《计算机图形学》
上机实验指导书

专　　业 _____

班级学号 _____

姓　　名 _____

指导教师 _____

学　　期 _____

南京工业大学测绘学院

目录

上机守则

1. 学生必须按指导教师安排的上机实验时间进入

机房上机。

2. 进入机房时必需穿上鞋套，否则不得进入机房。
3. 认真填写上机情况登记表，若遇计算机有异常情况，应先向老师汇报，不得擅自处理。
4. 遵守计算机操作规程。即开机时先开显示器再开主机；结束时关闭计算机：先通过 Windows 功能关闭系统，主机电源指标灯灭了后再关闭显示器。
5. 禁止玩游戏和从事与上机实验无关的内容。
6. 保持机房安静和整洁；不得携带食物和饮料进入机房；严禁随地吐痰和乱仍垃圾杂物；禁止吸烟、大声喧哗和闲聊。
7. 爱护机房设施，严禁更改设置参数、添加口令和删除非本人文件。对于导致计算机不能正常工作、影响他人上机者，将取消其上机资格。
8. 禁止私自拆卸配件和将室内物品带出室外。一经发现，除要求按价赔偿外，将通报批评和取消其上机资格，情节严重者交有关行政部门和司法部门处理。

计算机图形学实习指导

本课程一共有六次实习，每次实习给出：基本要求和提高要求，但同学们只需完成其中的六次实习的基本要求内容即可。

1、实习一　直线和圆的显示

2、实习二　区域填充

3、实习三　二维图形的裁剪

4、实习四　图形变换

5、实习五　投影变换

6、实习六 曲线拟合

同学们在实习过程中，完成基本要求后，如果有兴趣可根据不同的实际情况和自己的程度对提高要求进行选做。

一、实习目的

1、培养学生动手编程解决实际问题的能力。

2、训练学生分析问题和调试程序的能力。

3、锻炼学生撰写科技实验论文的能力。

二、实习要求

1、问题分析

充分地分析和理解问题本身，弄清要求做什么，用什么算法。

2、程序设计

(1)根据所采用的算法，设计数据结构，画出流程图并编程。

(2)最后准备调试程序的数据及测试方案。

3、上机调试

(1)对程序进行编译，纠正程序中可能出现的语法错误。

(2)调试前，先运行一遍程序看看究竟将会发生什么。

(3)如果情况很糟，根据事先设计的测试方案并结合现场情况进行错误跟踪，包括单步调试、设置观察窗输出中间变量值等手段。

4、整理实习报告

三、实习报告

1、实习内容：采用的算法名称

2、问题描述：包括目标、任务、条件约束描述等。

3、设计：数据结构设计和核心算法设计。主要功能模块的输入，处理（算法框架）和输出。

4、测试范例：测试结果的分析讨论，测试过程中遇到的主要问题及所采用的解决措施。

5、心得：包括程序的改进设想，经验和体会。

6、程序清单：源程序，其中包括变量说明及详细的注释。

实习一 直线和圆的显示

实习要求

将几何上的直线及圆显示在光栅显示器的显示平面上是本次实习的任务。

实习中，首先我们应构造直线或圆的参数作为已知条件在题目中给出：
直线参数为直线的两个端点的坐标：p1（x1，y1）、p2（x2，y2）；
圆的参数为圆心坐标、圆半径：pc（xc，yc）、r。

构造中除了上述参数个数外还应特别注意参数范围，参数范围与显示器里安装的适配器类型及程序中所选择的图形模式值有关，例如：显示器的适配器为 VGA，在 Turbo C 中程序选择的图形模式值为 2，这时显示器的分辨率就被确定为 640×480，即显示平面上呈二维象素矩阵，象素由象素坐标进行标识，坐标原点在屏幕左上角，x 轴正方向朝右（坐标值：0—639），y 轴正方向朝下（坐标值：0—479）。

图形显示的任务就是要找出最佳逼近几何图形的那些象素的坐标值，并将这些象素置成所要求的颜色，这个过程就叫做图形的扫描转换。扫描转换的基本要求：图形准确、象素均匀、显示速度快。由此引入许多图形扫描转换算法，对于直线和圆来说，直线的算法比圆复杂得多，因为对于不同方向的直线都要求图形准确、对于不同斜率的直线都要求象素均匀、而直线在图形中出现的量最大因此对显示速度快的要求也最高。

程序中需要用到 Turbo C 画点函数：

格 式：putpixel（int x，int y，int color）；

例 如：putpixel（150,80,4）； 在屏幕坐标（150，80）位置上画一个红色的点。

色	符号名	值	颜色	符号名	值
黑	BLACK	0	深灰	DARKGRAY	8
兰	BLUE	1	浅兰	LIGHTBLUE	9
绿	GREEN	2	浅绿	LIGHTGREEN	10
青	CYAN	3	浅青	LIGHTCYAN	11
红	RED	4	浅红	LIGHTRED	12
洋红	MAGENTA	5	浅洋红	LIGHTMAGENTA	13
棕	BROWN	6	黄	YELLOW	14
浅灰	LIGHTGRAY	7	白	WHITE	15

颜色符号常数（color）

实习内容

1、从键盘输入：直线两个端点的坐标、颜色，编程实现直线的绘制；

2、从键盘输入：圆心坐标、圆的半径、颜色，编程实现圆的绘制。

基本要求

1、用直线 DDA 算法实现任何斜率、任何方向直线的绘制。

2、用直线 Bresenham 算法实现|m|<1，任何方向直线的绘制。

3、用正负画圆算法实现圆的绘制。

提高要求

1、用直线 Bresenham 算法实现何斜率、任何方向直线的绘制。

2、将一个闭合多边形的各顶点坐标顺序装入二维数组 p[][2]，通过循环调用你编制的画线函数，绘出多边形。

3、用你编制的画线函数，实现虚线的绘制。

一、直线 DDA 算法

直线 DDA 算法思想

该算法实现的关键是：如何步进。它涉及到以下两点：

1、步进的方向。即步进的正或负，决定能否正确的到达终点。

2、步进的大小。即哪个方向的步进取单位步进？它控制了变化最大的步进，令其为单位步进，而另一个方向的步进必小于 1 ，这样不论斜率|m|≤1 否，都会使直线的亮度均匀。

依公式：$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$ 则下一点坐标为：$x_{i+1} = x_i + \Delta x$，$y_{i+1} = y_i + \Delta y$

实习提示

上述算法步骤 3 很关键，xin、yin 的符号和大小分别决定了直线显示的准确及均匀。实习中编制两段函数，一段 main()函数，另一段是 dda()函数，从 main()函数里得到参数，并将参数传入 dda()函数。main()函数形式如下：

```
#include "graphics.h"
#include "math.h"
main()
{
  //硬件测试，将 gd 装入图形驱动器，gm 置入最大图形模式。
  int gd=0,gm=2,x1,y1,x2,y2,color;
  printf("input x1,y1,x2,y2,color is:");

  //输入直线参数。
  scanf("%d，%d，%d，%d，%d",&x1,&y1,&x2,&y2,&color);
```

```
    //图形初始化。
    initgraph(&gd,&gm,"c:\\tc");

    //设置兰背景。
    setbkcolor(1);
    dda(x1,y1,x2,y2,color);
    getch();
    closegraph();
}
```

二、直线 Bresenham 算法

直线 Bresenham 算法思想

在另一个方向（即直线变化较小的方向）坐标的 0.5 处，引出"象素分界线"栅条。按直线从起点到终点的顺序，当变化较大方向的坐标每次步进一个单位时，另一个方向依误差 e_i(数学点 p_i 与最近的象素分界线的偏离值)的符号决定是否步进，当 $e_i \geq 0$ 时步进一个单位，否则不步进。为了避免实数运算，而且 e_i 两边同乘大于 0 的某个数又不影响符号的判别，从而可以将误差 e_i 变形成为整型数 d，推导过程请看教材。

直线 Bresenham 算法步骤

当直线满足条件：$0 \leq m \leq 1$，且 x1〈x2 时，Bresenham 算法步骤如下：

1、从主调函数得到参数：x1、y1、x2、y2、color；

2、计算：dx=x2-x1；dy=y2-y1；

3、计算初始误差：d=2·dy－dx；

4、令初始象素坐标为：x=x1；y=y1；

5、重复以下步骤，直到 x〉x2 为止：

5.1、用规定颜色在（x，y）处画象素点；

5.2、若 d≥0，则 y＝y＋1；d＝d－2·dx；

5.3、x ＝x＋1；d＝d＋2dy。

实习二 区域填充

实习要求

所谓区域填充,指的是在输出平面的闭合区域内完整地填充某种颜色或图案。以下所述及的区域填充算法或相关程序,主要针对显示平面内的区域而言。

区域填充的问题一般分两大类,一是多边形填充；一是种子填充。多边形填充有一定难度；种子填充在学生掌握了"栈"这一抽象数据类型的实现方法的前提下,比较容易完成。而边标志填充算法却是介于这两类之间,部分地具有它们的痕迹,算法思想巧妙,实现起来更容易。

在编程时可能需要在屏幕上画线段,建议学生使用自己在上次实习中实现的画线程序,也允许学生使用 Turbo C 的画线函数,包括：line( )、lineto( )、linerel( )、moveto( )等；另外,在编写种子填充程序时除了可能用到函数 putpixel( )外,还要用到取屏幕上某象素颜色的函数 getpixel( ),上述这些函数的具体使用方法可参考 Turbo C 使用手册,也可查看 Turbo C 的联机帮助。请注意,在 Turbo C 图形模式下,设备坐标系原点在屏幕左上角,X 轴正方向向右,Y 轴正方向向下。

▾ 实习内容

将闭合的多边形内填充某种颜色或图案。

▾ 基本要求

1、用扫描线种子填充算法实现多边形内实面积填充。

2、用边标志填充算法实现多边形内实面积填充。

▾ 提高要求

1、实现多边形内图案填充。

2、实现多边形内剖线填充。

3、用边相关多边形扫描线填充算法实现多边形内实面积填充。

一、扫描线种子填充算法

▾ 扫描线种子填充算法思想

首先填充种子所在的尚未填充的一区段,然后确定与这一区段相邻的上下两条扫描线上位于该区段内是否存在需要填充的新区段,如果存在,则依次把每个新区段最右端的象素作为种子放入堆栈。反复这个过程,直到堆栈为空。

▾ 扫描线种子填充算法步骤

1、初始化堆栈。

2、种子压入堆栈。

3、While（堆栈非空）从堆栈弹出种子象素。

{

(1)如果种子象素尚未填充,则：

① 求出种子区段：x_{left}、x_{right}。

② 填充整个区段。

(2)检查相邻的上扫描线的 $x_{left}\le x\le x_{right}$ 区间内,是否存在需要填充的新区段,如果存在,则把每个新区段在 $x_{left}\le x\le x_{right}$ 范围内的最右边的象素,作为新的种子象素依次压入堆栈。

(3)检查相邻的下扫描线的 $x_{left}\le x\le x_{right}$ 区间内,是否存在需要填充的新区段,如果存在,则把每个新区段在 $x_{left}\le x\le x_{right}$ 范围内的最右边的象素,作为新的种子象素依次压入堆栈。

}

有关堆栈操作的辅助代码

1、定义栈结构：

```
# define MAX 100 /*定义最大栈空间*/
struct stack
{
  int top; /*指向栈顶的计数器*/
  int xy[MAX][2]; /*种子点（二维）*/
}s;
```

2、初始化堆栈

```
s.top=－1;
```

3、进栈操作

```
pushxy(int x,int y)
{
  if(s.top= =MAX－1)
  {
    printf("Overflow!");
    exit(1);
  }
  else
  {
    s.top=s.top+1;
    s.xy[s.top][0]=x;
    s.xy[s.top][1]=y;
  }
```

```
}
```

4、出栈操作

```
popxy(int *x,int *y)
{
  if(s.top<0)
  {
    printf("underflow!");
    exit(1);
  }
  else
  {
    *x=s.xy[s.top][0];
    *y=s.xy[s.top][1];
    s.top=s.top－1;
  }
}
```

5、堆栈非空

s.top!=－1 或者 s.top>=0

扫描线种子填充算法伪代码

```
scanline_seed_fill(int x,int y,int boundarycolor,int newcolor)
{
  int savex,xleft,xright,pflag,xenter;
  //初始化堆栈;
  pushxy(x,y); /*种子压入堆栈*/
  while(堆栈非空)
  {
    popxy(&x,&y); /*栈顶象素出栈*/
    savex=x; /*保存种子坐标 x 分量的值*/
    while(getpixel(x,y)!=boundarycolor) /*获取该点的颜色值*/
    {
      putpixel(x,y, newcolor ); /*填充种子右侧的象素*/
      x++;
```

```
    }
    xright=x－1; /*得到种子区段的右端点*/
    x=savex－1; /*准备向种子左侧填充*/
    while(getpixel(x,y)!=boundarycolor) /*获取该点的颜色值*/
    {
      putpixel(x,y, newcolor ); /*填充种子左侧的象素*/
      x--;
    }
    xleft=x+1; /*得到种子区段的左端点*/
    x=xleft;
    y=y+1; /*考虑种子相邻的上扫描线*/
    while(x<=xright)
    {
      pflag=0; /*找到新种子的标志：0 为假；1 为真*/
      while(getpixel(x,y)!=boundarycolor && getpixel(x,y)!=newcolor&& x<xright)
      {
        if(pflag= =0)
        pflag=1;
        x++;
      }
      if(pflag= =1)
      {
        if((x= =xright)&&(getpixel(x,y)!=boundarycolor)&&(getpixel(x,y)!=newcolor))
          pushxy(x,y); /*新区间超过 xright,将代表该区段的象素进栈*/
        else
          pushxy(x－1,y); /*新区段右端点作为种子进栈*/
        pflag=0;
      }
      xenter=x;
      while((getpixel(x,y)==boundarycolor||getpixel(x,y)==newcolor)&&x<xright)
      {
        x++;/*向右跳过分隔带*/
      }
      if(xenter==x) x++;/*处理特殊情况,以退出 while(x<=xright)循环*/
    }
    x=xleft; /*为下扫描线的处理作准备*/
    y=y－2;
    /*检查相邻的下扫描线,找新区段,并将每个新区段右端的象素作为种子
    入栈,其方法与上扫描线的处理一样,这里省略。要求同学补充完整。*/
  }
}
```

二、边相关多边形扫描线填充算法

边相关多边形扫描线填充思想

边相关扫描线填充算法的实现需要建立两个表：边表（ET）和活动边表（AET）。

ET 用来对除水平边外的所有边进行登记,即建立边的记录。

AET 则是在 ET 建立的基础上进行扫描转换。对不同的扫描线,与之相交的边线也是不同的,当对某一条扫描线进行扫描转换时,我们只需要考虑与它相交的那些边线,为此 AET 建立了只与当前扫描线相交的边记录链表,以提供对当前扫描线上的区段进行填充。

边相关多边形扫描线填充算法步骤

1、根据给出的顶点坐标建 ET 表；并求出顶点坐标中最大 y 值 y_{max} 和最小 y 值 y_{min}。

2、定义 AET 指针,并使它为空。

3、使用扫描线的 y_j 值作为循环变量,使其初值为 y_{min}。

4、对于循环变量 y_j 的每一整数值,重复作以下事情,直到 y_j 大于 y_{max},或 ET 与 AET 表都为空为止：

① 如果 ET 中 y_j 桶非空,则将 y_j 桶中的全部记录合并到 AET 中。
② 对 AET 链中的记录按 x 的大小从小到大排序。
③ 依次取出 AET 各记录中的 x_i 坐标值,两两配对,对每对 x_i 之间的象素填上所要求的颜色。
④ 如果 AET 中某记录的 $y_{max}=y_j$,则删除该记录。
⑤ 对于仍留在 AET 中的每个记录,用 x_i+1/m 代替 x_i,这就是该记录边线与下一条扫描线 y_j+1 的交点。
⑥ 使 y_j 加 1,以便进入下一轮循环。

边相关多边形扫描线填充为伪代码

```
#include <stdlib.h>
#include <graphics.h>
#include <stdio.h>
#define round(x) ((x>0)?(int)(x+0.5):(int)(x-0.5)) /*求舍入的宏*/
struct edge{ /*边记录结构*/
  int ymax;
  float xi;
  float m;
  struct edge *next;
};
void poly_fill(int,int *,int);
void main()
{
  int polypoints[]={ /*多边形顶点坐标: x0,y0,x1,y1,... */
  100,300, 200,200, 300,200, 300,350,
```

```
    400,250, 450,300, 300,50, 100,150};
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"");
    poly_fill(8,polypoints,4); /*用红色填充*/
    getch();
    closegraph();
}
/*将一条边记录插入边记录构成的链表的表头*/
void insert_et(struct edge *anedge,struct edge **p_edges)
{
    struct edge *p;
    p=*p_edges;
    *p_edges=anedge;
    anedge->next=p;
}
/*复制一条边记录插入有效边表,维持有效边表的有序性*/
short insert_aet(struct edge *p,struct edge **p_aet)
{
    struct edge *q,*k,*l;
    if(!(q=(struct edge *)malloc(sizeof(struct edge))))
    {
        printf("\nOUT MEMORY IN INSERTING EDGE RECORD TO AET\n");
        return(0);
    }
    q->ymax=p->ymax; q->xi=p->xi;
    q->m=p->m; q->next=NULL;
    if(!(*p_aet)||((*p_aet)->xi>q->xi)||(((*p_aet)->xi==q->xi)&&((*p_aet)->m>q->m)))
    {
        l=*p_aet; *p_aet=q; q->next=l;
    }
    else
    {
        l=*p_aet;
        k=l->next;
        while(k&&(k->xi<q->xi))
        {
            l=k;
            k=k->next;
        }
        if(k&&(k->xi==q->xi)&&(k->m<q->m))
        {
            l=k;
            k=k->next;
```

```
      }
      l->next=q;
      q->next=k;
    }
    return(1);
}
/*从（x1,y）到（x2,y）用 color 色绘水平直线*/
void draw_line(int x1,int x2,int y,int color)
{
    int i;
    y=getmaxy()-y; /*进行坐标变换*/
    for(i=x1;i<=x2;i++)putpixel(i,y,color);
}
/*多边形扫描线填充:
numpoint 是多边形顶点个数;
points 存放多边形顶点坐标（x0,y0,x1,y1,...);
color 是填充色*/
void poly_fill(int numpoint,int *points,int color)
{
    struct edge **et=NULL,*aet,*anedge,*p,*q;
    int i,j,maxy,miny,x1,y1,x2,y2,yi,znum;
    maxy=miny=points[1];
    znum=2*numpoint;
    for(i=3;i<znum;i++)
    {
      if(maxy<points[i]) maxy=points[i];
      else if(miny>points[i])miny=points[i];
        i++;
    }
    if(!(et=(struct edge **)malloc((maxy-miny+1)*sizeof(struct edge *))))
    { /*建立边表 ET */
      printf("\nOUT MEMORY IN CONSTRUCTING ET\n");
      return;
    }
    for(i=0;i<maxy-miny+1;i++) et[i]=NULL;
    x1=points[znum-2]; y1=points[znum-1];
    for(i=0;i<znum;i+=2)
    { /*处理多边形所有边,为每条非水平边建立一个边记录,并将其插到 ET 表中的合适位置 */
      x2=points[i]; y2=points[i+1];
      if(y1!=y2) /*只考虑非水平边*/
      {
        if(!(anedge=(struct edge *)malloc(sizeof(struct edge))))
        {
```

```
            printf("\nOUT MEMORY IN CONSTRUCTING EDGE RECORD.\n");
            goto quit;
          }
          anedge->m=(float)(x2-x1)/(y2-y1);
          anedge->next=NULL;
          if(y2>y1) /*处理奇异点*/
          {
            j=i+1;
            do{ /*向后划过所有水平边*/
              if((j+=2)>=znum)j-=znum;
            }while(points[j]==y2);
            if(points[j]>y2) anedge->ymax=y2-1;
/*若(x2,y2)不是局部极值点,边记录的 ymax 域为 y2-1,这样处理
扫描线 y=y2 时此边记录将不在 AET 中,从而不会产生交点 */
            else anedge->ymax=y2; /*若(x2,y2)是局部极值点,边记录的 ymax 域为 y2,
这样处理扫描线 y=y2 时此边记录将在 AET 中,从而会产生一个交点 */
            anedge->xi=x1;
            insert_et(anedge,&et[y1-miny]);
          }
          else
          {
            j=i+1; /*向前划过所有水平边*/
            do{
              if((j-=2)<0)j+=znum;
            }while(points[j]==y1);
            if(points[j]>y1) anedge->ymax=y1-1;
/*若(x1,y1)不是局部极值点,边记录的 ymax 域为 y1-1,这样处理
扫描线 y=y1 时此边记录将不在 AET 中,从而不会产生交点 */
            else anedge->ymax=y1; /*若(x1,y1)是局部极值点,边记录的 ymax
域为 y1,这样处理扫描线 y=y1 时此边记
录将在 AET 中,从而会产生一个交点 */
            anedge->xi=x2;
            insert_et(anedge,&et[y2-miny]);
          }
        }
        x1=x2;
        y1=y2;
      }
      aet=NULL; /*初始化有效边表 AET*/
      for(yi=miny;yi<=maxy;yi++) /*从低到高逐条处理扫描线*/
      { /*将 ET 表中与 yi 对应的边记录链表中的全部边记录
        p=et[yi-miny]; 都按序并入 AET 中*/
        while(p)
```

```
    {
      if(!insert_aet(p,&aet)) goto quit;
      p=p->next;
    }
    p=aet;
    while(p) /*依次取出 AET 各记录中的 xi 坐标值,两两配对,*/
    {/*对每对 xi 之间的象素填上所要求的颜色*/
      draw_line(round(p->xi),round(p->next->xi),yi,color);
      p=p->next->next;
    }
    p=aet;
    while(p&&(p->ymax==yi)) /*对 AET 中的每个记录,若它的 ymax==yi, */
    {/*则删除该记录,否则用 xi+1/m 代替 xi,这就是该记录所对应的*/
      aet=p->next; /*边线与下一条扫描线 y=yi+1 的交点 */
      free(p);
      p=aet;
    }
    while(p)
    {
      if(p->ymax==yi)
      {
        q->next=p->next;
        free(p);
        p=q->next;
      }
      else
      {
        p->xi+=p->m;
        q=p;
        p=p->next;
      }
    }
  }
  quit:
  if(et) /*释放动态申请的内存*/
  {
    for(yi=miny;yi<=maxy;yi++)
    {
      q=p=et[yi-miny];
      while(p)
      {
        q=p->next;
        free(p);
```

```
            p=q;
          }
        }
        free(et);
      }
    }
```

三、 边标志填充算法

边标志填充算法思想

扫描线具有连贯性,这种连贯性只有在扫描线与多边形相交处才会发生变化,而每次的变化结果：无非是在前景色和背景色之间相互"切换"。

边标志填充算法正是基于这一发现,先在屏幕上生成多边形轮廓线,然后逐条扫描线处理。处理中：逐点读取象素值,若为边界色,则对该象素值进行颜色切换。

边标志填充算法步骤

1、用边界色画出多边形轮廓线,也就是将多边形边界所经过的象素打上边标志。

2、为了缩小范围,加快填充速度,须找出多边形的最小包围盒：x_{min}、y_{min}、x_{max}、y_{max}。

3、逐条扫描线进行处理,初始时标志为假,对每条扫描线依从左往右的顺序,逐个访问该扫描线上的象素。每遇到边界象素,标志取反。然后,按照标志是否为真决定象素是否为填充色。

边标志填充算法伪代码

```
EdgeMarkFill(int p[][2],int n,int boundarycolor,int newcolor)
{
  int i,x,y,flag,xmin,xmax,ymin,ymax;
  setcolor(boundarycolor); /*设置画笔色*/
  for(i=0 ;i<n;i++)/*画出多边形的 n 条边*/
  line(p[i][0], p[i][1], p[(i+1)%n][0], p[(i+1)%n][1]);
  /*用求极值的算法,从多边形顶点数组 p 中,求出 xmin,xmax,ymin,ymax*/
  for(y=ymin;y<=ymax;y++)
  {
    flag=－1;
    for(x=xmin;x<=xmax;x++)
    {
      if(getpixel(x,y)= = boundarycolor) flag=－flag;
      if(flag= =1)putpixel(x,y, newcolor);
    }
  }
}
```

实习三 二维图形的裁剪

- 实习要求

在使用计算机处理图形信息时，往往计算机内部存储的图形比较大，而屏幕显示只是图的一部分。为了看到图形的局部细节，常利用缩放技术，把图形的局部区域放大显示。在放大显示图形的一部分区域时，必须确定图形中哪些部分落在显示区域之内，哪些部分落在显示区域之外，以便只显示落在显示区域内的那部分图形，这个选择过程称为裁剪。

在进行裁剪时，画面中对应于屏幕显示的那部分区域称为窗口。一般把窗口定义为矩形，也可以是其它多边形。裁剪算法包括：对线段的裁剪及对多边形的裁剪。

- 实习内容

实现二维图形的裁剪。

- 基本要求

用 Sutherland－Hodgeman 多边形裁剪算法实现对矩形窗口某一边界的裁剪。

- 提高要求

1、用 Sutherland－Hodgeman 多边形裁剪算法实现对矩形窗口四个边界的全裁剪。
2、用 Sutherland－Hodgeman 多边形裁剪算法实现对任意凸多边形窗口的全裁剪。

一、 Cohen—Sutherland 线段裁剪算法

- Cohen—Sutherland 线段裁剪算法思想

该算法也叫线段编码裁剪算法，其裁剪对象是线段，裁剪窗口是矩形。裁剪窗口及其延长线把二维平面分成九个区域，每个区域各用一个四位二进制代码标识。算法首先对线段的两个端点按所在区域进行编码，然后根据编码可以迅速地判明线段相对于窗口的位置。对于每条线段 P_1P_2 分为三种情况处理：

1、若 P_1P_2 完全在窗口内，则显示该线段，简称"保留"；

2、若 P_1P_2 同在窗口某一外侧（左侧、右侧、上侧、下侧），则不显示该线段，简称"舍弃"；

3、若线段不满足上述两个条件，则求交点。交点将线段分为两段，其中一段完全在窗外，可舍弃；然后对另一段重复上述处理。

- Cohen—Sutherland 线段裁剪算法步骤

1、设初值：visible＝0；done＝1；

2、端点 p_1 编码得 c_1；端点 p_2 编码得 c_2 ；

3、do｛ if（ c_1 为 0 且 c_2 为 0） ｛ visible＝1；done＝0； ｝

else if（ c_1 与 c_2 按位逻辑乘不为 0） done＝0；

else

｛

if（c_1 不为 0） /*将窗外端点编码记在 c 中。*/

c ＝c_1；

```
            else
              c =c2;
            if（在上边界外） 与上边界求交点 p（x，y）；
            else if（在下边界外） 与下边界求交点 p（x，y）；
            else if（如在左边界外） 与左边界求交点 p（x，y）；
            else 与右边界求交点 p（x，y）；
            if（c 等于 c1） /* 将窗外端点移到交点处并重新编码。*/
             { p1=p；并对端点 1 重新编码；}
            else
           { p2=p；并对端点 2 重新编码；}
        }
    } while（done）；
```

二、多边形裁剪的 Sutherland—Hodgman 算法

在裁剪问题中，要求：封闭的多边形裁剪后仍应是封闭的多边形。为了达到这个目的，可以使用 Sutherland 和 Hodgman 发明的逐边多边形裁剪算法。

Sutherland—Hodgman 多边形裁剪算法思想

该算法的基本思想是每次用窗口的一条边界及其延长线来裁剪多边形的各边。多边形通常由它的顶点序列来表示，经过裁剪规则针对某条边界裁剪后，结果形成新的顶点序列，又留待下条边界进行裁剪，...，直到窗口的所有边界都裁剪完毕，算法形成最后的顶点序列，才是结果多边形（它可能构成一个或多个多边形）。

当多边形一个顶点 P_i 相对于窗口某条边界及其延长线进行剪裁时，不外乎下列四种情况（即裁剪规则）：

1、顶点 P_i 在内侧，前一顶点 P_{i-1} 也在内侧，则将 P_i 纳入新的顶点序列；
2、顶点 P_i 在内侧，前一顶点 P_{i-1} 在外侧，则先求交点 Q，再将 Q、P_i 依次纳入新的顶点序列；
3、顶点 P_i 在外侧，前一顶点 P_{i-1} 在内侧，则先求交点 Q，再将 Q 纳入新的顶点序列；
4、顶点 P_i 与前一顶点 P_{i-1} 均在外侧，则顶点序列中不增加新的顶点。

Sutherland—Hodgman 多边形裁剪算法步骤

考虑多边形相对于一条边界及其延长线进行裁剪的算法：

1．从主函数得到待裁剪多边形的顶点序列 P[][2]、顶点序列数 n、窗口一条边界参数 x_l（假如为矩形窗口的左边界）；

2．赋初值：将顶点序列中的最后一个顶点赋给前一顶点 S；

设置初始标志 flag：
if(S 在边界内侧)flag=0;
else flag=1；
设新的顶点序列数 j=0；

3．对多边形各顶点进行裁剪规则处理，结果放入新的多边形顶点序列 Q[][2]中：

```
for(对第一个顶点直到最后一个顶点，逐一处理）
{
  if(Pi 在边界内侧)
  {
    if(flag!=0)
    {
      flag=0;
      求交点并放入新的多边形顶点序列 Qj 中；
      j++；
    }
    将当前顶点放入新的多边形顶点序列 Qj 中：Qj=Pi；
    j++；
  }
  else
  {
    if(flag==0)
    {
      flag=1;
      求交点并放入新的多边形顶点序列 Qj 中；
      j++；
    }
  }
  将当前顶点赋给 S：S=Pi；
}
```

4．做返回准备：

将新的多边形顶点序列 Q 又逐一放回原多边形顶点序列 P 中：P=Q；
将新的多边形顶点数 j 放回原多边形顶点数 n 中：n=j；

实习提示

1、算法步骤中的顶点都是矢量表示，因此程序中要换成坐标分量的形式；

2、判断顶点是否在边界内侧，可用：

（1）坐标法：if(p[i][0]>=xl)； /*当前点是否在左边界内侧*/

（2）也可用向量叉积的符号判别法，具体内容请看"网络教室"点在边界内侧的判断方法。

3、尽量将参数设置成全局变量，以减少主函数和被调函数间参数的传递。

例如：窗口边界参数、顶点数等均可作为全局变量处理。

实习四 图形变换

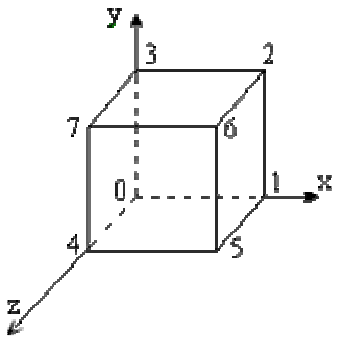
实习要求

本次实习内容包括：二维几何变换、二维观察变换、三维几何变换、三维观察变换。要求学生从二维变换和三维变换中各选一个题目予以实现。

图形变换的实质就是对二维点或三维点的坐标进行变换，形成新的坐标。由于直线经这些变换后仍是直线，所以不必对组成多边形或多面体的每一个象素点都进行坐标变换，而只需要变换所有顶点坐标，在相邻接的顶点间画线段就可以再现变换后的多边形或多面体的线框图。因此，多边形和多面体在计算机内的表示，既要存储各顶点的坐标，又要体现出顶点间的邻接关系。多边形的表示比较简单，只需要依顺时针或逆时针次序存放各顶点坐标即可表示一个多边形。下面介绍一种简单的多面体表示法：点表、边表表示法。

点表是一个 N×3 数组（N 是多面体顶点数），用来存储各顶点的 X、Y、Z 坐标；边表是一个 M×2 数组（M 是多面体边数），用来记录各边的前后顶点在点表中的序号。

例如，如图所示的单位正方体就可以用如下点表和边表共同表示：



点 表

序号	X	Y	Z
0	0	0	0
1	1	0	0
2	1	1	0
3	0	1	0
4	0	0	1
5	1	0	1
6	1	1	1
7	0	1	1

边 表

边序号	前顶点序号	后顶点序号
0	0	1
1	1	2
2	2	3
3	3	0
4	4	5
5	5	6
6	6	7
7	7	4
8	0	4
9	1	5
10	2	6
11	3	7

实习内容

1、编程实现多边形的二维几何变换并在屏幕上绘出变换后的多边形。
2、编程实现对多面体由窗口到视口的转换显示。
3、编程实现多面体的三维几何变换并在屏幕上绘出变换后的多面体。
4、编程实现多面体的投影变换并在屏幕上绘出得到的二维图形。

基本要求

1、二维几何变换。程序运行时，应先由用户输入多边形各顶点坐标，而后显示一个简单的菜单，由用户选择实施以下变换之一：旋转（任意中心）、变比（任意中心）和平移；然后根据用户选择的不同变换，提示用户输入相应的参数（对旋转变换，参数是旋转中心坐标和旋转角度；对变比变换，参数是变比中心坐标和 X、Y 方向上的变比因子；对平移变换，参数是 X、Y 方向上的平移量）；最后在屏幕上绘出变换后的多边形,如果变换前的多边形能在屏幕上显示，最好用不同颜色绘出变换前的多边形。

2、二维观察变换。可由用户在程序运行时输入多个多面体数据，接着输入窗口数据和屏幕视口数据，然后进行显示。注意，可多次运行该程序，通过输入不同的参数，以显示不同的效果。

3、三维几何变换。多面体可由用户在程序运行时输入（注意，这时用户不仅要输入各顶点坐标，还要输入各条边的信息），也允许在程序中用相应的数据结构设定好（这样可简化编程），显示一个简单的菜单，由用户选择实施以下变换之一：绕 X 轴旋转、绕 Y 轴旋转、绕 Z 轴旋转、变比（以原点为中心）和平移，然后根据用户选择的不同变换，提示用户输入相应的参数（对旋转变换，参数是旋转角度；对变比变换，参数是 X、Y、Z 方向上的变比因子；对平移变换，参数是 X、Y、Z 方向上的平移量）；最后在屏幕上绘出变换后的多面体,如果变换前的多面体能在屏幕上显示，最好用不同颜色绘出变换前的多面体。

如何在二维屏幕上绘出三维的多面体呢？这其实就是投影的问题，这里，为了将关注焦点集中于三维几何变换，要求学生在绘出多面体时一律采用主视图，即三维坐标变为二维坐标时只是简单地舍弃 Z 坐标，X、Y 坐标不变。

4、三维观察变换。多面体可由用户在程序运行时输入（注意，这时用户不仅要输入各顶点坐标，还要输入各条边的信息），也允许在程序中用相应的数据结构设定好（这样可简化编程），显示一个简单的菜单，由用户选择显示以下投影之一并在屏幕上绘出得到的二维图形：

（1）主视图；
（2）侧视图；
（3）俯视图；
（4）任意投影方向的平行投影，设 XY 平面为投影平面，由用户指定投影方向；
（5）一点透视投影，设 XY 平面为投影平面，由用户指定灭点位置（三维坐标）。

提高要求

1、在二维几何变换中除上述三种基本变换之外，增加以下变换选择：

（1）相对于 X 轴的反射变换。
（2）相对于 Y 轴的反射变换。
（3）相对于直线 y=x 的反射变换。
（4）相对于直线 y=-x 的反射变换。
（5）相对于原点的对称反射变换。
（6）错切变换（Shearing），参数是 X 方向和 Y 方向上的"错切因子"。X 方向上的"错切因子"指的是 Y 坐标为 1 的点经变换后 X 坐标的增加量；Y 方向上的"错切因子"指的是 X 坐标为 1 的点经变换后 Y 坐标的增加量。

2、为程序增加组合变换的处理能力，允许用户实施多次变换，每次的变换种类及相关参数均通过上述菜单系统进行选择，最后在屏幕上绘出多次变换后的多边形。

3、在三维几何变换里，除了上述三种基本变换之外，增加以下变换选择：

（1）相对于 YZ 平面的反射变换。
（2）相对于 XZ 平面的反射变换。
（3）相对于原点的对称反射变换。
（4）相对 Z 轴的错切变换（Shearing），参数是 X 方向和 Y 方向上的"错切因子"。X 方向上的"错切因子"指的是 Z 坐标为 1 的点经变换后 X 坐标的增加量；Y 方向上的"错切因子"指的是 Z 坐标为 1 的点经变换后 Y 坐标的增加量。
（5）相对 X 轴的错切变换，参数是 Y 方向和 Z 方向上的"错切因子"。Y 方向上的"错切因子"指的是 X 坐标为 1 的点经变换后 Y 坐标的增加量；Z 方向上的"错切因子"指的是 X 坐标为 1 的点经变换后 Z 坐标的增加量。
（6）相对 Y 轴的错切变换，参数是 Z 方向和 X 方向上的"错切因子"。Z 方向上的"错切因子"指的是 Y 坐标为 1 的点经变换后 Z 坐标的增加量；X 方向上的"错切因子"指的是 Y 坐标为 1 的点经变换后 X 坐标的增加量。

4、为程序增加组合变换的处理能力，允许用户实施多次变换，每次的变换种类及相关参数均通过上述菜单系统进行选择，最后在屏幕上绘出多次变换后的多面体。

5、在上述菜单中增加以下投影选择：

（1）正等轴测投影;

（2）正二轴测投影，设原用户坐标系中 X 和 Y 方向单位长度的投影长度相等，由用户指定原用户坐标系中 Z 方向单位长度的投影长度（大于等于 0 小于等于 1）。

（3）斜等测投影;

（4）斜二测投影。

实习五 投影变换

实习内容

编程实现多面体的投影变换并在屏幕上绘出得到的二维图形。

基本要求

多面体可由用户在程序运行时输入（注意，这时用户不仅要输入各顶点坐标，还要输入各条边的信息），也允许在程序中用相应的数据结构设定好（这样可简化编程），显示一个简单的菜单，由用户选择显示以下投影之一并在屏幕上绘出得到的二维图形：

（1）主视图;

（2）侧视图;

（3）俯视图;

（4）任意投影方向的平行投影，设 XY 平面为投影平面，由用户指定投影方向（三维空间矢量）；

（5）一点透视投影，设 XY 平面为投影平面，由用户指定灭点位置（三维坐标）。

提高要求

在上述菜单中增加以下投影选择：

（1）正等轴测投影;

（2）正二轴测投影，设原用户坐标系中 X 和 Y 方向单位长度的投影长度相等，由用户指定原用户坐标系中 Z 方向单位长度的投影长度（大于等于 0 小于等于 1）。

（3）斜等测投影;

（4）斜二测投影。

实习六 曲线拟合

实习要求

本次实习内容包括：插值方式的拉格朗日插值曲线和三次样条曲线以及逼近方式的贝齐尔曲线和 B 样条曲线。要求学生从两种方式中各选一个题目予以实现。

实习内容

1、编程实现用三次多项式来得到多于四个控制点的拉格朗日插值曲线。

下面是用三次拉格朗日插值公式画曲线的算法：

```
langran(xa,ya,za,k,l_section)
int *xa,*ya,*za,k,l_section;
/*xa,ya,za 为存放 k 个控制点坐标的数组，l_section 为每一曲线段的直线段数。 */
```

```
{
  for(i=0;i<l_section;i++)
  {
    u=i/l_section;
    /*求中间段混合函数值*/
    blend[0][i]=u*(u-1)*(u-2)/(-6);
    blend[1][i]=(u+1)*(u-1)*(u-2)/2;
    blend[2][i]=(u+1)*u*(u-2)/(-2);
    blend[3][i]=(u+1)*u*(u-1)/6;
    v=u-1; /*求第一段混合函数值*/
    f_blend[0][i]=v*(v-1)*(v-2)/(-6);
    f_blend[1][i]=(v+1)*(v-1)*(v-2)/2;
    f_blend[2][i]=(v+1)*v*(v-2)/(-2);
    f_blend[3][i]=(v+1)*v*(v-1)/6;
    w=u+1; /*求最后一段混合函数值*/
    l_blend[0][i]=w*(w-1)*(w-2)/6;
    l_blend[1][i]=(w+1)*(w-1)*(w-2)/2;
    l_blend[2][i]=(w+1)*w*(w-2)/(-2);
    l_blend[3][i]=(w+1)*w*(w-1)/6;
  }
  for(i=0;i<4;i++) /*取 0～3 控制点坐标*/
  {
    xsm[i]=xa[i];
    ysm[i]=ya[i];
    zsm[i]=za[i];
  }
  moveto(xa[0],ya[0];za[0]);
  for(j=0,j<l_section;j++)
  /*画 0—1 控制点间的曲线*/
  {
    x=y=z=0;
    for(i=0;i<4;i++)
    /*求第 j 直线段的插值点坐标*/
    {
      x=x+xsm[i]*f_blend[i][j];
      y=y+ysm[i]*f_blend[i][j];
      z=z+zsm[i]*f_blend[i][j];
    }
    lineto(x,y,z); /*画第 j 个直线段*/

  }
  for(m=4;m<k;m++)
  /*依次画出各四个控制点之间的中间曲线段*/
```

```
    {
      for(j=0;j<line_section;j++)
      {
        x=y=z=0;
        for(i=0;i<4;i++)
        {
          x=x+xsm[i]*blend[i][j];
          y=y+xsm[i]*blend[i][j];
          z=z+xsm[i]*blend[i][j];
        }
        lineto(x,y,z);
      }
      for(i=0;i<3;i++)
      /*推进下一个样本点*/
      {
        xsm[i]=xsm[i+1];
        ysm[i]=ysm[i+1];
        zsm[i]=zsm[i+1];
      }
      xsm[3]=xa[m];
      ysm[3]=ya[m];
      zsm[3]=za[m];
    }
    for(j=0;j<l_section;j++)
    /*画曲线的最后一个曲线段 k-1～k 点*/
    {
      x=y=z=0;
      for(i=0;i<4;i++)
      {
        x=x+smx[i]*l_blend[i][j];
        y=y+smx[i]*l_blend[i][j];
        z=z+smx[i]*l_blend[i][j];
      }
      lineto(x,y,z);
    }
    lineto(xsm[3],ysm[3],zsm[3]);
  }
```

2、实现三次样条的算法步骤：

（1）输入控制点坐标$Q_0, Q_1, \cdots, Q_n$

（2）输入 $Q_0^{'}, Q_n^{'}$,

（3）用式(7-2-5)和式(7-2-6)的等式右边矩阵计算 $\alpha_i, \beta_i, \gamma_i$,

$$\begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & \cdots & 0 & 0 \\ & & \cdots & & & \cdots & & \cdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} Q_1^{'} \\ Q_2^{'} \\ Q_3^{'} \\ \cdots \\ Q_{n-1}^{'} \end{bmatrix} = 3 \begin{bmatrix} Q_2 - Q_0 - Q_0^{'}/3 \\ Q_3 - Q_1 \\ Q_4 - Q_2 \\ \cdots \\ Q_n - Q_{n-2} - Q_n^{'}/3 \end{bmatrix} \tag{7-2-5}$$

$$\beta_i = -\frac{1}{4 + \beta_{i-1}}$$
$$\alpha_i = (\alpha_{i-1} - \gamma_i) * \beta_i \qquad \text{(i=1,2,...,n-1)} \tag{7-2-6}$$

```
α0 = β0 = 0;
for(i = 1;i <= n -1;i ++)
{   βi = -1/(4 + βi-1)   ;
    if (i == 1)
      {γi = 3*(Q2 - Q0) - Q0'   ;}
    else   if (i == n - 1)
             {γi = 3*(Qn - Qn-2) - Qn'   ;}
           else
             {γi = 3*(Qi+1 - Qi-1)   ;}
    αi = (αi-1 - γi)*βi   ;
}
```

（4）用式(7-2-7)求 $Q_i^{'}$,

$$Q_i^{'} = \alpha_i + \beta_i \cdot Q_{i+1}^{'} \qquad \text{(i=n-1,...,2,1)} \tag{7-2-7}$$

```
for(i = n -1;i >= 1;i - -)
{   Qi' = αi + βi *Qi+1')   ;}
```

（5）用式（7-2-3）求第 i 个曲线段的三次多项式系数 a_i, b_i, c_i, d_i,并绘出第 i 段曲线。

$$
\begin{aligned}
a_i &= Q_i' + Q_{i-1}' - 2(Q_i - Q_{i-1}) \\
b_i &= -Q_i' - 2Q_{i-1}' + 3(Q_i - Q_{i-1}) \\
c_i &= Q_{i-1}' \\
d_i &= Q_{i-1}
\end{aligned}
\qquad (7\text{-}2\text{-}3)
$$

```
for(i=1;i<=n;i++)
 {
    求第 i 个曲线段的三次多项式系数 ai,bi,ci,di;
    for(t=0;t<=1.0;t+=0.01)
     {
        pi=ait3 + bit2 + cit + di;
        if (t==0) moveto(pi);
        lineto(pi);
      }
  }
```

算法结束。

提示：以上算法中各量的求解，均是指对矢量的求解，其中包括坐标点 Q_i、切线向量Q_i'、a_i 、b_i 、c_i、三次多项式系数 a_i,b_i,c_i,d_i 以及曲线上的各点 p_i。

3、编程实现贝齐尔曲线拟合。

4、编程实现 B 样条曲线拟合。

基本要求

1、弄懂上述三次拉格朗日插值曲线算法，并予以实现。

2、弄懂三次样条曲线算法并予以实现。

3、用贝齐尔曲线定义来实现曲线拟合。

4、编程实现均匀 B 样条曲线拟合。

提高要求

1、用贝齐尔曲线的几何作图法实现曲线拟合。

2、用三次贝齐尔曲线来拼成多于四个控制点的曲线。

3、用三次 B 样条曲线来拼成多于四个控制点的曲线。

4、编程实现非均匀 B 样条曲线拟合，并与均匀 B 样条曲线进行比较。