《计算机图形学》6 月报告

161271033，赵铭南，zhaomn@smail.nju.edu.cn

2019 年 6 月 11 日

1 综述

本实验目的是利用课堂中学习到的各种算法实现能够绘制图形的命令行界面程序以及图形用户界面程序。命令行界面程序要求读取含有图元绘制指令序列的文本文件，依据指令调用自己实现的算法绘制图形以及保存图像；图形用户界面程序要求以鼠标交互的方式，通过鼠标事件获取所需参数并调用中的算法将图元绘制到屏幕上。

本报告第二部分算法介绍主要归纳了已完成算法的原理，第三部分系统介绍主要介绍已完成功能，第四部分总结。

2 算法介绍

目前已实现的算法有：绘制直线的 DDA 算法和 Bresenham 算法、绘制多边形的算法、绘制椭圆的中点圆算法、图元平移、图元旋转、图元缩放。

2.1 绘制线段

2.1.1 DDA 算法

数字差分分析 (DDA) 方法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的线段扫描转换算法。基本思路是，在一个坐标轴上以单位间隔对线段离散取样，确定另一个坐标轴上最靠近线段路径的对应整数值。

假设直线斜率为正，且起始点在终点左侧。设起始点为 (x_s, y_s)，终点为 (x_e, y_e)。则可计算出斜率 $k = \dfrac{y_s - y_e}{x_s - x_e}$。若斜率 $k \leq 1$，那么从起始点开始，在 x 坐标上每间隔 $\Delta x = 1$ 取样：$x_s, x_s + 1, ..., x_e$ 并且计算对应的 y：$y_0 = y_s, ..., y_i = y_{i-1} + k, ...$。若斜率 $k > 1$，那么从起始点开始，在 y 坐标上每间隔 $\Delta y = 1$ 取样：$y_s, y_s + 1, ..., y_e$ 并且计算对应的 x：$x_0 = x_s, ..., x_i = x_{i-1} + 1/k, ...$。最后将计算出的坐标两个分量都取整数，就得到这条直线应该显示的像素位置。

若直线斜率为正，且起始点在终点右侧。若斜率 $k \leq 1$，则取样变为 $x_s, x_s - 1, ..., x_e$，计算对应的 y：$y_0 = y_s, ..., y_i = y_{i-1} - k, ...$；若斜率 $k > 1$，则取样变为 $y_s, y_s - 1, ..., y_e$，计算对应的 x：$x_0 = x_s, ..., x_i = x_{i-1} - 1/k, ...$。

若直线斜率为负，且起始点在终点左侧。若斜率 $|k| \leq 1$，则取样变为 $x_s, x_s + 1, ..., x_e$，计算对应的 y：$y_0 = y_s, ..., y_i = y_{i-1} + k, ...$；若斜率 $k > 1$，则取样变为 $y_s, y_s - 1, ..., y_e$，计

算对应的 x：$x_0 = x_s, ..., x_i = x_{i-1} - 1/k, ...$。

若直线斜率为负，且起始点在终点右侧。若斜率 $|k| \leq 1$，则取样变为 $x_s, x_s - 1, ..., x_e$，计算对应的 y：$y_0 = y_s, ..., y_i = y_{i-1} - k, ...$；若斜率 $k > 1$，则取样变为 $y_s, y_s + 1, ..., y_e$，计算对应的 x：$x_0 = x_s, ..., x_i = x_{i-1} + 1/k, ...$。

DDA 算法是计算机图形学中最简单的绘制直线算法，其主要思想是由直线公式 $y = kx + b$ 推导出来的，但是存在浮点数运算与取整，因此该算法不利于硬件实现。

2.1.2 Bresenham 算法

Bresenham 算法的主要思想是采用递推步进的办法，令每次最大变化方向的坐标步进一个像素，同时另一个方向的坐标依据误差判别式的符号来决定是否也要步进一个像素。

假设斜率 m 小于 1，当前已确定第 k 个像素位于 (x_k, y_k) 处，通过计算

$$p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2b - 1)$$

来决定下一个像素点取 $(x_k + 1, y_k + 1)$ 还是 $(x_k + 1, y_k)$。若 $p_k > 0$，取 $(x_k + 1, y_k + 1)$；若 $p_k \leq 0$，取 $(x_k + 1, y_k)$。

另外由

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

我们可以同时计算出 p_{k+1}。若 $p_k > 0$，取 $(x_k + 1, y_k + 1)$，$p_{k+1} = p_k + 2\Delta y - 2\Delta x$；若 $p_k \leq 0$，取 $(x_k + 1, y_k)$，$p_{k+1} = p_k + 2\Delta y$。

因此，Bresenham 算法只需从端点 (x_0, y_0) 出发，先计算

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

通过不断迭代根据（1）计算 p_{k+1} 确定每个像素点的位置，直到到达另一个端点。

Bresenham 算法使得在求直线各点的过程中只涉及整数运算，减少了计算的复杂性，很好的符合了显示器（屏幕或打印机）系由像素构成的特性。

2.2 绘制多边形

绘制多边形即在相邻的两个顶点之间用 DDA 算法或者 Bresenham 算法画直线即可。

2.3 绘制椭圆

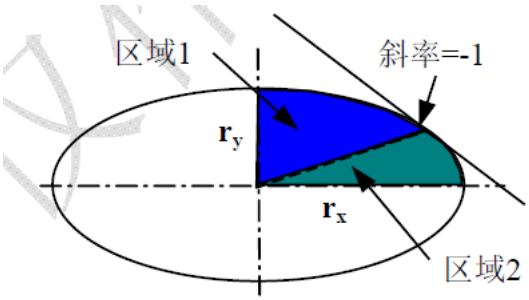
绘制椭圆只需绘制椭圆右上部分的图形，再将右上对称至其余部分即可。

2.3.1 中点圆算法

中点圆算法类似 Bresenham 画线算法，令每次最大变化方向的坐标步进一个像素，同时另一个方向的坐标依据判别式的符号来决定是否也要步进一个像素。

假设长轴长为 r_x，短轴长为 r_y。绘制椭圆右上部分图形时，需要将图形划分成两个区域进行绘制，如图所示，区域 1 进入区域 2 的条件是 $r_y^2 x \geq r_x^2 y$。

（1）区域 1



第一象限椭圆两个区域划分：

区域1中，椭圆切线斜率小于1；

区域2中，椭圆切线斜率大于1。

图 1: 椭圆的中点圆画法

假设当前确定了 (x_k, y_k)，x 方向以单位长度增加，y 方向根据决策参数决定不变或是减一，直到区域 2。决策参数为

$$p_k = r_y^2(x_k+1)^2 + r_x^2(y_k - \frac{1}{2})^2 - r_x^2 r_y^2.$$

从 $k=0$ 开始，$p_0 = r_y^2 - r_x^2 r_y + \frac{r_x^2}{4}$。若 $p_k \geq 0$，选择 (x_k+1, y_k-1) 作为下一个像素点，且更新 $p_{k+1} = p_k + 2r_y^2 x_k - 2r_x^2 y_k + 2r_x^2 + 3r_y^2$；若 $p_k < 0$，选择 (x_k+1, y_k) 作为下一个像素点，且更新 $p_{k+1} = p_k + 2r_y^2 x_k + 3r_y^2$。

（2）区域 2

假设当前确定了 (x_k, y_k)，y 方向以单位长度递减，x 方向根据决策参数决定不变或是加一，直到 y 到 0。决策参数为

$$p_k = r_y^2(x_k + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2.$$

若 $p_k > 0$,选择 (x_k+1, y_k-1) 作为下一个像素点,且更新 $p_{k+1} = p_k + 2r_y^2 x_k - 2r_x^2 y_k + 2r_y^2 + 3r_x^2$；若 $p_k \leq 0$，选择 (x_k, y_k-1) 作为下一个像素点，且更新 $p_{k+1} = p_k - 2r_x^2 x_y + 3r_x^2$。

2.4 绘制曲线

曲线上的每一点的坐标均可以被表示为某个参数的函数，如参数用 u 表示，则曲线上每一点笛卡尔坐标的参数式为：

$$\begin{cases} x = x(u), \\ y = y(u). \end{cases}$$

两个坐标分量组成曲线上该点的位置矢量，曲线就可以表示为参数 u 的失函数：

$$P(u) = (x(u), y(u)).$$

2.4.1 Bezier 曲线

n 次 Bernstein 基函数的多项式形式为:

$$BEZ_{i,n}(u) = C(n,i)u^i(1-u)^{n-i},$$

其中，$C(n,i) = \frac{n!}{i!(n-i)!}$。

Bezier 曲线是通过一组控制顶点唯一定义出的曲线。Bezier 曲线可以拟合任何数目的控制顶点，控制顶点也决定了 Bezier 曲线的阶数。Bezier 曲线可以由给定边界条件、特征矩阵或混合函数决定。对一般 Bezier 曲线来说，最方便的是混合函数形式。

给定 $n+1$ 个控制顶点位置：$P_i = (x_i, y_i)(i = 0, 1, 2..., n)$。这些控制顶点混合产生位置向量 $P(u)$：

$$P(u) = \sum_{i=0}^{n} P_i BEZ_{i,n}(u),\ (0 \le u \le 1).$$

利用 Bernstein 基函数的降（升）阶公式，可以递归计算 Bezier 曲线上点的坐标位置。对某一特定的参数 u，其计算公式为：

$$P_i^r = \begin{cases} P_i, (r = 0) \\ (1-u)P_i^{r-1} + uP_i^{r-1}, (r = 1, 2, ..., n; i = 0, 1, 2, ..., n-r). \end{cases}$$

当 $r = 0$ 时，计算结果为控制点本身；曲线上的型值点为：$P(u) = P_0^n$。

2.4.2 B-spline 曲线

B 样条基函数：给定参数 u 轴上的节点分割：$U_{n,k} = \{u_i\}(i = 0, 1, 2, ..., n+k)$，称由 deBoox-Cox 递推关系所确定的 $B_{i\,k}(u)$ 为 $U_{n,k}$ 上的 k 阶 B 样条基函数。

已知 $n+1$ 个控制顶点 $\{P_i\}(i = 0, 1, 2, ..., n)$ 及参数节点向量 $U_{n,k} = \{u_i\}(i = 0, 1, 2, ..., n+k; u_i \le u_{i+1})$，称如下形式的参数曲线 $P(u)$ 为 k 阶 B 样条曲线：

$$P(u) = \sum_{i=0}^{n} P_i B_{i,k}(u),\ (u_{k-1} \le u \le u_{n+1}).$$

3 阶 B 样条曲线的生成可通过对每 3 个连续的控制顶点 $P_i, P_{i+1}, P_{i+2}(i = 0, 1, 2, ..., n-1)$ 进行拟合成一条 3 阶均匀 B 样条曲线完成。三阶均匀 B 样条曲线段的矩阵表示为：

$$P(u) = \frac{1}{2}\begin{bmatrix} u^2 & u & 1 \end{bmatrix}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}.$$

2.5 图元平移

像素点 (x, y) 以 (dx, dy) 为向量平移后的坐标为：

$$\begin{cases} x_{new} = x + dx, \\ y_{new} = y + dy. \end{cases}$$

对于线段和多边形平移，只需平移端点即可，然后绘制出相应的线段；对于椭圆，只需平移中心，然后绘制出相应的椭圆即可; 对于曲线，只需平移控制顶点，然后绘制相应曲线即可。

2.6 图元旋转

像素点 (x, y) 以 (x_0, y_0) 为旋转中心，顺时针旋转 r 后的坐标为：

$$\begin{cases} x_{new} = (x - x_0)\cos r - (y - y_0)\sin r + x_0, \\ y_{new} = (y - y_0)\cos r + (x - x_0)\sin r + y_0. \end{cases}$$

对于线段和多边形，只需旋转端点即可，然后绘制出相应的线段；对于曲线，只需旋转控制顶点，然后绘制相应曲线即可。

2.7 图元缩放

像素点 (x, y) 以 (x_0, y_0) 为缩放中心，缩放 s 倍后的坐标为：

$$\begin{cases} x_{new} = (x - x_0)s + x_0, \\ y_{new} = (y - y_0)s + y_0. \end{cases}$$

对于线段和多边形，只需缩放端点即可，然后绘制出相应的线段；对于椭圆，只需缩放中心以及将长短轴缩放对应倍数，然后绘制出相应的椭圆即可；对于曲线，只需缩放控制顶点，然后绘制相应曲线即可。

2.8 线段裁剪

2.8.1 Cohen-Sutherland 算法

Cohen-Sutherland 算法也叫编码算法，编码算法以如图所示的 9 个区域为举出，依据每条线段的端点所在的区域，赋予每个端点区域码（四位二进制码）。

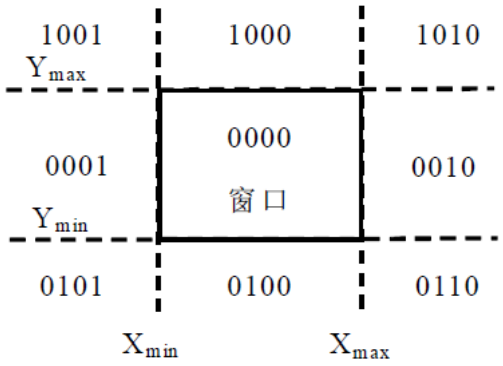


图 2: 区域编码

算法流程大致如下：

1. 给两个端点 P_1, P_2 赋予区域码；
2. 如果两个区域码都是 0000，说明线段在窗口内部，不用裁剪，直接结束；
3. 如果 $C1\&C2 == 0$ 成立，说明线段完全在窗口外部，直接舍弃，结束；
4. 计算线段与窗口的交点，用该交点替代一个端点，不断重复 1-4;

Cohen-Sutherland 算法用编码方法可快速判断线段完全可见或完全不可见，减少了一部分求交点计算，直观而方便，速度也较快。

2.8.2 Liang-Barsky 算法

Liang-Barsky 算法主要思想是：用参数方程表示直线，把被裁剪的线段看成是一条有方向的线段，把窗口的四条边分成入边和出边。裁剪结果的线段起点是直线和两条入边的交点以及始端点三个点里最前面的一个点，即参数 u 最大的那个点；裁剪线段的终点是和两条出边的交点以及端点最后面的一个点，即参数 u 最小的那个点。

一条端点为 $P_1(x_1, y_1), P_2(x_2, y_2)$ 的线段的参数方程为：

$$\begin{cases} x = x_1 + (x_2 - x_1)u = x_1 + u \cdot \Delta x, \\ y = y_1 + (y_2 - y_1)u = y_1 + u \cdot \Delta y. \end{cases}$$

那么裁剪后线段上的点的参数 u 都应满足：

$$\begin{cases} x_{min} \leq x_1 + u \cdot \Delta x \leq x_{max}, \\ y_{min} \leq y_1 + u \cdot \Delta y \leq y_{max}. \end{cases}$$

可以统一表示成 $u \cdot p_k \leq q_k, k = 1, 2, 3, 4$。

因此可以通过 p_k 来确定线段和裁剪窗口的相对关系。任何平行于窗口某边界的直线，其 $p_k = 0$，k 值对应于相应的边界（$k = 1, 2, 3, 4$ 对应于左、右、下、上边界），如果还满足 $q_k < 0$，则线段完全在边界外，应舍弃该线段。如果 $p_k = 0$ 并且 $q_k \geq 0$，则线段平行于窗口某边界并在窗口内。如果 $p_k < 0$，线段从线段从裁剪边界延长线的外部延伸到内部；如果 $p_k > 0$，线段从裁剪边界延长线的内部延伸到外部。

当 $p_k \neq 0$ 时，可以计算出参数 $u = q_k/p_k$ 的值，它对应于无限延伸的直线与延伸的窗口边界 k 的交点。对于每条直线，可以计算出参数 u_1 和 u_2，该值定义了位于窗口内的线段部分：

1. u_1 的值由线段从外到内遇到的矩形边界所决定（$p_k < 0$），对这些边界计算 $r_k = q_k/p_k$，u_1 取 0 和各个 r 值之中的最大值;
2. u_2 的值由线段从内到外遇到的矩形边界所决定（$p_k > 0$），对这些边界计算 $r_k = q_k/p_k$，u_2 取 0 和各个 r 值之中的最小值;
3. 如果 $u_1 > u_2$，则线段完全落在裁剪窗口之外，应当被舍弃；否则，被裁剪线段的端点可以由 u_1 和 u_2 计算出来。

算法流程大致如下：

1. 初始化线段交点的参数：$u_1 = 0$，$u_2 = 1$；
2. 计算出各个裁剪边界的 p、q 值；
3. 根据 p、q 来判断：是舍弃线段还是改变交点的参数。
4. p、q 的四个值经判断后，如果该线段未被舍弃，则裁剪线段的端点坐标由参数 u_1 和 u_2 的值决定。

Liang-Barsky 算法较 Cohen-Sutherland 算法而言，更为有效，求交点的次数减少了，计算也简便。

3 系统介绍

3.1 命令行界面程序

命令行界面程序 (CGConsole.exe) 要求输入含有图元绘制指令序列的文本文件的保存路径和图像保存路径，然后它会读取含有图元绘制指令序列的文本文件，执行每条指令相对应的动作。

目前命令行界面程序实现的功能有：重置画布、保存画布、设置画笔颜色、绘制线段、绘制多边形、绘制椭圆、绘制曲线、图元平移、图元旋转、图元缩放、线段裁剪。

3.1.1 重置画布

图元绘制指令：restCanvas width height

功能：清空当前画布，并重新设置宽高

实现函数：QImage resetCanvas(int width, int height)

3.1.2 保存画布

图元绘制指令：saveCanvas name

功能: 将当前画布保存为位图 name.bmp

实现函数：void saveCanvas(QImage &picture,string name)

3.1.3 设置画笔颜色

图元绘制指令：setColor R G B

功能: 将当前画笔颜色设置为 RGB 颜色

实现函数：void setColor(QPainter &painter, int R, int G, int B)

3.1.4 绘制线段

图元绘制指令：drawLine id x1 y1 x2 y2 algorithm

功能: 使用算法绘制从 (x_1, y_1) 到 (x_2, y_2) 的线段。

实现函数：void drawLine(QPainter &painter, float x1, float y1, float x2, float y2, string algorithm);

3.1.5 绘制多边形

图元绘制指令：drawPolygon id n algorithm x_1 y_1 $\cdots x_n$ y_n

功能: 使用算法绘制以 $(x_1, y_1)...(x_n, y_n)$ 为顶点的多边形。

实现函数：void drawPolygon(QPainter &painter, int n, vector<float> &x, vector<float> &y, string algorithm);

3.1.6 绘制椭圆

图元绘制指令：drawEllipse id x y rx ry

功能: 绘制以 (x, y) 为中心，rx 为长轴长，ry 为短轴长的椭圆。

实现函数：void drawEllipse(QPainter &painter, float x, float y, float rx, float ry);

3.1.7 绘制曲线

图元绘制指令：drawCurve id n algorithm x_1 y_1 $\cdots x_n$ y_n

功能: 使用算法绘制以 $(x_1, y_1)...(x_n, y_n)$ 为顶点的 Bezier 曲线或 B-spline 曲线。

实现函数：void drawCurve(QPainter &painter, int n, vector<float> &x, vector<float> &y, string algorithm);

3.1.8 图元平移

图元绘制指令：translate id dx dy

功能：以 (dx, dy) 为平移向量，将 id 对应的图元进行平移。

实现函数：void translate(QPainter &painter, int id, float dx, float dy);

3.1.9 图元旋转

图元绘制指令：rotate id x y r

功能：以 (x, y) 为旋转中心，将 id 对应的图元旋转 r 角度。

实现函数：void rotate(QPainter &painter, int id, float x, float y, float r);

3.1.10 图元缩放

图元绘制指令：scale id x y s

功能：以 (x, y) 为缩放中心，将 id 对应的图元进行缩放 s 倍。

实现函数：void scale(QPainter &painter, int id, float x, float y, float s);

3.1.11 线段裁剪

图元绘制指令：clip id x1 y1 x2 y2 algorithm

功能：将 id 对应的线段进行裁剪，裁剪窗口由 (x_1, y_1) 与 (x_2, y_2) 确定。

实现函数：void clip(QPainter &painter, int id, float xmin, float ymin, float xmax, float ymax, string algorihtm);

3.2 图形用户界面程序

图形用户界面程序 (CGGUI.exe) 目前实现的功能有：重置画布、保存画布、设置画笔颜色、绘制线段、绘制多边形、绘制椭圆、绘制曲线、图元平移、图元旋转、图元缩放、线段裁剪，该程序中图元的 id 是其被绘制的顺序（从 1 开始记录），重置画布后 id 会从 1 开始记录。

图形用户界面程序 (CGGUI.exe) 主界面如图 3 所示，蓝色区域的两个按钮分别是保存画布和重置画布功能按钮，红色区域的按钮是绘制按钮，可以绘制线段、绘制多边形、绘制椭圆、绘制曲线，黄色部分的按钮和输入框和图元平移、图元旋转、图元缩放、线段裁剪功能有关，黑色部分的滑动条和输入框可以用来改变画笔颜色。

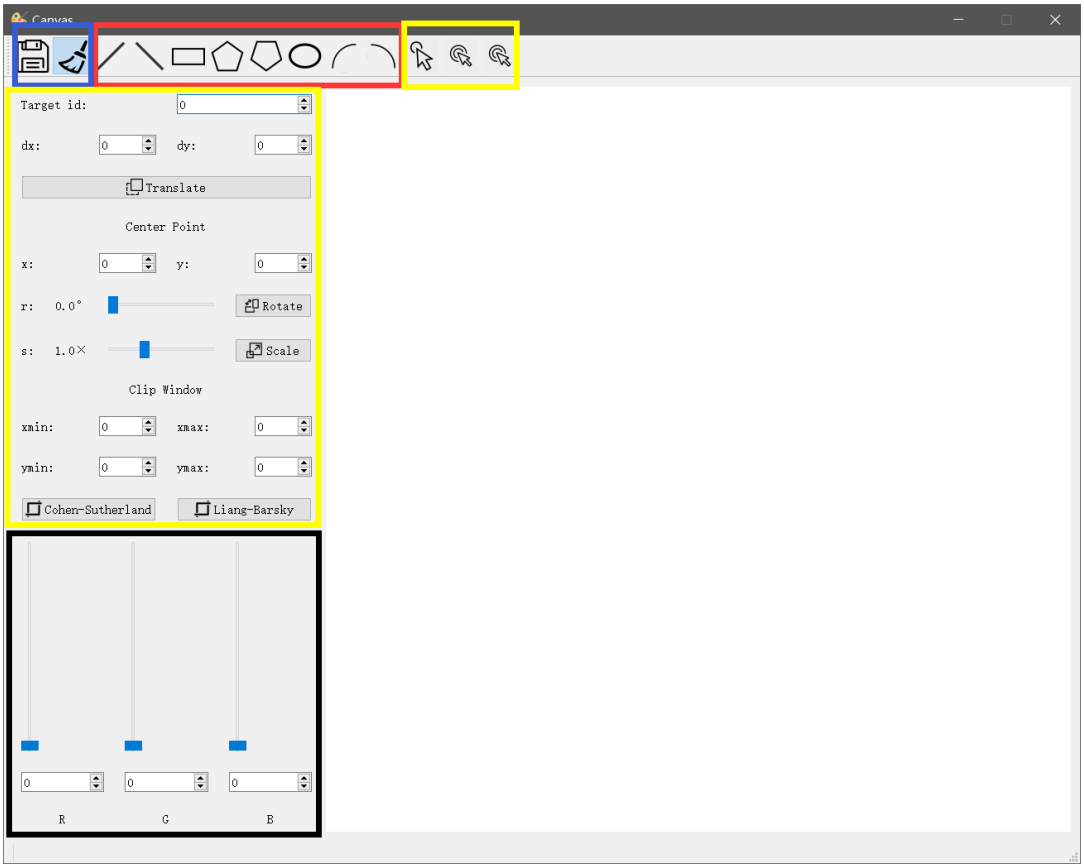


图 3: CGGUI.exe

下面介绍使用方法，更详细的介绍可以看系统使用书。

3.2.1 重置画布

点击按纽栏的第二个按钮即可清空画布，重置画布后新绘制的图元的 id 会从 1 开始记录。

3.2.2 保存画布

点击按纽栏的第一个按钮后，选择路径并输入文件名，即可保存画布。

3.2.3 设置画笔颜色

在黑色区域，通过滑动滑动条或者在输入框中输入数字或者点击箭头改变 R、G、B 对应的值就可以设置画笔颜色。

3.2.4 绘制线段

点击按纽栏的第三个（第四个）按钮即可进入用 DDA 算法 (Bresenham 算法) 绘制直线的模式。左键点击画布中任意一处，并按住左键移动即可绘制直线，松开左键代表绘制结束。

3.2.5 绘制矩形

点击按纽栏的第五个按钮即可进入绘制矩形的模式。左键点击画布中任意一处，并按住左键移动即可绘制矩形，松开左键代表绘制结束。

3.2.6 绘制多边形

点击按纽栏的第六个（第七个）按钮即可进入用 DDA 算法 (Bresenham 算法) 绘制多边形的模式。通过左键单击画布中的任意一处添加顶点，左键每点击一次就添加一个顶点，画布将显示所有顶点生成的多边形。再次点击按钮或者画布中单击右键表示目前多边形绘制结束，可开始新的多边形绘制。

3.2.7 绘制椭圆

点击按纽栏的第八个按钮即可进入绘制椭圆的模式。左键点击画布中任意一处，并按住左键移动即可绘制椭圆，松开左键代表绘制结束。

3.2.8 绘制曲线

点击按纽栏的第九个（第十个）按钮即可进入用绘制 Bezier 曲线（B-spline 曲线）的模式。通过左键单击画布中的任意一处添加控制点，左键每点击一次就添加一个控制点，画布将显示所有控制点生成的曲线。再次点击按钮或者画布中单击右键表示目前曲线绘制结束，可开始新的曲线绘制。

3.2.9 图元平移

在 tartget id 处输入被平移的图元 id，该程序中图元的 id 是其被绘制的顺序（从 1 开始记录），在 dx 和 dy 处输入平移向量后，点击 translate 按钮就可以完成对 target id 对应图元的平移操作。

3.2.10 图元旋转

在 tartget id 处输入被旋转的图元 id，该程序中图元的 id 是其被绘制的顺序（从 1 开始记录）。其次选择旋转中心，两种方法：1、在 x 输入框处输入旋转中心的 x 坐标，在 y 输入框处输入旋转中心的 y 坐标；2、点击按纽栏的第十一个按钮进入选择选择中心的模式，单击画布中的任意一处即可将对应位置设置成旋转中心。然后选择旋转角度，通过拖动滑动条选择旋转角度，最后点击 rotate 按钮就可以完成对 target id 对应图元的旋转操作。

3.2.11 图元缩放

在 tartget id 处输入被缩放的图元 id，该程序中图元的 id 是其被绘制的顺序（从 1 开始记录）。其次选择缩放中心，两种方法：1、在 x 输入框处输入缩放中心的 x 坐标，在 y 输入框处输入缩放中心的 y 坐标；2、点击按纽栏的第十一个按钮进入选择缩放中心的模式，单击画布中的任意一处即可将对应位置设置成缩放中心。然后选择缩放倍数，通过拖动滑动条选择缩放倍数，最后点击 scale 按钮就可以完成对 target id 对应图元的缩放操作。

3.2.12 线段裁剪

在 tartget id 处输入被缩放的图元 id，该程序中图元的 id 是其被绘制的顺序（从 1 开始记录）。其次选择裁剪窗口，两种方法：1、在 xmin 输入框处输入裁剪窗口左下顶点的 x 坐标，在 ymin 输入框处输入裁剪窗口左下顶点的 y 坐标，在 xmax 输入框处输入裁剪窗口右上顶点的 x 坐标，在 ymax 输入框处输入裁剪窗口右上顶点的 y 坐标；2、单击按纽栏的第十二个按钮进入选择裁剪窗口左下顶点的模式，单击画布中的任意一处即可将对应位置设置成裁剪窗口左下顶点，然后单击按纽栏的第十三个按钮进入选择裁剪窗口右上顶点的模式，单击画布中的任意一处即可将对应位置设置成裁剪窗口右上顶点（确保单击第十二个按钮后选择的点位于单击第十三个按钮后选择的点的左下方）。最后点击 Cohen-Sutherland（Liang-Barsky）按钮就可以使用 Cohen-Sutherland（Liang-Barsky）算法完成对 target id 对应线段的裁剪操作。

4 总结

利用课堂中学习到的各种算法实现能够绘制图形的命令行界面程序以及图形用户界面程序，目前有命令行界面程序以及图形用户界面程序，实现的功能有包括：重置画布、保存画布、设置画笔颜色、绘制线段、绘制多边形、绘制椭圆、绘制曲线、图元平移、图元旋转、图元缩放、线段裁剪。

参考文献

孙正兴. 计算机图形学教程. DynoMedia Inc.; 2006.