计算机图形学作业一

一、名词解释

1 计算机图形学

研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法。

2 涮新频率

场频又称为刷新频率，即显示器的垂直扫描频率，指显示器每秒所能显示的图象次数，单位为赫兹(Hz)。场频越大，图象刷新的次数越多，图象显示的闪烁就越小，画面质量越高。

3 区域填充

区域---指已经表示成点阵形式的填充图形，是象素的集合

区域填充是指将区域内的一点(常称种子点)赋予给定颜色 ,然后将这种颜色扩展到整个区域内的过程。

4 分辨率

阴极射线管在水平或垂直方向单位长度上能识别的最大像素个数

5 扫描转换

图形生成算法针对后一种图形的光栅化的情形，给出在光栅扫描显示器等数字设备上确定一个最佳逼近于图形的像素集的过程称图形的扫描转换。

二、简答题

1 计算机图形学研究的主要内容是什么？

简单地说，计算机图形学的主要研究内容就是研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法。

2 光栅扫描的显示子系统由哪几个逻辑部件组成，各组成部分的功能?

帧缓冲处理器（存储屏幕上像素的颜色值）、视频控制器（建立帧缓存与屏幕像素之间的一一对应，负责刷新）、显示处理器（扫描转换待显示图形）、CRT（图形显示设备）。

3 什么是走样？常见的走样现象有哪些?反走样技术有哪些?

走样：光栅系统在理论上只能用光栅网格上的象素近似地描绘平滑的直线、多边形和诸如圆与椭圆那样的曲线图元的边界。它引起了锯齿或阶梯状，这种视觉人工痕迹是信号处理理论中被称为走样的错误采样的表现。

反走样：在光栅图形显示器上绘制非水平且非垂直的直线或多边形边界时，或多或少会呈现锯齿状或台阶状外观。这是因为直线、多边形、色彩边界等是连续的，而光栅则是由离散的点组成，在光栅显示设备上表现直线、多边形等，必须在离散位置采样。由于采样不充分重建后造成的信息失真，就叫走样(aliasing)。而用于减少或消除这种效果的技术，就称为反走样(antialiasing)。

计算机生成图像时通常存在三种走样现象中的两种：锯齿形边以及图形细节或纹理绘制失真。第三种现象出现在显示非常微小对象的场合。

基本上反走样方法可分为两类。第一类是提高分辨率 即增加采样点(提高采样频率)。然而，CRT 光栅扫描设备显示非常精细光栅的能力是有限的，因此人们通常是在较高分辨率上对光栅进行计算，然后采用某种平均算法(滤除高频分量)得到较低分辨率的象素的属性，并显示在分辨率较低的显示器上 。这种方法称为超采样或后置滤波。另一类反走样是把像素作为一个有限区域，对区域采样来调整像素的亮度，以光顺边界来减小锯齿现象 J。这种方法等价于图像的前置滤波。

4 多边形填充扫描线算法包括哪些计算步骤？

对于一条扫描线，多边形的填充过程可以分为四个步骤：

(1) 求交：计算扫描线与多边形各边的交点；

(2) 排序：把所有交点按x值递增顺序排序；

(3) 配对：第一个与第二个，第三个与第四个等等；每对交点代表扫描线与多边形的一个相交区间；

(4) 填色：把相交区间内的象素置成多边形颜色；

5 什么是裁剪？二维编码裁剪法的基本编码方式？

裁剪：确定一个图形的哪些部分在窗口内，必须显示；哪些部分在窗口外，不该显示的过程。

原理：对窗口及其延长线分割出的平面 9 个区域进行编码， 根据线段两个端点的编码判断其与窗口之间的关系。算法如下：

① 对线段的两个端点 P_1、P_2 进行按其所处的位置进行编码，分别记为 code1、code2；

② 如果 code1=0 且 code2=0,说明线段在窗口内，全部可见；否则到③；

③ 若 code1&code2 ≠0 则说明线段在某一窗口边的延长线的外侧，全部不可见；否则到④；

④ 在线段与窗口延长线的交点处把线段分为两段，并对两段分别编码。其中交点看作是两个点，一个在与之相交的窗口延长线外侧，一个在内侧。两段线段分别进行①～④的判断过程，则其中一段必然符合③的条件，可弃之；另一段重复①～④的处理，直到剩余部分的线段完全可见或完全不可见。

三、综合题

1 简述 DDA 画线算法的基本思想，使用中点画线算法画斜率介于 0° 和 45° 之间的直线的 C 语言代码。

// 实现 DDA 算法。

1．算法描述：设直线方程为：$y = mx + b$，

$$m = \frac{y_2 - y_1}{x_2 - x_1}, b = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$$

对于直线，其数值微分是 $m = \frac{\Delta y}{\Delta x} = \frac{y(i+1) - y(i)}{x(i+1) - x(i)}$,由上式得，y(i+1)=y(i)+m(x(i+1)-x(i))，于是知，使 x(i)增 1，即 x(i+1)=x(i)+1 时，y(i+1)=y(i)+m,为画线精确，应使相邻的画出点的坐标值相差最大值为 1，这样可以得到画线段的数值微分分析器（Digital Differential Analyzer,简称 DDA），算法如下：

```
void DDADrawLine::LineDDA(int x0, int y0, int x1, int y1)
{
    float x = 0.0;
    float y = 0.0;
    float m = 0.0;
    // 添加增量，实现增量思想
    float dx = x1 - x0;
    float dy = y1 - y0;
    if (dx != 0)
    {
        m = dy / dx;
        if (m <= 1 && m >= -1)
        {
            y = y0;
            for (x = x0; x <= x1; x++)
            {
                glVertex2i(x, int(y + 0.5));
                y += m;
            }
        }
        if (m > 1 || m < -1)
        {
            m = 1 / m;
            x = x0;
            for (y = y0; y <= y1; y++)
            {
                glVertex2i(int(x + 0.5), y);
                x += m;
            }
        }
    }
    else
    {
        int x = x0;
        int y = 0;
```

```
            y = (y0 <= y1) ? y0 : y1;
            int d = fabs((double)(y0 - y1));
            while (d >= 0)
            {
                glVertex2i(x, y);
                y++;
                d--;
            }
        }
}
```

2 写出中点画圆算法的 C 程序

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#define x0 400
#define y0 300                              //定义全局变量 x0,y0:坐标轴中心（x0,y0)
void Middle_point_draw_circle(int x1, int y1, int r)
{
    int d0, x = 0, y = r;//d0 是判别式的值
    d0 = 1.25 - r;     //判别式的初始值，1.25 可以改为 1
    while (x < y)
    {
        if (d0 >= 0)
        {
            d0 = d0 + 2 * (x - y) + 5;                  //d0 一定要先比 x,y 更新
            x += 1;                          //因为 d0 表达式中的 x,y 是上一个点
            y -= 1;
            putpixel(((x + x1) + x0), (y0 - (y + y1)), RED);              //(x,y)
            putpixel(((-x + x1) + x0), (y0 - (y + y1)), RED);             //(-x,y)
            putpixel(((y + x1) + x0), (y0 - (x + y1)), RED);              //(y,x)
            putpixel(((-y + x1) + x0), (y0 - (x + y1)), RED);             //(-y,x)
            putpixel(((x + x1) + x0), (y0 - (-y + y1)), RED);             //(x,-y)
            putpixel(((-x + x1) + x0), (y0 - (-y + y1)), RED);            //(-x,-y)
            putpixel(((y + x1) + x0), (y0 - (-x + y1)), RED);             //(y,-y)
            putpixel(((-y + x1) + x0), (y0 - (-x + y1)), RED);            //(-y,-x)
            Sleep(50);
        }
        else
        {
            d0 = d0 + 2 * x + 3;
            x += 1;
            y = y;
            putpixel(((x + x1) + x0), (y0 - (y + y1)), RED);              //(x,y)
            putpixel(((-x + x1) + x0), (y0 - (y + y1)), RED);             //(-x,y)
            putpixel(((y + x1) + x0), (y0 - (x + y1)), RED);              //(y,x)
```

```
                putpixel(((-y + x1) + x0), (y0 - (x + y1)), RED);          //(-y,x)
                putpixel(((x + x1) + x0), (y0 - (-y + y1)), RED);          //(x,-y)
                putpixel(((-x + x1) + x0), (y0 - (-y + y1)), RED);         //(-x,-y)
                putpixel(((y + x1) + x0), (y0 - (-x + y1)), RED);          //(y,-y)
                putpixel(((-y + x1) + x0), (y0 - (-x + y1)), RED);         //(-y,-x)
                Sleep(50);
            }
        }
}
void main()
{
    int x1, y1, r;
    printf("请输入中点画圆算法圆心坐标(x1,y1)和圆的半径 r:\n");
    scanf("%d %d %d", &x1, &y1, &r);
    initgraph(x0 * 2, y0 * 2);                  //初始化图形窗口大小
    setbkcolor(WHITE);
    cleardevice();
    setcolor(BLACK);
    line(x0, 0, x0, y0 * 2);                //坐标轴 X
    line(0, y0, x0 * 2, y0);                 //坐标轴 Y
    Middle_point_draw_circle(x1, y1, r);                    //中点画圆算法
    _getch();                                                   //等待一个任意输入结束
    closegraph();                                               //关闭图形窗口
}
```