哈尔滨工业大学

<<计算机图形学>>

实验报告

(2017 年度春季学期)

姓名:	张茗帅
学号:	1140310606
学院:	计算机科学与技术学院
教师:	唐好选

第一次实验

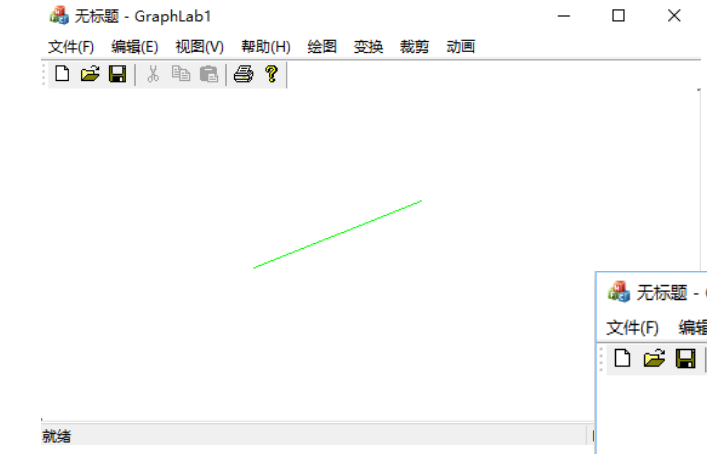
一、实验基本信息

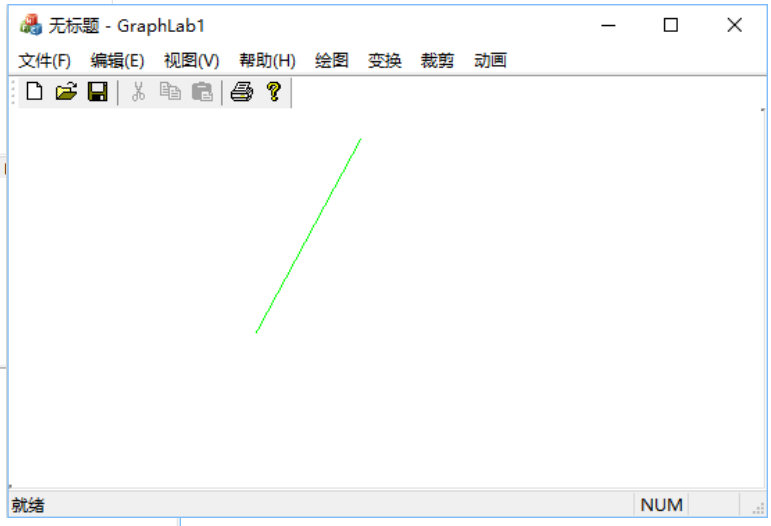
实验题目	多边形生成算法实验
实验时间	第 13 周 7、8、9、10 节
实验地点	格物楼 207
实验目的	通过编程掌握基本的直线绘制算法
实验内容	1、中点画线算法或 Bresenham 画线算法绘制直线 2、任意斜率直线段生成算法
实验的算法基础	我采用的是中点画线法 直线方程为 F(x,y)=ax+by+c=0 输入的两个点的坐标为(x0,y0),(x1,y1) a=y0-y1 b=x1-x0 c=x0y1-x1y0 分两种情形考虑再下一个象素的判定： 若 d≥0，中点 M 在直线上方，取正右方像素 P1 (Xp+1,Yp) 再下一个象素的判别式为： d1=F((Xp+1)+1,Yp+0.5)=a(Xp+2)+b(Yp+0.5)+c = d + a d 的增量为 a 若 d＜0，中点 M 在直线下方，取右上方象素 P2 (Xp+1,Yp+1) 再下一个象素的判别式为： d2=F((Xp+1)+1,(Yp+1)+0.5)= a(Xp+2)+b(Yp+1.5)+c =d + a+b d 的增量为 a+b d 的初始值 d0=F(X0+1,Y0+0.5) =F(X0,Y0)+a+0.5b =a+0.5b 用 2d 代替 d 后，d0=2a+b=a+a+b d 的增量都是整数
预期的实验结果	1 鼠标左键输入若干点，利用 Bresenham 算法依次连接各点，生成多边形。 2 能够处理任意斜率的直线段并在鼠标左键点击点后再点击右键可以成功绘制

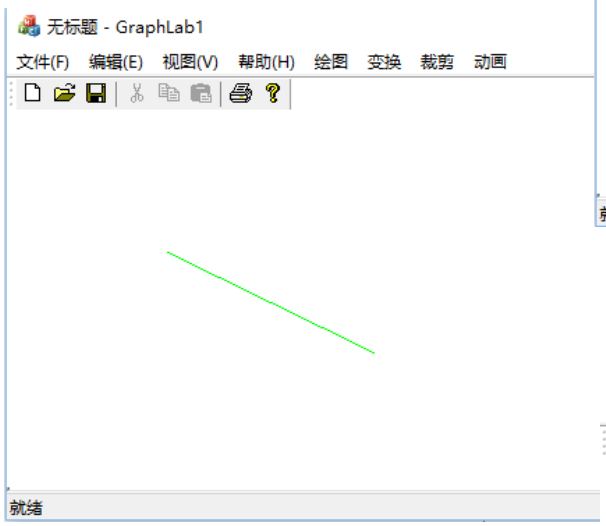
二、实验过程及结果

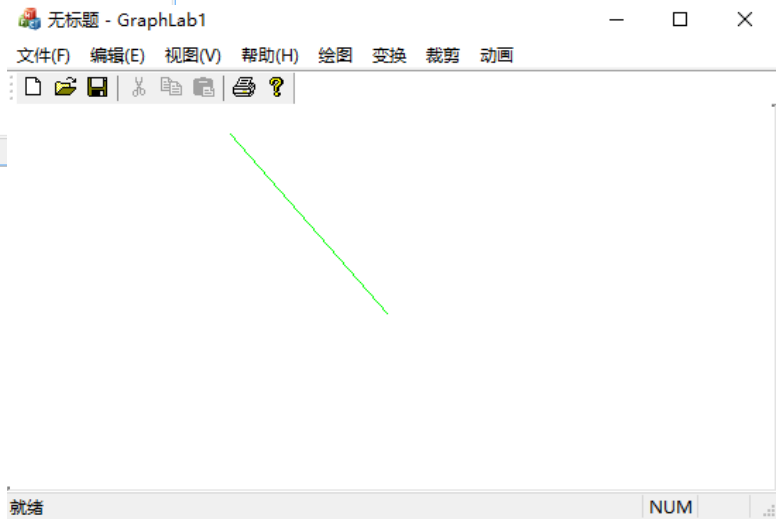
获取到鼠标左键输入的点后，存在 m_point[]这个 vector 容器中，然后点击鼠标右键后，按照顺序利用中点画线法进行直线段的绘制。

直线绘制展示（四种斜率的情况）：

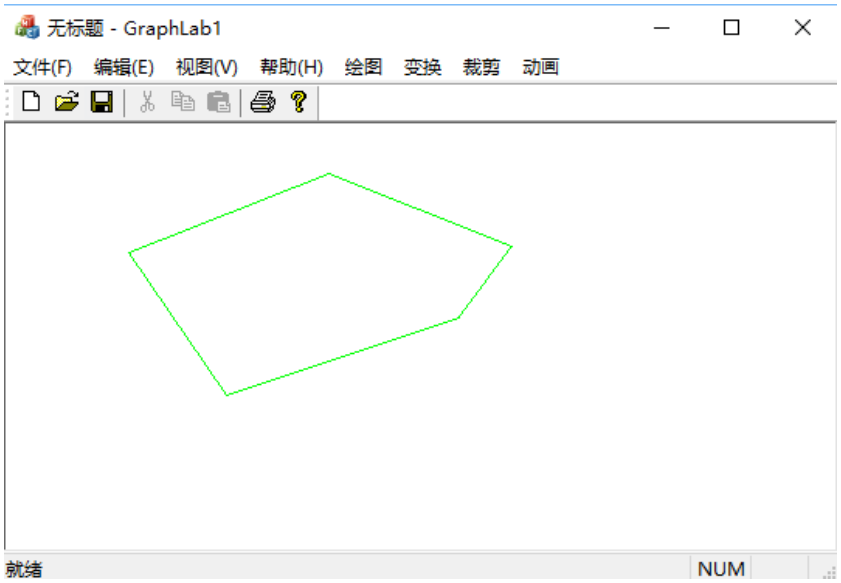


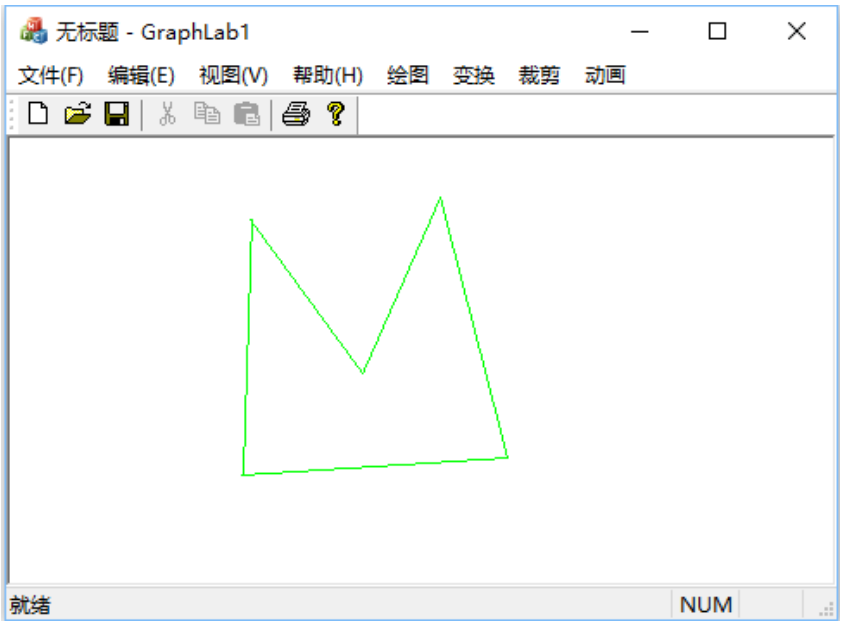






绘制多边形示例：





注：具体代码请参见文件文件夹 GraphLab1 文件夹

三、实验总结

中点画线法是只能绘制斜率在 0 到 1 之间的线段，因此，要绘制全斜率的直线段就必须进行相应的变换，我的变换就是首先将整条直线转换到 0<k<1,并且传入标志位，标识原始斜率，然后通过中点画线法求所有的点，然后映射到原始斜率上进行绘制。对于 k=0 以及 k=无穷的情况进行单独判断并特殊处理。

第二次实验

一、实验基本信息

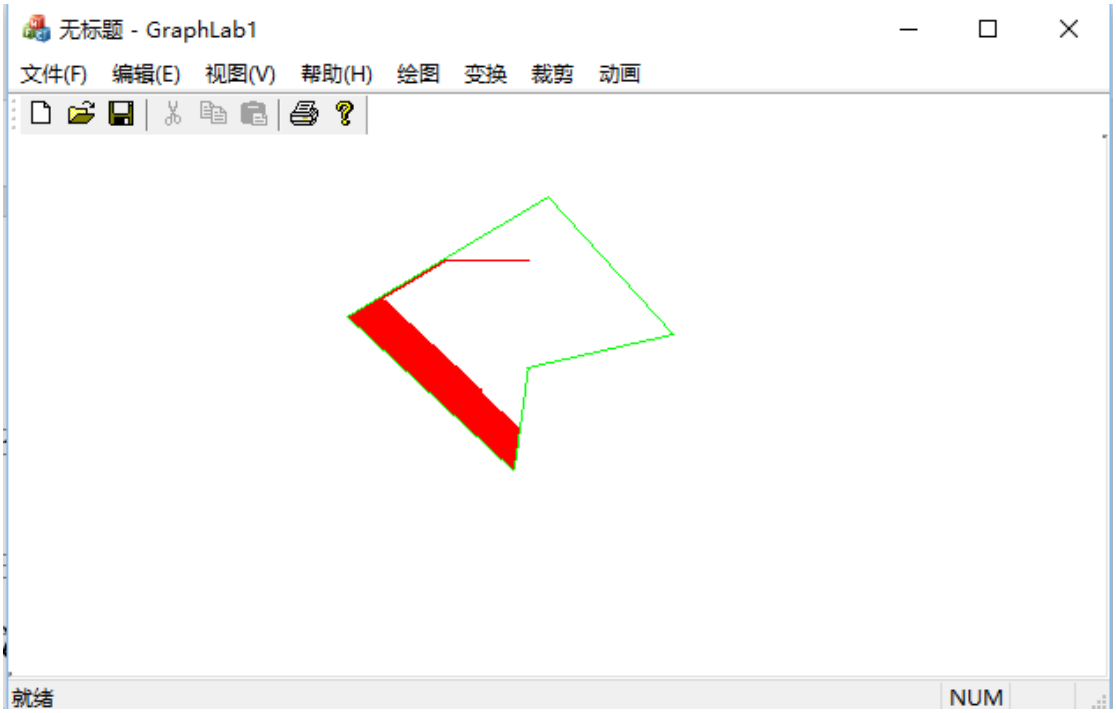
实验题目	多边形填充算法实验
实验时间	第 13 周 7、8、9、10 节
实验地点	格物楼 207
实验目的	熟悉多边形填充的算法，扫描线填充和种子填充等等
实验内容	利用扫描线填充和种子填充对实验一绘制的多边形进行填充
实验的算法基础	种子填充算法和边扫描线算法 **种子填充算法：** 种子像素入栈，当栈非空时，重复以下步骤：栈顶像素出栈，将出栈象素置成填充色，按左、上、右、下顺序检查与出栈象素相邻的四象素，若其中某象素不在边界上且未被置成填充色，则将其入栈 **扫描线算法：** 扫描线算法就是先确定图形的整体轮廓，可以用一个矩形将其圈起来，然后在这个轮廓内遍历每一条扫描线，绘制偶数对点内的区域。
预期的实验结果	能够填充凸多边形以及凹多边形完全区域，能够填充所有绘制出的点构成的多边形，但是不能绘制因为相交而新增的点。比如说五角星，中间的五边形是由五个未绘制的点构成的，不能绘制，其余区域能够绘制。

二、实验过程及结果

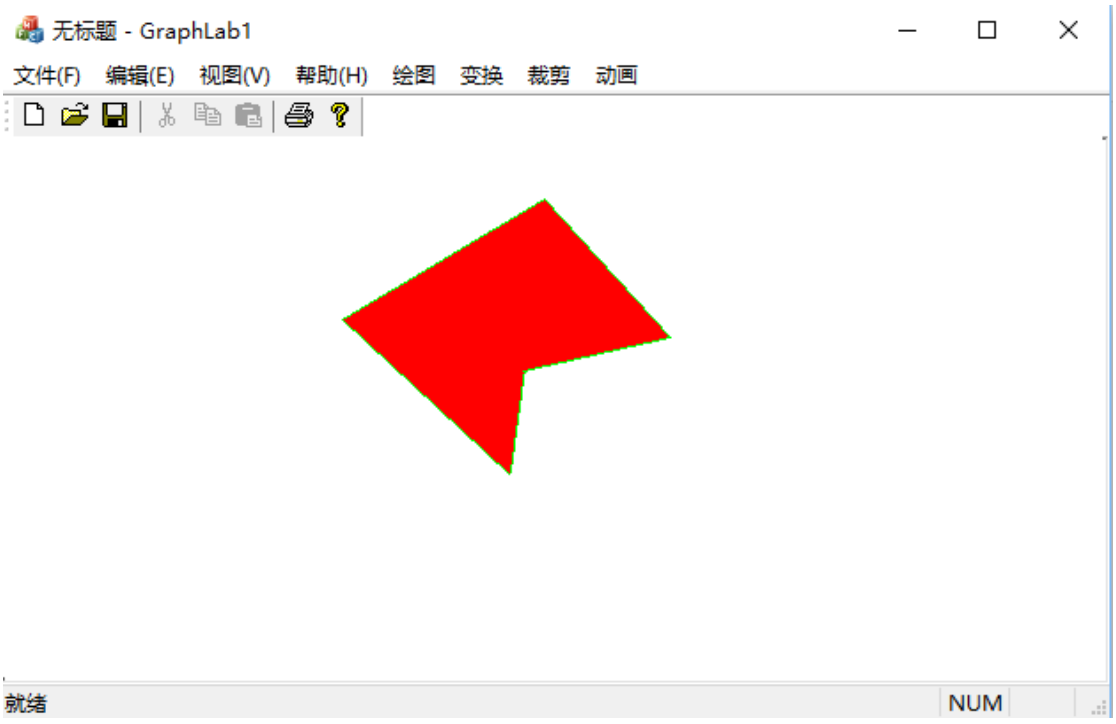
最重要的事情就是如何快速找到一个在多边形区域内的点，我采用的办法是对于先画的前两条边，取到这两条线段的中点，再将这两个中点连成线段后再取中点作为种子填充的起始入栈点。（前提是前两条边必须是上凸的）

种子填充示例：

下图为填充过程中



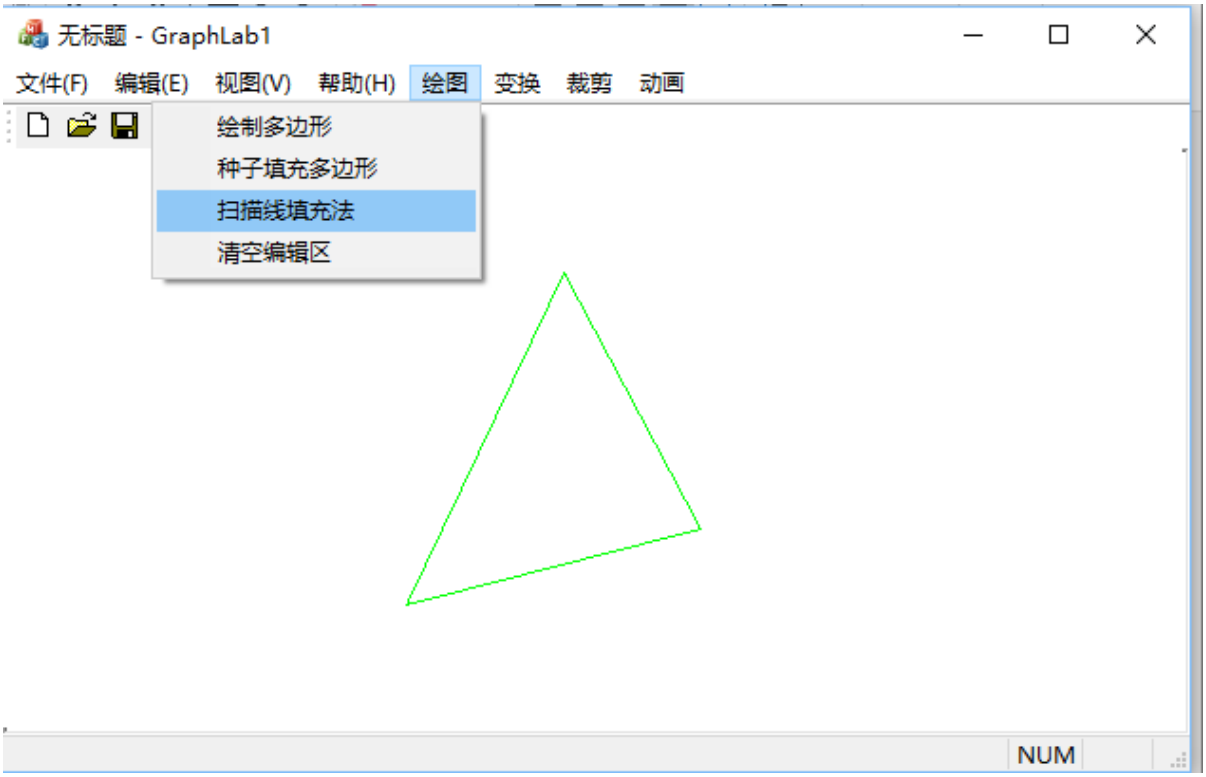
下图为填充完毕后



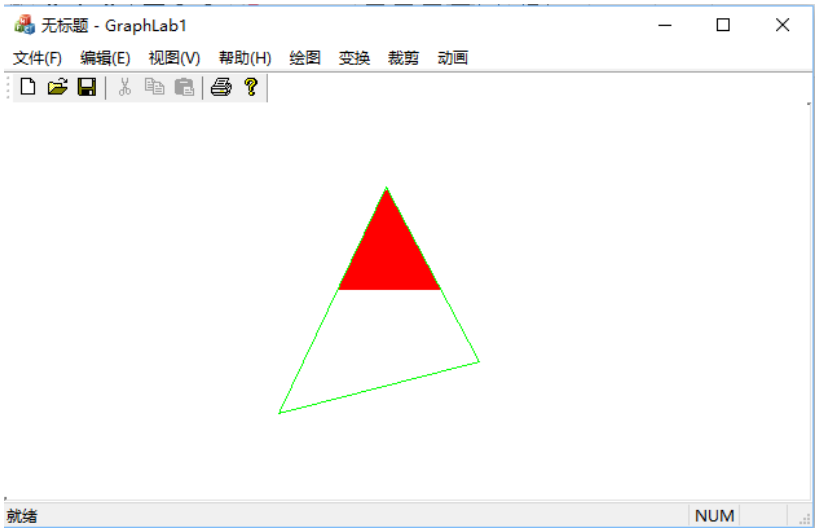
对于大图

扫描线填充示例：

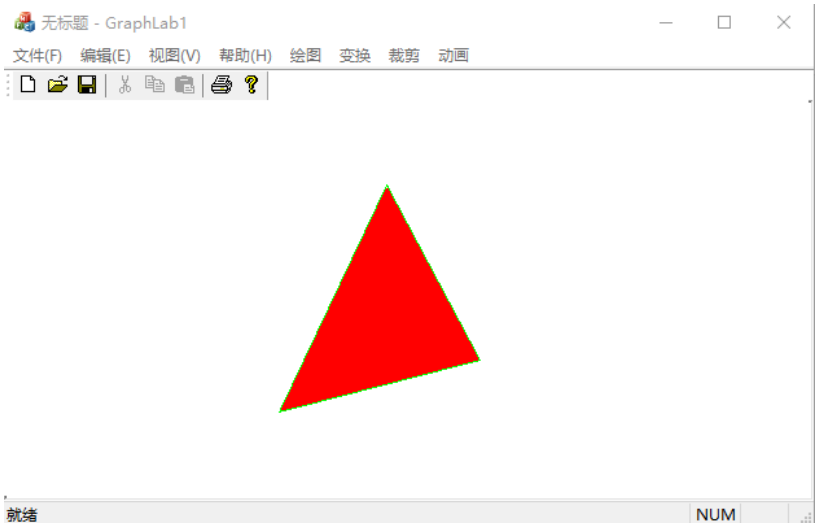
如下图，这一次采用扫描线填充



填充进行中



填充完毕后



注：具体代码请参见文件文件夹 GraphLab1 文件夹

三、实验总结

种子填充算法十分简单，不过对于在绘制大图的时候非常占用空间，往往会造成内存溢出，绘制大型的图形时比较慢，因此，我对于 VS 中该项目的属性进行了改造，扩大了默认的栈分配空间，只有这样，绘制大图的时候才不会系统崩溃。

扫描线填充法，在做实验的过程中我发现了一个问题，就是一条扫描线在扫描的过程中遇到边界后会出现点统计的个数问题，因为一条扫描线对于一条边界可能会扫到 2 个甚至更多的点（例如这条边界斜率很小），这个时候对于扫到奇数偶数点的判断就会出问题，从而导致填充错误，于是我对于连续扫到边界颜色的点就行特殊处理（即计数只为 1，不再累加），通过这种手段，我的边扫描填充得以正确实现。

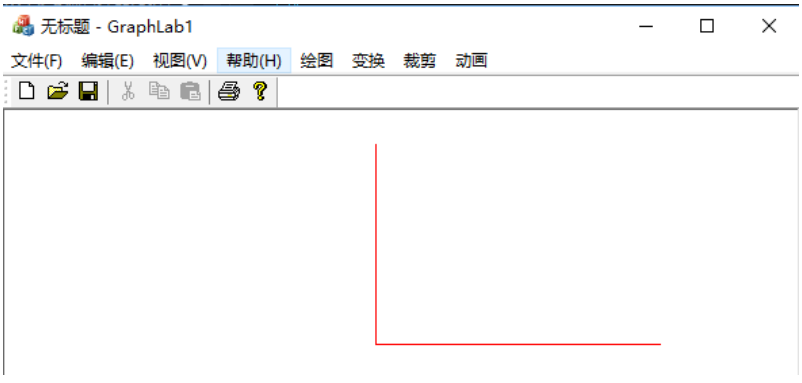
第三次实验

一、实验基本信息

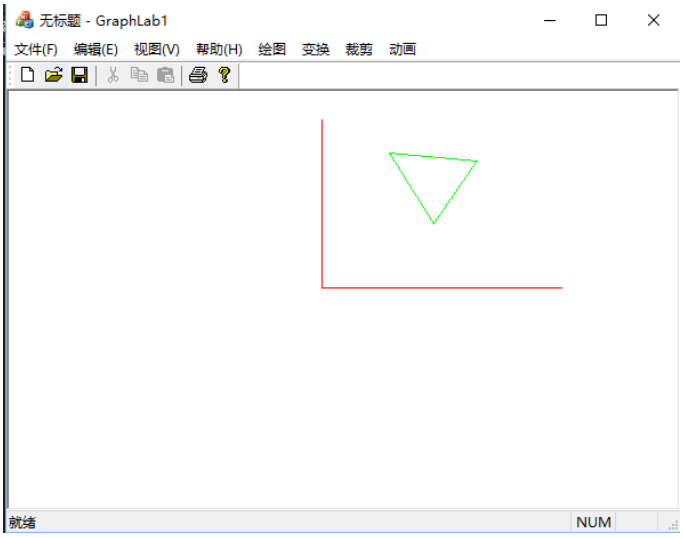
实验题目	二维图形的图形变换实验
实验时间	第 14 周 7、8、9、10 节
实验地点	格物楼 207
实验目的	掌握二维图形的变换，掌握齐次坐标系的变换方法
实验内容	实现平移、旋转、缩放、对称、错切等基本变换，并按照自己画的坐标系进行变换
实验的算法基础	1、平移 x’ =x+m, y’ =y+n 2、旋转 $\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$ 3、相对于原点对称 x’ =-x, y’ =-y 4、相对于直线 y=x 对称 x’ =y, y’ =x 5、放缩 $\begin{cases} x' = x * S_x \\ y' = y * S_y \end{cases}$ 6、相对于 X 轴正向错切 $\begin{cases} x' = x + ay \\ y' = y \end{cases}$
预期的实验结果	能够正确的将图形对称，平移，缩放，旋转到可视区域内

二、实验过程及结果

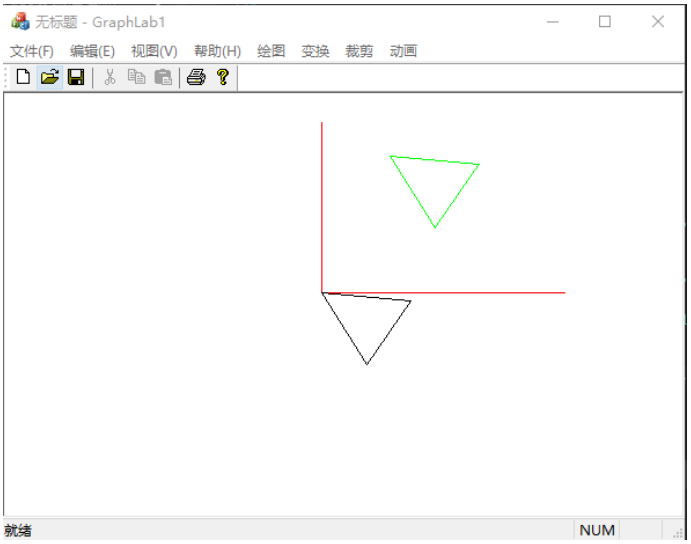
首先画坐标系



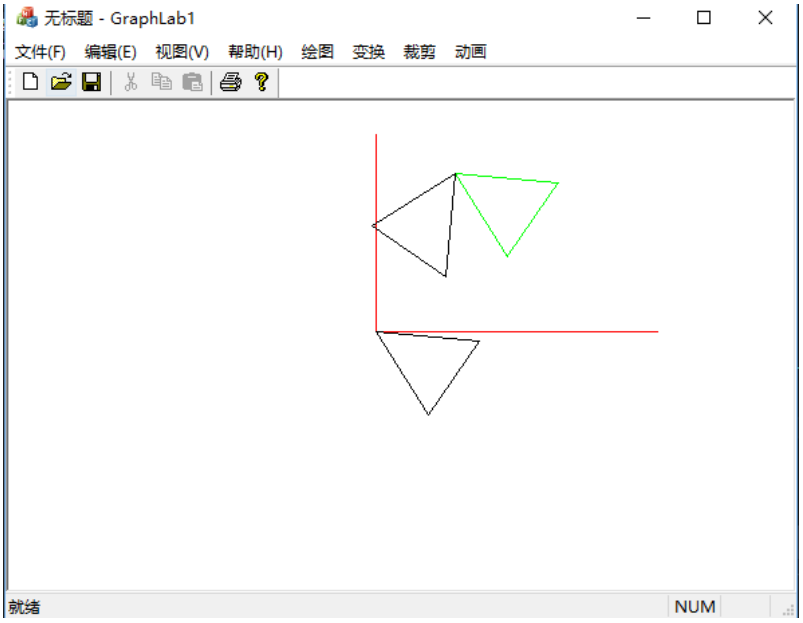
绘制图形用于变换



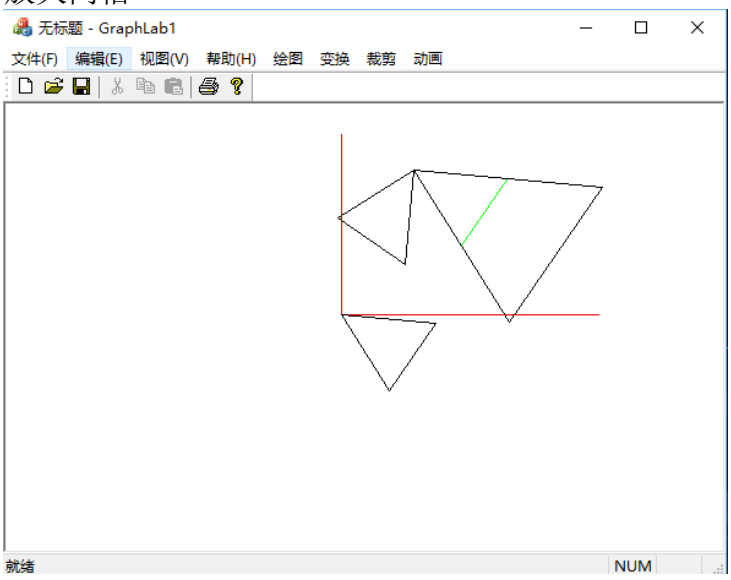
平移



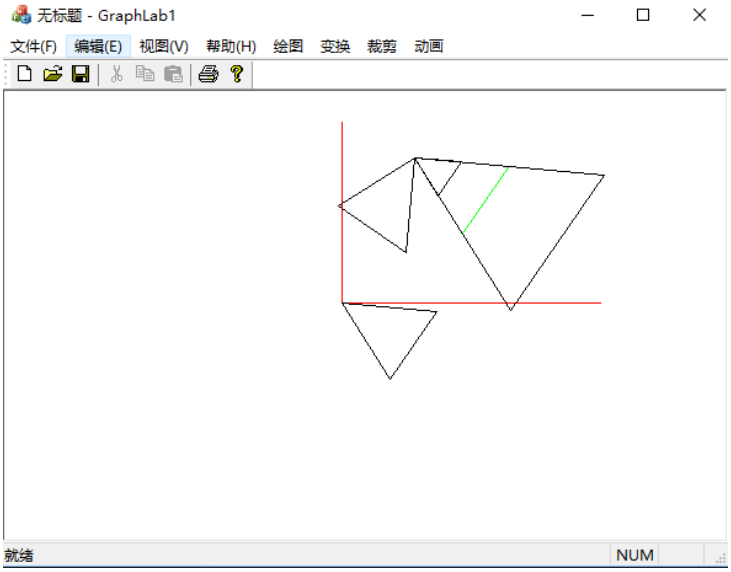
旋转（顺时针旋转 90 度）



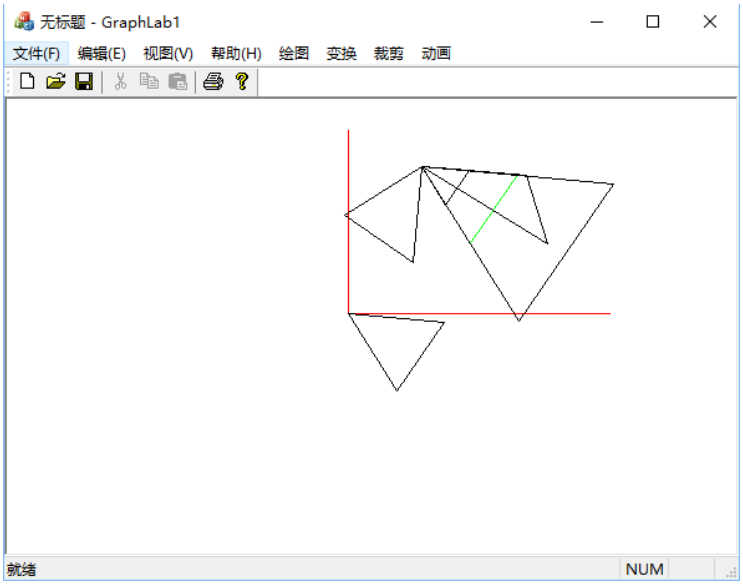
放大两倍



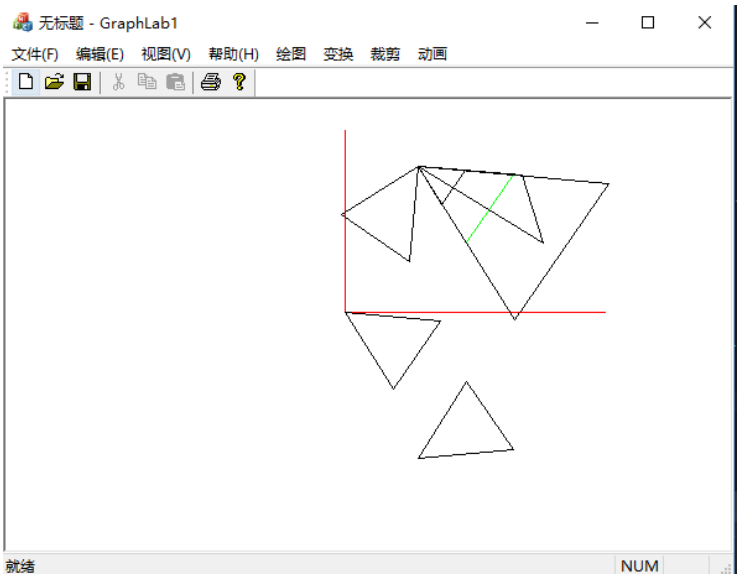
缩小两倍



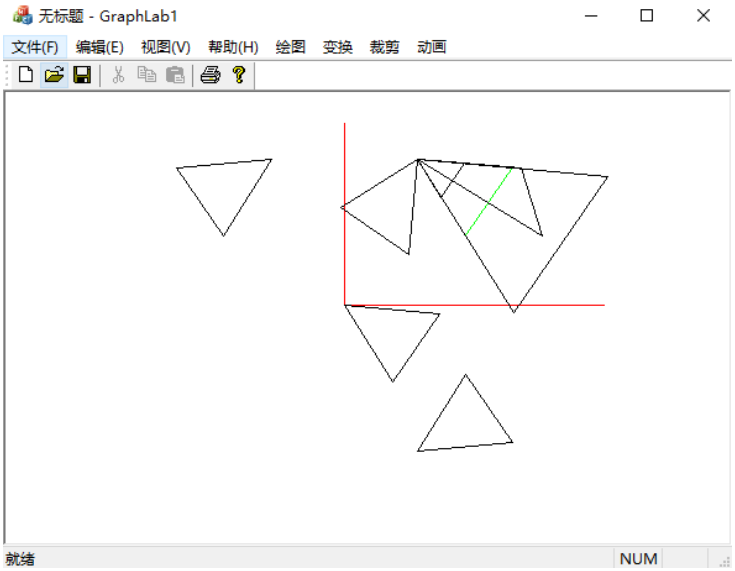
沿 X 轴错切 45 度



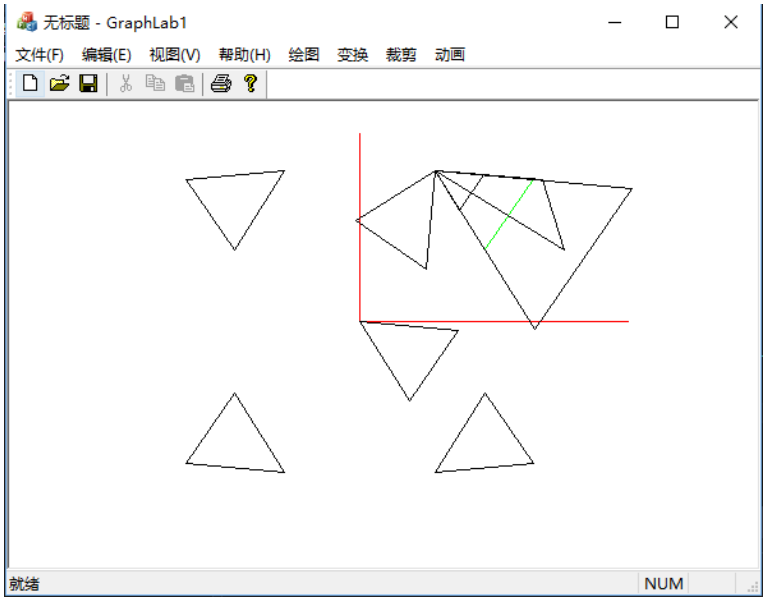
关于 X 轴对称



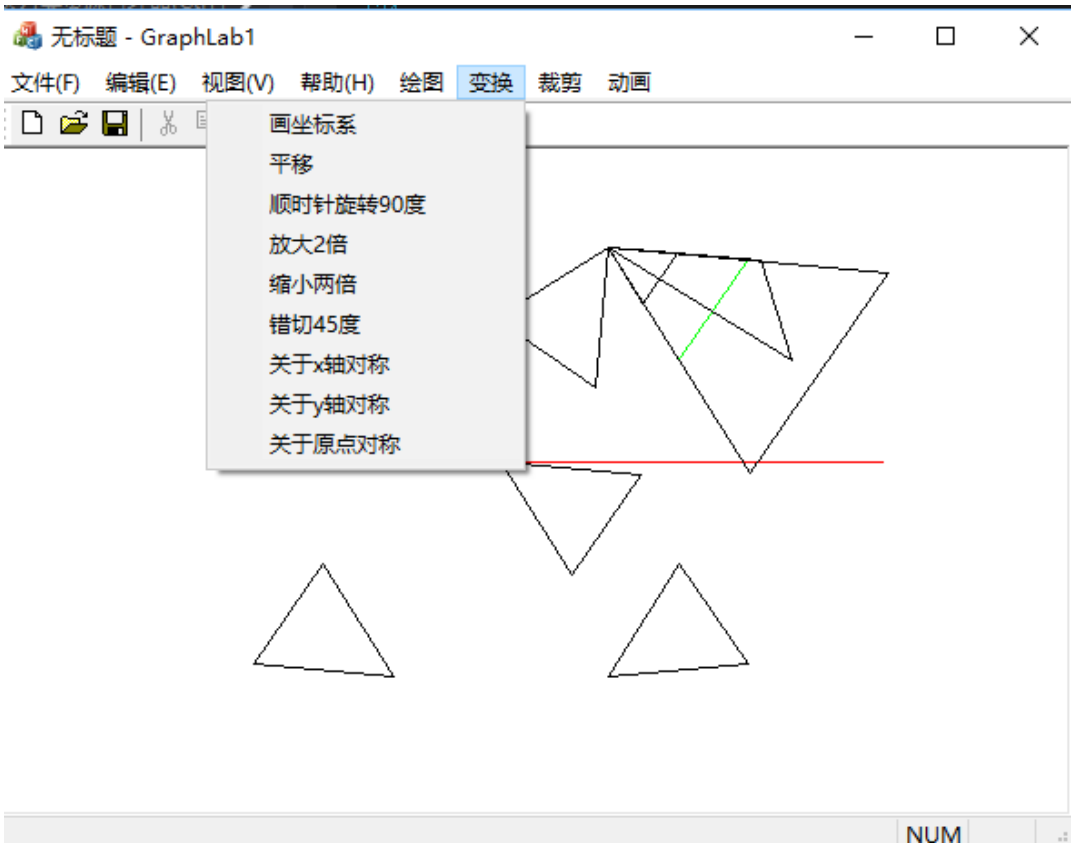
关于 Y 轴对称



关于原点对称



我的窗口列表如下：



注：具体代码请参见文件文件夹 GraphLab1 文件夹

三、实验总结

本次实验相比之下感觉十分简单，只需利用简单的坐标变换思想和以及利用齐次变换矩阵做乘法就可以得到目标图形各个顶点的坐标，然后按照顺序绘制各点再连接成图形即可。

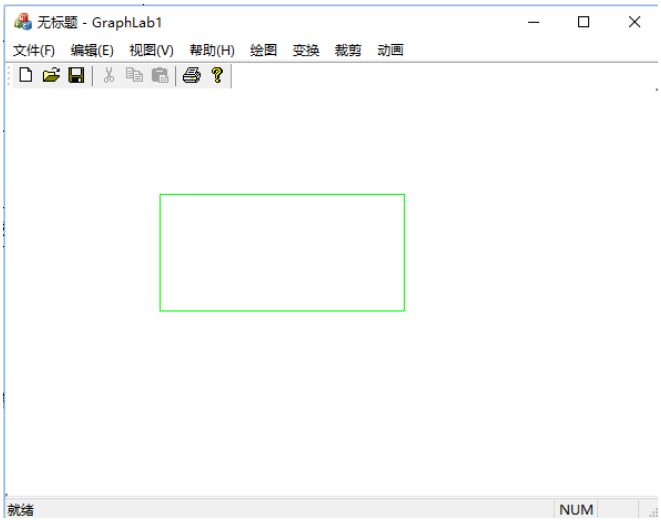
第四次实验

一、实验基本信息

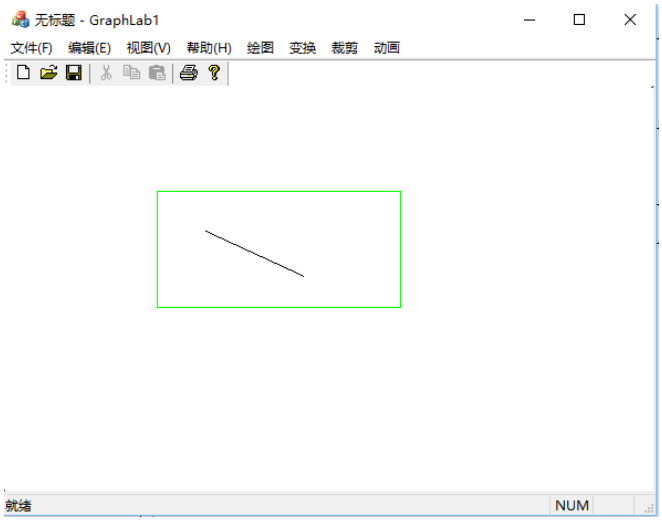
实验题目	直线段裁剪算法实验
实验时间	第 14 周 7、8、9、10 节
实验地点	格物楼 207
实验目的	掌握直线段的裁剪算法 例如编码(Cohen-Sutherland)裁剪算法、中点分割裁剪算法、Liang Barsky 裁剪算法
实验内容	利用编码(Cohen-Sutherland)裁剪算法、中点分割裁剪算法、Liang Barsky 裁剪算法，选择一个进行实现，
实验的算法基础	**我采用的是编码裁剪算法** 其算法基本步骤如下： 第一步：判别线段两端点是否都落在窗口内，如果是， 则线段完全可见；否则进入第二步； 第二步：判别线段是否为显然不可见，如果是，则裁剪结束；否则进行第三步 ； 第三步：求线段与窗口边延长线的交点，这个交点将线段分为两段，其中一段显然不可见，丢弃。对余下的另一段重新进行第一步，第二步判断，直至结束 中点分割算法：
预期的实验结果	能够正确的裁剪全部在可视域内，可视域外，一内一外，相交的所有线段。

二、实验过程及结果

首先画矩形裁剪框

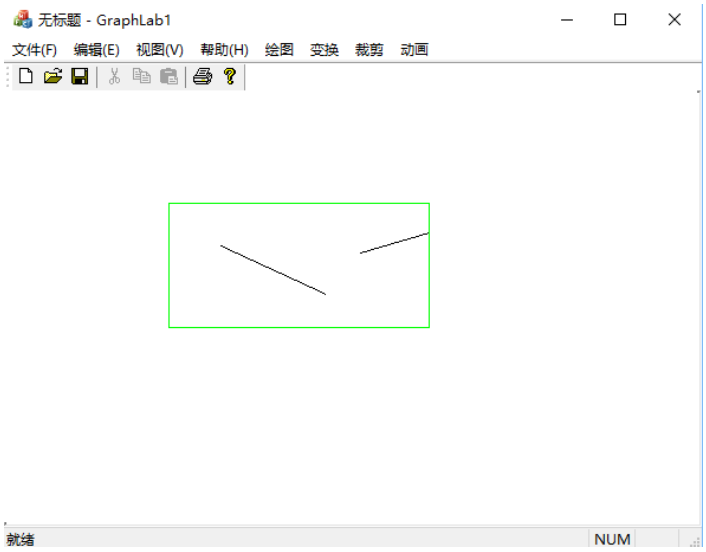


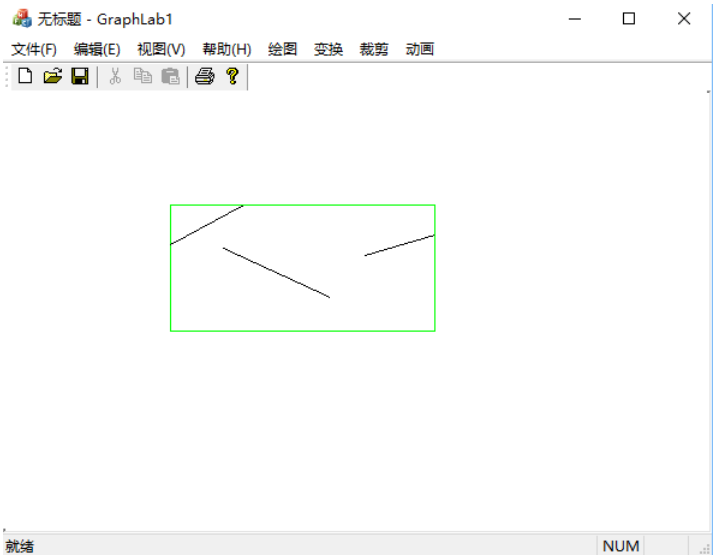
先画一个完全可见的线段

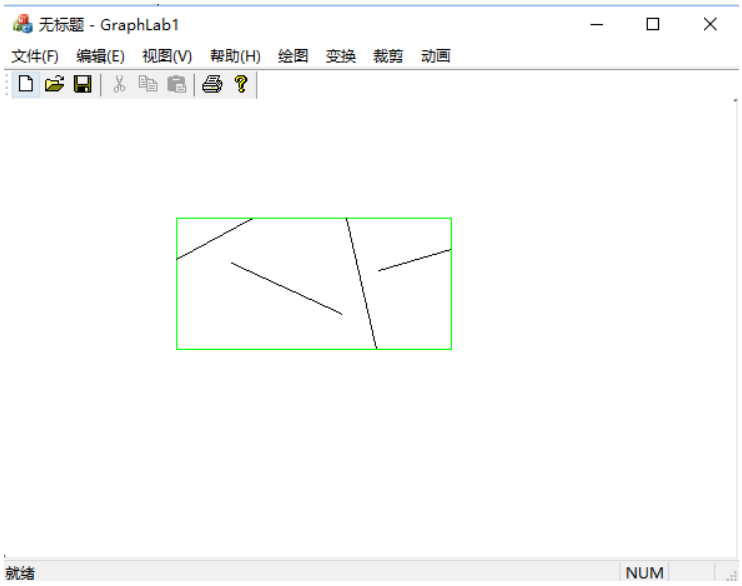


再先画一个完全不可见的线段（这个通过截图难以显示）

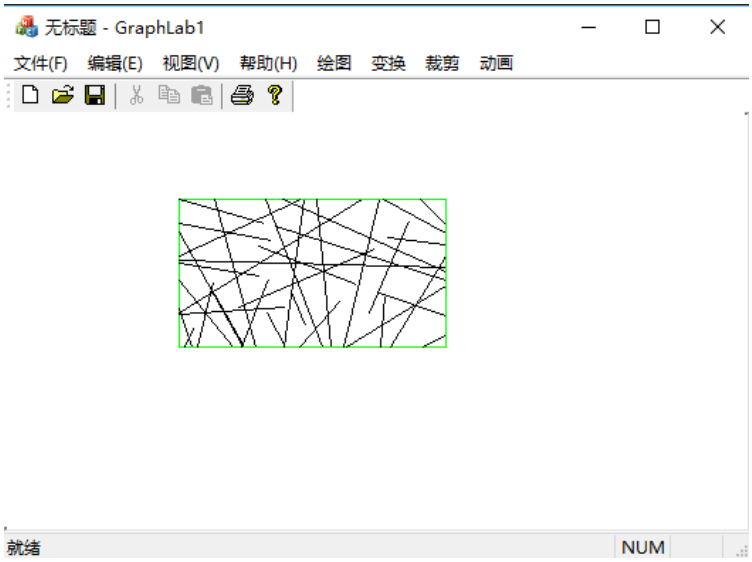
以下为部分可见线段的裁剪结果







画了若干条线段之后



注：具体代码请参见文件文件夹 GraphLab1 文件夹

三、实验总结

编码裁剪算法，首先按照已经画出的矩形裁剪框对各个区域进行编码，之后每当输入两个点后，对每一个点放入编码生成子程序中找到该点的编码值，用于之后判断，根据两个点的编码值，可以快速判断“完全可见”和“完全不可见”，对于部分可见，也可以按照两个点的编码值分类讨论，确定求交边的顺序（也就是说先与哪个边进行求交运算），最坏的情况就是与四条边进行求交。

第五次实验

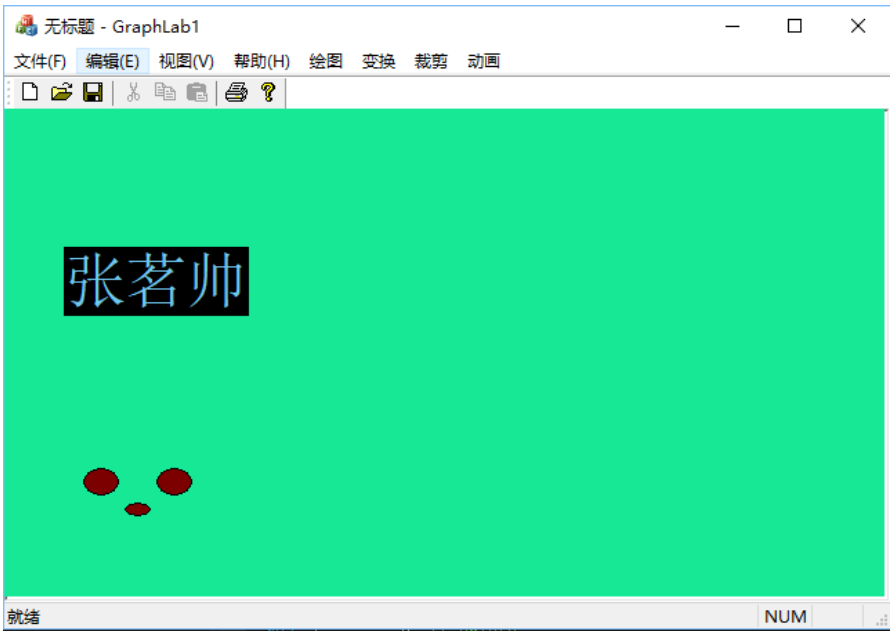
一、实验基本信息

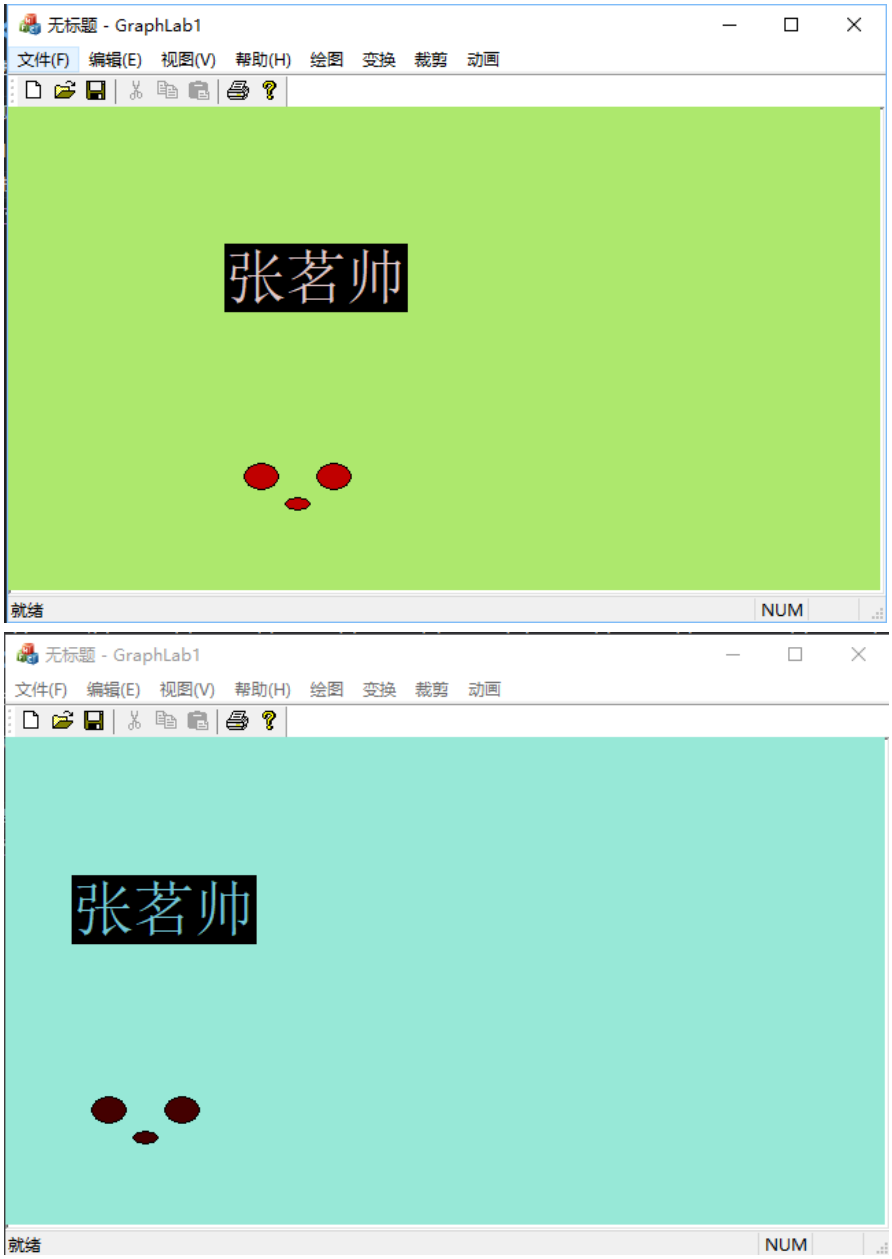
实验题目	简单动画实验
实验时间	第 14 周 7、8、9、10 节
实验地点	格物楼 207
实验目的	掌握简单动画的制作方法
实验内容	实时动画的制作
实验的算法基础	在实时动画中，一种最简单的运动形式是对象的移动，它是指屏幕上一个局部图像或对象在二维平面上沿着某一固定轨迹运动。运动的对象或物体本身在运动时的大小、形状、色彩等效果是不变的。 x+=mx y+=my 利用 SetTimer 和 OnTimer 这两个系统函数即可以实现（在 Ontimer 中给出坐标变换）
预期的实验结果	实现了简单的实时动画，图形沿一定轨迹移动

二、实验过程及结果

我设计的简单动画就是我的名字和一副简单的笑脸在屏幕上来回进行左右移动，与此同时，颜色也是随着移动而发生改变。

截图展示如下：





注：具体代码请参见文件文件夹 GraphLab1 文件夹

三、实验总结

我认为动画的主要思想就是提高相应的帧率使得人们看不出这是由不连续的片段构成，在进行我的简单动画的制作过程中，我出现了一个比较低级的失误，，就是刚开始的 OnTimer 函数我是自己定义的，在 debug 的过程中发送屏幕上总是一片空白，感觉很不知所措，后来经过分析感觉 OnTimer 可能就像生成“鼠标左右键点击触发”的那个函数似的，有可能是从类向导中导出的相关系统函数，于是我到那里进行了寻找，发现真的有，修改过后，我的动画得以成功运行。