浙江大学

# 面向对象程序设计

# 大 程 序 报 告


**大程名称：** GeoPad 的设计与实现

**姓名 ：** *******

**学号：** *******

**电话：** *******

**指导老师：** 李际军

**2021~2022 春夏学期** 2022 年 6 月 25 日

# 目录

# GeoPad 大程序设计

## 1 大程序简介

### 1.1 选题背景及意义

GeoGebra 是一个结合「几何」、「代数」与「微积分」的动态数学软件，它是由美国佛罗里达州亚特兰大学的数学教授 Markus Hohenwarter 所设计的。 一方面来说，GeoGebra 是一个动态的几何软件。您可以在上面画点、向量、线段、直线、多边形、圆锥曲线，甚至是函数，事后你还可以改变它们的属性。曾获得数个欧洲和美国的教育软件大奖。

使用 C++自主设计一个类似于 Geogebra 的软件，将对于面向对象的工程能力有很大的提升。

### 1.2 目标要求

本选题要求实现类似 https://www.geogebra.org/geometry 的几何画板的功能，能够创建基本几何图形，能够对几何图形进行拖拽、缩放，可以进行中点、垂线、切线的构造等。

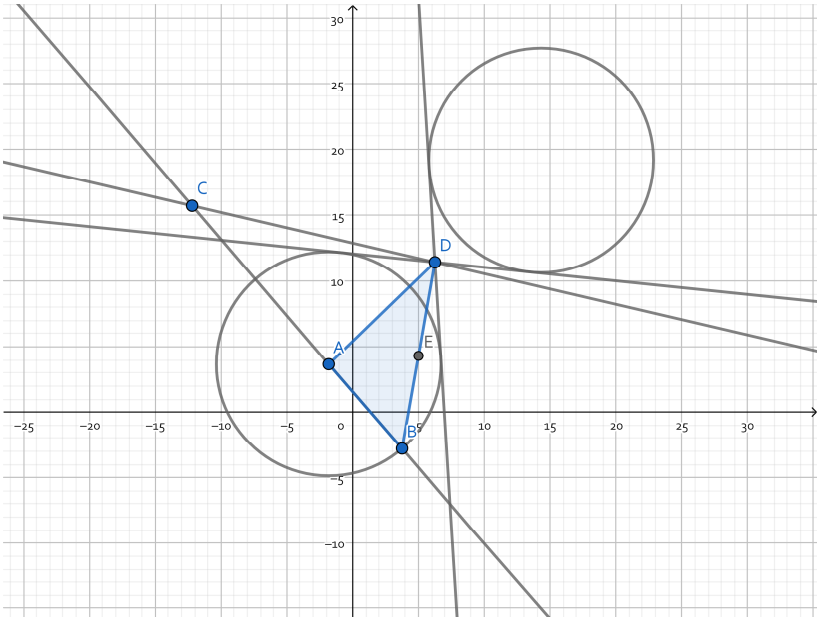



图 1.2 Geogebra 界面

## 1.3 术语说明

- GLFW：是一个开源的多平台库，用于 Windows 上的 OpenGL、OpenGL ES 和 Vulkan 开发。它提供了一个简单的 API，用于创建窗口、上下文和表面，接收输入和事件。GLFW 是用 C 语言编写的，支持 Windows、macOS、X11 和 Wayland。
- GLAD：OpenGL 辅助库。
- JsonCpp：JsonCpp 是用于生成和解析 JSON 的 C++ 开源库。
- ImGUI：ImGUI 又称为 Dear ImGui，它是与平台无关的 C++轻量级跨平台图形界面库，没有任何第三方依赖，可以将 ImGUI 的源码直接加到项目中使用，也可以编译成 dll, ImGUI 使用 DX 或者 OpenGL 进行界面渲染。
- Delete（级联删除）：对于构造出的几何图形，当删除其中一个图形时，与被删除图形有几何关联的图形也需要一并删除，引起级联删除。
- Center（构造中点）：构造两个点的中点。
- Vertical（构造垂线）：构造点到直线的垂线。
- Tangent（构造切线）：构造点到圆的切线。
- Reflect about point（中心对称）：构造图形关于某个点的中心对称图形。

# 2 需求分析

## 2.1 业务需求

大程序主要的需求是实现类似于 Geogebra 的几何图形功能，能够创建基本的几何图形（点、圆、直线等），对几何图形进行编辑（删除、移动等），实现几何关系的构造（中点、垂线、切线等）。

## 2.2 功能需求

**创建几何图形**

- 创建点：创建一个可编辑的点对象
- 创建圆：创建一个可编辑的圆对象

- 创建矩形：创建一个可编辑的矩形对象
- 创建直线：创建一个可编辑的直线对象
- 创建多边形：创建一个可编辑的多边形对象

**移动和删除**

- 移动：对选中的图形进行移动
- 删除：级联删除有几何关联的所有图形

**几何构造**

- 中点：构造两个点的中点
- 垂线：构造点到直线的垂线
- 切线：构造点到圆的切线

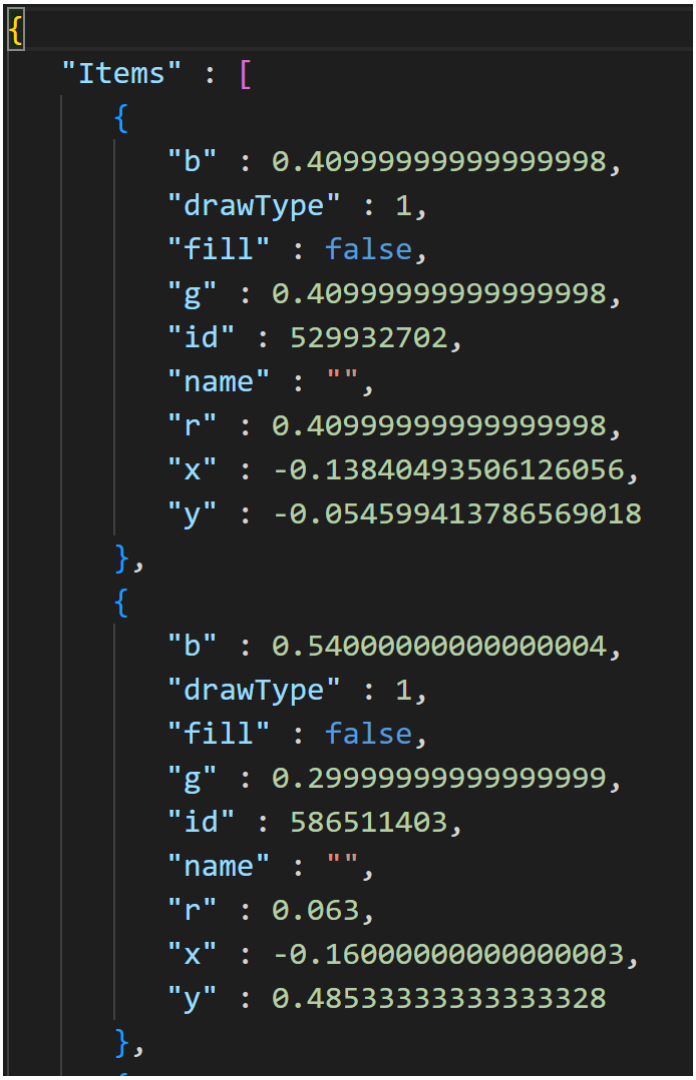
**几何变换**

- 中心对称：构造图形关于点的中心对称图形（目前只实现了圆）

## 2.3 数据需求

本大程序主要对几何图元进行操作，因此在储存时要保存几何图元的坐标以及几何构造关系。本项目中，文件均以 json 格式保存。

例如对于所有点的保存：



```
{
    "Items" : [
        {
            "b" : 0.40999999999999998,
            "drawType" : 1,
            "fill" : false,
            "g" : 0.40999999999999998,
            "id" : 529932702,
            "name" : "",
            "r" : 0.40999999999999998,
            "x" : -0.13840493506126056,
            "y" : -0.054599413786569018
        },
        {
            "b" : 0.54000000000000004,
            "drawType" : 1,
            "fill" : false,
            "g" : 0.29999999999999999,
            "id" : 586511403,
            "name" : "",
            "r" : 0.063,
            "x" : -0.16000000000000003,
            "y" : 0.48533333333333328
        },
```


图 2.1 点对象数据格式

其中 r g b 表示点的 RGB 值，drawType 表示绘制点时所用的模式（虚、实），fill 表示是否填充，id 表示全局唯一标识符，name 表示点的名字，x y 表示坐标。

对于几何关系的储存，以垂线关系的储存为例：


图 2.2 垂线关系数据格式

lineDst 和 lineSrc 表示直线的两个端点，pedal 表示垂足，point 表示过哪个点做的垂线，均记录了其全局唯一标识符。

### 2.4 性能需求

图形显示界面需要流畅，用户在对几何图形进行拖动或删除时时不能有卡顿现象，我们对画布上的几何图形进行更新均是平凡的遍历更新，仅在级联删除时使用了 BFS 进行删除，但 BFS 的复杂度仍然是线性的。总体来说，整个程序的性能很好，所有操作均可以在人感知不到卡顿的情况下完成。

## 3 类库已有功能分析

由于本项目没有使用 libgl 库，因此这里对 OpenGL 进行简要分析。

### 3.1 总体架构设计

OpenGL(Open Graphics Library)是跨语言、跨平台的编程图形程序接口，它采用 C 语言编写，所以可以支持多语言集成；同时，OpenGL 仅对 GPU 进行编程，不支持视图窗口的编写，所以可以在多平台之间通用。它将计算机的资源抽象成一个个 OpenGL 对象，对这些资源的操作抽象成一个个 OpenGL 指令。

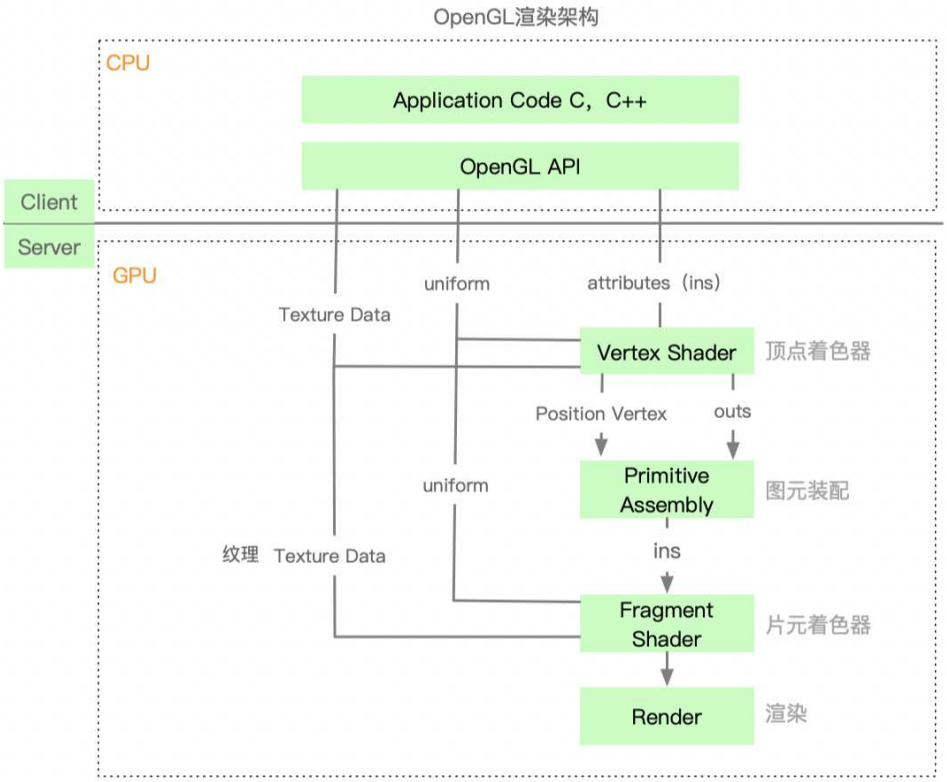



图 3.1 OpenGL 渲染流程

## 3.2 类体系设计

### C 语言实现“对象”

OpenGL 是用 C 语言编写的，但由于 OpenGL 需要为很多面向对象的语言提供支持，因此 OpenGL 的开发当中引入了对象（Object）的概念。

在 OpenGL 中一个对象是指一些选项的集合，它代表 OpenGL 状态的一个子集。比如，我们可以用一个对象来代表绘图窗口的设置，之后我们就可以设置它的大小、支持的颜色位数等等。可以把对象看做一个 C 风格的结构体(Struct)：

```
struct object_name {
    GLfloat  option1;
    GLuint   option2;
    GLchar[] name;
};
```

### 状态机

OpenGL 可以记录自己的状态，比如下面的这几行代码（比如：当前所使用的颜色、是否开启了混合功能，等等，这些都是要记录的）

```
glClearColor(1.0,1.0,1.0.1.0);

glEnable(GL_DEPTH_TEST);//开启深度测试
glDisable(GL_DEPTH_TEST);//关闭深度测试

glEnable(GL_BLEND);//开启混合
glDisable(GL_BLEND)//关闭混合
```

- OpenGL **可以接收输入**（当我们调用 OpenGL 函数的时候，实际上可以看 OpenGL 在接收我们的输入），根据输入的内容和自己的状态，修改自己的状态，并且可以得到输出（比如我们调用 glColor3f，则 OpenGL 接收到这个输入后会修改自己的“当前颜色”这个状态；我们调用 glRectf，则 OpenGL 会输出一个矩形）
- OpenGL **可以进入停止状态**，不再接收输入。这个可能在我们的程序中表现得不太明显，不过在程序退出前，OpenGL 总会先停止工作的。

## 3.3 关键功能类及函数设计描述

**函数原型：**

```
void glEnable(GLenum cap);

void glDisable(GLenum cap);
```

**函数参数：**

```
cap  //指明一个 GL 功能的标识符。
```

**描述：**

glEnable/glDisable 可以用来开启和关闭各种功能。使用 glIsEnabled 或 glGet 来获取当前设置的 GL 功能。GL_DITHER 和 GL_MULTISAMPLE 的初始值为 GL_TURE，其他功能的初始值都是 GL_FALSE。

glEnable/glDisable 都只接受一个 cap 参数，cap 的取值可以是（等）：

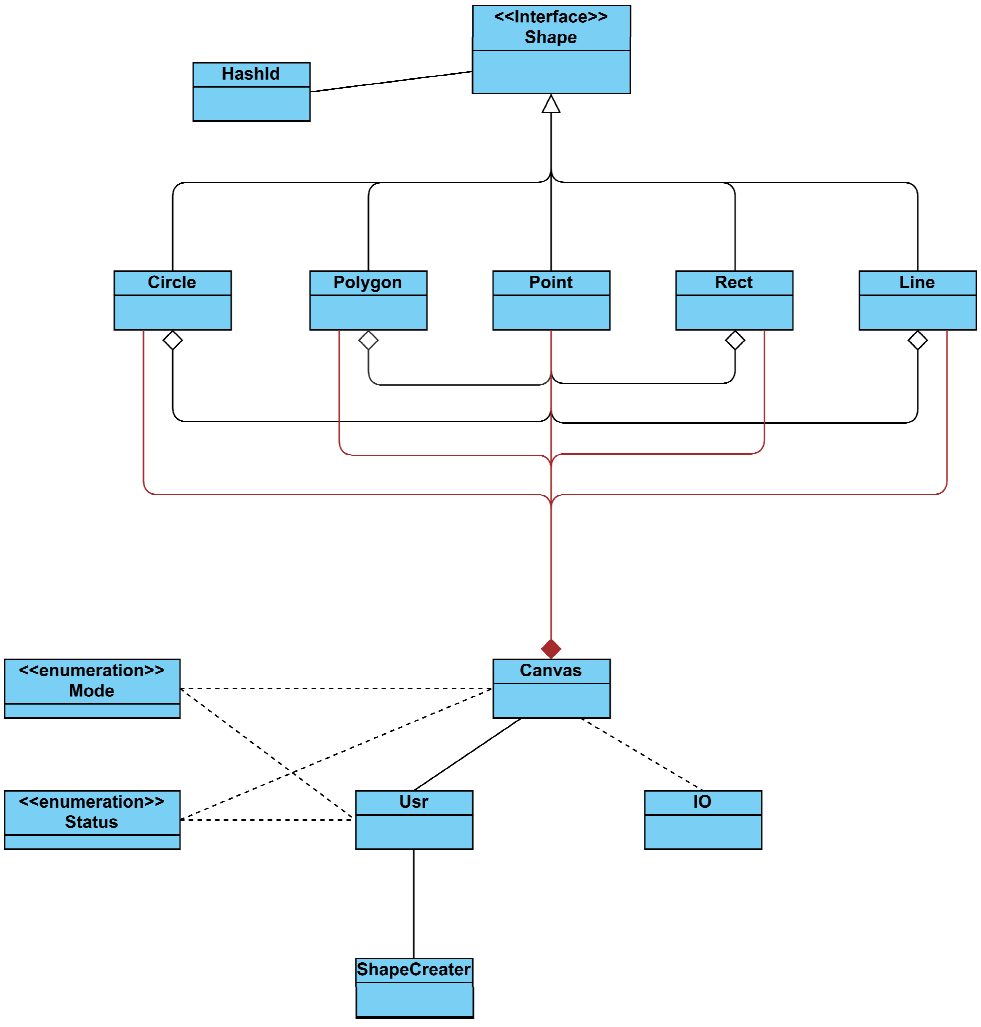
```
GL_ALPHA_TEST //alpha 测试功能，参考 glAlphaFunc 函数。
```

```
GL_BLEND //混合功能，将片元颜色和颜色缓冲区的颜色进行混合，参考 glBlendFunc。
```

# 4 新设计类功能说明

## 4.1 总体架构设计

下图是本项目整体设计的 UML 图（类的方法已经略去）：




## 4.2 类模块体系设计

- 顶层类是 `Shape` 类，各个图形要素抽象成 `Circle` `Polygon` `Point` `Rect` `Line` 类，均继承自 `Shape` 类，重写父类方法，同时自定义新的方法。

- HashId 是为画布上的每一个图形生成一个全局唯一的 Id 以供区分。
- Canvas 类负责启动初始化（销毁）窗口、控件绘制和各种图形的绘制，提供用户交互接口。
- User 类负责处理由 Canvas 类传来的用户的各个操作请求，如创建、删除图形，进行几何关系的构造等。
- Mode 是用户可选择的模式的枚举类。目前支持的模式有：
  - Move ：移动
  - Delete： 删除
  - Point：创建点
  - Line：创建直线
  - Circle：创建圆
  - Rect：创建矩形
  - Ploygon：创建多边形
  - Center：构造中点
  - Vertical：构造垂线
  - Tangent：构造切线
  - Reflect about point：中心对称

- Status 是在创建模式下的状态的枚举类。
- ShapeCreater 类是一个全局的构造器类，负责图形的创建和销毁，防止内存泄漏。
- IO 类是文件读写类，调用了 JsonCpp 库，负责保存文件和打开文件。

## 4.3 数据结构类设计

想要实现类似 Geogebra 的功能，图形的几何关联的保持势必是第一个要克服的问题。在本项目的设计中，Point 类和其他图形类有着非常密切的关系，其他图形的构造皆是由点构造出来的，比如圆的构造是圆和圆心，多边形的构造是每个顶点。

在 Circle, Polygon, Rect, Line 类中，均定义了 Point* 对象，使用

指针的原因是，这样在移动某个点时，造成的影响可以通过几何关系级联传递，视觉上的效果就是有几何关联的图形都会受到影响而发生改变。如果我们用不用指针而使用 `Point` 对象，造成的问题是：

- 丧失了几何关系的约束，几何上的交点在这种设计下并不是同一个点。
- 更新变得很麻烦，需要沿着有几何联系的图形逐个更新。

因此，本项目几何图形的储存结构是以**点为基本构造对象，其他几何图形保留点的指针来进行构造。**

## 4.4 源代码文件组织设计

**<文件目录结构>**

1）文件函数结构

```
│  CMakeLists.txt
│  JetBrainsMono-ExtraLight.ttf
│  main.cpp
├─geofiles
│  ├─another_sample
│  └─sample
├─lib
├─src
│  │  Canvas.cpp
│  │  Circle.cpp
│  │  IO.cpp
│  │  Line.cpp
│  │  Point.cpp
│  │  Polygon.cpp
│  │  Rect.cpp
│  │  ShapeCreater.cpp
│  │  Usr.cpp
│  └─include
│
└─thirdparty
    │  glad.c
    ├─dist
    ├─glad
    ├─GLFW
    ├─imgui
    └─KHR
```

2） 多文件构成机制

1. 文件包含

本项目的文件包含较为复杂，详情请看源代码。

2. #define 保护

```
#ifndef GEOPAD_CANVAS_H
#define GEOPAD_CANVAS_H

//...

#endif //GEOPAD_CANVAS_H
```

3. 外部变量使用

外部变量的使用会破坏封装性，本项目中除了在 Config.h 中使用了如下宏外，没有使用任何全局变量或函数。

```
#ifndef GEOGEBRA_CONFIG_H
#define GEOGEBRA_CONFIG_H

#define Pixel_X_2_Frame_X(x,w) (-1.0f + 2 * (x) / (w))
#define Pixel_Y_2_Frame_Y(y,h) (1.0f - 2 * (y) / (h))

#define Frame_2_Pixel_X(x,w) ((x)+1.0f)*(w)/2
#define Frame_2_Pixel_Y(y,h) (1.0f-(y))*(h)/2

#define NOT_ADDING -1

#define ADD_CENTER 0
#define ADD_FIRST_POINT 0
#define POLYGON_ADDING 0

#define THRESHOLD 0.008f

#define PRECISION 0.00000000001f

#endif //GEOGEBRA_CONFIG_H
```

## 4.5 重点类及函数设计描述

**1. 各个图形的绘制函数，这里以 Polygon 的绘制函数为例：**

```
void Draw() const override;
```

**功能描述：**通过判断是否填充、画线类型和颜色进行图形的绘制。

**参数描述：**无参数。

**重要局部变量：**无。

**算法实现步骤：**

Step1:

设置相关参数：

- 判断是否填充（`glPolygonMode(GL_FRONT, GL_FILL)`）
- 画线类型（`glEnable(GL_LINE_STIPPLE)`）
- 混合参数（`glEnable(GL_BLEND);`
  `glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);`）
- 颜色（`glColor4f(r, g, b, 0.2)`）

Step2: 调用 `glVertex2f(x, y)` 绘制点。

Step3: 设置 `glBegin(GL_LINE_LOOP)`，再调用 `glVertex2f(x, y)` 绘制闭合多边形。

### 2. 级联删除函数

```
void Usr::TryDelete(Canvas *canvas)
```

**功能描述：**对于选中的图形进行删除，级联删除所有相关的几何图形。

**参数描述：**当前画布的指针。

**重要局部变量：**`delete_queue` 队列。

**局部变量解释：**由于需要删除和被删除图形有几何关联的图形，我们可以把几何联系理解为一张**无向图（含有多个连通分量）**。每个节点是一个图形，节点之间有边表示有几何联系，我们需要从被删除的图形所在的节点出发，删除其所在的连通分量，`delete_queue` 表示当前在 BFS 队列中的图形。

算法实现步骤：

Step1: 首先将删除的图形加入 `delete_queue`。

Step2: 当队列不为空时，取队列头的图形节点，将所有与其有几何联系的图形加入队列并标记删除（`mark as deleted`）这些图形

Step3: 结束 BFS 后，真正的删除这些图形。

**3. 图形的创建**

在本设计中，图形的创建也依靠点的创建，以创建一个圆为例，我们先创建一个点做为圆心，再创建一个点做为圆周上的点，这两步中均判断创建点是否是已有点。“两步创建策略”适用于 `Circle` `Line` `Rect` 。对于 `Polygon` 的创建，则是不断创建点，直到最后一个点和第一个点重合。这一部分的代码均在 `User.cpp` 中的 `TryCreate` 开头的函数中实现。

**4. 图形的几何构造/变换**

前面提到我们创建的图形是基于点的创建的，而由点构造出的点如果可以拖动，那么图形的更新将会变得异常麻烦，而且也不符合几何关系的约束。在 `Geogebra` 中，构造出的图形是不允许拖动的。我的处理方法是在 `Point` 类中加入这个点是否是被构造出的标记，同时改变这个点在画布上的颜色以示区分。**主动点是蓝色，构造点是灰色。**

目前实现的构造有三个：中点、垂线和切线，实现方法就是数学计算出点的位置然后构造点，具体实现 `User.cpp` 的 `TryConstruct` 和 `TryTransform` 开头的函数里。

**5. 文件读写**

使用了 `JsonCpp` 库来保存，每个图形、构造和变换均单独写入一个文件。

# 5 部署运行和使用说明

## 5.1 编译安装

初次开发在 Ubuntu 20.04 上进行，因为环境配置方便快捷，但是考虑到在 Windows 平台占主流，于是在开发结束后迁移到了 Windows 上。Windows 平台的配置参考了这篇博客。

上交的是完整的 CLion 工程，如果有 CLion 可以直接打开。如果没有的话，工程里也有 CMakeLists.txt ，可以根据其重建工程。在迁移时**必需的文件结构**如下（文件夹内部文件没有展开列出）：

```
│  CMakeLists.txt
│  JetBrainsMono-ExtraLight.ttf
│  main.cpp
├─geofiles
│  ├─another_sample
│  └─sample
├─lib
├─src
│  │  Canvas.cpp
│  │  Circle.cpp
│  │  IO.cpp
│  │  Line.cpp
│  │  Point.cpp
│  │  Polygon.cpp
│  │  Rect.cpp
│  │  ShapeCreater.cpp
│  │  Usr.cpp
│  └─include
│
└─thirdparty
    │  glad.c
    ├─dist
    ├─glad
    ├─GLFW
    ├─imgui
    └─KHR
```

此外还上交了静态编译的版本，在 `GeoPad-Release` 文件夹中，可以直接点击 `GeoPad-Realease\bin` 文件夹内的可执行文件使用。

## 5.2 运行测试

详细请看操作演示视频。本项目的功能测试在操作演示中全部进行了展现。

## 5.3 使用操作

在**操作演示**文件夹内，我录制了四段操作演示视频可供理解操作方法。

### 侧栏


图 5.1 侧栏图片

左侧菜单栏是目前所有支持的模式，由上到下分别是：

- Move ：移动，**拖动点**进行移动
- Delete： 删除
- Point：创建点
- Line：创建直线，连续点击两个点进行创建
- Circle：创建圆，连续点击两个点进行创建。第一个点为圆心，第二个点为圆周上的点
- Rect：创建矩形，第一个点是左上角，第二个点是右下角
- Ploygon：创建多边形，连续创建点直到最后一个点和第一个点重合
- Center：构造中点
- Vertical：构造垂线，先选择一个点，再选择一条线
- Tangent：构造切线，先选择一个点，再选择一个圆
- Reflect about point：中心对称，先选择中心点，再选择一个图形（目前支持圆）

## 顶部菜单


图 5.2 顶部菜单栏

**顶部菜单栏是文件和设置，目前支持：**

- 打开文件，Ctrl-O
- 保存文件，Ctrl-S
- 调整画布颜色

## 右键菜单

将鼠标放在图形上单击右键可以选择改变颜色或者是线的类型（虚线或者是实线）。

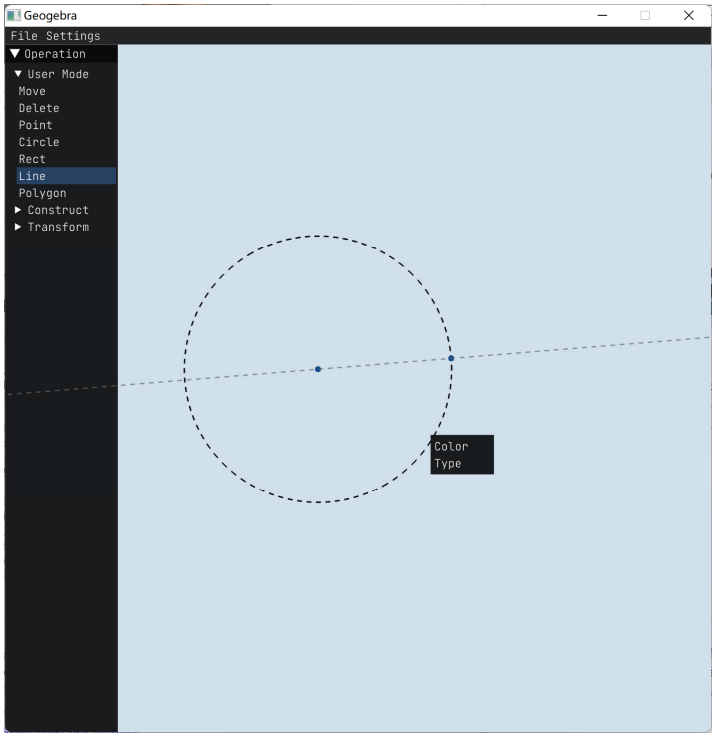



图 5.3 右键菜单

更多的操作演示请看**操作演示**的视频。

### 5.4 收获感言

开发过程遵循了 OOP 的三大基本特性：继承、封装和多态。继承在图形元素均继承自顶层 Shape 类中体现；封装则是在各个模块中都有体现，数据私有而方法公有；多态则体现在 Shape 的虚函数中，由子类实现，实现运行时多态。

代码规范方面，类名和函数名均使用大驼峰，变量名均使用下划线，具有良好的命名风格。此外在缩进和换行方面风格也统一。

总体来说，本次 GeoPad 的设计历时一周，花费了不少时间，因为时间有限，并没有设计很多功能，只是选取 Geogebra 的一部分功能进行了实现。但是在本设计下，其他功能是可扩展的，因此本设计具有良好的可扩展性。

## 6 参考文献资料

[1]https://stackoverflow.com/questions/401847/circle-rectangle-collision-detection-intersection

[2] https://www.daimajiaoliu.com/daima/4798f43a91003e8

[3] https://www.programminghunter.com/article/1730208776/

[4] https://www.jianshu.com/p/fd4cc689a20d

[5]https://stackoverflow.com/questions/2945337/how-to-detect-if-an-ellipse-intersects collides-with-a-circle