计算机图形学实验 进度报告/系统报告

学号：**171860635** 姓名：陈振宇

10月30日报告

目前已经完成的内容：

1. 使用java语言和swing/awt框架，搭建了图形化界面；
2. 完全完成了命令解析模块，完成了基本的错误报告机制；
3. 为后续可能添加的鼠标操作功能做了准备；
4. 完成了DDA直线绘制算法和Bersenham直线绘制算法，在此之上完成了对应的两种多边形绘制算法；
5. 完成了图形平移功能，若当前画布上的所有图形都使用已经完成的算法，则可以完成平移操作；
6. 完成了对于 `<Integer,Shape>` 键值对的保存功能，用于实现平移/旋转/缩放时的重绘。

框架/图形化界面相关内容：

目前的代码调用可以分为3个层次：Main->MyMouseListener->MyGraphics，“->”代表调用。其中，Main类中包含了main函数和层次调用信息，MyMouseListener继承了MouseListener，主要包含的功能是命令解析和鼠标操作解析。MyGraphics中包含了主界面中的JFrame，JPanel和对应JPanel的Graphics，主要完成了绘图逻辑。从下面这段main函数中的代码可以看出层次调用关系，其中g为 `p.getGraphics()` ， p为 `JPanel` ， f为 `JFrame` ；

```
MyGraphics mg=new MyGraphics();
mg.init(g,p,f);
MyMouseListener mml=new MyMouseListener();
mml.init(mg);
p.addMouseListener(mml);
```

考虑到画板大小不会超过1000x1000，主界面的分辨率被设置为1280x1024，且不可更改，

命令解析模块相关内容：

命令解析模块完成的功能很容易理解，鉴于所需要实现的命令不多且后续不会更改，在这里没有使用表驱动而是使用了普通的 `if{}else{}` 进行解析。命令解析模块每次将文件中的一行读入 `String` 中，再将这个 `String` 通过 `split()` 分解为 `String[] token` 。

```
BufferedReader in=new BufferedReader(new FileReader(path));
String str;
while((str=in.readLine())!=null)
{
    commands.add(str);
}


String[] token=commands.get(i).split(" ");
```

为了防止读取过程中代码的错误和自己完成的测试用例中产生的错误，在命令解析模块中实现了简单的报错机制。这个报错机制主要处理 token 中的数量错误，在词组数量不正确时打印错误信息和命令文件中产生错误的行号，同时抛出一个 Exception 。

```
private void printError(String[] token,int len,int line)
{
    try {
        if (token.length != len) {
            System.out.println("Error @ MyMouseListener @ resolveCommands: invalid command @ lir
            throw (new Exception());
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    }

}
```

算法模块相关内容：

算法部分在 MyGraphics 模块中完成。考虑到现有的框架不提供对于像素的操作，但是 BufferedImage 提供对于像素的操作，同时 Graphics 类中提供了显示 BufferedImage 的简洁接口， ImageIO 类中提供了保存 BufferedImage 为各种格式的接口，所以采用的思路是在 BufferedImage 中绘制需要的图片，在需要时与 JPanel 同步，同时保存时仅涉及对于 BufferedImage 的保存。

```
public void save(String name)
{
    print("save");
    //保存为bmp格式
    try{
        ImageIO.write(img,"bmp",new File(name+".bmp"));
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}


private void panelSync() {originalGraphics.drawImage(img,0,0,null);}


private void drawPixel(int x,int y)
{
    int rgb_int=originalGraphics.getColor().getRGB();
    img.setRGB(x,y,rgb_int);
    //panelSync();//效率太低。
}
```

算法相关内容：

DDA直线算法相比Bresenham直线算法较为简单。对于特殊情况的直线，如垂直于x轴或y轴的直线，要单独做特殊处理。对于其他所有情况下的直线，考虑直线的斜截式：

$$y = kx + b$$

先考虑$|k| < 1$的情况。在这种情况下，以x作为算法的循环体。直线的斜率可以用另一种形式表达：

$$k = \frac{\Delta y}{\Delta x}$$

在$\Delta x = 1$的情况下，也就是一般化DDA算法中循环体的步长，$\Delta y = k$，所以在每次循环中要将y（作为一个double变量）加上k。同理，在$|k| \geq 1$的情况下，以y作为循环体，每次循环中x（作为一个double变量）加上$\frac{1}{k}$即可。每次取点时，利用 `Math.round()` 操作，将x或者y四舍五入取整即可完成画点操作。

Bresenham算法相较而言更为复杂。除去两种特殊情况之外，算法整体分为8种情况，最终可以归类为2种情况，方便代码完成。两种特殊情况同DDA直线算法中的特殊情况，分别是直线垂直于x轴和直线垂直于y轴，这两种情况都很好完成。
下面给出一种泛用式Bresenham算法的推导过程，只需要考虑两种情况即可。先考虑第一种情况，直线

的斜率绝对值在$(0,1]$中。
设需要画线的两个点分别是(x_b,y_b)和(x_b,y_b)。其中b和e分别代表着begin和end。设通过算法画出的第i个点为(x_i,y_i)，其中i从0开始计数。
定义$\Delta x=x_e-x_b$，$\Delta y=y_e-y_b$。定义x_s和y_s，对应代码中的 sx 和 sy ，代表x和y的步长，取值为1或-1。当$x_e>x_b$时，$x_s=1$，否则为-1。y_s同理。
设决策参数为T，T的定义将会在之后完成。定义所需要解决的问题A：已知$(x_i,y_i)=(x_t,y_t)$，以及T_i，其中T_i是用来决策第$i+1$个点的决策参数。现在要求根据Bresenham直线算法求(x_{i+1},y_{i+1})和T_{i+1}。
决策参数应该如何规定？假设现在我们要在两个整数点之间做决策，两个整数点的y坐标分别是y_{bias}和y_{same}，定义$d_1=y_{bias}-y$，$d_2=y-y_{same}$，其中y为在这个横坐标处真正的y值。一个简单易懂的定义是$T=|\Delta x|(|d_2|-|d_1|)$，这样当$T>0$时，显然要选择$d_1$对应的点，也就是$y_{bias}$。当$T\leq 0$时，选择另一个点。这是对本程序中采用算法的决策参数的一个直观的理解。
下面给出T的严格定义：T_{i+1}为算法中预测(x_{i+2},y_{i+2})的决策参数。$T_{i+1}=|\Delta x|(|d_2|-|d_1|)$，其中$d_1=y_{bias}-y_{real}$，$d_2=y_{real}-y_{same}$，$y_{bias}=y_{i+1}+y_s$，$y_{same}=y_{i+1}$，$y_{real}=k(x_{i+1}+x_s)+b$。
y_{bias}和y_{same}可以理解为与前一个点（也就是(x_{i+1},y_{i+1})，这个点在算法中要计算T_{i+1}的时候是已知的）在同一行的坐标和“偏差一行”的坐标。y_{real}是所预测点的准确y坐标。根据上一段，在$T_{i+1}>0$时，我们选择$(x_{i+1}+x_s,y_{bias})$这个点，否则选择$(x_{i+1}+x_s,y_{same})$。
T_{i+1}之所以用$i+1$作为下标而不是用$i+2$作为下标，是因为T_{i+1}使用了(x_{i+1},y_{i+1})的信息作为判断依据。在不使用迭代计算的时候，使用上一个点的信息和直线方程信息就可以判断出下一个点的位置。
根据定义很容易完成对于T_{i+1}的计算。绝对值是一个难以处理的地方，但是好在我们已经定义了x_s和y_s，可以将这两个值与许多值相乘得到绝对值。d_1和d_2的符号（可以理解为“走向”）和Δy是相同的（考虑y_s的定义）,同时Δx和x_s的符号也是相同的。这样可以将T_{i+1}表示为：

$$T_{i+1}=(\Delta x x_s)(d_2y_s-d_1y_s)=(\Delta x x_s)(d_2-d_1)y_s$$

考虑到$k=\frac{\Delta y}{\Delta x}$，上式可以被化简为

$$T_{i+1}=2x_sy_s\Delta yx_{i+1}-2x_sy_s\Delta xy_{i+1}+C$$

其中C是一个常数，这个常数将会在接下来的迭代运算中消除。
考虑到$\Delta x x_s=|\Delta x|$，$\Delta y y_s=|\Delta y|$：

$$T_{i+1}=2y_s|\Delta x|x_{i+1}-2y_s|\Delta x|y_{i+1}+C$$

进行迭代运算：

$$T_{i+1}-T_i=2|\Delta y|-2y_s|\Delta x|(y_{i+1}-y_i)$$

这样就得出了迭代表达T的公式。
若$y_{i+1}=y_i$，即$T_i<0$时（T_i和y_{i+1}可以互相进行判断）：

$$T_{i+1}=T_i+2|\Delta y|$$

否则

$$T_{i+1}=T_i+2|\Delta y|-2y_s^2|\Delta x|=T_i+2|\Delta y|-2|\Delta x|$$

现在迭代公式已经完成了，只剩下一个初始值T_0。鉴于之前的公式中常数都被C隐藏了，现在从原始公式开始计算。将(x_0,y_0)代入原始公式即可，其中b利用(x_0,y_0)消去。

$$T_0=\Delta x x_s y_s(2y_{real}-y_{same}-y_{bias})$$

代入计算之后：

$$T_0=2x_s|\Delta y|-|\Delta x|$$

此时整个算法完成。

对于剩下一种情况的计算，将上述公式中所有的x和y互相替换即可。