
321.
322.    paintSelected();
323.}
```

4 部署运行和使用说明

4.1 编译安装

本程序使用 Qt Creator IDE 进行开发。借助 qmake 工具来便利程序的编译。

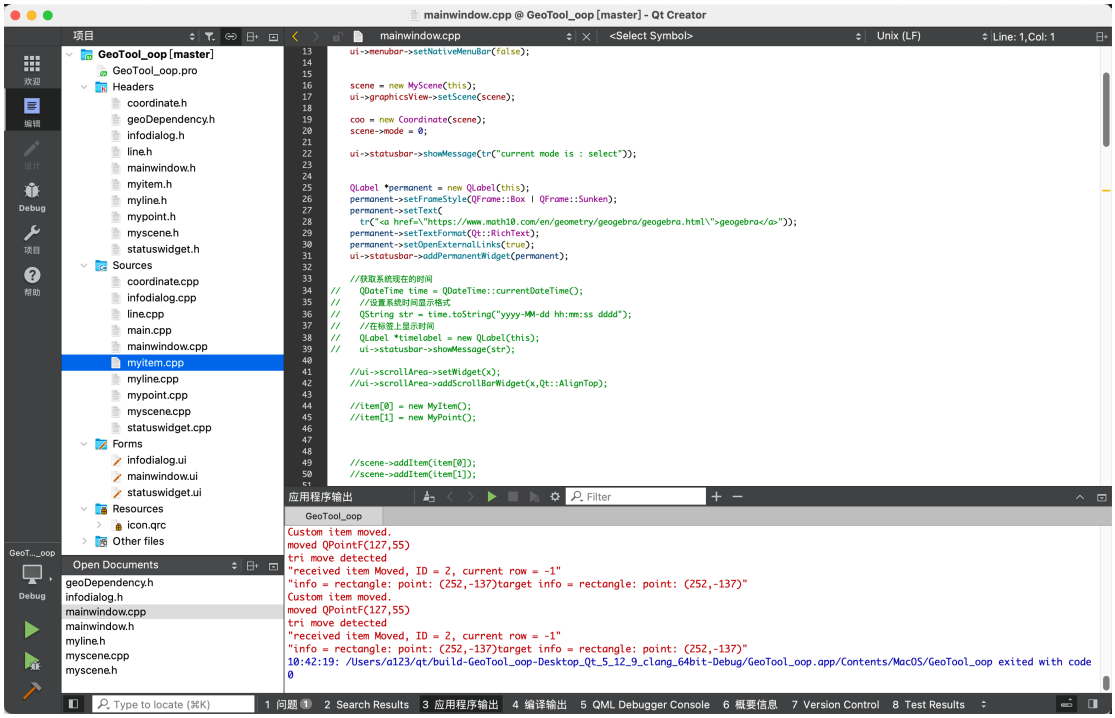
qmake 是 Trolltech 公司创建的用来为不同的平台和编译器书写 Makefile 的工具。是 qt 工具包的一部分。 Trolltech 公司使用 qmake 作为 Qt 库和 Qt 所提供的工具的主要连编工具。

对于 qmake 工具，必要的信息在.pro 文件中列举如下：

```
SOURCES += \
    coordinate.cpp \
    infodialog.cpp \
    line.cpp \
    main.cpp \
    mainwindow.cpp \
    myitem.cpp \
    myline.cpp \
    mypoint.cpp \
    myscene.cpp \
    statuswidget.cpp

HEADERS += \
    coordinate.h \
    geoDependency.h \
    infodialog.h \
    line.h \
    mainwindow.h \
    myitem.h \
    myline.h \
    mypoint.h \
    myscene.h \
    statuswidget.h

FORMS += \
    infodialog.ui \
    mainwindow.ui \
    statuswidget.ui

DISTFILES += \
    png/001-circle.png \
```

```
    png/002-square.png \
    png/003-point.png \
    png/004-triangle.png \
    png/005-segment.png \
    png/006-square-1.png \
    png/007-triangle-1.png \
    png/008-question.png \
    png/009-radius.png \
    png/010-measuring-tape.png

RESOURCES += \
    icon.qrc
```

当然，也可以在 IDE 中直接编译生成可以执行文件。点击左下角的绿色开始符号即可。



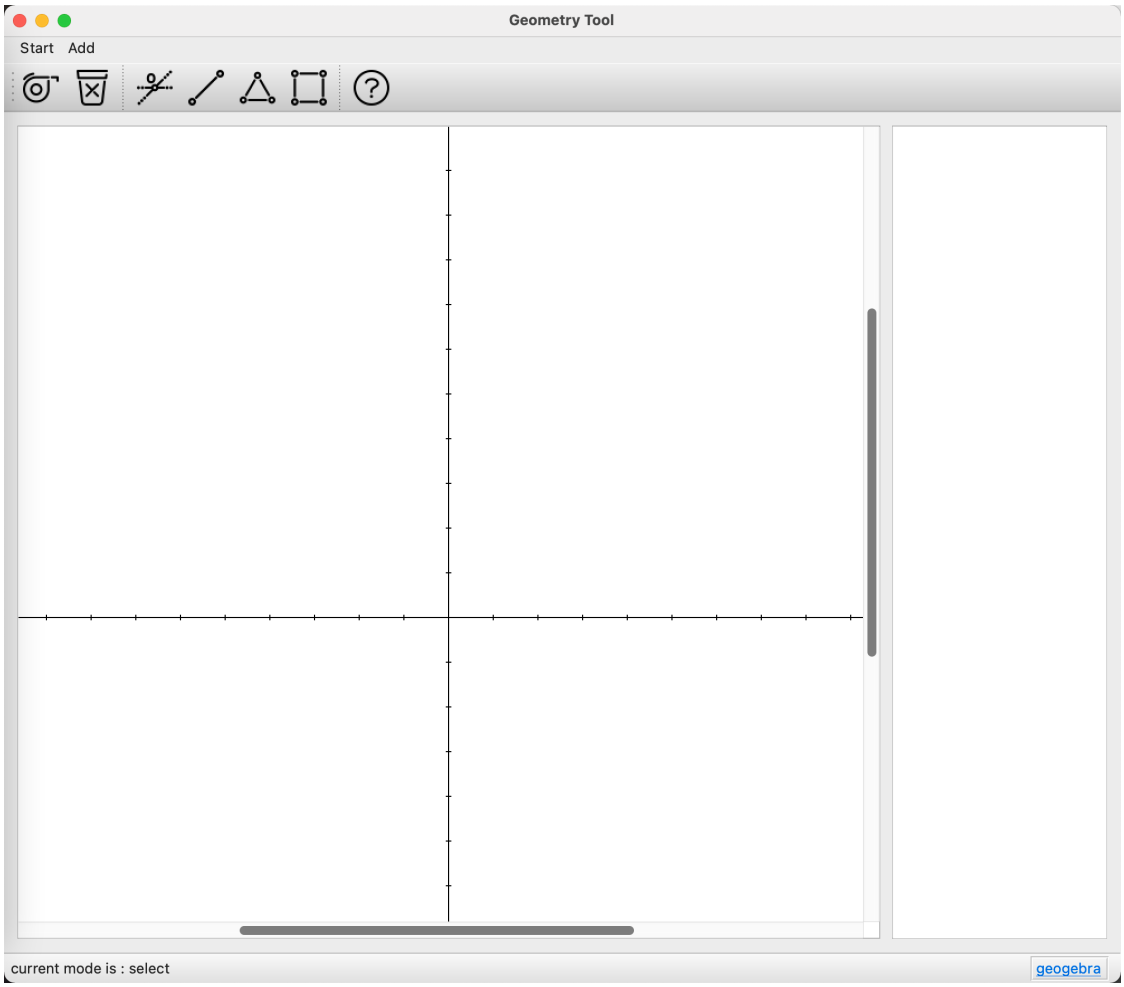
我的个人电脑是 mac ，编译的时候使用的 kit 是 Desktop Qt 5.12.9 clang 64bit ，程序可以顺利编译。在 windows 平台上测试使用 Desktop Qt 6.3.9 MSVC2019 64bit 同样可以顺利编译。

我在文档中同时附上了 Unix 可执行文件和 exe 文件，不过我也发现 exe 文件在部分电脑运行的时候显示缺少 Qt6.0Core.dll 文件依赖，基于 windows 平台的同学在测试的时候需要在环境变量里加上这一文件。

如果测试的时候无法顺利编译或者无法顺利打开 exe 文件或 Unix 可执行文件，关于详细的操作演示，我在文档中另附了一份操作演示视频，可以观看。

4.2 使用操作

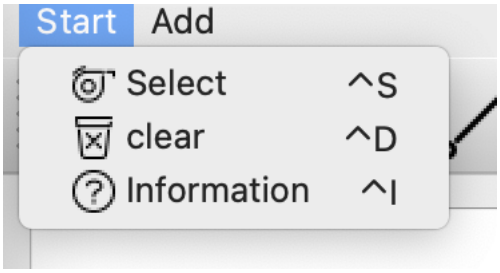
进入程序首页，首先是用户主界面。

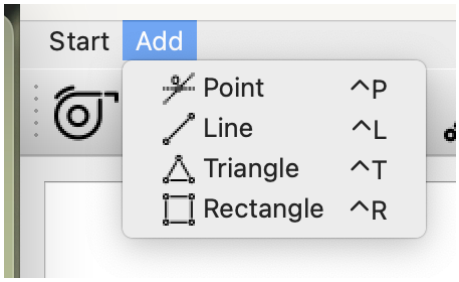


分为多个模块：

- 居于左上角的是菜单栏，包含了文字菜单栏和图标菜单栏两个部分。其中图标菜单栏从左到右分别是选择、删除、添加点、添加线、添加三角形、添加多边形、关于信息等。
  在每一个菜单栏中，各个功能还添加了快捷键。方便用户用键盘进行交互。

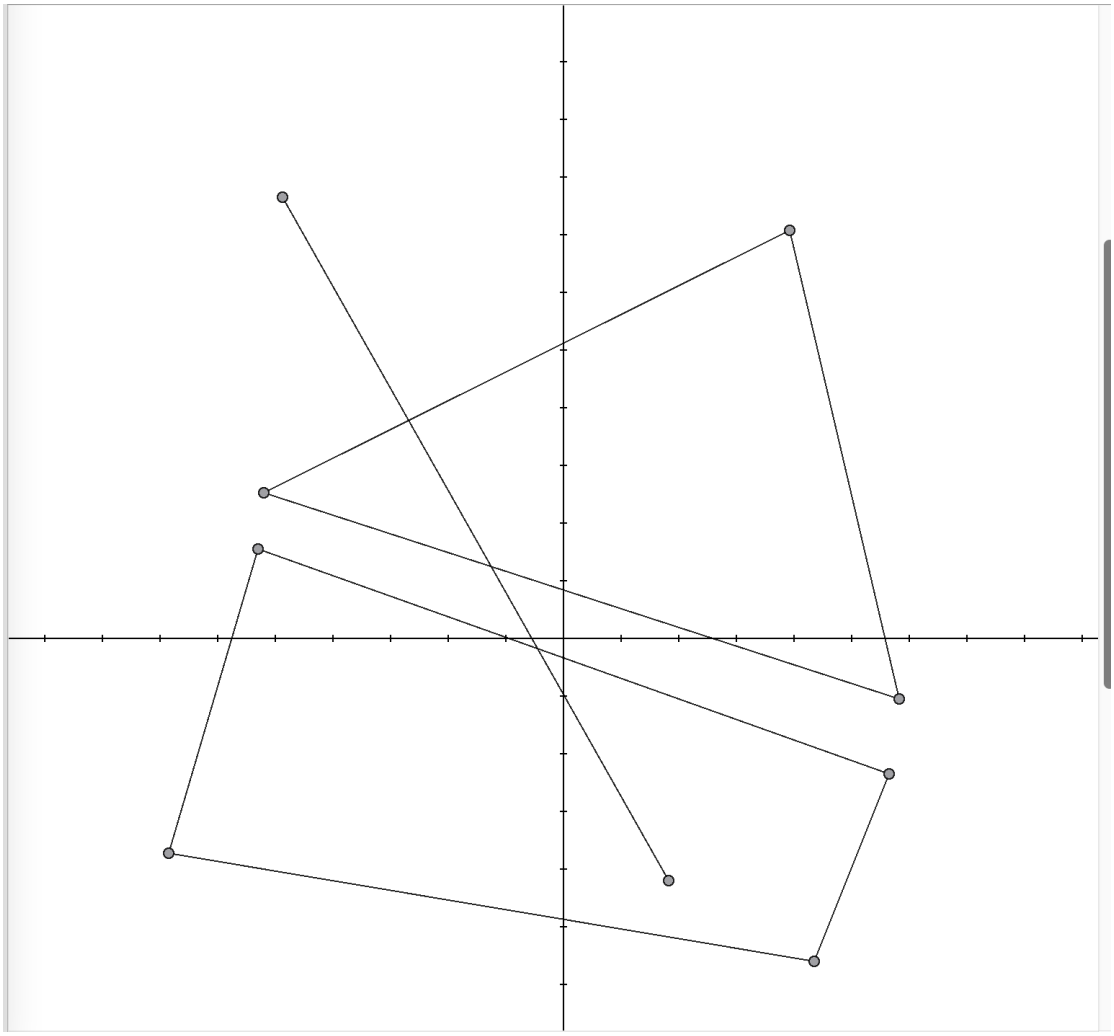
比如选择的快捷键是 ctrl+S，删除的快捷键是 ctrl+D 等等，完整的快捷键表列在下图。



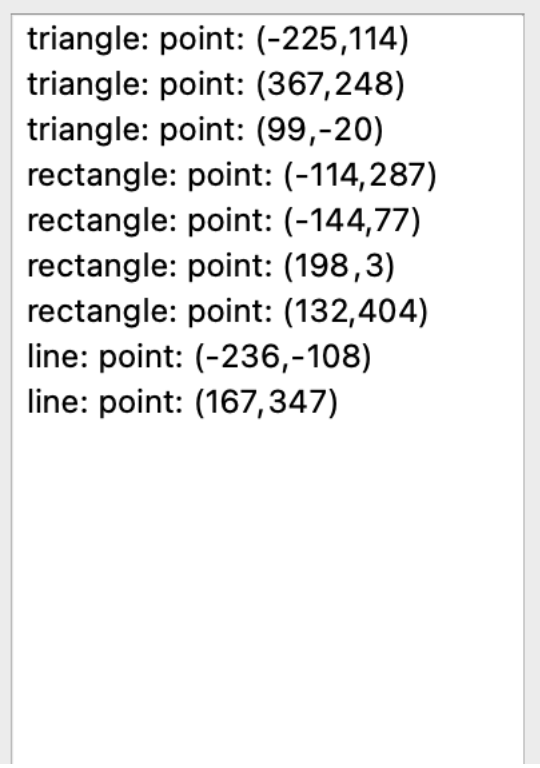


名称	使用	文本	快捷键	可选的	工具提示
actionSelect	☑	Select	^S	☐	Select
actionPoint	☑	Point	^P	☐	Point
actionLine	☑	Line	^L	☐	Line
actionTriangle	☑	Triangle	^T	☐	Triangle
actionRectangle	☑	Rectangle	^R	☐	Rectangle
actionInformation	☑	Information	^I	☐	Information
actionclear	☑	clear	^D	☐	clear

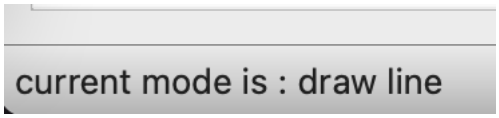
- 居于正中间的是核心的画图板模块，所有的图形信息直接反应在画图板上。同时，用户可以通过鼠标直接和画图板进行互动。



- 在右侧的是信息栏，标识了各个图形的具体信息，标注了每一个点的实时位置。

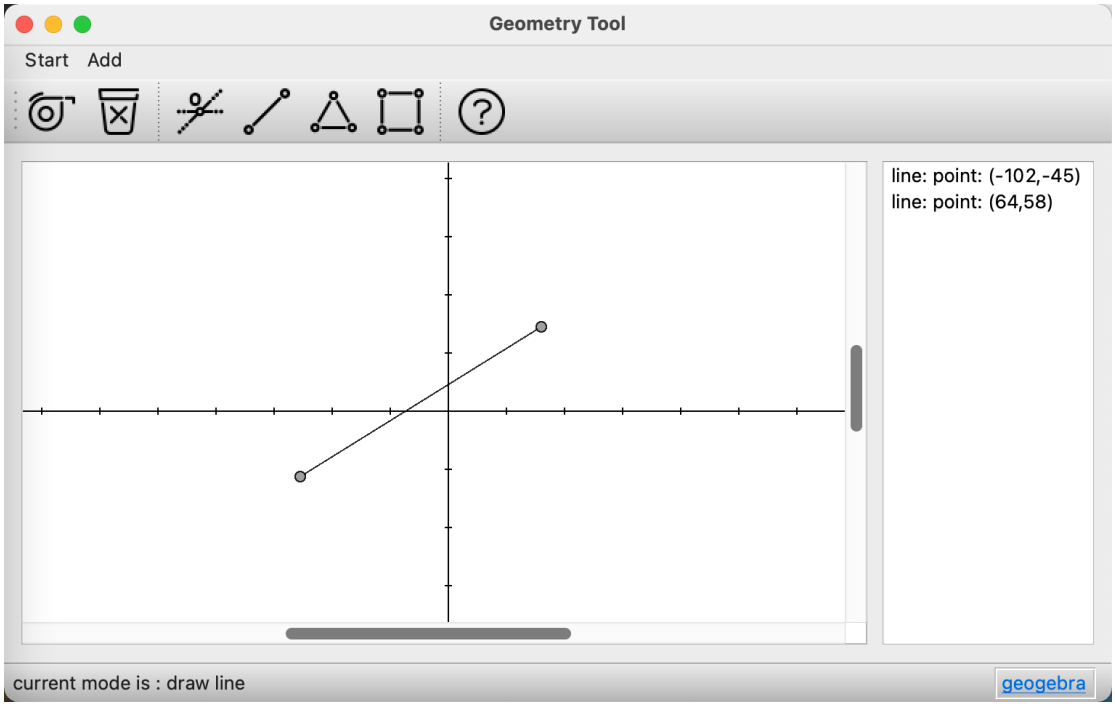


- 在最下方是状态栏，注释了当下的模式选择。

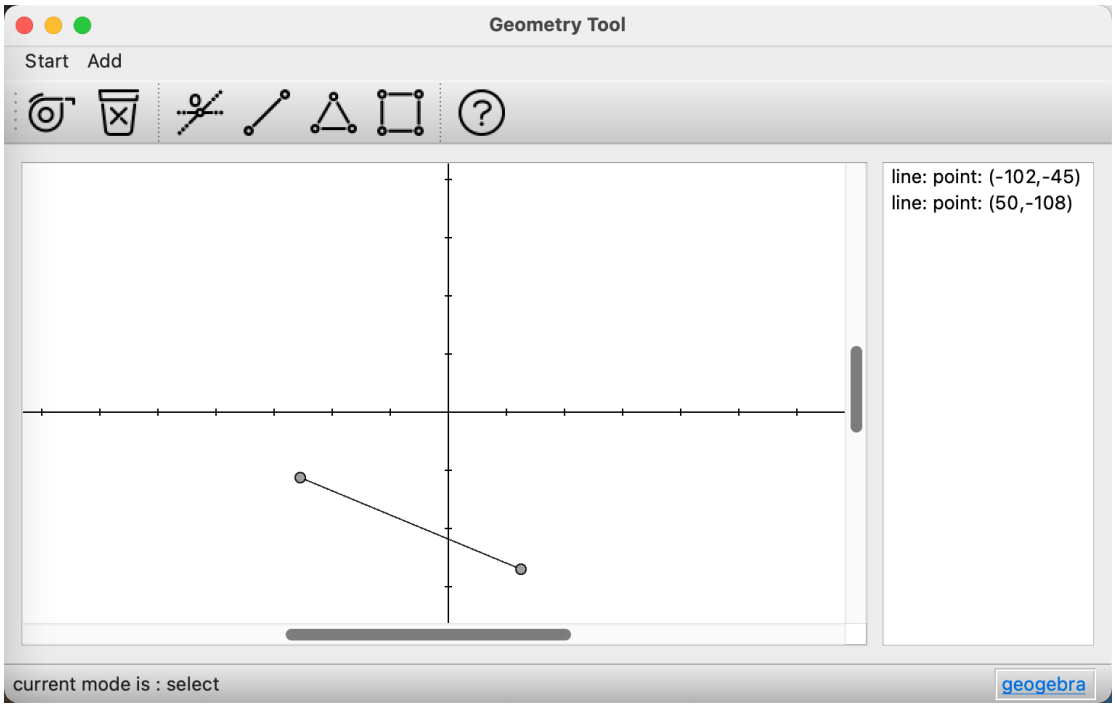


并且在最右下角还有一个超链接，可以打开 geogebra 画图网站进行对比学习。

- 在选择画线模式后，用户可以通过在画图板上点击两次的方式画线，标识线的起始和结束位置。信息会实时更新。对于其他的几何图形，同理。



- 在选择选择模式后，用户可以通过鼠标长按其中一个点进行拖动，就可以改变任意点的位置。可以见到在图中右侧的信息也实时更新了。



4.3 收获感言

本项目中可以说是我的第一个产品设计项目，也是让我第一次感到开发一个这样的产品非常有成就感的项目。为了完成这项工作，我从零开始学习了 Qt 框架，理解了内部的信号和槽机制，搞明白了用户接口应该怎样设计，并且花费了大量的时间进行 debug 和测试。

这个程序的开发让我深入锻炼了 C++的语言特性，多文件系统的开发手段，以及怎样进行一个程序模块的设计，才能让开发的时候思路更加明确。类的设计是非常重要的，尤其是类的继承，非常考验开发者的编程能力和设计思维。

设计一个良好的 ui 也非常关键，我为此也特意筛选了很多图标，选择了最接近最直观的几个来作为我程序的用户交互模块图标。

因为本人能力有限，在一开始构建这个程序的时候，完全是摸爬滚打，因此在一开始设计的时候出现了大量的冗余代码，并且在类的设计上有不妥当的地方。也正是因为如此，当我在想到要重新添加一些新的功能（比如，两个点的合并等）的时候，发现这与我原本的代码的逻辑是冲突的，并不好加上。如果我在最初设计的时候能够有更清晰的蓝图，那么整个程序都会更容易开发的多。

现在的功能实现尚还局限，本来计划加上的一些功能或是因为时间限制或是因为逻辑冲突不方便添加，因此这份工程就只能先做到这一步了。但是这一次的经验和教训我都已掌握，这次工程也极大的锻炼了我的编程能力和 c++上的开发能力，下一个项目我必能做的更好。

5 参考文献资料

- 徐文鹏主编；王玉琨，刘永和副主编. 计算机图形学基础（OpenGL 版）：清华大学出版社，2014.06
- https://doc.qt.io/
- http://c.biancheng.net/qt/