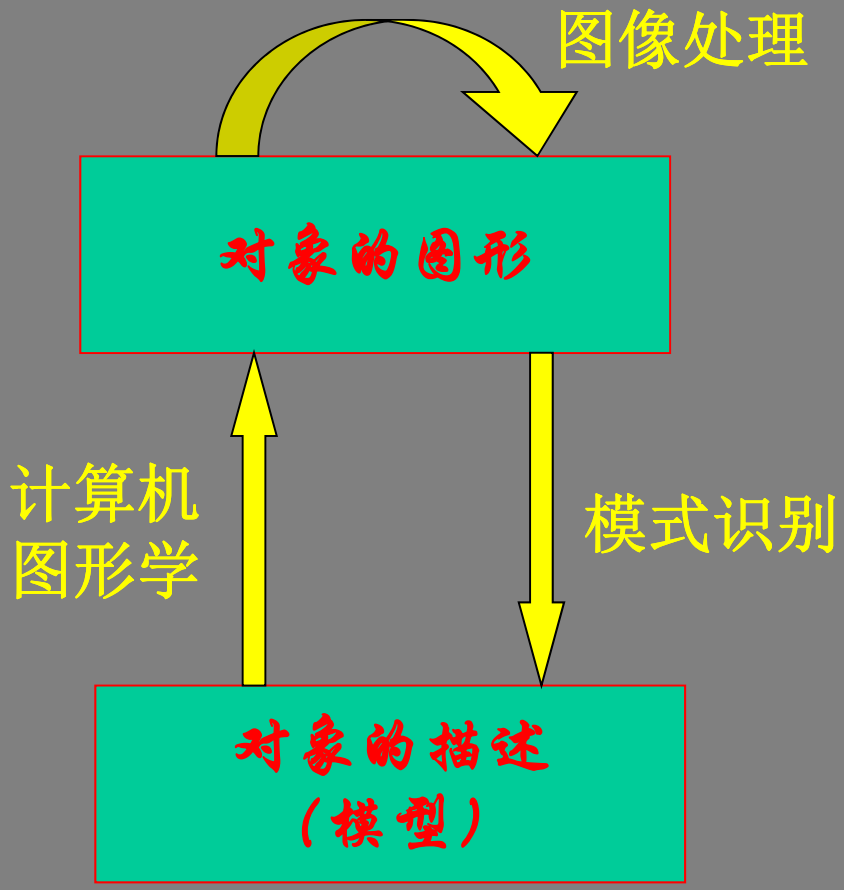
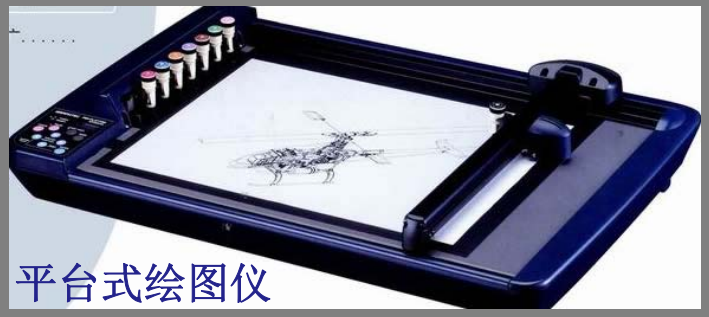
# 计算机图形学、图像处理、模式识别的相互关系



# 1.2 计算机图形学的起源

## 计算机图形学产生前

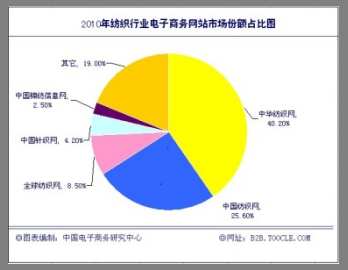
- 1950年，美国麻省理工学院Whirlwind计算机上用CRT显示操作界面和摄像信号
- 1952年，美国Gerber科学仪器公司研制成平台式绘 图仪
- 50年代中期，美国SAGE空中防御系统可用光笔在显示器上选择目标
- 1959年，美国Calcomp公司研制成滚筒式绘图仪


平台式绘图仪

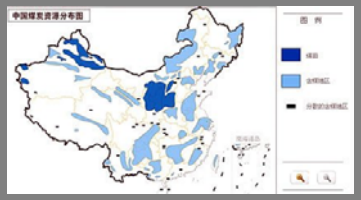

滚筒式绘图仪

# 1.3 计算机图形学的应用及发展动向

## 计算机图形学的应用

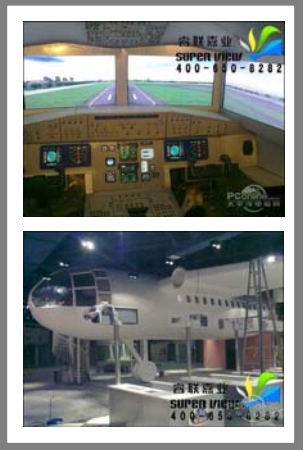



- 事务和商务数据图形显示



- 自然资源图

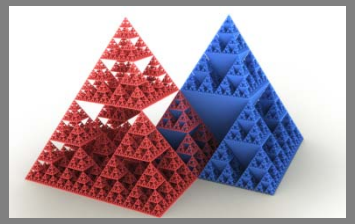
- CAD/CAM



- 计算机仿真与动画

- 生产过程控制

- 办公室自动化

- 计算机艺术

# 计算机图形学的研究 内容

- 图形生成和表示
- 图形操作
- 图形输入（交互技术）
- 图形数据结构
- 几何模型构造
- 动画技术
- 图形标准化

# 1.4 图形系统的硬件

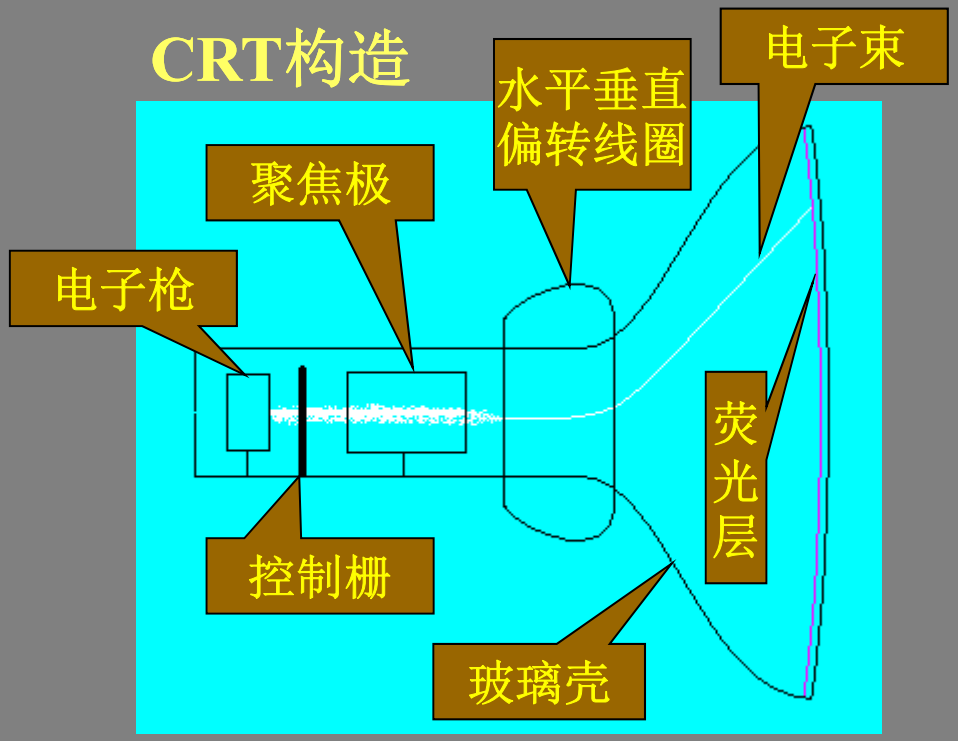
## 计算机图形系统的硬件

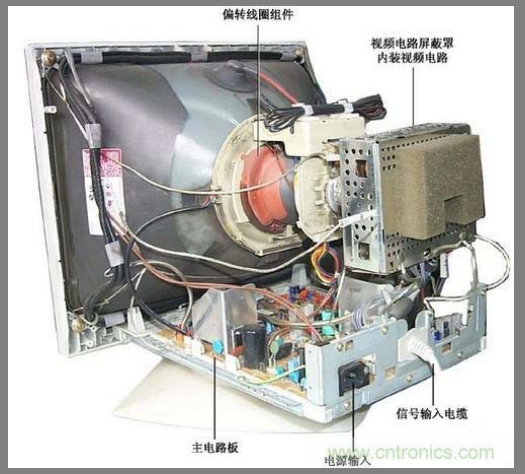
- 计算机
- 显示处理器(DPU)：专用以图形输出控制
- 图形显示器
- 输入设备
- 硬拷贝设备

# CRT显示器(cathode-ray tube阴极射线管显示器)

CRT是一类很重要的图形显示器

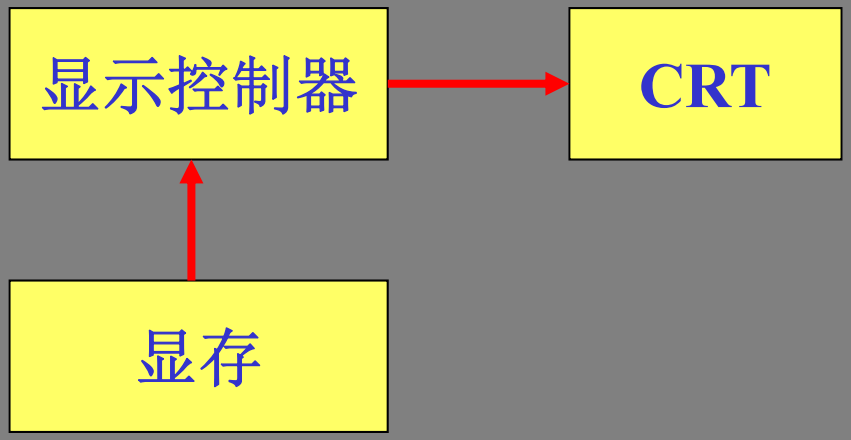
CRT构造



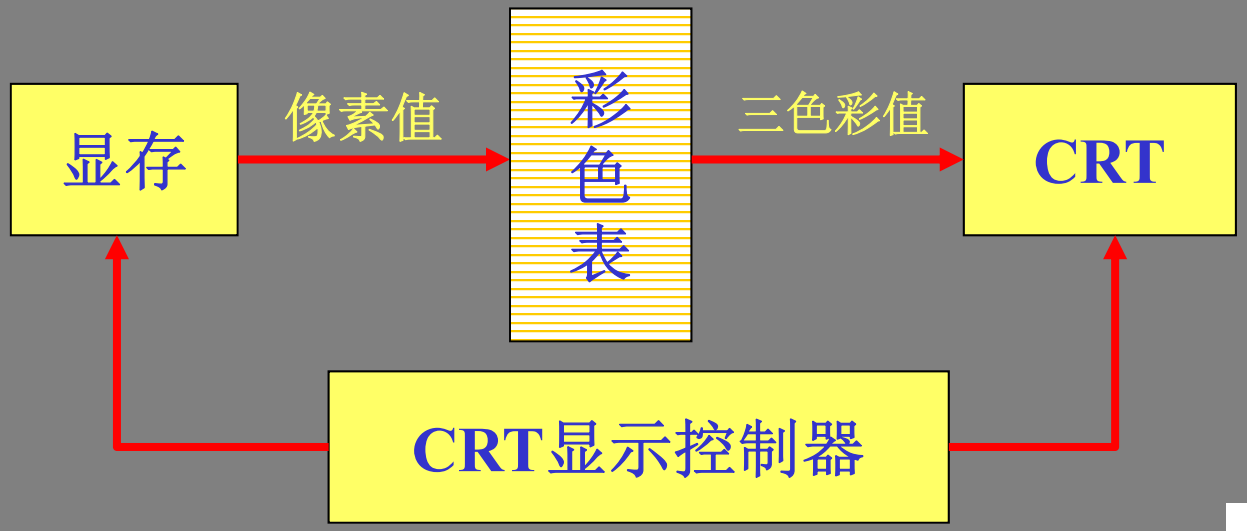



CRT的工作方式分 { 随机扫描方式
光栅扫描方式

随机扫描显示器：由偏转线圈画线

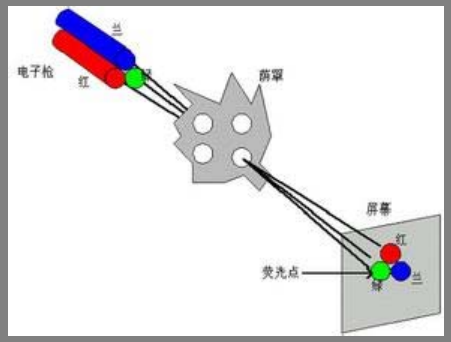


## 光栅扫描显示器：由偏转线圈画光栅，由控制栅画图




彩色显示器颜色由红、绿、蓝三原色按不同比例合成

彩色显示原理




光栅扫描显示器涉及像素、分辨率

- 像素：光栅扫描显示器中，屏幕上可以点亮或熄灭的最小单位称像素
- 分辨率：屏幕上像素的总数称为分辨率。如 1024×768

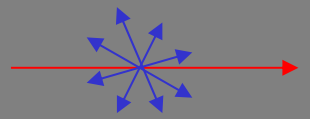
# LCD显示器(liquid crystal display 液晶显示器)

LCD是利用液晶显示的一类图形显示器

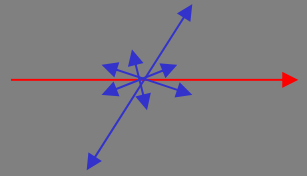
液晶显示原理：

光的偏振现象：光是横波，振动方向垂直于光传播方向。光分自然光和偏振光。

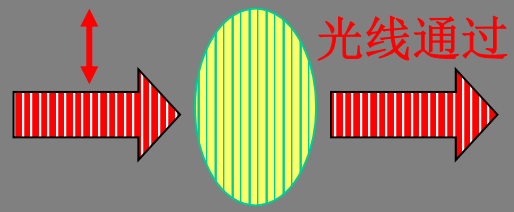
- 自然光：各方向振动强度相等

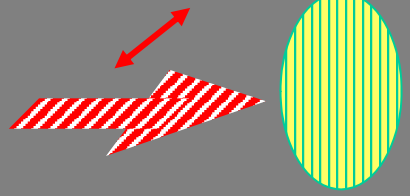


- 偏振光：某方向振动最强

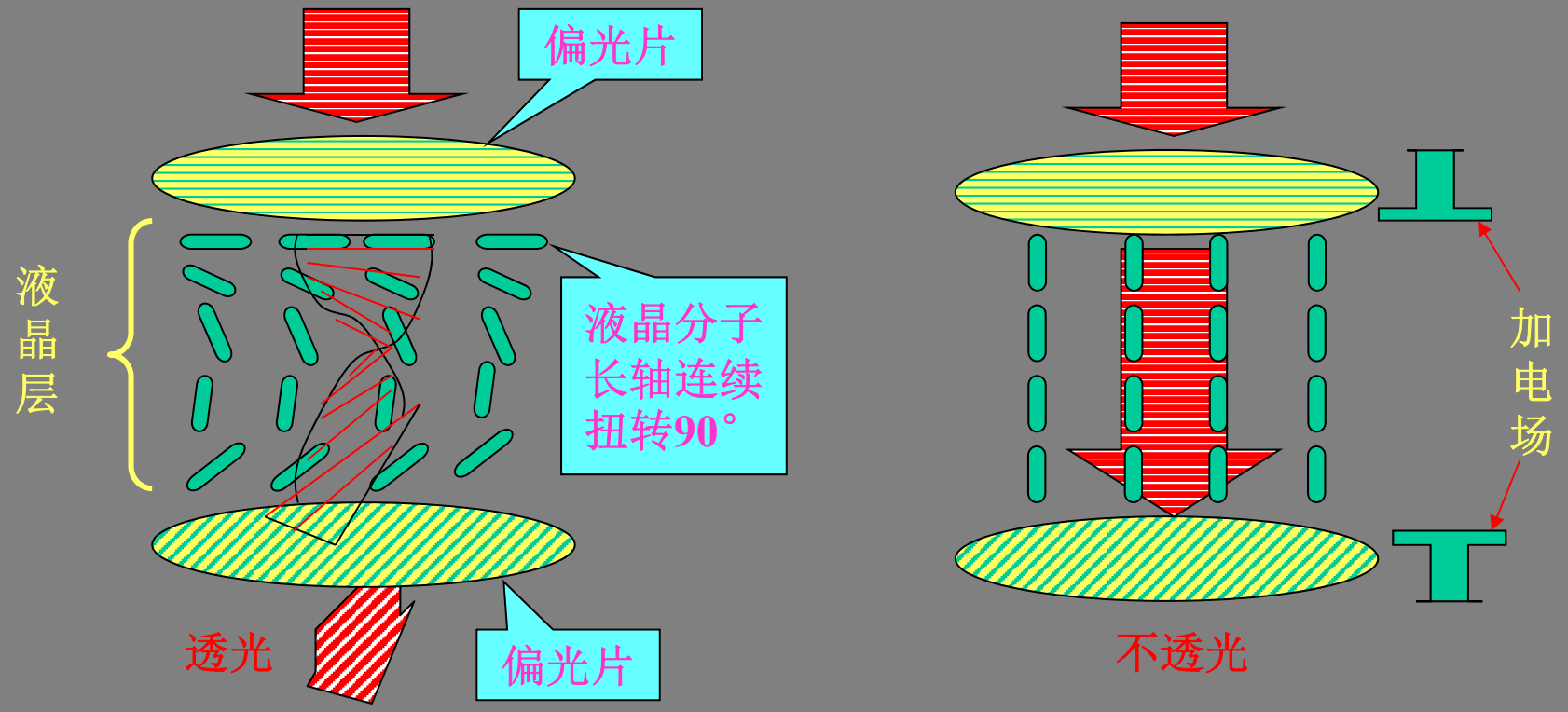


偏振片只让某方向振动的光通过

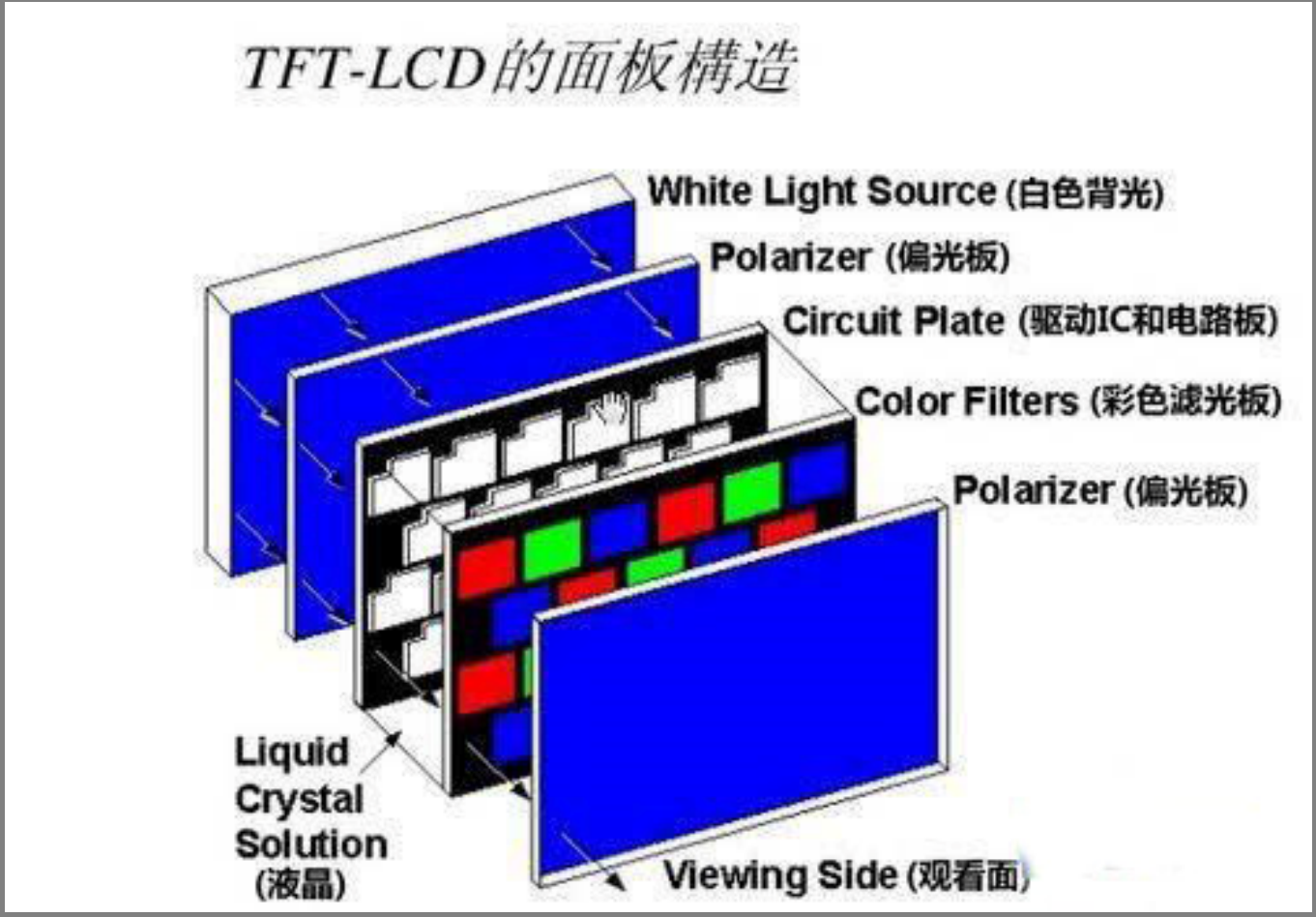


# 显示原理：



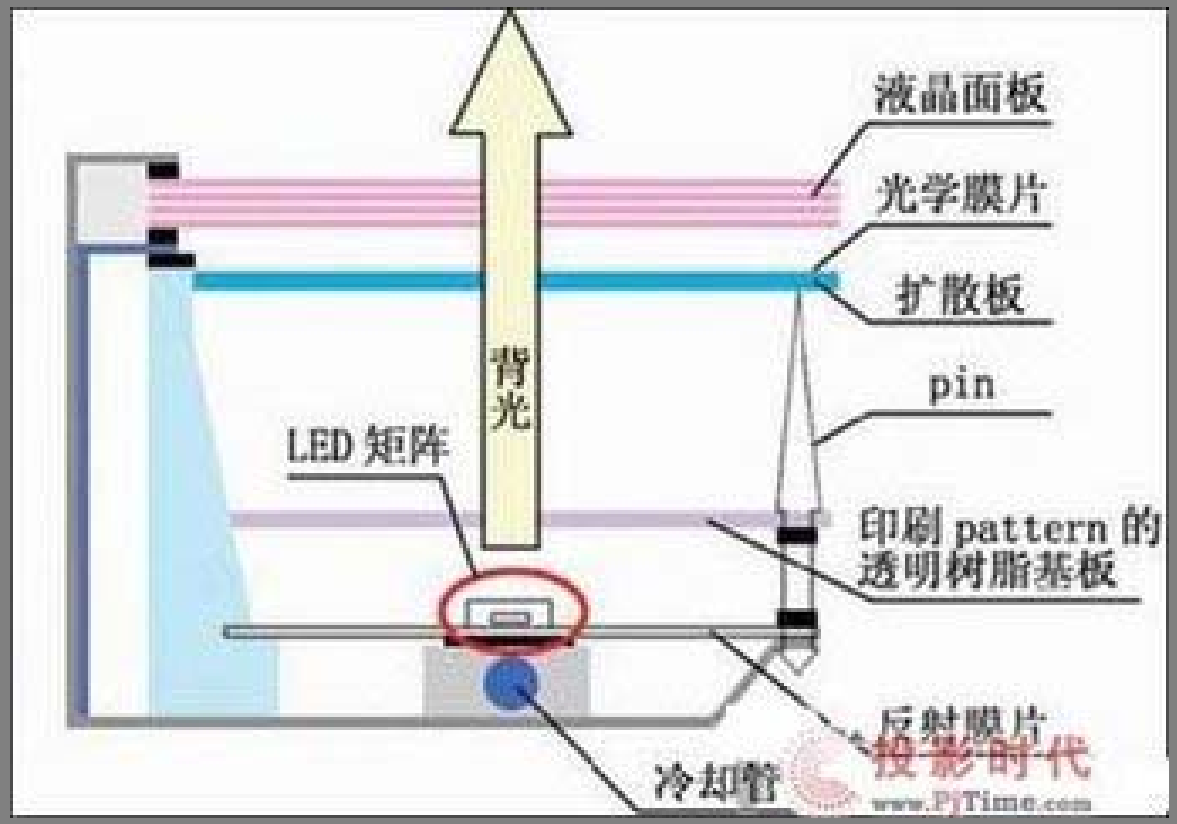
# 彩色液晶显示器原理：




**TFT（Think Film Transistor)薄膜晶体管(控制像素亮度)**

# LED背光的彩色液晶显示器原理

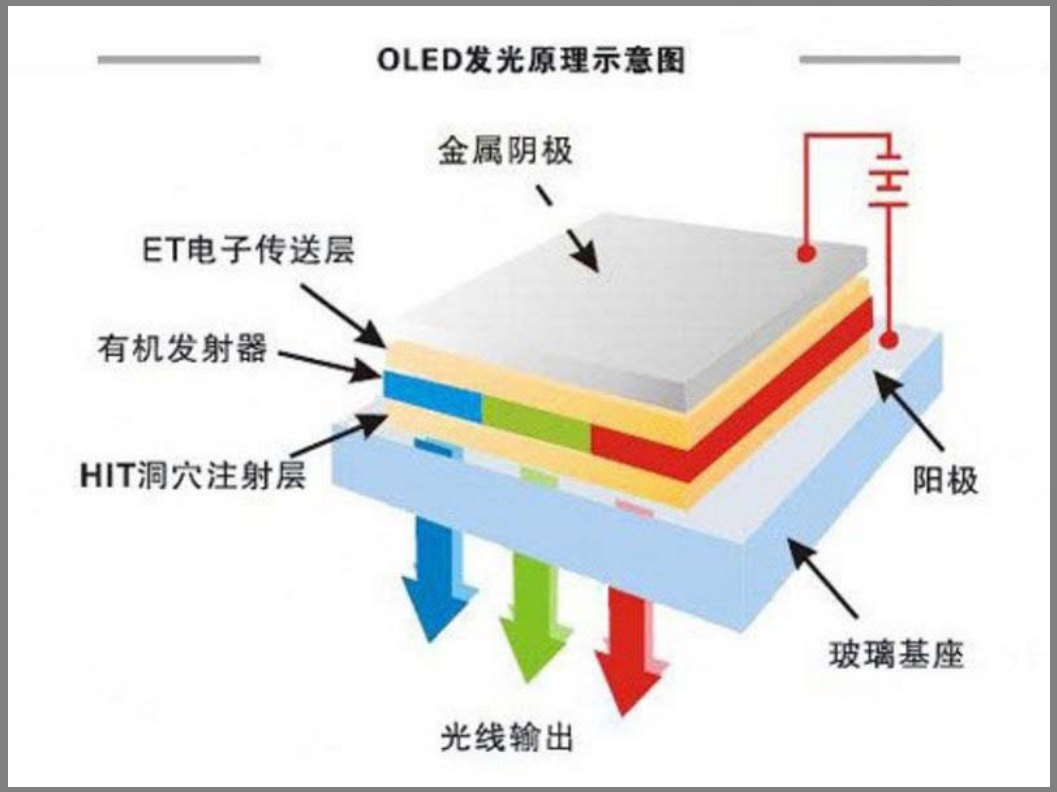
## (LED代替冷阴极灯管CCFL，用于手机屏、部分电脑屏和LED电视)

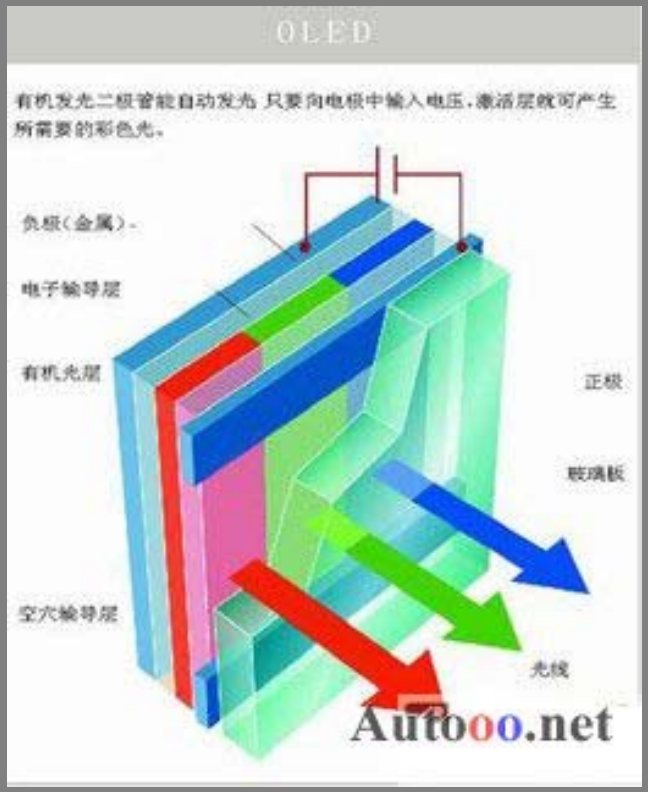


# LED显示器(light-emitting diode 发光二极管显示器)

LED是利用发光二极管显示的一类图形显示器，大量用于室外显示屏
OLED是用有机材料代替金属作为发光材料的发光二极管,用于三星和LG手机

## LED、OLED显示原理：



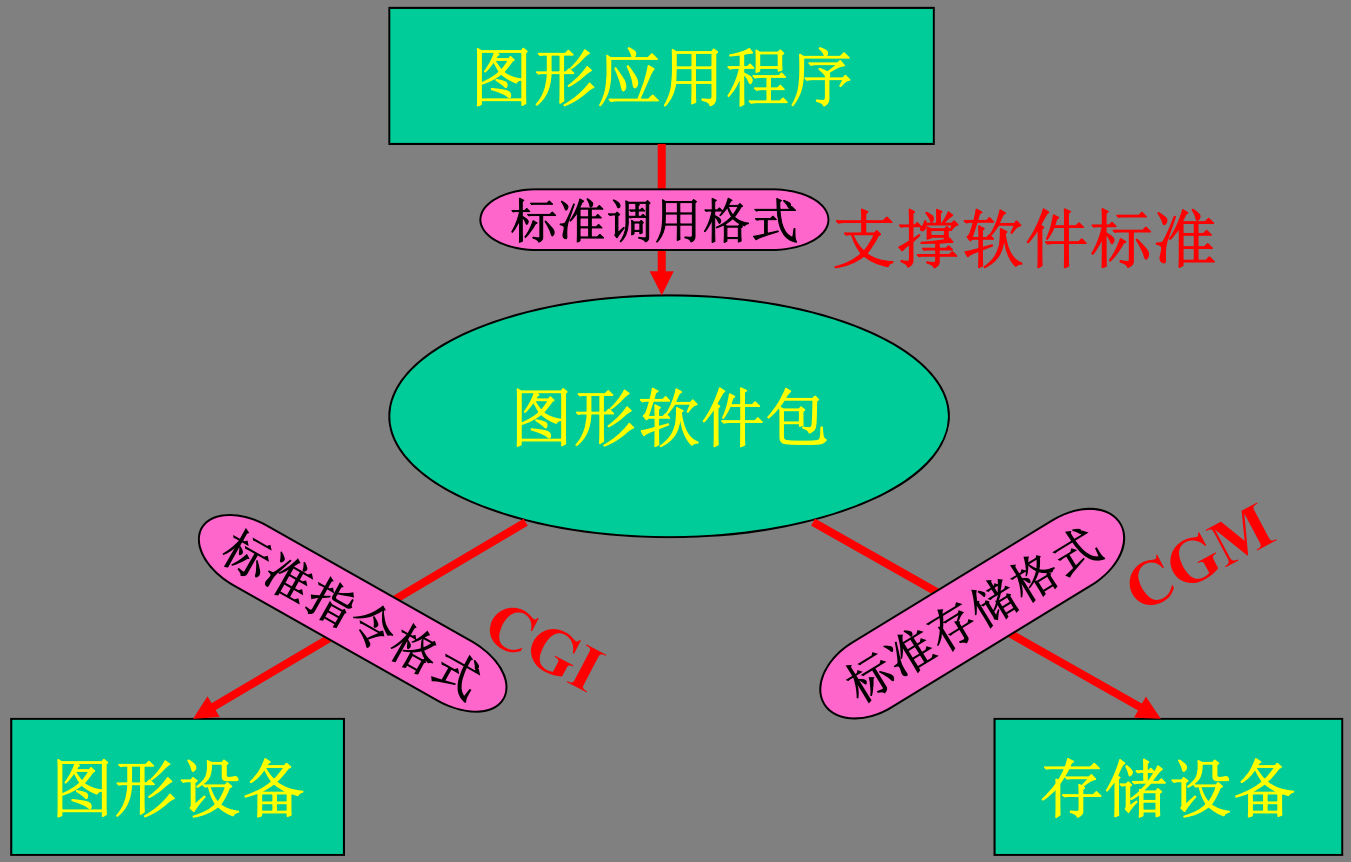
# 1.5 计算机图形标准

计算机产生图形，通常是应用程序调用图形软件包来实现，具有标准调用接口的图形软件包可以使图形应用程序很容易在不同系统上运行。

# 计算机图形标准

- 图形支撑软件标准
  - Core核心图形系统
  - GKS
  - GKS －3D
  - PHIGS，PHIGS+
- 图形设备接口标准 CGI
- 图形数据交换标准 CGM

# 图形标准图示



# 3、 对话框和控件

新建单文档工程：**Dialog**

## (1) 创建模式对话框

设计对话框

- 编辑对话框资源**Dialog** IDD_MODELDLG
- 创建对话框类：右击添加类 CModelDlg

CModelDlg通过成员函数管理对话框

显示对话框

- 菜单消息映射ID_DIALOG(COMMAND) → CDialogView 添加代码

```
CModelDlg dlg;  dlg.DoModal( ) ;
```

- 消息处理代码文件前添加：#include "ModelDlg.h"

使用控件

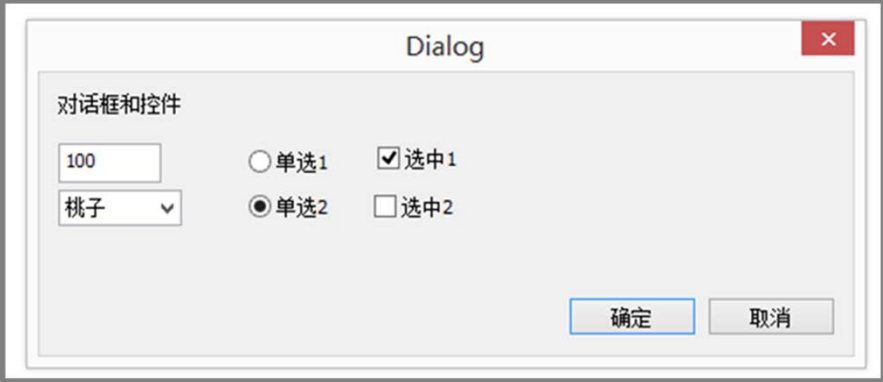
在对话框资源中添加控件：对话框编辑器工具箱

- 静态文本(Static Text)：ID：IDC_STATIC Caption：控件
- 下压按钮(Button)：ID：IDC_BUTTON1 Caption：正

消息映射 IDC_BUTTON1 ( BN_CLICKED ) → CModelDlg添加代码

```
CWnd *pw=GetDlgItem( IDC_BUTTON1);
CString  s ;  pw->GetWindowTextW(s);
if( s= =_T("正"))  pw->SetWindowTextW(_T("负"));
else               pw->SetWindowTextW(_T("正"));
```

# (2) 对话框类成员与控件的数据交流




静态文本(Static Text)
Caption改为： 对话框和控件

在对话框资源中添加：
编辑框(Edit Control) IDC_EDIT1
右击编辑框，添加关联的成员变量：类别：value ， int m_edit
组合框(Combo Control) IDC_COMBO1
（数据：苹果; 梨子; 香蕉；有序(Sort)：False）
右击组合框，添加关联的成员变量：类别：value ，CString m_combo
单选按钮(Radio Button)
IDC_RADIO1 (组Group:True)单选1 , IDC_RADIO2单选2
右击单选按钮，添加关联的成员变量：类别：value ， int m_radio
复选框(Check Box) IDC_CHECK1 选中1，右击添加：value，BOOL m_check1
IDC_CHECK2 选中2，右击添加：value，BOOL m_check2

在菜单消息的处理函数 CDialogView::OnDialog( ) 中改写代码如下：

```
CModelDlg dlg;   CString s;
dlg.m_edit=100;  dlg.m_combo=_T("桃子");
dlg.m_radio=1;   // 0表示选按钮组中第0个单选按钮，其余类推
dlg.m_check1=TRUE;   dlg.m_check2=FALSE;
if( dlg.DoModal( )==IDOK )
{
    s.Format(_T("IDOK\n \n单选%d\n%d\n"),dlg.m_radio+1,dlg.m_edit);
    s=s+dlg.m_combo;
    if(dlg.m_check1)   s=s+ _T("\n选中1");
    if(dlg.m_check2)   s=s+_T("\n选中2");
}
else  s=_T("IDCANCEL\n\n单选2\n100\n桃子\n选中1");
AfxMessageBox( s ) ;
```

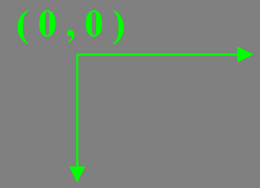
Windows系统消息(WM开头的消息)如鼠标消息的处理函数可以在类的属性窗口删除，其余的成员变量和成员函数，需手工删除：

- 非控件关联、非消息处理成员：在类的定义和构造函数/处理函数定义两处删除
- 控件关联成员变量：在类的定义和构造函数及DoDataExchange函数处删除
- 消息处理成员函数：在类的定义和处理函数定义及消息映射处删除
  (在.cpp文件BEGIN_MESSAGE_MAP和END_MESSAGE_MAP之间)

此外，可以利用右击快捷菜单中的查找所有引用功能

# 4、 图形设备环境

## (1) 绘图和填充函数




视图类的OnDraw( CDC *pDC ) 成员函数用以画图，坐标系为

关于坐标和矩形的结构和类：

```
typedef  long       LONG ;

typedef  struct  tagPOINT
{
      LONG  x , y ;
}   POINT , FAR * LPOINT ;

typedef  struct  tagRECT
{
      LONG  left , top , right , bottom ;
} RECT , FAR * LRECT;
```

色彩类型：

```
typedef unsigend long  COLORREF;
```

RGB( r , g , b ) 即： $r+g*2^8+b*2^{16}$

```
class  CPoint : public  tagPOINT
{
public:
   CPoint( ) ;
   CPoint( int initX , int initY ) ;
                  ... ...
} ;

class  CRect : public  tagRECT
{
public:
   CRect( ) ;
   CRect( int l , int t , int r , int b ) ;
   CRect( POINT topLeft , POINT bottomRight ) ;
                  ... ...
} ;
```

绘图模式设置：绘图模式是指画笔颜色与屏幕上原有颜色的结合方式

绘图模式设定函数： int SetROP2( int nDrawMode ) ;

nDrawMode 常用取值 R2_BLACK(像素取黑色)、 R2_COPYPEN(像素取画笔色)

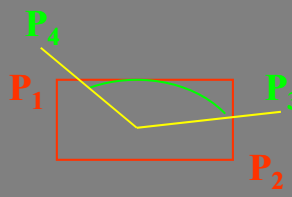
R2_XORPEN(画笔色与背景色异或作为像素色)

绘图和填充函数：

画点 SetPixel( int x , int y , COLORREF crColor ) ;
SetPixel( POINT point , COLORREF crColor ) ;

移光标 MoveTo( int x , int y ) ; MoveTo( POINT point ) ;

画直线 LineTo( int x , int y ) ; LineTo( POINT point ) ;




画圆弧 Arc( int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4) ;
Arc( LPCRECT lpRect , POINT ptStart , POINT ptEnd) ;

填充

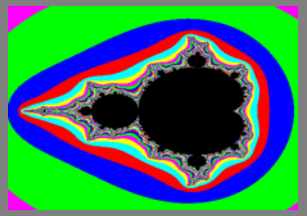
画矩形 Rectangle( int x1 , int y1 , int x2 , int y2 ) ;
Rectangle( LPCRECT lpRect ) ;

画椭圆和圆 Ellipse( int x1 , int y1 , int x2 , int y2 ) ;
Ellipse(LPCRECT lpRect ) ;

画多边形 Polygon( LPPOINT lpPoints , int nCount ) ;

种子填充 FloodFill( int x , int y , COLORREF crColor ) ;

绘制 Mandelbrot 分形集 $M=\{ c \in C \mid z_{n+1}=z_n^2+c , z_0=0 \text{ 序列有界} \}$



新建单文档工程：**Draw1**

在 CDraw1View::OnDraw( CDC *pDC ) 中添加

```
COLORREF  clr[6]={0x0000FF,0xFF0000,0x00FF00,    //红蓝绿
          0xFF00FF,0x00FFFF,0xFFFF00};  //洋红黄青
CRect  rc ;    GetClientRect( &rc ) ;
int    i,j,n,w=rc.Width( ),h=rc.Height( ),wc=(4.5/7)*w,hc=h/2;
double  x,y,t ,cx,cy, d=(3.0/h>3.5/w)?3.0/h:3.5/w;  // hd>=3,wd>=3.5
for(i=0;i<w;i++)  for(j=0;j<h;j++)
{    cx=(i-wc)*d;  cy=(j-hc)*d;
     for(x=y=0,n=100;n>0;n--)
     {     t=x*x-y*y+cx;        y=2*x*y+cy;    x=t;
           if(x*x+y*y>16)  break;
     }
     pDC->SetPixel(i,j,n>0?clr[n%6]:RGB(0,0,0));
}
```

## (2) 图形设备环境

程序通过图形设备接口（**GDI**）进行输出

**GDI提供了：**
- 画笔————线的颜色、线型等
- 画刷————填充图案、颜色等
- 字体————字体风格、大小、方向等
- 设备环境——内含画笔、画刷等的一个画图平台

**MFC封装了这些GDI对象属性及其操作**

设备环境类：

| | |
|---|---|
| **CDC** | 所有设备环境的基类，绘图和坐标为CDC类的操作 |
| **CClientDC** | 客户区设备环境类，输出范围为边框之内的区域 |
| **CPaintDC** | 刷新绘图时用的设备环境 |

**OnDraw(CDC *pDC)中的pDC为CPaintDC类对象的指针。**
在视图类的其它地方我们可用CClientDC dc(this);来申请客户区设备环境

新建单文档工程：**Draw2**

消息映射： CDraw2View(WM_RBUTTONDOWN) 添加代码

```
CClientDC  dc(this);
CRect  rc(point,point+CPoint(100,100) );
dc.Rectangle( &rc ) ;
```

## (3) 画笔画刷

定义画笔：

画笔线型 粗细 画笔颜色

```
CPen  penRed( PS_SOLID ,  1  , RGB(255,0,0) ) ;
```

画笔线型：PS_SOLID ————————

PS_DOT ----------

PS_DASH – – – – – – – – – –

PS_DASHDOT –·–·–·–·–·–·–·–

PS_DASHDOTDOT –··–··–··–··–

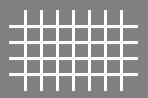
PS_NULL

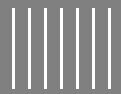
定义画刷：

阴影风格 填充颜色

```
CBrush  brGreen(RGB(0,255,0)) , brGray( HS_CROSS ,  RGB(192,192,192) ) ;
```

阴影风格：HS_BDIAGONAL  HS_ FDIAGONAL 

HS_CROSS  HS_DIAGCROSS 

HS_HORIZONTAL  HS_VERTICAL 

# 6.3 分形

- 欧氏几何描述边界光滑的图形
- 分形几何描述边界极其毛糙的图形。

## 一、分形的概念

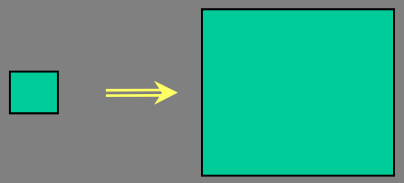
对图形根据所含点的多少进行分类，以二维图形为例说明。

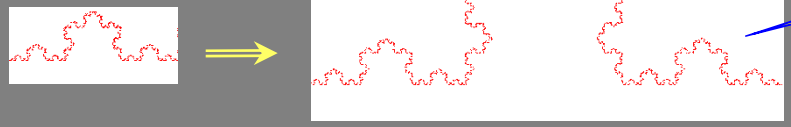
首先，直线和某个有面积的图形如正方形，它们都包含无穷个点，故无法直接比较，我们改用其它方式：

考虑将图形沿x和y轴都放大为原来的a倍，观察图形实际放大多少倍？我们可以这样认为：图形放大倍数大的含点多。

直线段 

实际放大为原来的$3=3^1$倍；

正方形  实际放大为原来的$9=3^2$倍

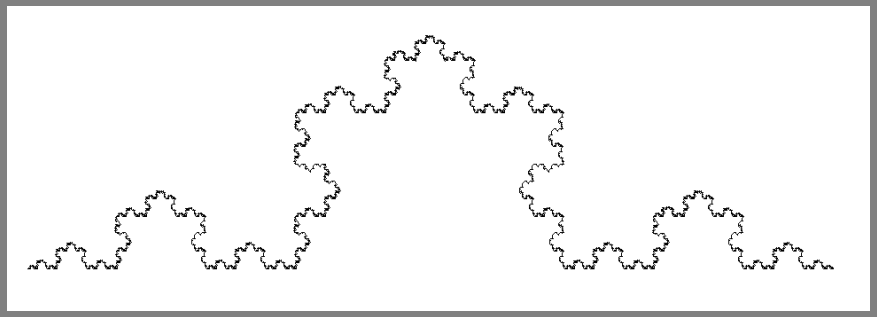
某些曲线如 


实际放大为原来的$4\approx3^{1.26}$倍。

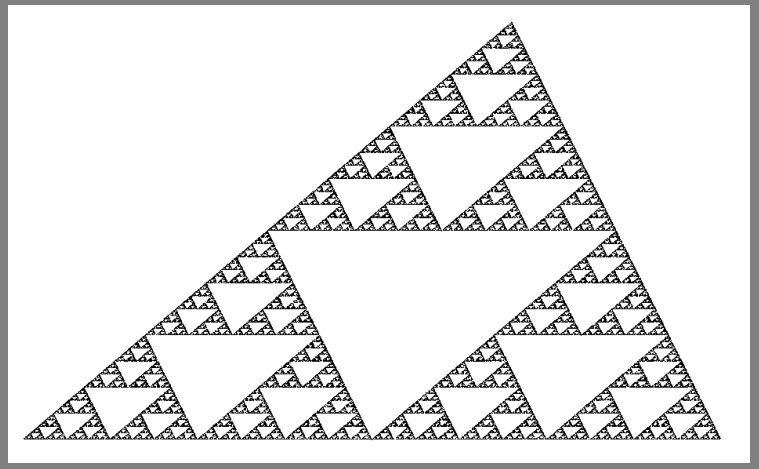
从上可知， 直线段点数<Koch曲线点数<正方形点数

易知直线段为1维，正方形为2维，Koch曲线应为1.26维。若图形在各方向放大a倍后，图形实际放大b倍，则 $D=\log_a b$ 称为图形维数，若D不是整数，则称为分数维，具有分数维的图形称为分形。

# 二、Koch曲线和Sierpinski三角



Koch曲线



Sierpinskip三角

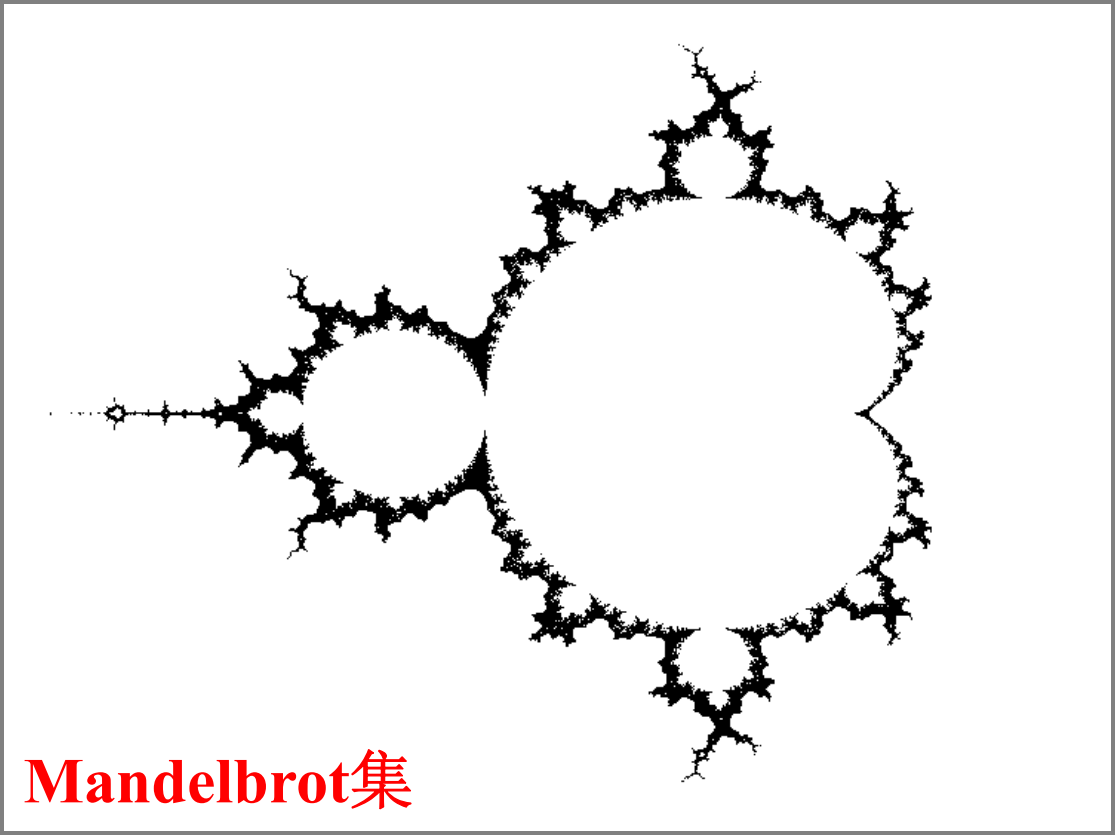
# 三、Julia和Mandelbrot集

考虑复数函数 $f(z)=z^2+c$ , $c,z\in C$ 的迭代性态：

将迭代序列 $z_{n+1}=z_n^2+c$ , $n=0,1,\ldots$ 分成有界和无界两类

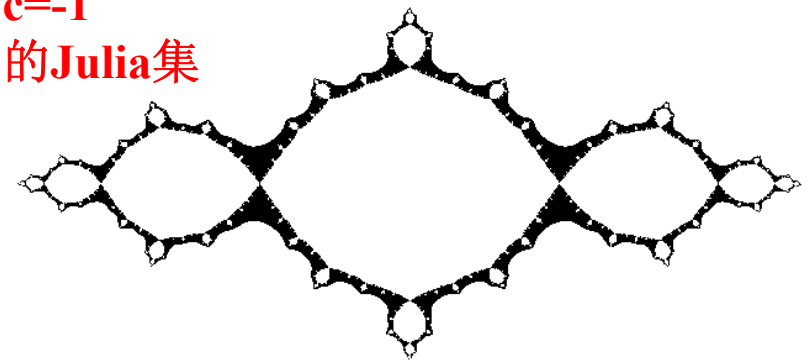
- Julia集=区域 $\{ z_0\in C \mid z_{n+1}=z_n^2+c , c\in C$ 序列有界 $\}$
- Mandelbrot集 $M=\{ c\in C \mid z_{n+1}=z_n^2+c , z_0=0$ 序列有界 $\}$

# Mandelbrot集

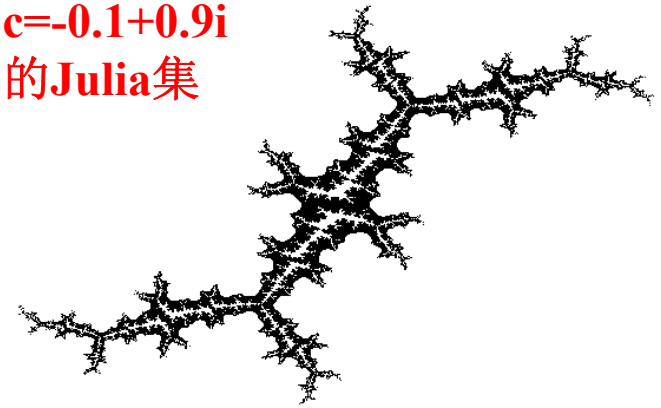


# Julia集

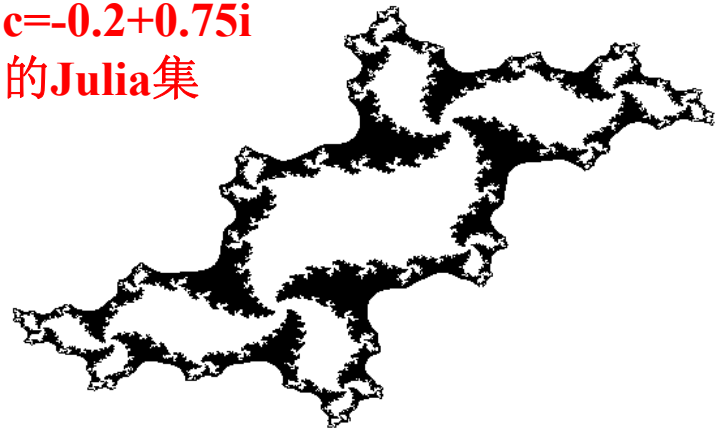
c=-1
的Julia集



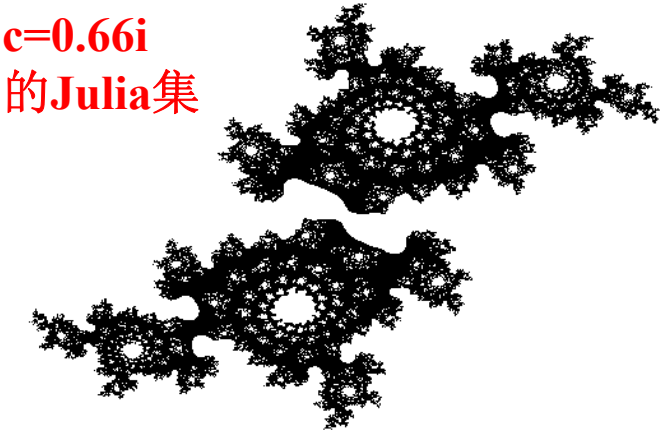
c=-0.1+0.9i
的Julia集



c=-0.2+0.75i
的Julia集

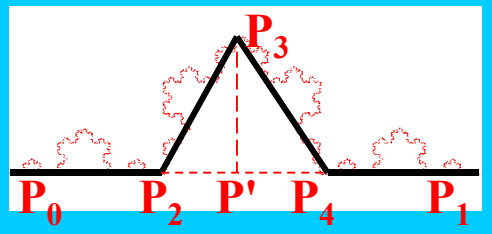


c=0.66i
的Julia集



# 四、分形算法

**画Koch曲线的递归算法：**




Koch($P_0$,$P_1$)

若 | $P_0P_1$ | <ε 则画 $P_0P_1$

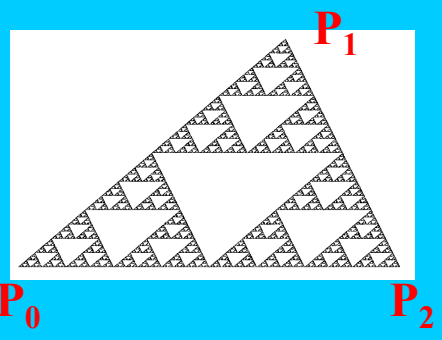
否则执行

Koch($P_0P_2$),Koch($P_2P_3$),Koch($P_3P_4$),Koch($P_4P_1$)

其中：$P_2=\frac{2}{3}P_0+\frac{1}{3}P_1$，$P_4=\frac{1}{3}P_0+\frac{2}{3}P_1$

$$P_3=\frac{P_0+P_1}{2}+\frac{\sqrt{3}}{6}(P_1-P_0)R(90^o)$$

## 画 Sierpinski三角的随机算法：




$P=P_0$

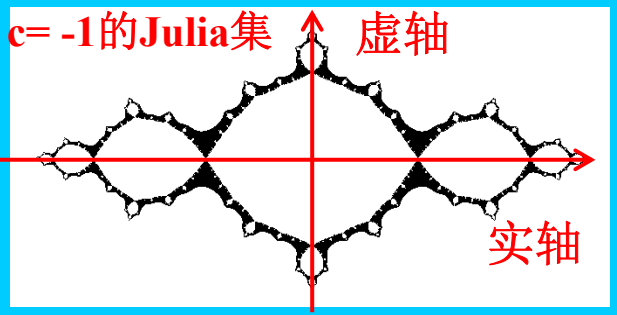
反复执行下列操作

随机取$P_0,P_1,P_2$中的一个点为$P_k$

$P=(P+P_k)/2$ （取中点）

画点P

## 画 Julia集的生成算法：




对矩形区域的每一个点作如下处理

计算点对应的复空间的复数$z_0$

用公式 $z_{n+1}=z_n^2+c$, 迭代20次

若 $|z_n|>5$ 则作为发散点，否则作为收敛点画黑色点

# Ch8 真实感图形的绘制

真实感图形绘制：在屏幕上绘制出有明暗效果的立体图形

## 8.1 漫反射及具体光源的照明

物体表面各点明暗(反射光强)影响因素
- 光源位置、强度
- 视点方向
- 表面方向
- 表面光洁度

### 光线反射分类

表面反射
- 漫反射(向四周反射)
  - 环境光漫反射(周围射向物体的无方向性光的反射)
  - 点光源漫反射(点光源射向物体的有方向性光的反射)
- 镜面反射(向特定方向反射)

# 光照模型

环境光反射模型：$I_1=I_aK_a$

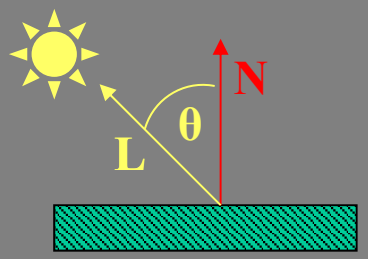
反射光强　入射环境光强　环境反射系数：0~1之间

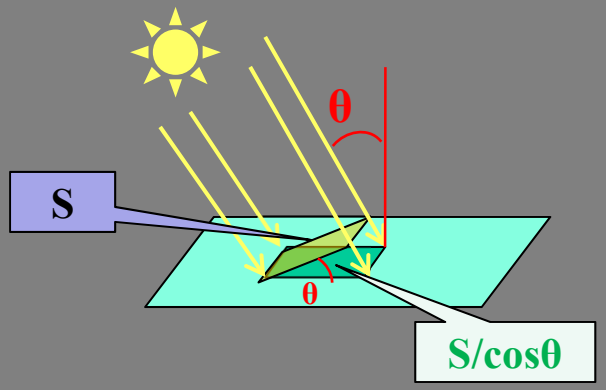
点光源漫反射模型：$I_2=I_PK_d\cos\theta$　　$(0<\theta<\pi/2)$

入射光强　漫反射系数：0~1之间　入射角

点光源光强为 $I_P$ 的漫反射模型 $I_2=\frac{I_P}{r+k}K_d\cos\theta$

光源距离　经验常数



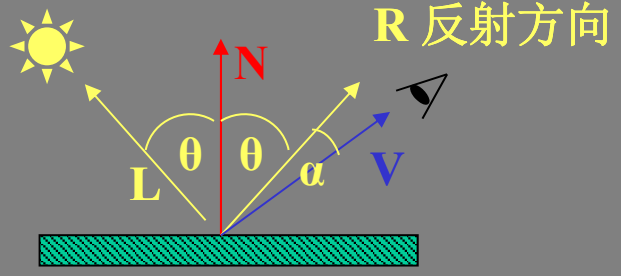



镜面反射模型：
( Phong光照模型 )

$I_3=I_P K_s \cos^n\alpha$

- $I_P$：入射光强
- $K_s$：镜面反射系数：近似代替 Phong Bui-Tuong模型的W(θ)
- $n$：0~200整数：越大越光洁
- $\alpha$：镜面反射光方向与视点方向夹角

点光源光强为 $I_P$的镜面反射模型 $I_3=\frac{I_P}{r+k}K_s\cos^n\alpha$




光照模型公式(局部)：

$$I=I_aK_a+\frac{I_P}{r+k}[K_d\cos\theta+K_s\cos^n\alpha]$$

或 $$I=I_aK_a+\frac{I_P}{r+k}[K_d(L\cdot N)+K_s(R\cdot V)^n]$$ ，L,N,R,V单位向量

RGB颜色光照模型： (红、绿、蓝为三原色)

$$\begin{cases} I=I_{aR}K_{aR}+\frac{I_{PR}}{r+k}[K_{dR}(L\cdot N)+K_s(R\cdot V)^n] \\ I=I_{aG}K_{aG}+\frac{I_{PG}}{r+k}[K_{dG}(L\cdot N)+K_s(R\cdot V)^n] \\ I=I_{aB}K_{aB}+\frac{I_{PB}}{r+k}[K_{dB}(L\cdot N)+K_s(R\cdot V)^n] \end{cases}$$

# 8.2 多边形网格的明暗处理

由多边形网格表示的物体(多面体或曲面物体)上使用光照模型来显示物体表面的明暗与颜色，有三种处理方法：

- 常数明暗法
- Gouraud明暗法(亮度插值明暗法)
- Phong明暗法(法向量插值明暗法)

# 一、常数明暗法

基本思想：网格上各多边形内部用统一的亮度。

该方法使用条件：

1) 光源在无穷远处，故每个平面的L·N是常数

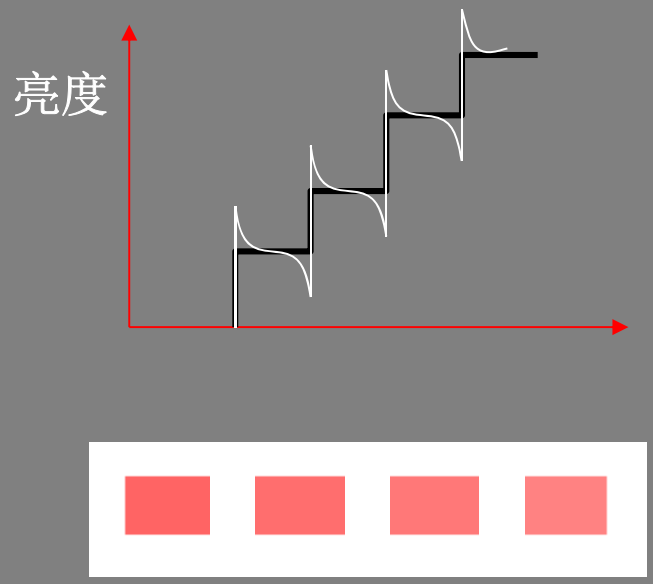
2) 视点在无穷远处，故每个平面的R·V是常数

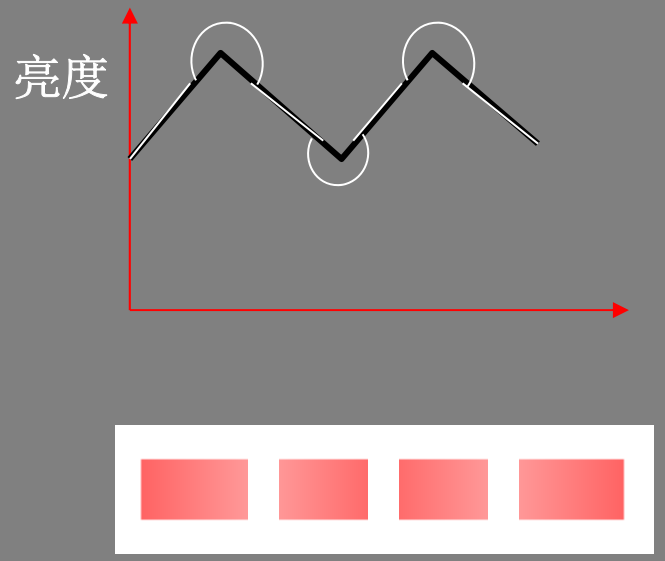
3) 多边形网格是物体表面，不是曲面的近似

- 1) 2) 3) 满足，处理效果很好
- 1) 2) 不满足，可取多边形各顶点的平均L和V，或多边形中点的L和V，也有较好效果
- 3) 不满足，用更小的网格近似曲面，效果可得到改进，但Mach带效应带来的网格痕迹不易消除

**Mach带效应：边界上变化不连续的亮度变化将被夸大。**

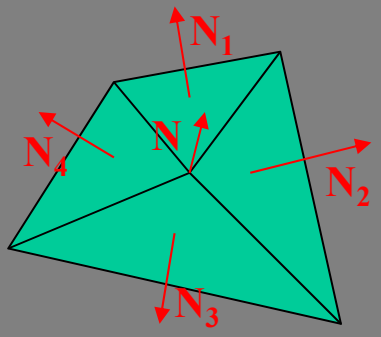
**视觉对不均匀的亮度变化更敏感**



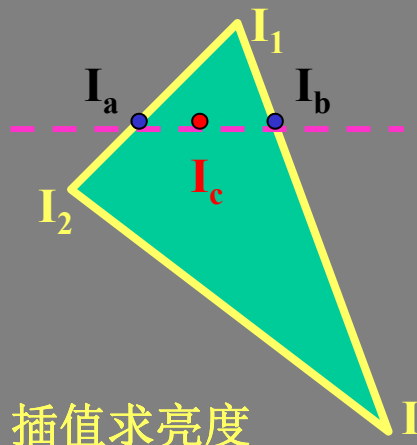
# 二、Gouraud明暗法———设法让亮度均匀变化

基本思想：算出多边形网格顶点处曲面亮度，作为顶点亮度。顶点亮度插值求出边的各点亮度，然后在扫描多边形时，对每条扫描线插值求点的亮度。




求顶点法向量

$$N=\frac{\sum_{i=1}^{4}N_i}{\left|\sum_{i=1}^{4}N_i\right|}$$




插值求亮度

$$I_a=(1-t_1)I_1+t_1I_2$$
$$I_b=(1-t_2)I_1+t_2I_3$$
$$I_c=(1-t_3)I_a+t_3I_b$$

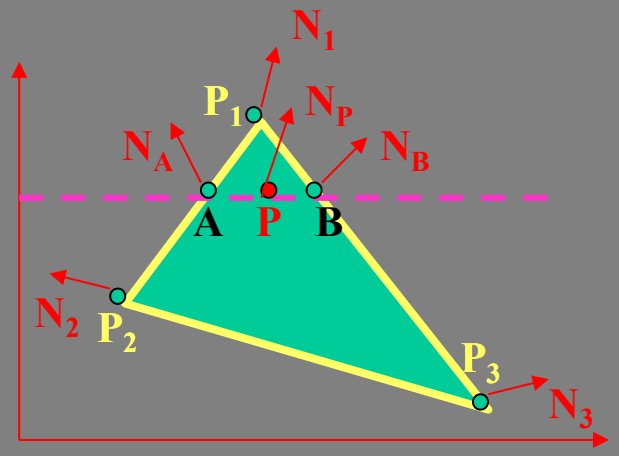
**算法：**

1) 计算各多边形平面的单位法向量
2) 对拥有同一顶点的各多边形平面的法向量取平均值，得顶点法向量
3) 用光照模型求出各顶点的亮度值
4) 对每条边用两顶点的亮度插值，再沿每一条扫描线在各边之间作亮度插值，从而实现多边形的明暗处理

该法对漫反射光照模型效果较好，但对镜面反射效果不好，因镜面反射的亮度变化并不均匀。

# 三、Phong明暗法———设法让方向均匀变化

基本思想：算出多边形网格顶点处曲面法向量，由顶点插值求出边上各点法向量，再由边插值求面上各点法向量，最后由各点的法向量用光照模型求出亮度值。




插值求法向量

$$N_A=((1-t_1)N_1+t_1N_2)/|\ (1-t_1)N_1+t_1N_2\ |$$
$$N_B=((1-t_2)N_1+t_2N_3)/|\ (1-t_2)N_1+t_2N_3\ |$$
$$N_P=((1-t_3)N_A+t_3N_B)/|\ (1-t_3)N_A+t_3N_B\ |$$

**算法：**

1) 计算各多边形平面的单位法向量

2) 对拥有同一顶点的各多边形平面的法向量取平均值，得顶点法向量

3) 对每条边用两顶点的单位法向量插值，再沿每一条扫描线在各边之间作单位法向量的插值，求出扫描线各点的法向量

4) 利用法向量求出亮度值

该法处理后的明暗效果很好，不仅对漫反射有效，对镜面反射也适用。

# 3种明暗算法画网格图

使用z缓冲算法进行消隐和渲染，在Ch7的z缓冲算法画网格图的代码基础上，通过将类point3D、zPoint进行派生，扩展point3D的顶点法向量、zPoint的顶点亮度和zPoint的顶点法向量，并利用多态性实现3种明暗的处理。

光照模型的类

1、光源参数类型 light

含成员数据： Lx , Ly , Lz 光源方向

Iar , Iag , Iab 环境光强度 (红绿蓝三色)

Ipr , Ipg , Ipb 点光源光强(红绿蓝三色)

r , k 点光源距离和距离调整参数

2、网格材料参数类型 material

含成员数据： Kar , Kag , Kab 环境反射系数 (红绿蓝三色)

Kdr , Kdg , Kdb 漫反射系数红绿蓝三色)

Ks 镜面反射系数

类的派生：

1、point3D 派生出 point3Ddir，扩展point3D norm 记录顶点法向量

2、zPoint 派生出 GouraudPoint，扩展 COLORREF intensity 记录颜色

3、zPoint 派生出 PhongPoint，扩展 point3D norm 记录屏幕点的法向量

工程名：DrawGouraudPhong 使用z缓冲算法中的代码

成员函数 CDrawGouraudPhongView::OnDraw (CDC *pDC) 中更改代码

```
point3D *mesh[M+1][N+1];
CRect rc; GetClientRect(&rc);
point3D::setViewport(rc); point3D::setWindow(-12,12,12,-12);
observe3D ob; ob.lookAt(1,-1,2,0,1,0); point3D L(-0.4,0.9,1);   L=ob.trans(L).unitize();
light src0={L.x,L.y,L.z,150,150,150,1500,1500,100,10,0};
material mat0={0.8,0.8,0.8,0.8,0.8,0.8,0.7};  src=src0;  mat=mat0;
point3Ddir meshData[M+1][N+1];
for(int i=0;i<=M;i++) for(int j=0;j<=N;j++) mesh[i][j]=&meshData[i][j];
createVertex(meshData,ff,ob);
zPoint a0[5],*pm[5];  GouraudPoint a1[5];  PhongPoint a2[5]; int i;
if(shade==_T("Flat")) for(i=0;i<5;i++) pm[i]=&a0[i];
else if(shade==_T("Gouraud")) for(i=0;i<5;i++) pm[i]=&a1[i];
else                          for(i=0;i<5;i++) pm[i]=&a2[i];
process(pDC,mesh,pm);
```

光照模型的类及全局变量

```
struct light
{    double Lx,Ly,Lz; /*光源方向*/
     double Iar,Iag,Iab,Ipr,Ipg,Ipb; double r,k;
} src;
struct material { double Kar,Kag,Kab; double Kdr,Kdg,Kdb; double Ks; } mat;
CString shade=_T("Flat");
struct point3Ddir: public point3D      //带法向量的顶点坐标的类
{    point3D norm;
     point3Ddir() { }
     point3Ddir(point3D p):point3D(p) { }
     COLORREF toIntensity()
     {    double LN,RV,smooth=10,t1=src.r+src.k,t2;    int r,g,b;
          LN=norm.x*src.Lx+norm.y*src.Ly+norm.z*src.Lz; if(LN<=0) LN=0;
          RV=2*LN*(-norm.z)-(-src.Lz);  if(RV<0) RV=0;
          smooth=pow(RV,smooth); t2=mat.Ks*smooth;
          r=int(src.Iar*mat.Kar+src.Ipr/t1*(mat.Kdr*LN+t2));
          g=int(src.Iag*mat.Kag+src.Ipg/t1*(mat.Kdg*LN+t2));
          b=int(src.Iab*mat.Kab+src.Ipb/t1*(mat.Kdb*LN+t2));
          if(r>255) r=255; if(g>255) g=255; if(b>255) b=255;     return RGB(r,g,b);
     }
};
```

```
struct GouraudPoint: public zPoint      //带亮度和z的屏幕点坐标
{ COLORREF intensity;   //亮度
  GouraudPoint() { }
  GouraudPoint(CPoint p,double zz,COLORREF ii=0):zPoint(p,zz){ intensity=ii; }
  void combine(point3D *p)   //*p点转换为GouraudPoint点
  { zPoint::combine(p);  intensity=(reinterpret_cast<point3Ddir*>(p))->toIntensity(); }
  void interpolation(zPoint *p1,double t,zPoint *res)   //*this到*p1的t插值，存入*res
  {  GouraudPoint *pres=dynamic_cast<GouraudPoint*>(res);
     zPoint::interpolation(p1,t,res);
     pres->intensity=colorInterpolation(p1,t);
  }
  COLORREF colorInterpolation(zPoint *p1,double t)   //*this亮度到*p1亮度的t插值
  {  COLORREF c1=intensity,c2=(dynamic_cast<GouraudPoint*>(p1))->intensity;
     int r,g,b,r0,g0,b0,r1,g1,b1;
     if(t<0) t=0; else if(t>1) t=1;
     r0=c1&0xFF;  g0=(c1>>8)&0xFF;  b0=(c1>>16)&0xFF;
     r1=c2&0xFF;  g1=(c2>>8)&0xFF;  b1=(c2>>16)&0xFF;
     r=int(r0+t*(r1-r0)+0.5);  g=int(g0+t*(g1-g0)+0.5);  b=int(b0+t*(b1-b0)+0.5);
     return RGB(r,g,b);
  }
};
```

```
struct PhongPoint: public zPoint        //带法向量和z的屏幕点坐标
{ point3D norm;
  PhongPoint() { }
  PhongPoint(CPoint p,double zz):zPoint(p,zz){ }
  void combine(point3D *p)
  { zPoint::combine(p);  this->norm=(reinterpret_cast<point3Ddir*>(p))->norm; }
  void interpolation(zPoint *p1,double t,zPoint *res)
  { PhongPoint *pp1=dynamic_cast<PhongPoint*>(p1);
    PhongPoint *pres=dynamic_cast<PhongPoint*>(res);
    zPoint::interpolation(p1,t,res);
    point3D n1=norm,n2=pp1->norm,nn;
    nn.x=n1.x+t*(n2.x-n1.x); nn.y=n1.y+t*(n2.y-n1.y); nn.z=n1.z+t*(n2.z-n1.z);
    pres->norm=nn.unitize();
  }
  COLORREF colorInterpolation(zPoint *p1,double t)  //*this到*p1的t插值点亮度
  { PhongPoint pt;    point3Ddir pd;
    interpolation(p1,t,&pt);  pd.norm=pt.norm;  return pd.toIntensity();
  }
};
```

生成带顶点法向量的曲面网格

```
void createVertex(point3Ddir meshData[ ][N+1],double(*f)(double,double),observe3D ob)
{ int i,j,i0,j0,i1,j1;  double x,z,x0=-10,z0=-10,dx=20.0/M,dz=20.0/N;  point3D v;
  for(i=0;i<=M;i++) for(j=0;j<=N;j++)
  { x=x0+i*dx; z=z0+j*dz; meshData[i][j]=point3Ddir(ob.trans(point3D(x,f(x,z),z))); }
  for(i=1;i<=M;i++) for(j=1;j<=N;j++)  //计算面片(i,j,i+1,j+1)的单位平面法向量
  { v=(meshData[i-1][j]-meshData[i][j])^(meshData[i][j-1]-meshData[i][j]);
    meshData[i][j].norm=v.unitize();  }
  for(i=0;i<=M;i++) for(j=0;j<=N;j++)  //生成顶点法向量
  { i0=i; j0=j; i1=i+1; j1=j+1;
    if(i0<1) i0=1;  if(j0<1) j0=1;   if(i1>M) i1=M; if(j1>N) j1=N;
    v=meshData[i0][j0].norm+meshData[i0][j1].norm
          +meshData[i1][j0].norm+meshData[i1][j1].norm;
    meshData[i][j].norm=v.unitize();  }
}
```

菜单项消息映射：(捆绑工具栏按钮)

映射： ID_VIEW_FLAT(COMMAND) → CDrawGouraudPhongView添加代码

```
shade=_T("Flat");   InvalidateRect(NULL);
```

映射： ID_DRAW_GOURAUD(COMMAND) → CDrawGouraudPhongView添加代码

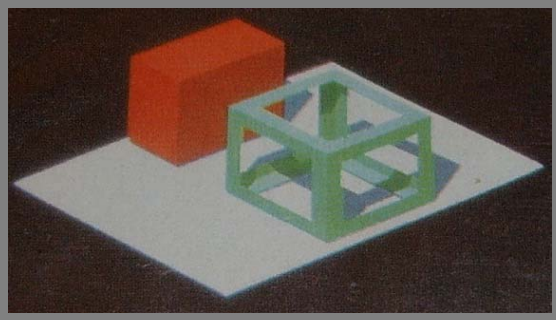
```
shade=_T("Gouraud");  InvalidateRect(NULL);
```

映射： ID_DRAW_PHONG(COMMAND) → CDrawGouraudPhongView添加代码

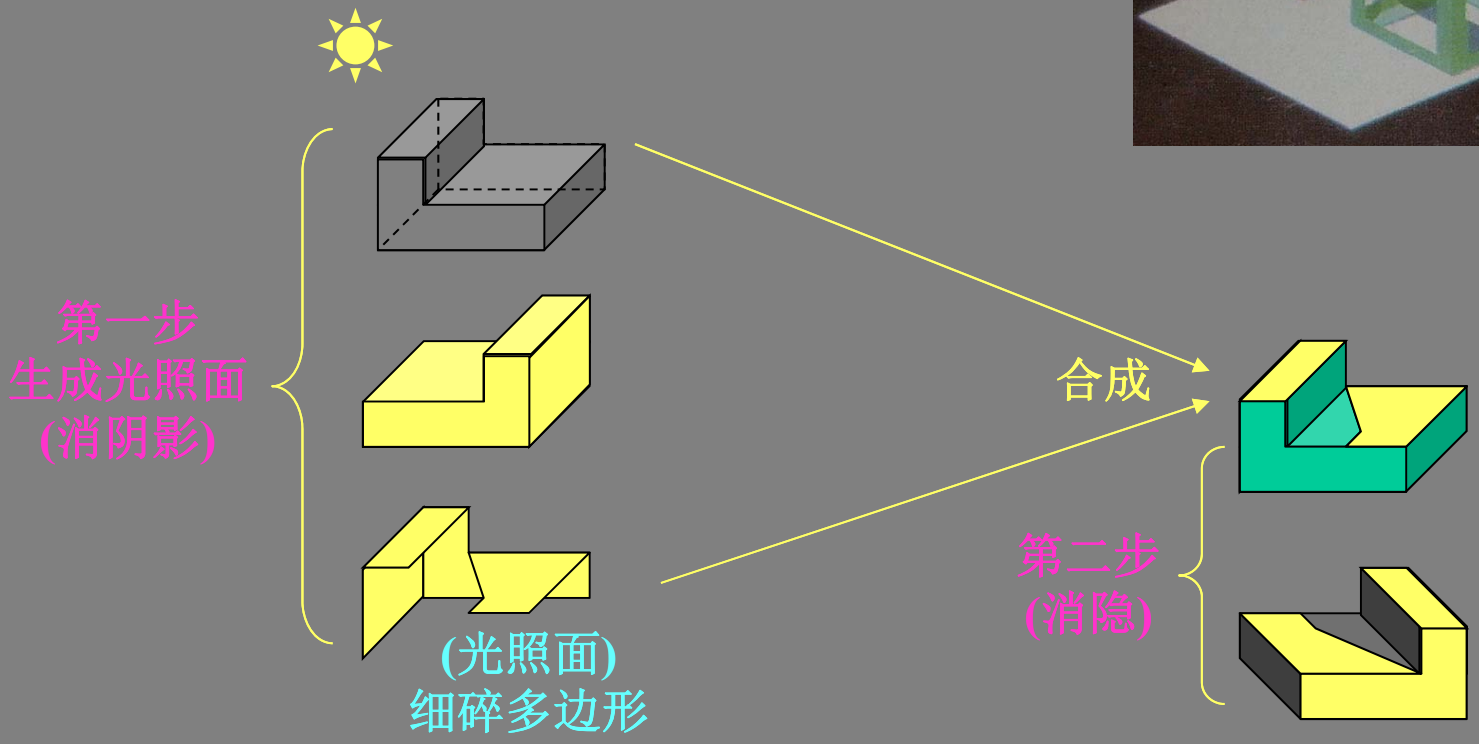
```
shade=_T("Phong";)   InvalidateRect(NULL);
```

# 8.3 阴影

对点光源来说，阴影的生成算法与消除隐藏面的算法相同，当从光源来看物体，看不到的隐藏面就是光源产生的阴影。



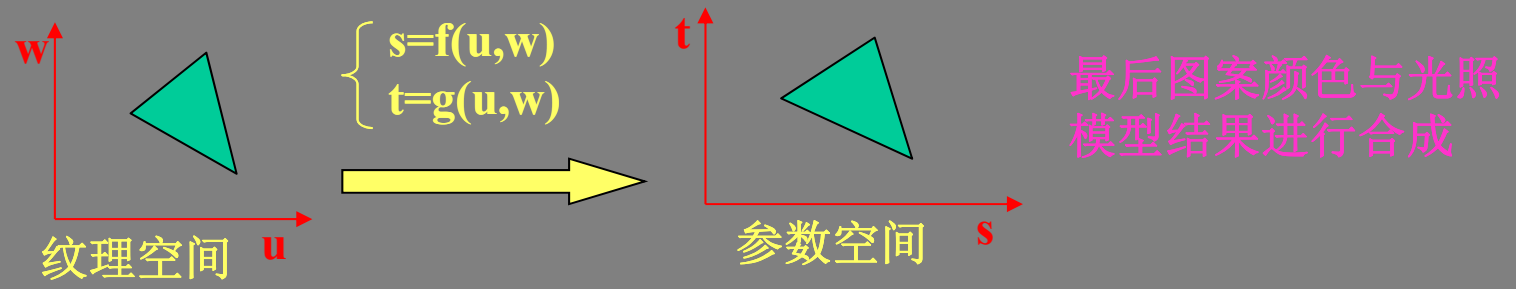
图示：一种两步阴影算法



# 8.4 纹理

纹理 { 光滑表面上的图案
表面的凹凸不平

◆ 图案映射：可用表现凹凸不平表面的图案贴纸进行图案映射反映表面的凹凸感。




◆ 法向量扰动：对原有表面用一干扰函数T(u,v)进行垂直扰动，从而使法向量也产生了扰动，以此可表现各种凹凸感。

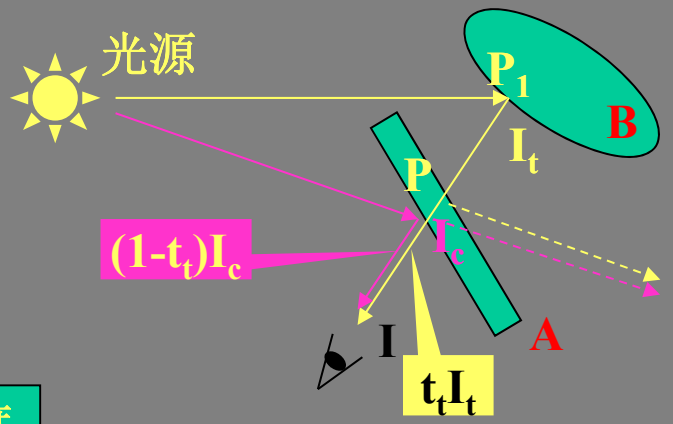
# 8.5 整体光照明模型

8.1 节介绍的是局部光照模型，只考虑光源的直接照射产生的漫反射和镜面反射，环境光认为是各方向光强相同。

若要更加精细地描述环境光的影响，则需要用整体光照模型。整体光照模型要考虑周围环境各种不同的反射光对物体表面的作用及物体的透射光的作用。

# 一、透射光亮度的简单模拟

简单的透明效果模拟可用透明体表面前后光强的插值合成产生透明光强，如右图




透射光模型：

$$I = (1-t_t)I_c + t_tI_t$$

- $(1-t_t)I_c$：P点实际光亮度
- $t_tI_t$：P点吸收后亮度
  - $t_tI_c$透射
  - $(1-t_t)I_c$反射
- 透射率
- $P_1$点亮度

P点非透射亮度$I_c$， $P_1$点亮度$I_t$ ,透射系数$t_t$ $(0 \le t_t \le 1)$

*此法对无折射变形的透明体效果很好

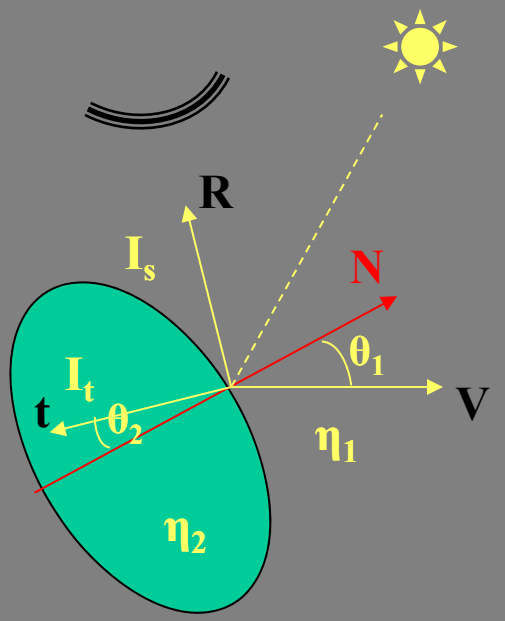
# 二、Whitted光照模型

**Whitted光照模型：**
**(整体光照模型)**

$$I = I_c + k_s I_s + k_t I_t$$

- $k_s$：镜面反射系数
- $k_t$：透射系数
- $I_c$：光源直接照射的反射亮度 (局部光照模型部分)
- $I_s$：镜面反射的环境光亮度
- $I_t$：规则透射的环境光亮度

图示：



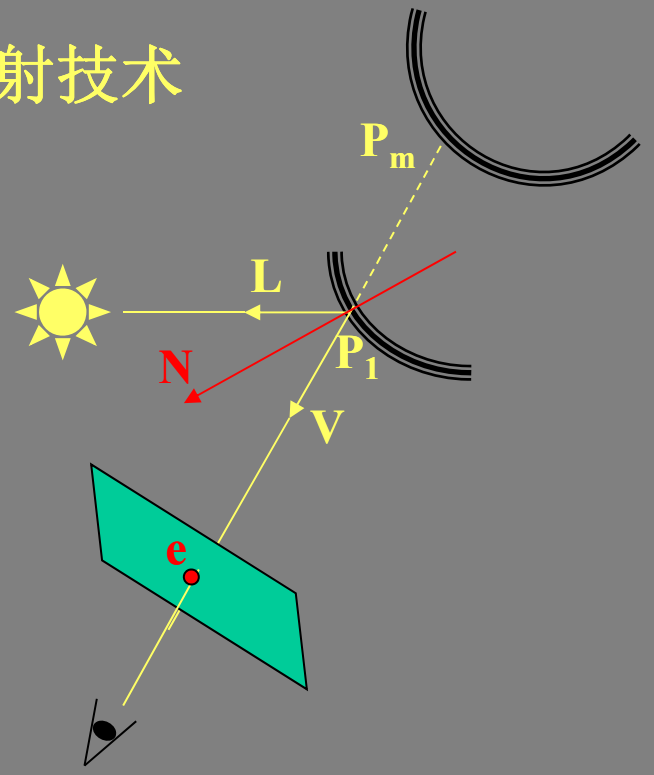

折射率：$\frac{\eta_1}{\eta_2} = \frac{\sin\theta_1}{\sin\theta_2}$

**Whitted光照模型的实现需用光线跟踪技术 ( 即如何求$I_t$ , $I_s$ )**

# 8.6 光线跟踪

光线跟踪技术来源于光线投射技术

光线投射原理：

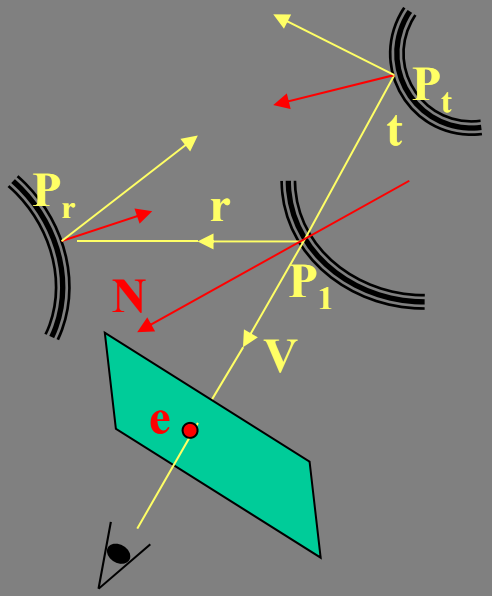



进一步逆向跟踪表面入射光线和透射光线，即得光线跟踪方法。

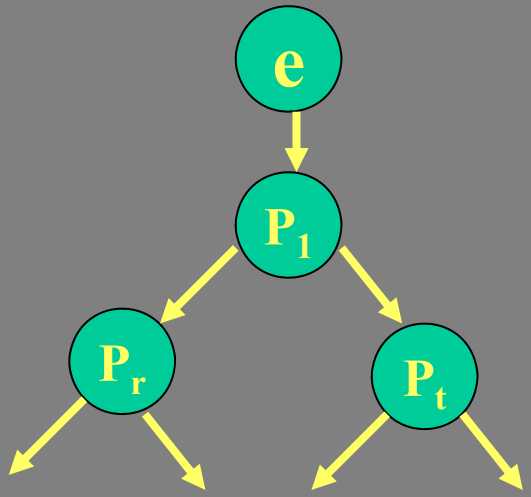
## 光线跟踪技术的基本思想：

逆向跟踪过像素到视点的光线，到第一个表面后，再进一步跟踪镜面反射的入射光线和透射体透射的入射光线，反复跟踪下去直至某个深度或光亮几乎衰耗完。

原理图示：




光线跟踪




光线树

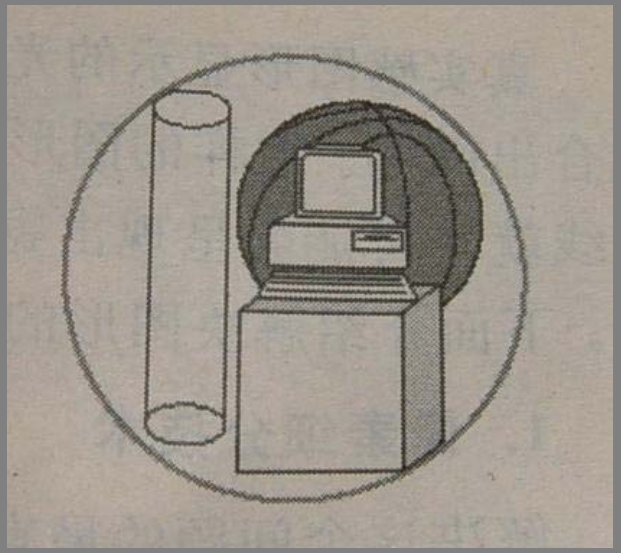
光线跟踪图例

# 8.7 加速光线跟踪算法

光线跟踪算法需大量的运算，主要是求交点的运算(95%)。为了减少求交点运算，可采取场景分层次结合包围盒技术或空间分割技术。

## 场景分层次结合包围盒技术

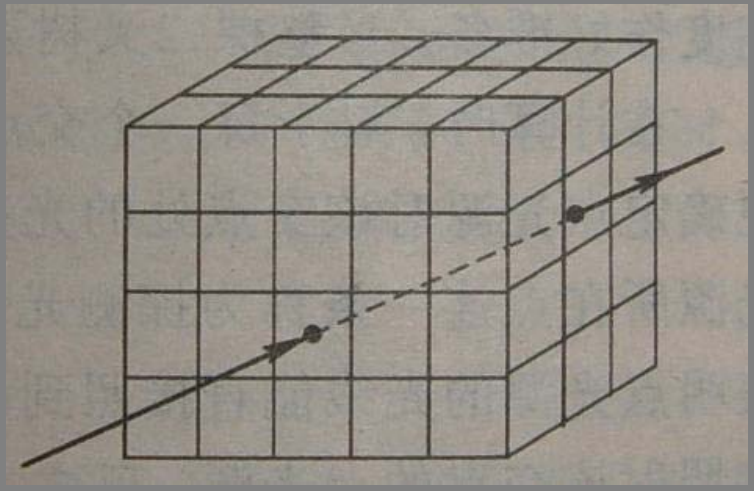
整个场景作为根节点分成几个局部场景作为子节点，子节点再细分成更小局部的节点，再由根到叶分层用包围盒(长方体或球)测试，对测试成功的叶节点内场景求交。



包围盒 (包围球)

# 空间分割技术

将包含场景的空间不断分割，直至每个子域中所含表面足够少。对每条光线我们只要对光线穿过的子域中包含的表面求交，求得交点后即终止该光线的求交运算。



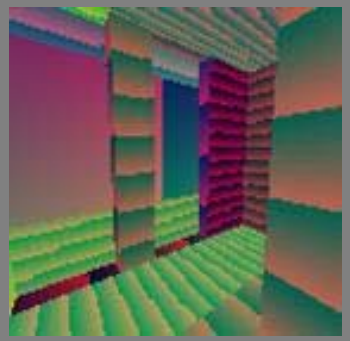
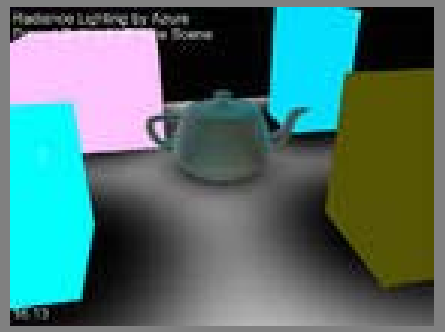
空间分割

# 8.8 幅射度方法

- 镜面反射、透射为主　　光线跟踪效果理想
- 漫反射为主　　辐射度方法效果理想

基本思想：从光能的角度出发，利用封闭环境中能量的守恒，列出能量交换达到平衡需满足的方程，解出方程进而确定各表面各点的光亮度。

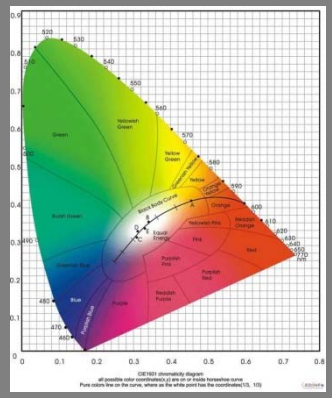


# 8.9 色彩模型

现实世界的颜色，是由不同波长可见光的不同分布产生的，而被人感受到的感觉则是各种色彩(红橙黄绿青蓝紫)、饱和度(颜色深浅)和亮度。

## 一、CIE色度图



CIE色度图是国际照明委员会(CIE)制订的基于三原色原理的国际颜色标准。

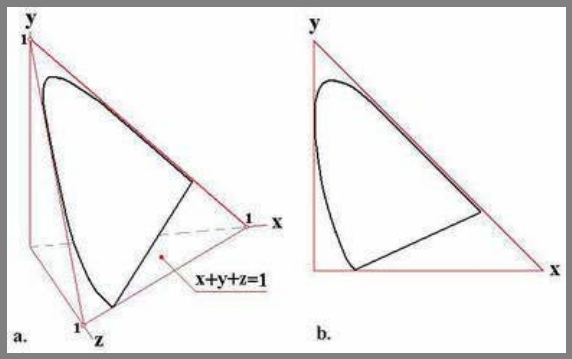
三刺激理论：人类对颜色的感受是由分别感受红、绿、蓝三种颜色光线的三种视觉细胞的感受在大脑中综合形成的感觉。三种颜色光的不同比例混合就可产生与不同波长光线对人的视觉作用同样的感受。

利用三刺激理论，可以用三种颜色作为原色 (基本颜色)来混合调制出各种颜色。

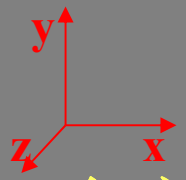
＊三原色选取：很自然首选纯红、纯绿、纯蓝，但这三种颜色无法调制出所有颜色(有个别颜色不能用纯红、纯绿、纯蓝调配)，故实际的CIE色度图三原色选为偏近红、绿、蓝的并不存在的一种抽象颜色，利用它们可以调配出所有的颜色。

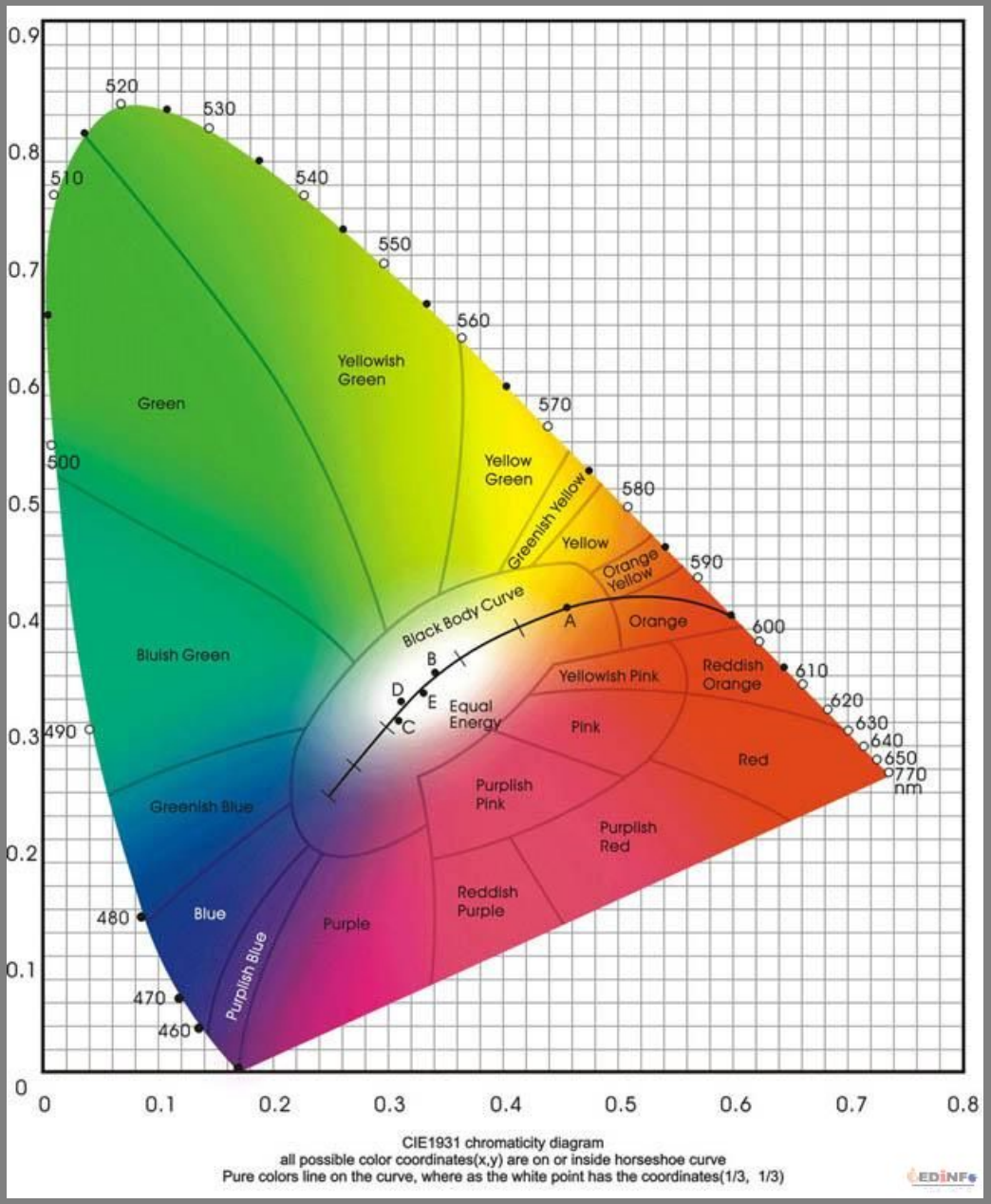
CIE色度图的构成：将三原色按不同比例(X,Y,Z)可调配出各种颜色，将比例标准化为(x,y,z)

其中
$$\begin{cases} x=X/(X+Y+Z) \\ y=Y/(X+Y+Z) \\ z=Z/(X+Y+Z) \end{cases}$$




并将对应颜色在三维坐标系中表示出来，再将颜色面区域正投影到xy平面上，即得CIE色度图。

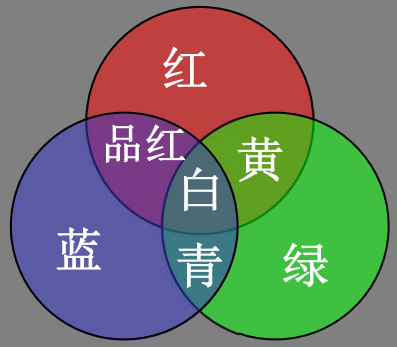


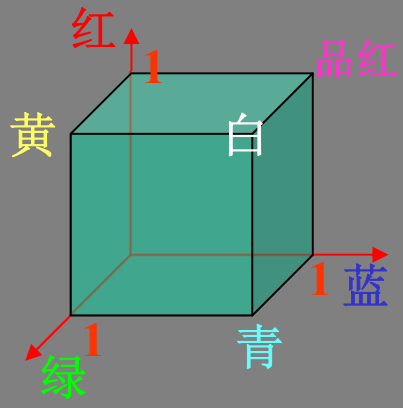



*CIE色度图上马蹄形区域为可见的颜色；c点对应白光；边界对应纯色光；c点到边界直线上的点对应不同深浅的颜色(饱和度)；过c点直线上c点两边分别对应互补的颜色(可混合成白色)

# 二、几种常用的颜色模型

**RGB模型：红、绿、蓝为三原色构成各种颜色，用于发光显示，是加色系统**

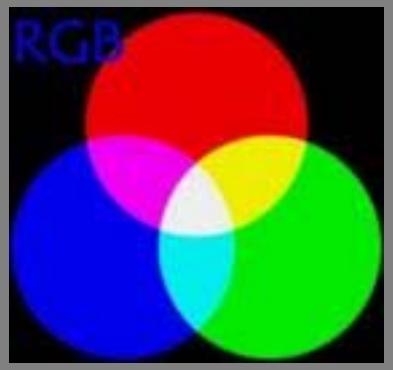


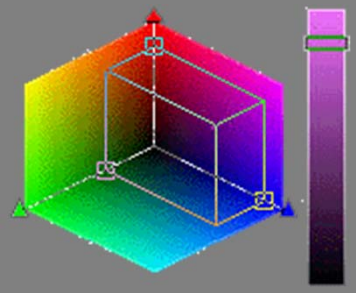




RGB颜色空间

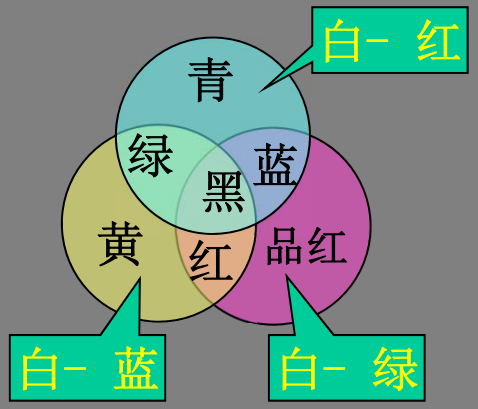
各种颜色用 $\begin{bmatrix} R \\ G \\ B \end{bmatrix}$ 表示

(R,G,B∈[0,1])



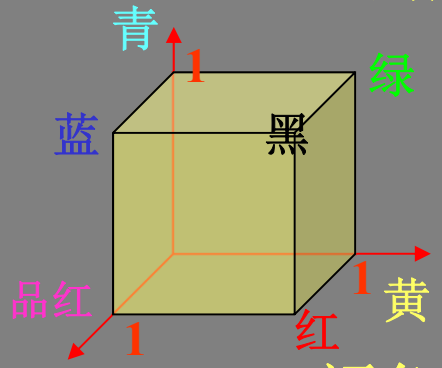



# CMY模型：青、品红、黄为三原色构成各种颜色，用于非发光显示，是减色系统





减色：白纸印上青色相当于印上吸收红色的颜料，故反射光成青色。




CMY颜色空间

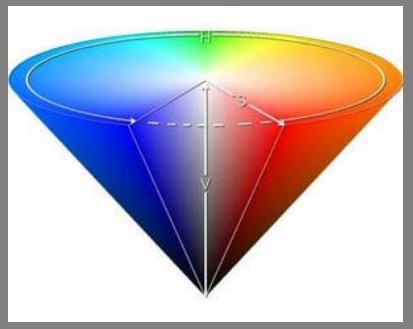
各种颜色用 $\begin{bmatrix} C \\ M \\ Y \end{bmatrix}$ 表示

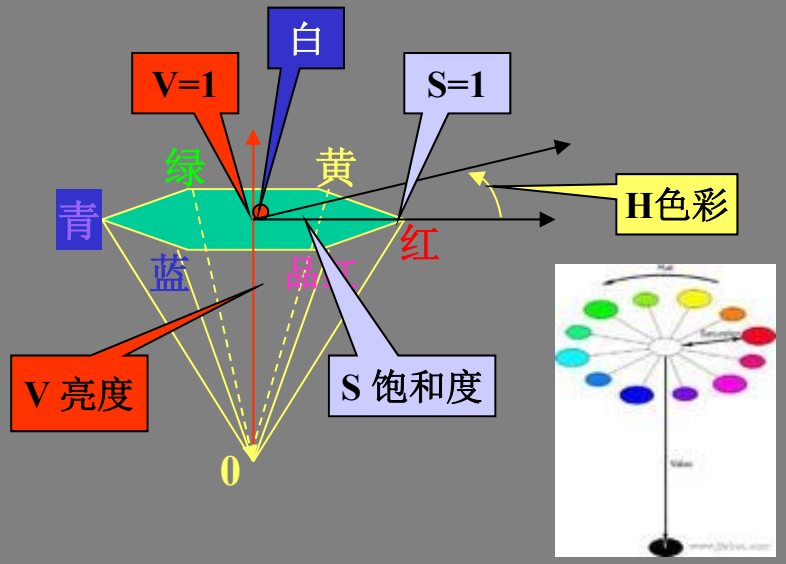
$(C,M,Y \in [0,1])$

同一颜色在RGB颜色空间的值 $\begin{bmatrix} R \\ G \\ B \end{bmatrix}$ 与在CMY颜色空间的值 $\begin{bmatrix} C \\ M \\ Y \end{bmatrix}$ 的关系为：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{或} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

**HSV模型：** 以色彩、饱和度和亮度来构成各种颜色，是面向用户的颜色模型，颜色空间用六棱锥表示。

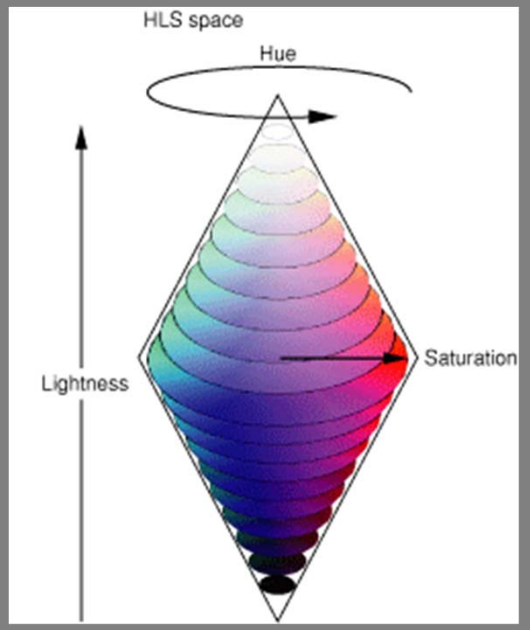


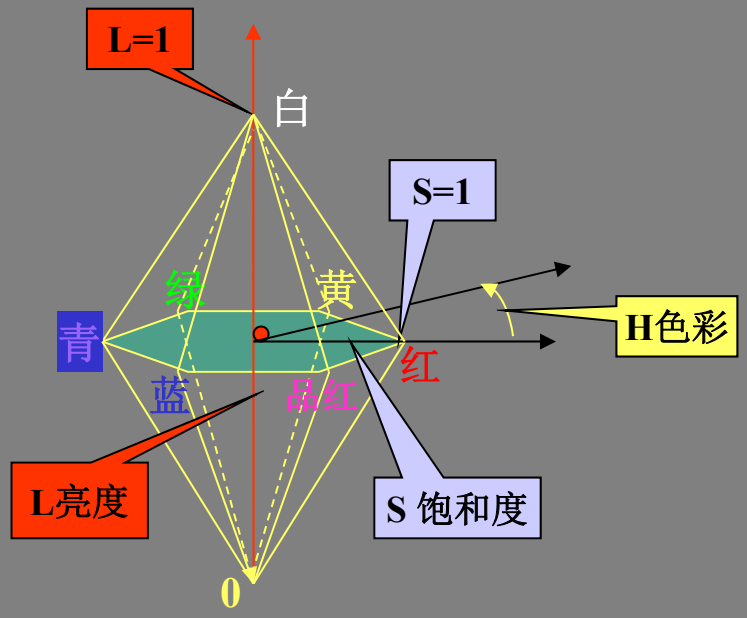



- 相同亮度在同一水平面
- 相同色彩在同一垂直半截面
- 相同饱和度在相似六边形为底的六棱锥侧面
- 过纵轴水平线两边是互补色

$S,V \in [0,1], H \in [0^o, 360^o]$

HLS模型：一种类似HSV模型的实用的色彩模型，用色彩、亮度、饱和度构成各种颜色，颜色空间用双六棱锥表示。




Tektronix公司使用的一种实用颜色模型

$S,L \in [0,1], H \in [0^o, 360^o]$