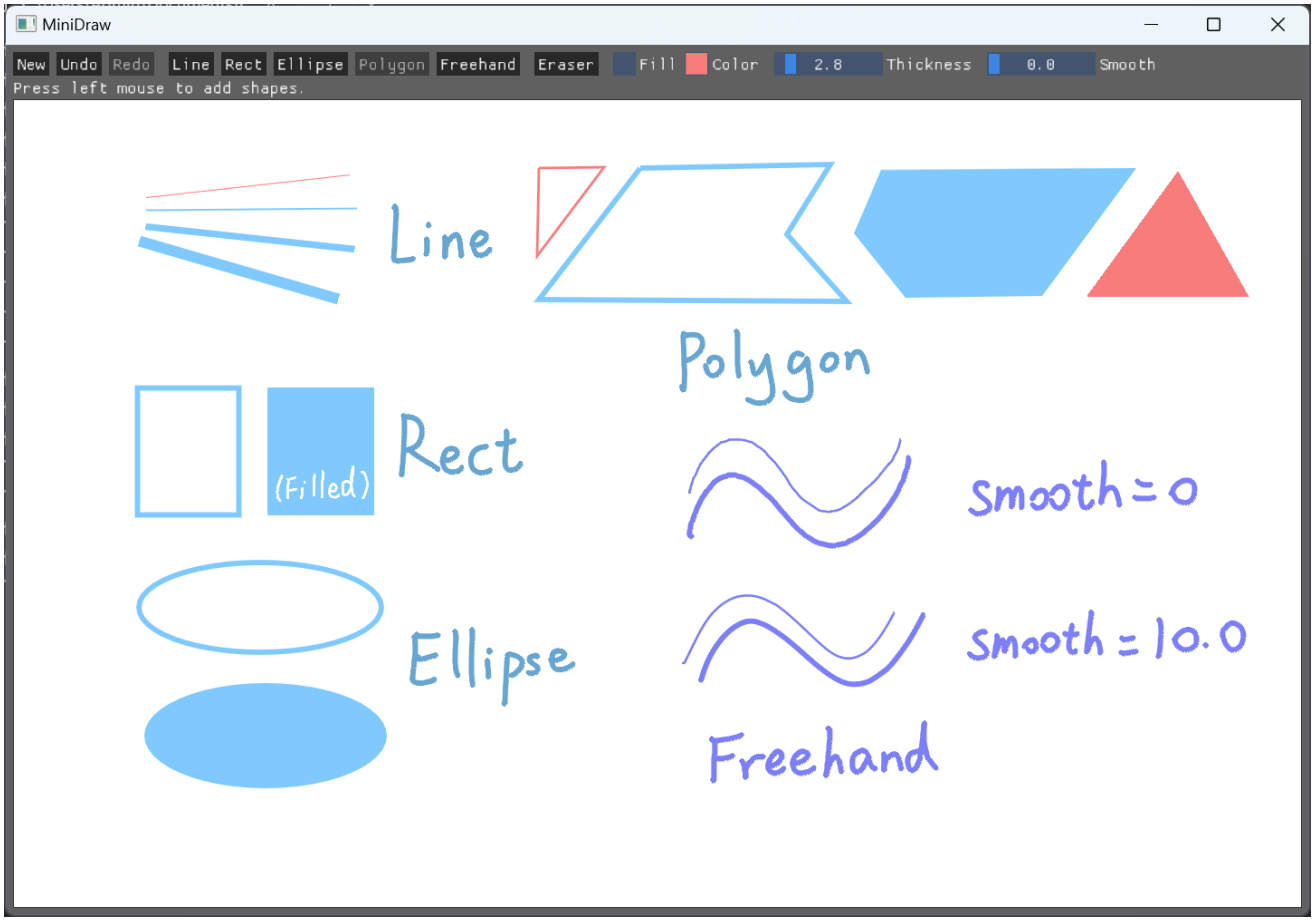
Homework 1 作业报告 by 76-朱雨田

本次作业要求在助教的框架上完成画图软件 MiniDraw，要求实现**椭圆、多边形的绘制**，以及可选的**自由图形绘制。**

我在实现了椭圆、多边形、自由图形的要求基础上进一步实现了**橡皮擦、线宽选项、颜色选项、图形填充选项、撤销重做、手抖修正（Smooth）**的额外功能，使得 MiniDraw 可以在一定程度上实现绘画的需要。

以下是效果展示：


MiniDraw
New
Undo
Redo
Line
Rect
Ellipse
Polygon
Freehand
Eraser
Fill
Color
2.8
Thickness
0.0
Smooth
Press left mouse to add shapes.
Line
Polygon
(Filled)
Rect
Smooth = 0
Ellipse
Smooth = 10.0
Freehand

以下是本次作业的类图：


Shape
Class
Fields
config_
Methods
~Shape
draw
setConfig
update
Nested Types
public
Freehand
Class
Shape
Fields
end_point_x_
end_point_y_
pointlist
start_point_x_
start_point_y_
Methods
~Freehand
addPoint
draw
Freehand (+ 1...
update
Line
Class
Shape
Fields
end_point_x_
end_point_y_
start_point_x_
start_point_y_
Methods
~Line
draw
Line (+ 1 overl...
update
Polygon
Class
Shape
Fields
end_point_x_
end_point_y_
FinishAreaRad...
finished
pointlist
start_point_x_
start_point_y_
Methods
~Polygon
addPoint
draw
finishDrawing
inFinishArea
Polygon (+ 1...
size
update
Ellipse
Class
Shape
Fields
end_point_x_
end_point_y_
start_point_x_
start_point_y_
Methods
~Ellipse
draw
Ellipse (+ 1 ov...
update
Rect
Class
Shape
Fields
end_point_x_
end_point_y_
start_point_x_
start_point_y_
Methods
~Rect
draw
Rect (+ 1 over...
update

1. 椭圆

椭圆形的绘制与矩形类似，都是通过确定左上角、右下角的顶点来确定整个图形。

和 Line，Rect 一样，我们为椭圆新开一个 class 并新建文件，继承 Shape 并实现 `draw()` 的虚函数接口。这是为了方便将所有的图元封装为 Shape 类，这样 `Canvas` 类中就不必考虑绘制图元的类型，而在每次更新事件统一调用它们的 `draw()` 接口。

2020 年的 Qt 框架中存在 **Ellipse 是 Qt 中的类名，定义 Ellipse 类会导致报错**的情况。ImGUI 框架中没有遇到这类问题，因此直接沿用示例代码中 Line、Rect 的命名方式将类名命名为了 Ellipse。

翻阅 ImGUI 的接口，就能发现可用的 `draw_list->AddEllipse()` 函数。计算出椭圆中心、半长轴、半短轴等数据后填入参数中即可。

2. 多边形

与椭圆同样地、为多边形新建一个 `Polygon` 类。与椭圆仅需实现 `draw()` 接口不同的是，多边形类还需要存储顶点位置的数据。这里以 `std::vector<std::pair<float, float>>` 存储了多边形所有的顶点，其中起点与终点不会重复记录。

绘制方面，ImGUI 有自带的 `draw_list->AddPolyline()` 可用于绘制折线，其中顶点位置以数组头指针的形式输入。这里**每次绘制事件**都动态申请了一个数组以向该函数传递顶点位置，并在绘制结束后释放。感觉这并不是非常漂亮的实现方式。

同时，我为多边形绘制添加了许多细节。当多边形绘制过程中，若鼠标光标靠近多边形的起点，则光标会被自动吸附至多边形的顶点上，此时再次点击左键添加顶点则会直接连接至起点，绘制结束。若在绘制过程中单击鼠标右键，则会直接将上一个顶点连接至起点。为了实现这样的目的，我在多边形类中继续添加了 `Polygon::finishDrawing()` 等方法，便于在 `Canvas` 类中实现多边形绘制的交互。

3. 自由绘制

自由绘制存储数据的方式与多边形相同，只是在鼠标左键按住时的每次 `update` 事件中都在鼠标位置添加一个顶点，而不是在按下左键时才添加顶点。鼠标松开时完成图形绘制，参照 Line 和 Rect 的实现即可。

绘制方面，`draw_list->AddPolyline()` 于自由绘制的效果并不佳。顶点连接处会有很不自然的裂缝，如图所示：

在线条起点和终点处因为笔迹角度变化较大，该函数会绘制很多不自然的尖角。在折角较大处也能看出不自然的痕迹。

因此我们转而使用更原始的方式，即循环绘制顶点间的每条线段。效果如下：

在折角较大处出现了我们不希望出现的，影响画面效果的裂隙。为了解决该问题，在所有线条保持不透明的情况下，我们可以略微偷懒，在每个顶点处绘制一个**半径为线宽一半的、已填充的正圆**（实际发现半径再略小一点能得到更好的效果）。效果如下：

可以看到折角处变得相当平滑，且起点与终点处都变成了圆形，达到了相当不错的绘制效果。我们认为做到这一步的 Freehand 已经相当令人满意了。

Photoshop、Sai、Krita、Procreate 等专业绘画软件的笔刷并不是 MiniDraw 中 Freehand 的折线的实现思路，而是在顶点及其连线上叠加图章以得到有纹理的线条。这样可以自然地避免裂隙的问题，但也决定了这种思路无法用于绘制矢量图形。

我还试图为线条加上数位板的压感，但尝试寻找后并没有找到可行的接口（这里感谢z）。之后，我试着给线条加上按速度变化宽度（速度越大，线条越窄）的特性，但实现效果不佳。

4. 橡皮擦

GoodNotes 等矢量笔记软件的橡皮擦实现思路是**删除橡皮擦笔刷所经之处的线段**。这里由于 `shape_list_` 中存储的数据是**图元**而不是**线段**，在 Freehand 图元中控制线段的显示与否又会相当麻烦（还要适应撤销重做功能！），因此并没有采用这样的矢量橡皮擦思路。专业绘画软件的橡皮擦是通过直接在图层上降低像素透明度来实现的。这在MiniDraw中也并不可行。

其实，这里的橡皮擦，就是**颜色为白色的 Freehand**！在不做**图层、自由变换、复制粘贴**等功能的情况下，这样的偷懒是可以做到橡皮擦的功能的，只是确实是怎么想都不太漂亮的实现。

但加入这样的橡皮擦功能，确实让 “真的用 MiniDraw 画东西” 成为了可能。

5. UI

助教提供的基础 UI 是灰色画布、红色笔刷。通过翻阅代码，我找到了控制画布、笔刷和背景颜色的代码，并对整个 UI 的颜色做出了调整，如以上展示图所示。

对于不同的功能，我翻阅文档，在 UI 上添加了不同的控件，在控件间添加了间隔，并大致理解了 ImGUI **用函数控制样式变化**的思路。生成控件的函数需要包裹在描述样式的函数块中，这与 Qt 等其他 GUI 软件的思路有相当大的不同，可能是 ImGUI 以面向过程为主的缘故。

特别地，我为撤销重做按钮添加了不可行时按钮变灰的特性。这一特性同样也被用于图元类型选择中，被选中的图元按钮颜色灰变灰，从而提示当前绘制的图元类型。

我还在 UI 中添加了 “New” 按钮。该按钮用于清空当前画布的所有数据。

6. 填充选项、线宽选项、颜色选项

助教的代码框架中为每个 `Shape` 对象都存储了单独的 `config_` 属性，用于存储偏移、线宽、颜色等数据。每次绘制新图形时都读取编辑器当前的填充、线宽、颜色数据，并将其赋值至绘制图元的 `config_` 中。同时，UI 在每帧都从画布中读取这些属性以作为 UI 显示的值。

`Canvas` 中存储当前编辑器的填充、线宽、颜色数据，并将其封装成了 `get()` `set()` 的属性，而不是直接将字段暴露在 `public` 中。

长方形、椭圆、多边形都有自带的填充绘制函数。其中多边形的填充绘制不能正常绘制非简单多边形，这个问题目前还没有得到很好的解决。

7. 撤消重做

如果只需实现撤销功能，只需在每次按下撤销键时从 `shape_list_` 中 `pop_back()` 即可达成所需效果。特别地，如果在**绘制过程中**按下撤销键，则取消该次绘制，即结束绘制，并不把图形添加至 `shape_list_` 中。

重做功能则是在 `Canvas` 中添加了一个 `std::stack<std::shared_ptr<Shape>> redo_stack_`，每次撤销时都向栈内压入 `shape_list_` 中被删除的数据，在重做操作时将栈顶弹出并压入 `shape_list_` 中。如果绘制了新的图形，则直接清空该栈。

如果 `shape_list_` 空，则撤销操作不可行；若 `redo_stack_` 空，则重做操作不可行。`Canvas` 向外暴露**能否撤销/重做**的方法，并在 `MiniDraw` 类中用于控制按钮是否变灰。

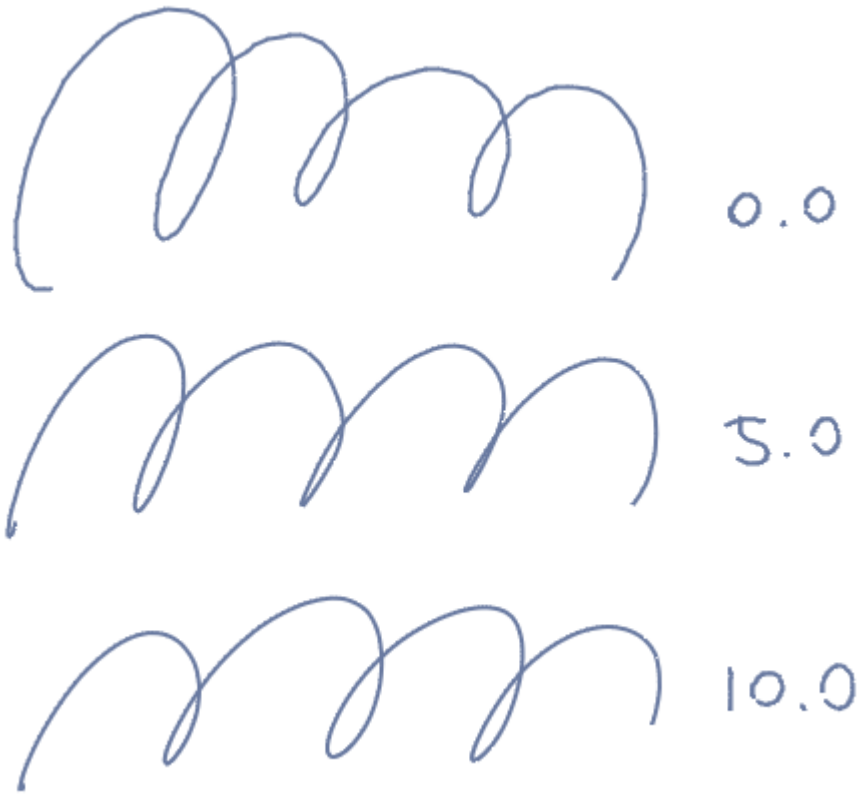
8. 手抖修正

手抖修正也是目前电子绘画所需的重要功能。Photoshop、SAI 等软件的手抖修正仅由一个参数控制，而 Procreate 的手抖修正更为复杂。这里**猜测了** Photoshop、SAI 实现手抖修正的原理，并将其实现在了 MiniDraw 中。

具体地，即记录一个不断延迟跟踪鼠标的位置，每次更新时在该位置上（而不是鼠标位置）添加 Freehand 的顶点。跟踪的方式采用 `lerp`，即跟踪点每次更新都把位置设置为和鼠标间线段按一定比例的取值。即每次更新计算以下公式，m 是一个在 $(0,1]$ 区间内的值，由控件中的 Smooth 值按一定规则映射得来。

$$P'_{Follow}=m\cdot P_{Follow}+(1-m)\cdot P_{Mouse}$$

这一方法也经常用于游戏的平滑移动中。可以观察到拉大平滑值后，线条的抖动明显消失了：



该实现得益于 ImGUI 框架**每帧更新一次画布**的机制。

松开左键后，延迟笔刷应该继续跟踪左键最后的位置，并在该点结束线条。这里因为跟踪点是在 `Canvas` 类中实现的，如果左键抬起后跟踪点仍然在运动，则会和下一笔产生矛盾，所以笔刷会立即停止绘制，这会在平滑值较大时使线条不合预期地提前结束。解决这一问题的思路是为跟踪点新建一个 class，并在每次绘制 Freehand 的时候实例化，由跟踪点控制 Freehand 的绘制，但这里并没有这样实现。

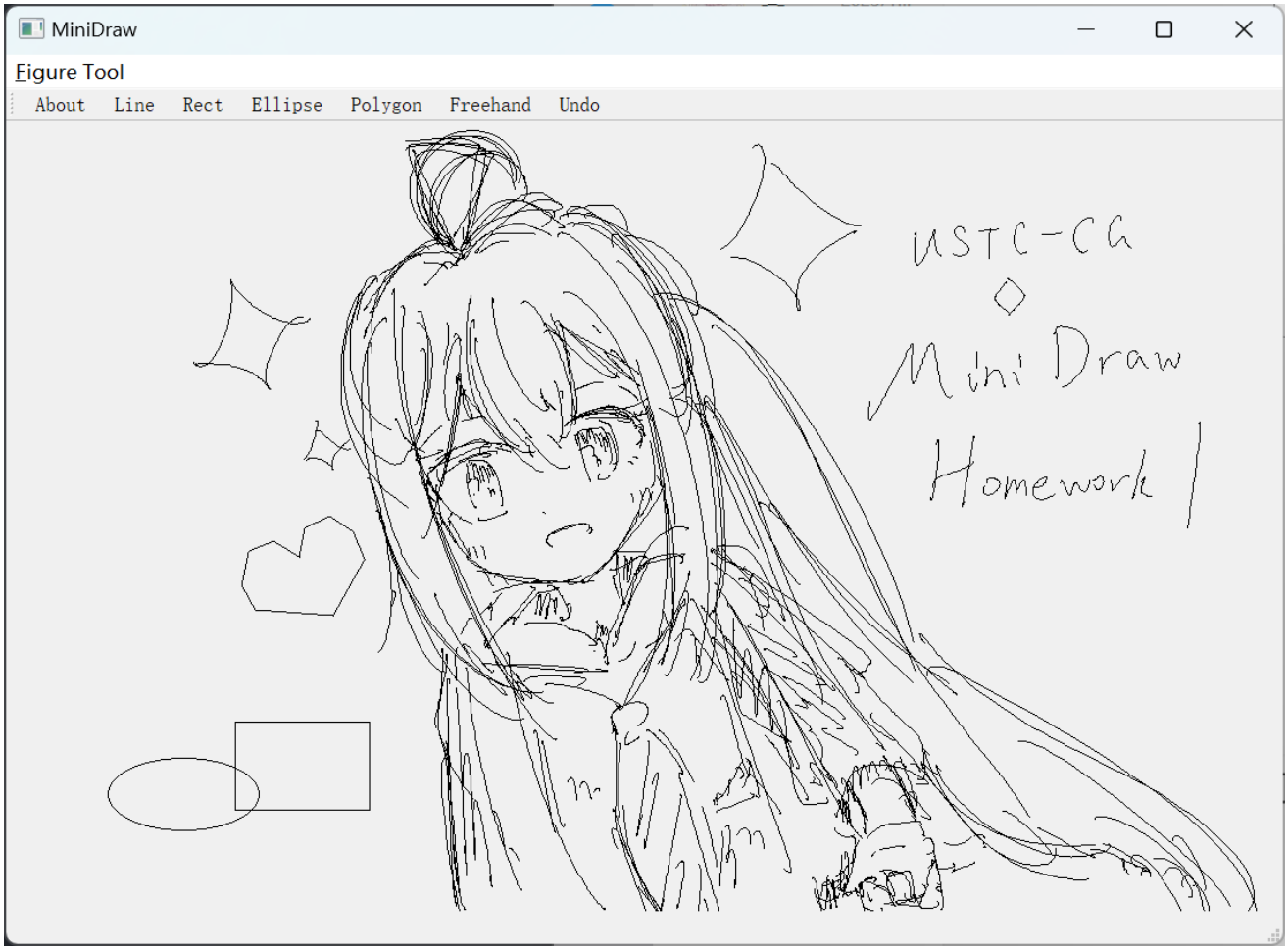
同时，该实现方法还有一个问题，即帧率不同时平滑的效果也有所不同。该问题的解决只需要给 m 乘上一个帧间间隔时间 Δt 值，并调整 m 的取值范围即可（数学上可以证明），但我这里也并没有这样实现。

待提升之处

- 橡皮擦的实现仍然比较偷懒。
- UI 还可以进一步优化，如将工具栏置于左侧（像 Photoshop 那样）、为工具添加图标等。
- 手抖修正中提到的两个问题：笔刷提前停止绘制，不同帧率下平滑效果不同。
- 多边形每次绘制都要申请一次内存，感觉在时间空间上有所浪费。
- 代码风格：在 `Canvas` 类中使用了下划线命名，而在 `Polygon` 等类中使用了小驼峰命名，命名规则不统一。
- `Canvas` 类添加的属性过多，显得比较冗杂。

学习心得

在 2020 年的 USTC-CG 作业中，我已经实现过了 Qt 框架版本 MiniDraw。当时配环境就配了一下午，而这次整个 MiniDraw 的实现只花了一晚上，不知道是框架更简洁了还是我的水平提升了。这是当时 MiniDraw 的实现效果，在 Ellipse Polygon Freehand 的基础上只新增了一个 **Undo** 功能：



可以看到手部和呆毛因为没有橡皮擦所以改不了了（笑）

毕竟是二周目，所以我希望在 USTC-CG 24 的作业中完成所有的 optional 项。身为常与绘画软件打交道的绘画爱好者，我对一个绘画软件该有怎样的功能也有比较充分的了解，因此把我能想到的能实现的功能都实现了个遍，并再次试着在我的 MiniDraw 里绘画。虽然没有笔压和图层功能，以我的水平还画不出完成度比较高的东西，但效果还不错！

因为我对面向对象编程（特别是C#）本身就有一定经验，且已经做过一遍 Qt 版本的 MiniDraw，因此这次作业对我的意义更在于熟悉 ImGUI 框架代码的写法，并了解绘画软件中各种功能的实现思路。用一晚上超量完成~~卷~~了图形学的 Homework1， 我仍然认为这次二周目的作业是相当值得的。**至少看到自己真的能用自己做出来的画图小软件画出点什么东西，这样的成就感就是所见即所得的计算机图形学啊。**