《计算机图形学》10 月报告

171860588，史文泰，171860588@smail.nju.edu.cn

2019 年 10 月 31 日

1 综述

计算机图形学课程的实习作业目标是完成简易绘图程序，主要内容分为 4 个模块：

- **核心算法模块**（图元生成算法，图元变换算法，以及整体程序的框架关系）
- **辅助功能模块**（为辅助核心算法以及用户交互模块而实现的功能，包括重置画布等）
- **文件输入输出接口**（包括读取指令文件并解析以及输出绘图结束后的图像）
- **用户交互模块**（包括鼠标，键盘与核心算法的交互，实现便捷绘制）

至本报告完成时，已在**核心算法模块**，**辅助功能模块**，**用户交互模块和文件输出接口**取得一定进展，并未涉及**文件输入接口**。

本报告只涉及**算法/功能原理概述**，代码实现详见《系统使用说明书》。

2 算法介绍

2.1 线段绘制算法（已完成）

2.1.1 DDA 算法

2.1.1.1 原理介绍

DDA 算法即**数值微分算法**，是一种基于直线的微分方程来生成直线的方法。它利用计算两个坐标方向的差分来确定线段显示的屏幕像素位置，是一种线段扫描转换算法。具体算法如下：

考虑直线具有正斜率$m=\frac{\Delta y}{\Delta x}$，且$0\leqslant m\leqslant 1$，则取单位$x$间隔（$\Delta x=1$）采样并计算每个对应的$y$值：

$$y_{k+1}=y_k+m$$

其中，下标k取整数值，并从第一个点开始递增，至最后端点。由于$0\leqslant m\leqslant 1$，为实数，计算出的y值必须取整。

若$m>1$，则取单位y间隔（$\Delta y=1$）采样，并计算连续的x值（增量为$\frac{1}{m}$）。

若$-1\leqslant m<0$，则取单位x间隔（$\Delta x=-1$）采样并计算每个对应的y值。

若$m<-1$，则取单位y间隔（$\Delta y=-1$）采样并计算每个对应的x值。

2.1.1.2 算法理解

DDA 算法直接利用了直线的坐标方程，但是并非使用方程直接计算出每个点的坐标，而是采用增量的思想，固定x轴和y轴方向的增量Δx和Δy中较大者，以此为单位间隔离散取样，逐步确定沿路每个像素的位置。

增量的思想消除了直线方程中的乘法运算，加快了算法执行速度。取增量较大者为单位间隔避免了取样点稀疏问题。

2.1.1.3 算法优化

DDA 算法在每次计算像素位置时涉及浮点运算和取整运算，两者均较为耗时。可将增量m和$\frac{1}{m}$分离成整数和小数部分使所有的计算都简化为整数操作。

DDA 算法的取整运算有累积误差，使长线段所计算的像素位置偏离实际线段。可采取增量叠加后取整的方式避免累积误差。以$0\leqslant m\leqslant 1$为例：

$$y_{k+1}=y_1+m_k$$
$$m_k=m_{k-1}+m$$

其中$m_0=0$，m_k（$k=1,2\ldots,n-1$）为实数。每次对y_k取整，可避免累积误差。

2.1.1.4 性能测试

因未实现文件输入模块，暂无法进行性能测试。

2.1.2 BresenHam 算法

2.1.2.1 原理介绍

BRESENHAM 算法是一种精确而有效的光栅线段生成算法。通过引入整形参量定义来衡量两候选像素与线段路径上实际数学点间在某方向上的相对偏移，并利用对整形参量符号的检测来确定最接近实际路径的像素。

令Δx和Δy分别为端点的水平和垂直偏离量，令$m=\frac{\Delta y}{\Delta x}$，定义算法执行到**第$k$步的决策参数**为$p_k$。以$0\leqslant m\leqslant 1$为例，则有：

$$y=mx+b$$
$$p_k=\Delta x(d_1-d_2)=2\Delta yx_k-2\Delta xy_k+c$$
$$c=2\Delta y+\Delta x(2b-1)$$

又由递推关系式可得：

$$p_{k+1}=2\Delta y x_{k+1}-2\Delta x y_{k+1}+c$$

两式相减可得：

$$p_{k+1}=p_k+2\Delta y(x_{k+1}-x_k)-2\Delta x(y_{k+1}-y_k)$$

像素的选择决定了$y_{k+1}-y_k$的值，故：

$$p_{k+1}=\begin{cases}p_k+2\Delta y-2\Delta x & (p_k>0，选择(x_k+1，y_k+1))\\ p_k+2\Delta y & (p_k<0，选择(x_k+1，y_k))\end{cases}$$

$$p_0=2\Delta y-\Delta x$$

若$m>1$，则交换x与y之间的规则，沿y方向为单位不长步进并计算即可。易知：

$$p_{k+1}=\begin{cases}p_k+2\Delta x-2\Delta y & (p_k>0，选择(x_k+1，y_k+1))\\ p_k+2\Delta x & (p_k<0，选择(x_k，y_k+1))\end{cases}$$

$$p_0=2\Delta y-\Delta x$$

若$m<0$，决策参数p_k与$|m|$的决策参数相同，仅是递进的方向改变，故不再赘述。

2.1.2.2 算法理解

BresenHam 算法利用了一个基本的事实，即实际直线方程与像素网格坐标系相交时，至多有两个候选点，而当限定增量较大者作为单位增量方向使得每次的增量较小候选坐标限定在当前坐标或其增量方向上的下一个坐标，避免了取整运算。

同时，算法巧妙地引入了决策参数p_k，由p_k的符号确定p_{k+1}和点的选择，使得整个运算过程只涉及加法运算，加快了算法的执行速度。

2.1.2.3 算法优化

当直线垂直，即$m=\infty$时，前述算法无法处理，需要单独考虑。此时只需要由y_0至y_n依次递增，即可得到对应的点。

拟采用线生成的并行算法，将线段分割成n_p个子段，并在每个子段中同时生成线段。

2.1.2.4 性能测试

因未实现文件输入模块，暂无法进行性能测试。

2.2 多边形绘制算法（已完成）

2.2.1 DDA 算法

2.2.1.1 原理介绍

对于多边形绘制的每一条边，使用线段绘制的 DDA 算法。

2.2.2 BresenHam 算法

2.2.2.1 原理介绍

对于多边形绘制的每一条边，使用线段绘制的 BresenHam 算法。

2.3 椭圆绘制算法（已完成）

2.3.1 中点椭圆生成算法

2.3.1.1 原理介绍

椭圆的生成可以简化为中心在原点的椭圆在第一象限内的椭圆弧的生成，其余三个象限利用对称性可以轻松得到，最后将椭圆平移至指定位置即可。由椭圆方程可知斜率m：

$$m=\frac{dy}{dx}=-2\frac{{r_y}^2x}{{r_x}^2y}$$

故在第一象限内由$m=-1$分割为两个区域，分割边界是：

$$2{r_y}^2x\geqslant 2{r_x}^2y$$

在上半部分，以x方向取单位步长；在下半部分以y方向取单位步长。

定义椭圆函数：

$$f_{ellipse}(x,\ y)={r_y}^2x^2+{r_x}^2y^2-{r_x}^2{r_y}^2$$

则在区域 1 中决策参数$p1_k$为：

$$p1_{k+1}=\begin{cases}p1_k+2{r_y}^2x_k+3{r_y}^2 & p_k<0\\ p1_k+2{r_y}^2x_k-2{r_x}^2y_k+2{r_x}^2+3{r_y}^2 & p_k\geqslant 0\end{cases}$$

$$p1_0={r_y}^2-{r_x}^2r_y+\frac{{r_x}^2}{4}$$

在区域 2 中决策参数$p2_k$为：

$$p2_{k+1}=\begin{cases}p2_k-2{r_x}^2y_k+3{r_x}^2 & p_k<0\\ p2_k+2{r_y}^2x_k-2{r_x}^2y_k+2{r_y}^2+3{r_x}^2 & p_k\geqslant 0\end{cases}$$

$$p2_0={r_y}^2\left(x+\frac{1}{2}\right)^2+{r_x}^2(y_k-1)^2-{r_x}^2{r_y}^2$$

2.3.1.2 算法理解

椭圆的中点生成算法其实是线段的 BresenHam 算法的另一个应用，均是通过决策参数p_k的符号从候选点中选出最接近实际方程的点。候选点的增多并

没有带来递推复杂性的增加。不同的一点是$|m|$会随着生成过程而跨越 1 的分界，故需要在分界处改变单位步长的选取。

2.3.1.3 算法优化

除了前述提及的利用对称性，可将$r_x{}^2$，$r_y{}^2$等常量提前算出，避免重复计算。**拟采用**并行算法，由x轴与y轴坐标轴处出发，通过不同的单位步长选取并行地处理两个区域，直至在分界处汇合。

2.3.1.4 性能测试

因未实现文件输入模块，暂无法进行性能测试。

2.4 圆生成算法（已完成）

2.4.1 中点圆生成算法

2.4.1.1 原理介绍

圆的生成可以视作椭圆生成的特例，其中$r_x = r_y$。也可以利用圆本身的对称性只计算$\frac{1}{8}$圆弧，避免了单位步长选取的改变。本作业中我选择了后一种方式。以下只给出决策参数p_k的推导式：

$$p_{k+1} = \begin{cases} p_k + 2x_k + 3 & p_k < 0 \\ p_k + 2x_{k+1} + 3 - 2y_k - 2 & p_k \geqslant 0 \end{cases}$$

$$p_0 = \frac{5}{4} - r$$

3 系统介绍

截至报告完成时，我使用了 Windows & QT & C++的环境，完成了初步的自由画图软件。在 QT 与 C++的交互中，我定义主窗口类 MainWindow 了一个画板类 Paint2DWidget，继承自 QWidget，用以进行绘画。**拟实现** Paint3DWidget 画板，用以进行简单的 3D 图形的绘制。

在 **MainWindow** 中进行软件外观的布局，并设置快捷键，完成读取文件，写出文件等操作。通过按钮、Spin Box 等控件的槽函数触发来向 Paint2DWidget 传递信号，控制绘画模式的改变。

在 **Paint2DWidget** 中接收信号，选择特定的绘画模式。监听鼠标的按下，释放和拖动事件，并实时调用对应的算法进行绘制，实现了实时交互的图形绘制。**拟实现**监听鼠标滚轮的滑动，并实现实时缩放图像，实时旋转，颜色填充等操作。

图像的绘制则采用**继承**的方式来实现。定义图像基类 Graphics，接着直线，椭圆，圆，多边形等图形继承 Graphics，便于使用动态绑定来管理画板上的所有图形。

系统框架的实现详见《系统使用说明书》

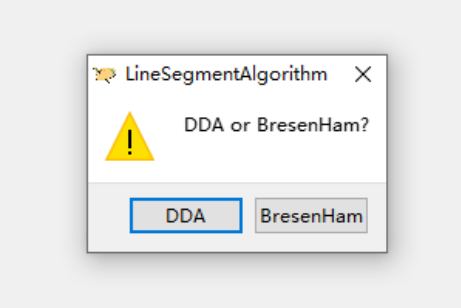
4 辅助功能介绍

- 实现了可自由画线的图形类：**RandomLine**，可任意绘画线段。
- 实现了图形橡皮擦，可以一笔擦除涉及到的所有图形。

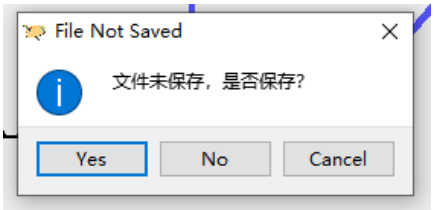
- 实现了撤回功能，可以撤回已经绘画完毕的图形。若当前正在画多边形，则可按撤回多边形的每一条边。**拟实现**撤回后的复原功能。
- 实现了线段与多边形的算法的自由选择。
- 实现了画笔宽度，画笔颜色的自由改变。
- 实现了生成带有图标的 exe 文件。

5 实验结果展示

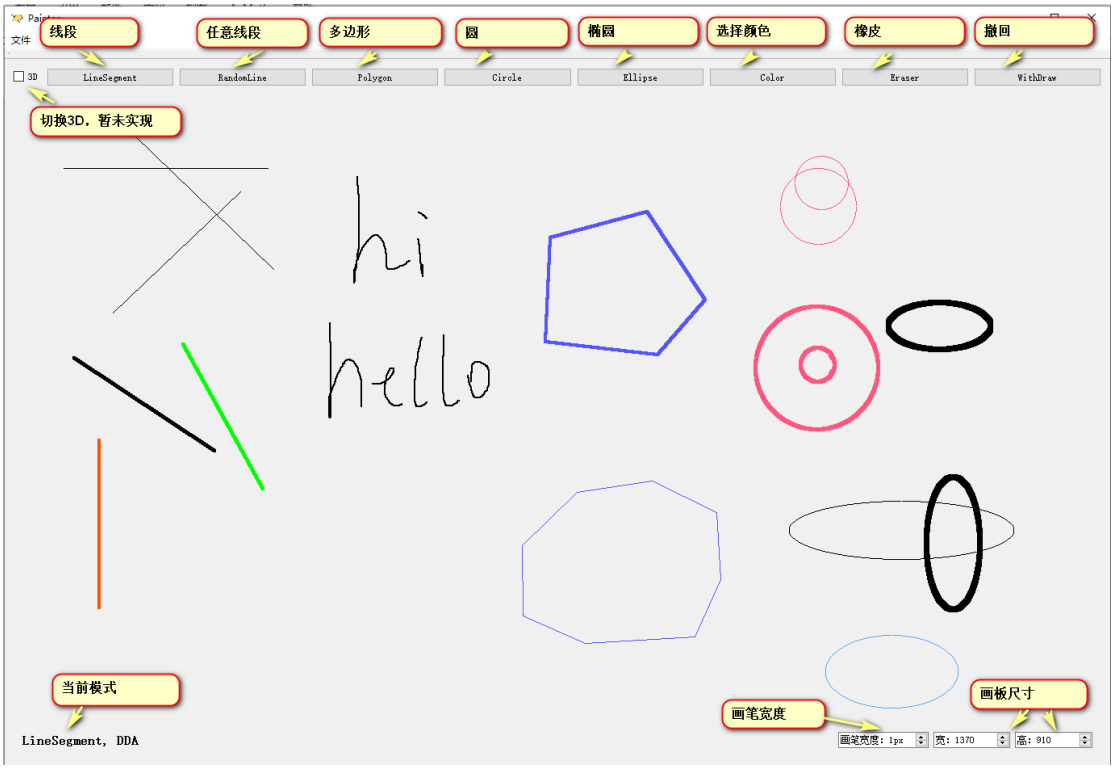
- 选择实现算法



- 保存文件



- 整体绘制效果及功能



6 总结

本次作业中我直接使用了 QT，而没有考虑读取输入文件，这导致性能测试无法进行，并不明智。但是实现了交互和核心算法后，命令的处理也不会成为大的困难。作业的难点在于算法的具体实现，交互的整体框架以及撤回等辅助功能的实现上。对于 QT，我刚刚入门，所以也遇到了很多困难，参考了很多技术博客，最终将他们解决。这次作业使我锻炼了编码能力，加深了对于图形绘制的算法的理解。

参考文献

[1] https://blog.csdn.net/qq_37233607/article/details/79303173
[2] https://blog.csdn.net/liang19890820/article/details/49874033
[3] https://blog.csdn.net/toby54king/article/details/78880227
[4] https://blog.csdn.net/fanyun_01/article/details/78379019
[5] https://blog.csdn.net/kabuto_hui/article/details/51288063
[6] https://blog.csdn.net/dong_zhihong/article/details/8208139
[7] http://www.voidcn.com/article/p-sflzniok-hx.html
[8] https://github.com/loveyu/simple-graph
[9] https://github.com/triumphalLiu/DrawingBoard
[10] https://github.com/Moya-Zhang/nju-Computer-Graphics
[11] https://blog.csdn.net/linuxwuj/article/details/78548043
[12] https://blog.csdn.net/king16304/article/details/52172115
[13] http://c.biancheng.net/qt/
[14] https://blog.csdn.net/dong_zhihong/article/details/8208139