三、填空题

1、按照工作原理分，目前显示器主要有CRT显示器、\_液晶显示器\_以及等离子显示器几种。

2、生成直线常用的算法主要有DDA法、＿中点画线法＿＿、直线的Bresenham算法。

3、二维图形的基本变换主要有平移、\_\_\_比例\_\_\_，对称，旋转，错切等。

4、在线段AB的区域编码裁剪算法中，如A、B两点的编码均为 零 （填汉字），则该线段位于窗口内；如A、B两点的码按位与不为零，则该线段在窗口外。

5、透视投影按照灭点个数可分为 一点透视 、两点透视、三点透视三种。

6、平面几何投影可以分为两大类，即 正平行投影 和斜平行投影。

7、刷新式CRT图形显示器按扫描方式分为随机扫描和 光栅扫描 两种。

8、常用的用户接口形式：子程序库，专用语言，交互命令

10、生成直线的四点要求是 生成直线要直,直线终止点要准,直线粗细要均,速度要快

11、可以用基本的交互技术形成三种组合交互任务：对话框，构造，操作。

12、图形的基本要素一般有 点,线,面,体 .

8. 计算机图形系统中常用的用户接口形式有子程序库、专用语言和交互命令三种形式。

13、处理图形时常涉及的坐标系有：用户坐标系、设备坐标系、建模坐标系、观察坐标系

1. 将多边形外部一点A 与某一点B 用线段连接，若此线段与多边形边界相交的次数为偶数， 则点B 在多边形外部，若此线段与多边形边界相交的次数为奇数，则点B 在多边形内部。

5、平面几何投影可以分为两大类，即＿正平行投影＿和斜平行投影

1. 通常投影根据 可以分为 投影和 投影。

4. 字符的图形表示可以分为 和 两种形式。

7. 平面几何投影根据 可以分为 和 。

4、简述图形系统的组成与功能

组成：计算机硬件、图形输入/输出设备、计算机系统软件和图形软件

功能：（1）计算功能（2）存储功能（3）交互功能（4）输入功能（5）输出功能

36、计算机图形学研究的内容：

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和算法

计算机动画，地理信息系统，人机交互，真实感图形显示，虚拟现实，科学计算可视化，并行图形处理

35、常涉及的四种坐标系：用户坐标系，设备坐标系，观察坐标系，建模坐标

1、计算机图形学：用计算机建立、存储、处理某个对象的模型，并根据模型产生该对象图形输出的有关理论、方法与技术，称为计算机图形学。

2、计算机图形标准：计算机图形标准是指图形系统及其相关应用程序中各界面之间进行数据传送和通信的接口标准。

3、列举有关计算机图形学的应用

（1）计算机辅助设计与制造

（2）计算机辅助绘图

（3）计算机辅助教学

（4）办公自动化和电子出版技术

（5）计算机艺术

（6）在工业控制及交通方面的应用

（7）在医疗卫生方面的应用

（8）图形用户界面

3、列举有关计算机图形学的应用电子出版及办公自动化

计算机动画及广告、计算机艺术、科学计算的可视化、工业模拟、计算机辅助教学

5、输入设备：键盘、鼠标、光笔、触摸屏、操纵杆、跟踪球和空间球、数据手套、图像扫描、数字化仪、声频视频输入系统

显示设备：CRT显示器、液晶显示器以及等离子显示器

6、生成直线常用的算法主要有DDA法、中点画线法、直线的Bresenham算法。

7、掌握：画圆、椭圆的算法（Bresenham）

8、填充算法：重点是种子填充算法（代码）

9、字符的图形表示分为点阵字符和矢量字符。

10、走样：在光栅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。这是由 于直线或多边形边界在光栅显示器的对应图形都是由一系列相同亮度的离散像素构成的。这种用离散量表 示连续量引起的失真，称为走样（aliasing）。

11、反走样：用于减少和消除用离散量表示连续量引起的失真效果的技术，称为反走样。

12、二维图形的基本变换主要有平移、比例，对称，旋转，错切等。

13、投影分为正投影和斜投影。

14、灭点：对于透视投影，若一组平行线平行于投影平面时，它们的透视投影仍然保持平行，而不平行于投影面的平行线的投影会聚到一个点，这个点就是灭点。

15刷新式CRT图形显示器按扫描方式分为随机扫描和光栅扫描两种。

16、图形消隐：计算机为了反映真实的图形，把隐藏的部分从图中消除。

17、几何变换：几何变换的基本方法是把变换矩阵作为一个算子，作用到图形一系列顶点的位置矢量，从 而得到这些顶点在几何变换后的新的顶点序列，连接新的顶点序列即可得到变换后的图形。

18、计算几何：计算几何研究几何模型和数据处理的学科，讨论几何形体的计算机表示、分析和综合，研 究如何方便灵活、有效地建立几何形体的数学模型以及在计算机中更好地存贮和管理这些模型数据。

19、裁剪：识别图形在指定区域内和区域外的部分的过程称为裁剪算法，简称裁剪。

20、透视投影：空间任意一点的透视投影是投影中心与空间点构成的投影线与投影平面的交点。

21、透视投影按照灭点个数可分为一点透视、两点透视、三点透视三种。

22、投影变换：把三维物体变为二维图形表示的变换称为投影变换。

23、窗口：世界坐标的范围是无限大的。为了使规格化设备坐标上所显示的世界坐标系中的物体有一个合 适的范围与大小，必须首先对世界坐标系指定显示范围，它通常是一个矩形，这个矩形被称为窗口。

24、视区：在规格化设备坐标系上也要指定一个矩形区域与窗口对应，显示窗口里的内容，这个矩形被称 为视区。

25、坐标系统：为了描述、分析、度量几何物体的大小、形状、位置、方向以及相互之间的各种关系使用 的参考框架叫做坐标系统。

26、刷新：荧光的亮度随着时间按指数衰减，整个画面必须在每一秒内重复显示许多次，人们才能看到一 个稳定而不闪烁的图形，这叫屏幕的刷新。

27、用户坐标系：用户坐标系用户为处理自已的图形时所采用的坐标系，单位由用户自己决定。

28、规范化设备坐标系：将各个设备坐标系中的数据化为统一的数据范围从而得到的设备坐标系。

29、规格化变换：图形软件根据窗口与视区的一一对应关系，自动实现从世界坐标到规格化设备坐标的转换，这种从窗口到视区的变换，称为规格化变换。

30、屏幕坐标系统：屏幕坐标系统也称设备坐标系统，它主要用于某一特殊的计算机图形显示设备(如光栅 显示器)的表面的点的定义，在多数情况下，对于每一个具体的显示设备，都有一个单独的坐标系统，在定 义了成像窗口的情况下，可进一步在屏幕坐标系统中定义称为视图区的有界区域，视图区中的成像即为实 际所观察到的图形对象。

31、观察坐标系：观察坐标系通常是以视点的位置为原点，通过用户指定的一个向上的观察向量来定义整 个坐标系统，缺省为左手坐标系，观察坐标系主要用于从观察者的角度对整个世界坐标系内的对象进行重 新定位和描述，从而简化几何物体在投影面的成像的数学推导和计算。

32、简要指出下列术语之间的主要不同之处：

图形和图象：前者是指矢量表示的图，后者是指用点阵表示的图。

随机扫描与光栅扫描：前者是电子束扫描路径随图形不同而不同，后者是电子束扫描路径固定不变。

窗口与视口：前者是在世界（用户）坐标系中定义的，后者是设备坐标系中定义的。

数组与矩阵：前者是计算机语言中的术语，后者是数学中的术语。

逼近与插值：前者是所求曲线大致通过数据点，后者是所求曲线通过每一个数据点。

1. 图形交互技术有哪些？

回显，引力域、草拟技术、旋转、形变、网格、约束

定位技术、定向技术、定路经技术、定量技术、文本技术、橡皮筋技术、拖动技术

1. 常用的用户接口形式：子程序库，专用语言，交互命令

1、在图形文件系统中，点、线、圆等图形元素通常都用其几何特征参数来描述，在图形系统中，图形处 理运算的精度不取决于显示器的分辨率，在彩色图形显示器中，使用 RGB 颜色模型。计算机图形学以计 算几何为理论基础。

2、深度缓存算法并不需要开辟一个与图像大小相等的深度缓存数组，深度缓存算法能并行实现，深度缓 存算法中没有对多边形进行排序。

3、计算机图形处理中，除了应用到各种算法外，还经常会处理大量的图形方面的数据，因而必须应用到 数据库技术，图形数据库设计的子库层次是一个简单的、具有普遍存储规则的许多物体的集合，图形数据 库的设计一般有物体和子库两个层次。

4、投影线从视点出发，主灭点最多有 3 个，任何一束不平行于投影面的平行线的透视投影将汇成一点。 在平面几何投影中，若投影中心移到距离投影面无穷远处，则成为平行投影。 5、实体模型和曲面造型是CAD 系统中常用的主要造型方法，曲面造型是用参数曲面描述来表示一个复杂 的物体，从描述复杂性和形状灵活性考虑，最常用的参数曲面是3 次有理多项式的曲面，在曲线和曲面定 义时，使用的基函数应有两个重要性质：凸包性和仿射不变性。

6、简单光反射模型，又称为 Phong 模型，它模拟物体表面对光的反射作用，简单光反射模型主要考虑物 体表面对直射光照的反射作用，在简单光反射模型中，对物体间的光反射作用，只用一个环境光变量做近 似处理。

7、定义了物体的边界也就唯一的定义了物体的几何形状边界，物体的边界上的面是有界的，而且，面的 边界应是闭合的，物体的边界上的边可以是曲线，但在两端之间不允许曲线自相交。

8、透视投影的投影线从视点出发，主灭点最多有 3 个，任何一束不平行于投影面的平行线的透视投影将 汇成一点。

9、图形数据按照目的不同一般可以分为图形的表示数据和图形的显示数据。

10、双线性法向插值法（Phong Shading）的优点是高光域准确。

11、画圆弧的算法有角度 DDA 法、逐点比较法、终点判断法、Bresenham画圆法四种。

12、Z 缓冲器消隐算法是最简单的消除隐藏面算法之一。

13、若要对某点进行比例、旋转变换，首先需要将坐标原点平移至该点，在新的坐标系下做比例或旋转变 换，然后再将原点平移回去。

14、在种子填充算法中所提到的八向连通区域算法同时可填充四向连通区。

15、多边形被两条扫描线分割成许多梯形，梯形的底边在扫描线上，腰在多边形的边上，并且相间排列， 多边形与某扫描线相交得到偶数个交点，这些交点间构成的线段分别在多边形内、外，并且相间排列，边 的连贯性告诉我们，多边形的某条边与当前扫描线相交时，很可能与下一条扫描线相交。

16、透视投影又可分为一点透视、二点透视、三点透视，斜投影又可分为斜等测、斜二测，正视图又可分 为主视图、侧视图、俯视图。

17、Bezier 曲线不一定通过其特征多边形的各个顶点，Bezier 曲线两端点处的切线方向必须与起特征折线 集（多边形）的相应两端线段走向一致，Bezier 曲线可用其特征多边形来定义。

18、扫描线算法对每个象素只访问一次，主要缺点是对各种表的维持和排序的耗费较大 ，边填充算法基 本思想是对于每一条扫描线与多边形的交点，将其右方象素取补，边填充算法较适合于帧缓冲存储器的图 形系统。

19、深度缓冲器算法最简单常用的面向应用的用户接口形式：子程序库、专用语言和交互命令。图形用户 界面的基本元素有窗口、图标、菜单、指点装置。在计算机图形学中，被裁剪的对象可以是线段、多边形 和字符三种形式。

20、扫描仪最重要的参数是光学精度和扫描精度。

DDA算法

原理：假定直线的起点、终点分别为：（x0,y0), (x1,y1)，且都为整数。已知过端点P0 (x0, y0), P1(x1, y1)的直线段L: y=kx+b。直线斜率为k=(y1-y0)/(x1-x0)。计算Yi+1= kxi+1+b= kxi+b+kΔx= yi+kΔx 。当Δx =1; yi+1 = yi+k

即：当x每递增1，y递增k(即直线斜率)；注意上述分析的算法仅适用于|k|≤1的情形。在这种情况下，x每增加1,y最多增加1。当|k|>1时，必须把x，y地位互换

实现:

void DDALine(int x0,int y0,int x1,int y1,int color)

{ int x；

float dx, dy, y, k;

dx = x1-x0; dy=y1-y0;

k=dy/dx ; y=y0;

for (x=x0; x<=x1; x++)

{ drawpixel (x, int(y+0.5), color);

y=y+k;

}

}

中点画线算法

void Midpoint Line (int x0,int y0,int x1, int y1,int color)

{ int a, b, d1, d2, d, x, y;

a=y0-y1; b=x1-x0; d=2\*a+b;

d1=2\*a ; d2=2\* (a+b);

x=x0; y=y0;

drawpixel(x, y, color);

while (x<x1)

{ if (d<0) {x++; y++; d+=d2; }

else {x++; d+=d1;}

drawpixel (x, y, color);

} /\* while \*/

} /\* mid PointLine \*/

试写出能获得整数Bresenham画线算法的c++程序（提示：①假定直线的斜率在0和1之间）

void MidBreshnhamLine(int x0,int y0,int x1,int y1,int color){

int dx,dy,d,UpIncre,DownIncre,x,y;

if(x0>x1){

x=x1;x1=x0;x0=x;

y=y1;y1=y0;y0=y;

}

x=x0;y=y0;

dx=x1-x0;dy=y1-y0;

d=dx-2\*dy;

UpIncre=2\*dx-2\*dy;DownIncre=-2\*dy;

while(x<=x1){

putpixel(x,y,color);

x++;

if(d<0){

y++;

d+=UpIncre;

}

else

d+=DownIncre;

}

}

Brsenham画线算法

void BreshnhamLine(int x0,int y0,int x1,int y1,int color){

int x,y，dx，dy,e;

dx=x1-x0;

dy=y1-y0;

e=-dx;x=x0;y=y0;

while(x<=x1){

putpixel(x,y,color);

x++;

e=e+2\*dy;

if(e>0){

y++;

e=e-2\*dx;

}

}

}

中点画圆算法

void MidBreshnhamCircle(int r,int color)

{

int x,y,d;

x=0;y=r;d=1-r;

while(x<=y){

CirclePoint(x,y,color);

if(d<0)d+=2\*x+3;

else{

d+=2\*(x-y)+5;

y--;

}

x++;

}

}

Breshnham画圆算法

circle (xc, yc, radius, c)

int xc, yc, radius, c;

{

int x, y, p;

x=0;

y=radius;

p=3-2\*radius;

while (x<y)

{

        plot\_circle\_points(xc, yc, x, y, c);

        if (p<0) p=p+4\*x+6;

        else{

                  p=p+4\*(x-y)+10;

                  y-=1;

                }

        x+=1;

}

if (x= =y)

plot\_circle\_points(xc, yc, x, y, c);

}

plot\_circle\_points(xc, yc, x, y, c)

int xc, yc, x, y, c;

{

set\_pixel(xc+x, yc+y, c);

set\_pixel(xc+x, yc+y, c);

set\_pixel(xc+x, yc-y, c);

set\_pixel(xc-x, yc-y, c);

set\_pixel(xc+y, yc+x, c);

set\_pixel(xc-y, yc+x, c);

set\_pixel(xc+y, yc-x, c);

set\_pixel(xc-y, yc-x, c);

}

中点画椭圆算法

MidpointEllipe(a,b, color)

int a,b,color;

{ int x,y; float d1,d2;

x = 0; y = b;

d1 = b\*b +a\*a\*(-b+0.25);

ellipesputpixel(x,y,color);

while( b\*b\*(x+1) < a\*a\*(y-0.5)) 生成上半部分椭圆

{ { if (d1<0)

d1 +=b\*b\*(2\*x+3); x++; }

else { d1 +=(b\*b\*(2\*x+3)+a\*a\*(-2\*y+2))

x++; y--; }

ellipesputpixel(x,y,color);

}//上部分

d2 = b\*b\*(x+0.5)\*(x+0.5)+a\*a\*(y-1)\*(y-1)-a\*a\*b\*b;

while(y >0) 生成下半部分椭圆

{ if (d2 <0) { d2 +=b\*b\*(2\*x+2)+a\*a\*(-2\*y+3);

x++; y--；}

else {d2 += a\*a\*(-2\*y+3); y--; }

ellipesputpixel(x,y,color); }

Void ellipespixel(int x,int y,int color)

{

putpixel(x,y,color)

putpixel(-x,y,color)

putpixel(x,-y,color)

putpixel(-x,-y,color)

}

种子填充算法

递归算法可实现如下：

void FloodFill4(int x,int y,int oldColor,int newColor)

{ if(GetPixel(x,y) == oldColor)

{ PutPixel(x,y,newColor);

FloodFill4(x,y+1,oldColor,newColor);

FloodFill4(x,y-1,oldColor,newColor);

FloodFill4(x-1,y,oldColor,newColor);

FloodFill4(x+1,y,oldColor,newColor);

}

}/\*end of FloodFill4() \*/

边界表示的4连通区域

void BoundaryFill4(int x,int y,int boundaryColor,int newColor)

{ int color;

color = GetPixel(x,y);

if((color != boundaryColor) && (color != newColor))

{ PutPixel(x,y,newColor);

BoundaryFill4(x,y+1,oldColor,newColor);

BoundaryFill4(x,y-1,oldColor,newColor);

BoundaryFill4(x-1,y,oldColor,newColor);

BoundaryFill4(x+1,y,oldColor,newColor);

}

}/\*end of BoundaryFill4() \*/