

# 计算机图形学

# Computer Graphics

计算机科学与技术学院

# 计算机图形学课前简介

## 1. 课程学时

总学时：48学时

## 2. 课程主要内容

二维图形基础

自由曲线和曲面

图形变换

图形裁剪

真实感图形生成技术

### 3. 学习成绩考核方法

平时，期末考试。

### 4. 本课程主要参考书

- [1] 《计算机图形学》 孙家广等编著. 清华大学出版社, 1995
- [2] 《计算机图形学C语言版》 Hearn D等. 影印版.  
清华大学出版社, 1998
- [3] 《计算机图形学教程》 唐荣锡等编著. 科学出版社出  
版, 1992
- [4] 《计算机图形学——原理、方法及应用》 潘云鹤等著高等教  
育出版社
- [5] 《计算机图形技术基础》 刘乃琦等 电子科技大学出版社
- [6] <http://eduinfo.hust.edu.cn/old/kjzz/gjcbss/sjsj/sjsjtxx/index.htm>
- [7] <http://www.ekany.com/wdg98/cg/txx.htm>

# 第一章 绪论

1.1 研究内容

1.2 发展历史

1.3 计算机图形学的应用及研究前沿

# 1.1 研究内容

- ?何谓图形
- ?构成图形的要素
- ?图形的两种表示法
- ?图形学所研究的内容

# 图形以及构成图形的要素

- 图形：计算机图形学的研究对象
  - 能在人的视觉系统中产生视觉印象的客观对象
  - 包括自然景物、拍摄到的图片、用数学方法描述的图形等等
- 构成图形的要素
  - 几何要素：刻画对象的轮廓、形状等
  - 非几何要素：刻画对象的颜色、材质等

# 计算机中表示图形的方法

- 点阵表示
  - 枚举出图形中所有的点(强调图形由点构成)
  - 简称为**图像**（数字图像）
- 参数表示
  - 由图形的形状参数(方程或分析表达式的系数，线段的端点坐标等)+属性参数(颜色、线型等)来表示图形
  - 简称为**图形**:
  - 图形主要分为两类:
  - 基于线条信息表示
  - 明暗图(**Shading**)

# 什么是计算机图形学

- 定义：计算机图形学是研究怎样用数字计算机生成、处理和显示图形的一门学科。

国际标准化组织(ISO)的定义：

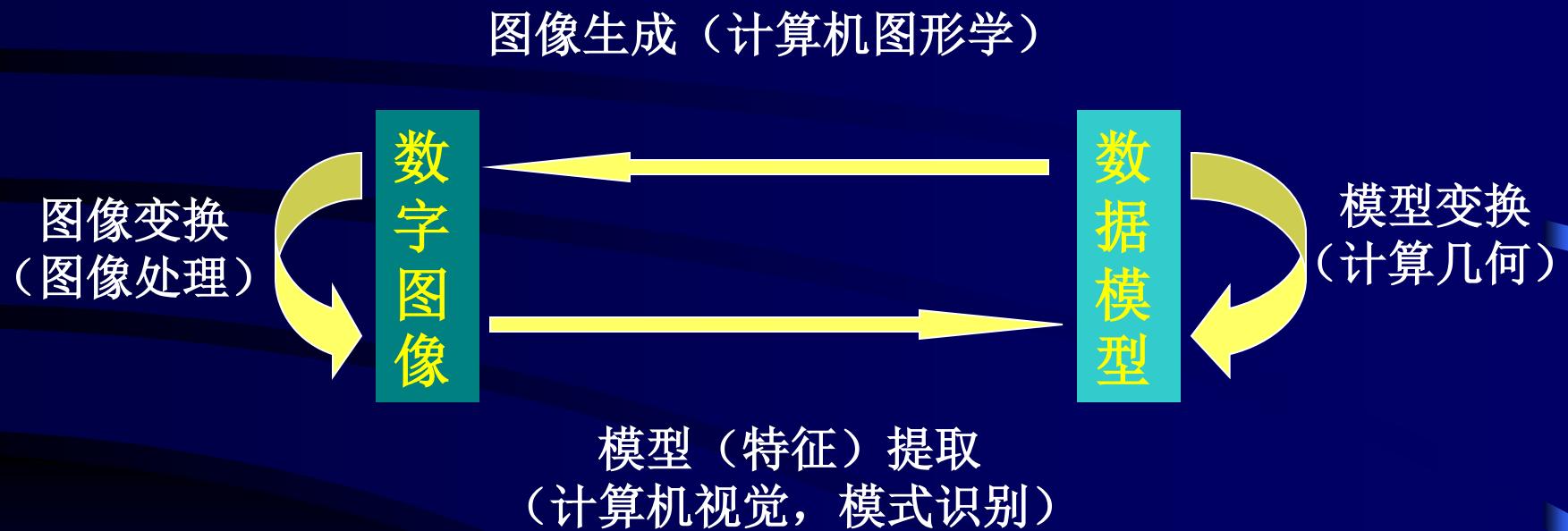
计算机图形学是研究通过计算机将数据转换为图形，并在专门显示设备上显示的原理、方法和技术的学科。

它是建立在传统的图学理论、应用数学和计算机科学基础上的一门边缘学科。

# 计算机图形学的研究内容

- 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与算法，构成了计算机图形学的主要研究内容。
  - 图形硬件、图形标准、图形交互技术、光栅图形生成算法、曲线曲面造型、实体造型、真实感图形计算与显示算法，以及科学计算可视化、计算机动画、自然景物仿真、虚拟现实等。

# 与相关学科的关系



发展特点：交叉、界线模糊、相互渗透

# CAGD (Computer Aided Geometric Design)

- 几何形体在计算机中的表示，分析、研究怎样灵活方便地建立几何形体的数学模型，提高算法效率，在计算机内更好地存储和管理这些模型等。研究曲线、曲面的表示、生成、拼接、数据拟合。

# 图象处理

- 研究如何对一幅连续图像取样、量化以产生数字图像，如何对数字图像做各种变换以方便处理，
- 如何滤去图像中的无用噪声，如何压缩图像数据以便存储和传输，图像边缘提取，特征增强和提取。

# 计算机视觉和模式识别

- 图形学的逆过程，分析和识别输入的图像并从中提取二维或三维的数据模型（特征）。手写体识别、机器视觉。

# 1.2 发展历史

?历史追溯

?硬件发展

图形显示器的发展

图形输入设备的发展

?图形软件及软件标准的发展

# 历史追溯

- 50年代
  - 1950年，第一台图形显示器作为美国麻省理工学院（MIT）旋风I号（Whirlwind I）计算机的附件诞生了
  - 1958年，美国Calcomp公司由联机的数字记录仪发展成滚筒式绘图仪，GerBer公司把数控机床发展成为平板式绘图仪
  - 50年代末期，MIT的林肯实验室在“旋风”计算机上开发SAGE空中防御体系

- 60年代

- 1962年，MIT林肯实验室的I. E. Sutherland发表了一篇题为“Sketchpad：一个人机交互通信的图形系统”的博士论文--**确定了交互图形学作为一个学科分支（提出基本交互技术、图元分层表示概念及数据结构...）。**
- 1962年，雷诺汽车公司的工程师Pierre Bézier 提出 Bézier 曲线、曲面的理论
- 1964年MIT的教授Steven A. Coons提出了超限插值的新思想，通过插值四条任意的边界曲线来构造曲面。

- 70年代
  - 光栅图形学迅速发展
    - 区域填充、裁剪、消隐等基本图形概念、及其相应算法纷纷诞生
  - 图形软件标准化
    - 1974年，ACM SIGGRAPH的与“与机器无关的图形技术”的工作会议
    - ACM成立图形标准化委员会，制定“核心图形系统”（Core Graphics System）
    - ISO发布CGI、CGM、GKS、PHIGS

- 真实感图形学
  - 1970年，Bouknight提出了第一个光反射模型
  - 1971年Gouraud提出“漫反射模型+插值”的思想，被称为Gouraud明暗处理
  - 1975年，Phong提出了著名的简单光照模型- Phong模型
- 实体造型技术
  - 英国剑桥大学CAD小组的Build系统
  - 美国罗彻斯特大学的PADL-1系统

- 80年代

- 1980年Whitted提出了一个光透視模型-Whitted模型，并第一次给出光线跟踪算法的范例，实现Whitted模型
- 1984年，美国Cornell大学和日本广岛大学的学者分别将热辐射工程中的辐射度方法引入到计算机图形学中
- 图形硬件和各个分支均在这个时期飞速发展

- 90年代：微机和软件系统的普及使得图形学的应用领域日益广泛。
  - 标准化、集成化、智能化
  - 多媒体技术、人工智能、科学计算可视化、虚拟现实
  - 三维造型技术

- ACM SIGGRAPH会议小知识
  - 全称 “the Special Interest Group on Computer Graphics and Interactive Techniques”
  - 60年代中期，由Brown 大学的教授Andries van Dam (Andy) 和IBM公司的Sam Matsa发起
  - 1974年，在Colorado大学召开了第一届 SIGGRAPH 年会，并取得了巨大的成功
  - 每年只录取大约50篇论文

# 硬件发展

- 图形显示器的发展  
    图形显示器是计算机图形学中关键的设备
- 60年代中期：画线显示器（亦称矢量显示器）  
    需要刷新。设备昂贵，限制普及
- 60年代后期：存储管式显示器  
    不需刷新，价格较低，缺点是不具有动态修改图形功能，不适合交互式。

# 硬件发展

- 70年代初，刷新式光栅扫描显示器出现，大大地推动了交互式图形技术的发展。
- 以点阵形式表示图形，使用专用的缓冲区存放点阵，由视频控制器负责刷新扫描。

# 图形显示设备的发展：

画线显示器（矢量显示器/随机扫描显示器）



存储管式显示器

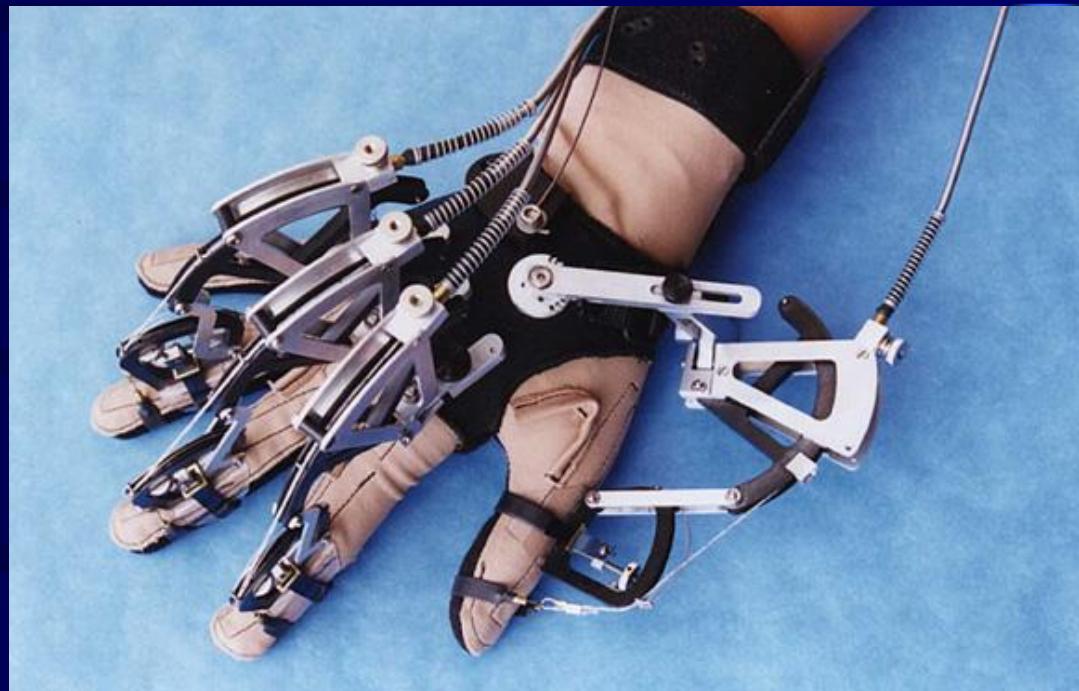


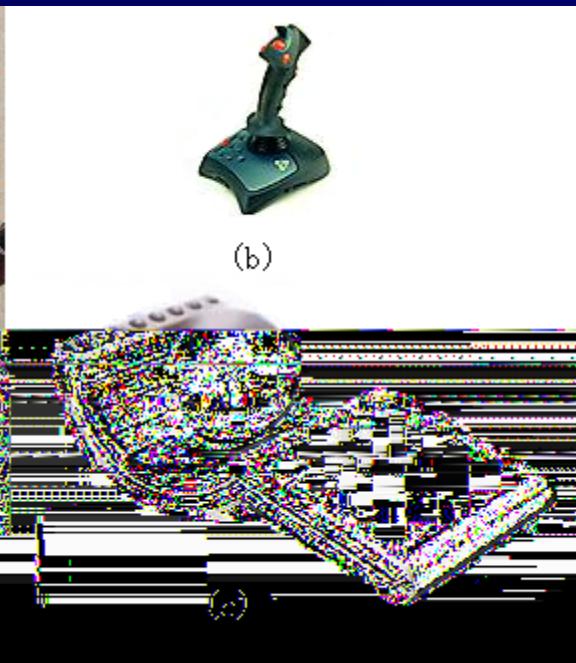
刷新式光栅扫描显示器

# 硬件发展

## 图形输入设备的发展

- 第一阶段：控制开关、穿孔纸等等
- 第二阶段：键盘
- 第三阶段：二维定位设备，如鼠标、光笔、图形输入板、触摸屏等等，语音
- 第四阶段：三维输入设备（如空间球、数据手套、数据衣），用户的手势、表情等等
- 第五阶段：用户的思维





# 图形软件发展及软件标准形成

三种类型的计算机图形软件系统：

(1)用某种语言写成的子程序包

如： GKS (Graphics Kernel System)

PHIGS(Programmer's Hierarchical Interactive Graphics system )

GL

便于移植和推广、但执行速度相对较慢，效率低

(2) 扩充计算机语言，使其具有图形生成和处理的功能

如： Turbo Pascal、 Turbo C， AutoLisp等。

简练、紧凑、执行速度快，但可移植性差

(3) 专用图形系统：效率高，但系统开发量大，可移植性差。

# 图形软件发展及软件标准的形成

## 发展历程



通用的、与设备无关的图形包，图形标准

GKS (Graphics Kernel System) (第一个官方标准，1977)

PHIGS(Programmer's Herarchical Iuteractive Graphics system)

一些非官方图形软件，广泛应用于工业界，成为事实上的标准

DirectX (MS)

Xlib(X-Window系统)

OpenGL(SGI)

Adobe公司Postscript

开放式、高效率的发展趋势

# 1.3 计算机图形学的应用及研究前沿

## 图形用户界面

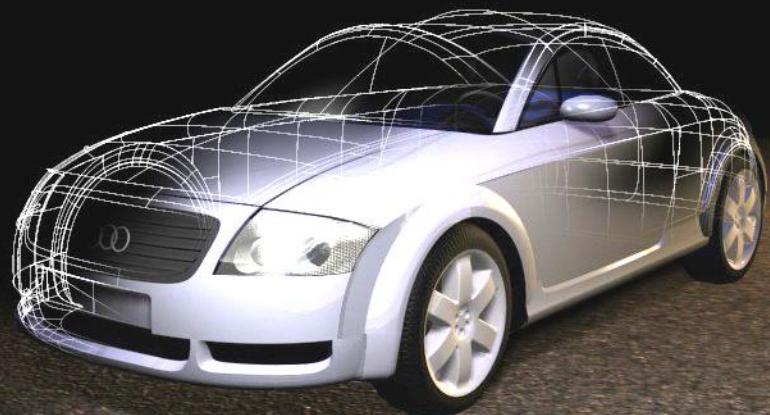
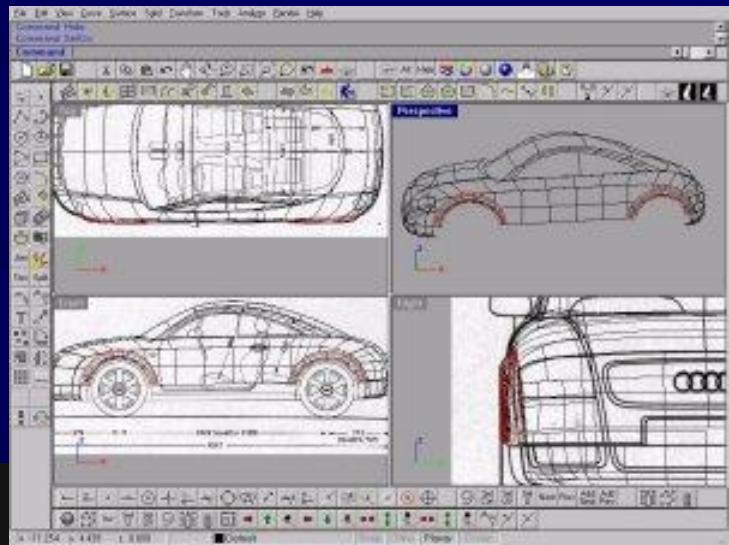
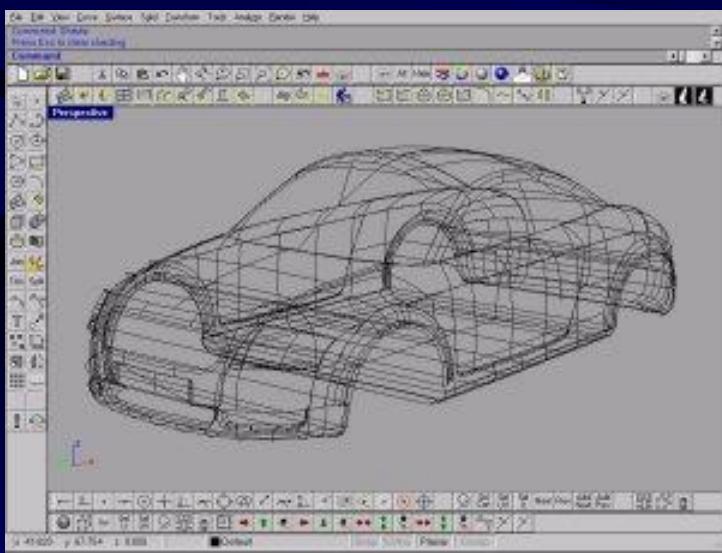
- 介于人与计算机之间，人与机器的通信，人机界面（HCI）：软件+硬件
- 发展：由指示灯和机械开关组成的操纵界面→由终端和键盘组成的字符界面（80年代）→由多种输入设备和光栅图形显示设备构成的图形用户界面（GUI），（90年代）PC，工作站，WIMP(W-windows、I-icons、M-menu、P-pointing devices)界面，所见即所得→VR技术（发展方向）

# 计算机辅助设计与制造 (CAD/CAM)

-CAD/CAM是计算机图形学在工业界最广泛、  
最活跃的应用领域

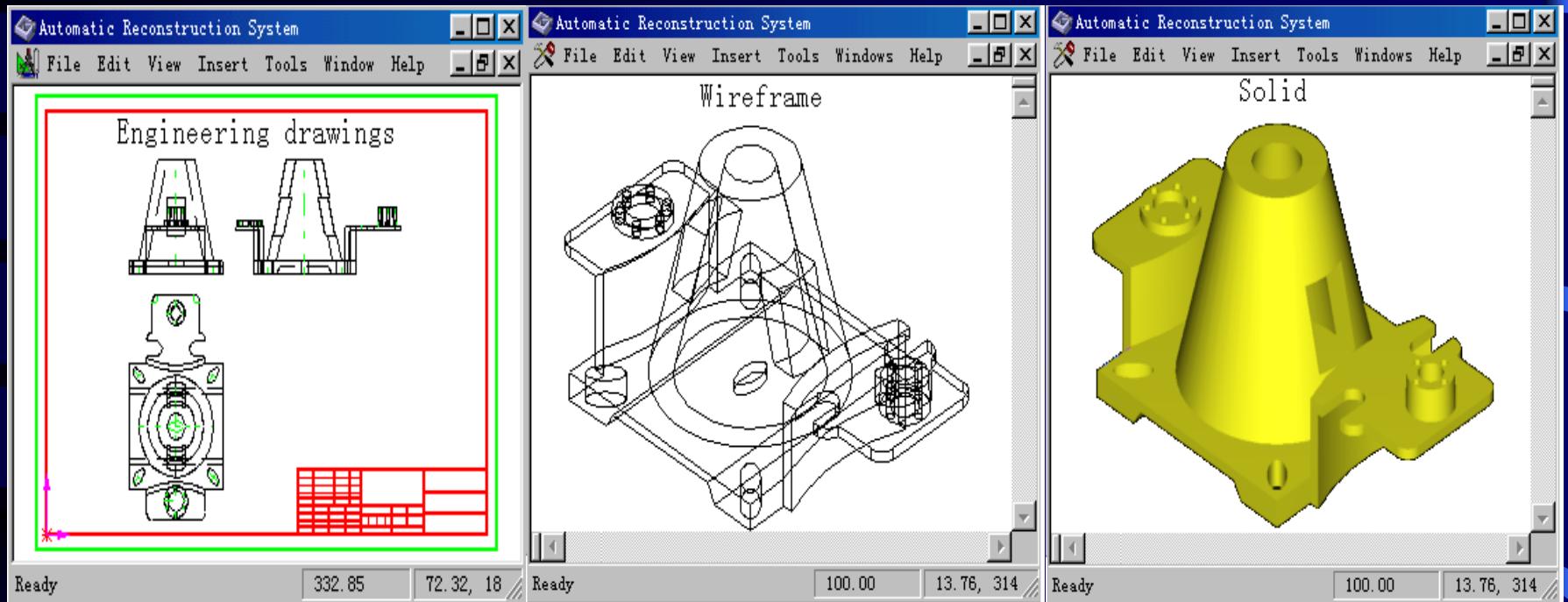
- 飞机、汽车、船舶的外形的设计
- 发电厂、化工厂等的布局
- 土木工程、建筑物的设计
- 电子线路、电子器件的设计
- 设计结果直接送至后续工艺进行加工处理，如波音777飞机的设计和加工过程

# 奥迪效果图和线框图

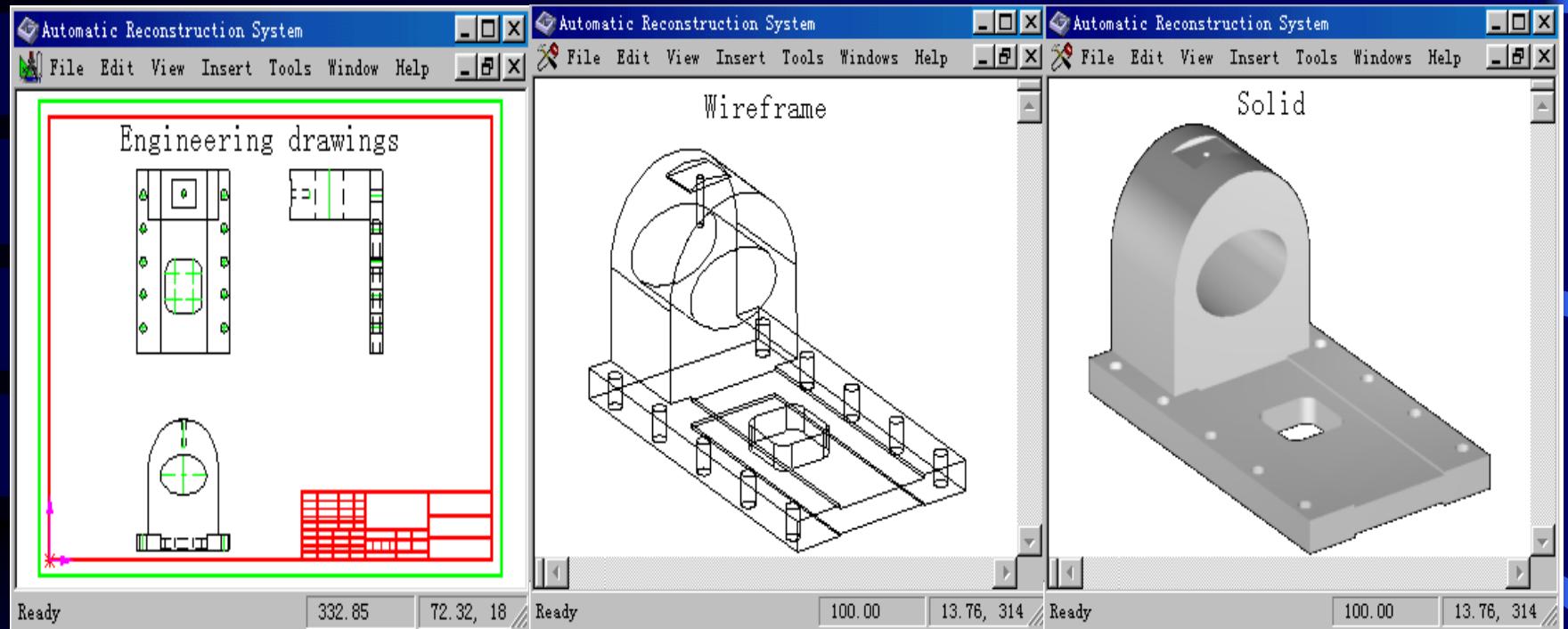


# 计算机辅助设计与制造 (CAD/CAM)

- 基于工程图纸的三维形体重建
  - 定义：从二维信息中提取三维信息，通过对这些信息进行分类、综合等一系列处理，在三维空间中重新构造出二维信息所对应的三维形体，恢复形体的点、线、面及其拓扑关系，从而实现形体的重建
  - 优势：可以做装配件的干涉检查、以及有限元分析、仿真、加工等后续操作，代表CAD技术的发展方向



工程图及其三维重建结果1



工程图及其三维重建结果2

# 可视化

- 科学计算可视化(Scientific Visualization)
  - 海量的数据使得人们对数据的分析和处理变得越来越难，用图形来表示数据的迫切性与日俱增
  - 1986年，美国科学基金会（NSF）专门召开了一次研讨会，会上提出了“科学计算可视化（Visualization in Scientific Computing）”
  - 科学计算可视化广泛应用于医学、流体力学、有限元分析、气象分析当中

- 在医学领域，可视化有着广阔的发展前途
  - 是机械手术和远程手术的基础
  - 将医用CT扫描的数据转化为三维图象，帮助医生判别病人体内的患处
  - 由CT数据产生在人体内漫游的图象
- 可可视化的前沿与难点
  - 可视化硬件的研究
  - 实时的三维体绘制
  - 体内组织的识别分割——Segmentation

# 真实感图形实时绘制与自然景物仿真

- 计算机中重现真实世界的场景叫做**真实感绘制**
- **真实感绘制**的主要任务是模拟真实物体的物理属性，简单的说就是物体的形状，光学性质，表面的纹理和粗糙程度，以及物体间的相对位置，遮挡关系等等

# 真实感图形实时绘制与自然景物仿真

- 光照模型

- 简单光照模型
- 局部光照模型
- 整体光照模型

- 绘制方法

- 光线跟踪
- 辐射度

- 加速算法

- 包围体树、自适应八叉树等等

# 地理信息系统（GIS）

- 建立在地理图形之上的关于各种资源的综合信息管理系统
- 数字地球，地形数据作为载体，（70%）全球信息化。
- 军事，政府决策，旅游，资源调查。

数字地球：1998年1月31日美国副总统戈尔在洛杉矶加利福尼亚科学中心召开的地理信息系统年会上提出了这一设想。

比如，可以设想一个小孩来到地方博物馆的一个数字地球陈列室，当他戴上头盔显示器，她将看到就象是出现在空中的地球。使用“数据手套”，她开始放大景物，伴随越来越高的分辨率，她会看到大洲，随之是区域、国家、城市、最后是房屋、树木以及其它各种自然和人造物体。在发现自己特别感兴趣的某地块时，她可乘上“魔毯”，即通过地面三维图象显示去深入查看。当然，地块信息只是她可以了解的多种信息中的一种。使用数字地球系统的语音识别装置，小孩还可以询问有关的土地覆盖、植物和动物种类的分布、实时的气候、道路、行政区线，以及人口等方面的信息。在这里，她还可以看到自己以及世界各地的学生们为“全球项目”收集的环境信息。这些信息可以无缝地融入数字地图或地面数据里。用数据手套向超连接部分敲击，她还可以获得更多的有关她所见物体的信息。比如，为了准备全家去国家黄石公园渡假，她策划一个完美的步行旅游，去观看刚从书中读到的喷泉、北美野牛和巨角岩羊。甚至在离开她家乡的地方博物馆之前，她就可以把要去步行旅游的地方从头到尾地浏览一遍。

# 娱乐

## ● 电脑游戏

- 实时性
- 逼实性
- 蕴含了先进的图形处理技术

## ● 电视广告，节目片头，科教演示（CAI）

- Quake III, “古墓丽影”，“侏罗纪公园”、“皇帝的新衣”、完美风暴.....
- MAYA, 3D-MAX, SOFTIMAGE...

应

用

- 计算机动画——商业领域
- 广告设计
- 电脑游戏
- 卡通动画片
- 影视特技

应

用

- 计算机艺术——艺术领域
  - 齐东旭
  - 潘云鹤
  - 智能CAD

# 应 用

## ➤过程控制

➤石油化工、金属冶炼、电网控制的工作人员根据设备关键部位的传感器送来得图像和数据，对设备运行过程进行监控

➤机场、铁路的调度人员通过计算机产生运行状态信息来调整空中交通和铁路运输

应

用

➤ 系统环境模拟

➤ 飞行模拟舱——用光栅扫描器产生驾驶员在驾驶舱中预期所能看到的景象，对飞行员进行单飞前的地面训练和飞机格斗训练等

应

用

- 事务和商务数据的图形显示
- 绘制表示经济信息的各类二、三维统计管理图表
- 信息可视化:信息流量, 商业统计数据, 股市行情

# 多媒体

在计算机控制下，对多种媒体信息进行生成、操作、表现、存储、通信、或集成的信息系统，其中媒体至少应包括一种“连续媒体”及一种

“离散媒体”

- 计算机处理的常见媒体：文本、图形、图像、语音、音频、视频、动画
- 特点：媒体的多样性、操作的交互性、系统的集成性
- CAI，教学娱乐。

# Virtual Reality (虚拟现实、灵境)

Virtual Reality 或称虚拟环境 (Virtual Environment)

- 是用计算机技术来生成一个逼真的三维视觉、听觉、触觉或嗅觉等感觉世界，让用户可以从自己的视点出发，利用自然的技能和某些设备对这一生成的虚拟世界客体进行浏览和交互考察。

- 输入输出设备



# 虚拟现实 (Virtual Reality简称VR)

虚拟现实是指用立体眼镜、传感手套等一系列传感辅助设施来实现的一种三维现实，人们通过这些设施以自然的方式（如头的转动、手的运动等）向计算机送入各种动作信息，并且通过视觉、听觉以及触觉设施使人们得到三维的视觉、听觉等感觉世界。随着人们不同的动作，这些感觉也随之改变。

## •QuickTime技术简介

QuickTime是苹果公司开发的新一代虚拟现实技术。它是一种基于静态图像处理的，在微机平台上能够实现的初级虚拟现实技术。它的出现使得以往专业实验室中成本昂贵的虚拟现实技术的应用普及有了广阔前景。

假定我们在一室空间进行观察，室内空间一般有六个面，如果我们获取了这六个面的许多不同距离，不同方位的实景照片并将它们按照相互的关系有机连接起来，就可以在视觉上形成这个房间整个空间的整体认识，这就是全景概念

# 当前研究热点

- 当前研究热点
  - 真实感图形实时绘制
    - 物体网格模型的面片简化: 对网格面片表示的模型, 在一定误差的精度范围内, 删除点、边、面, 从而简化所绘制场景的复杂程度, 加快图形绘制速度
    - 基于图象的绘制(IBR, Image Based Rendering): 完全摒弃传统的先建模, 然后确定光源的绘制的方法。它直接从一系列已知的图象中生成未知视角的图象, 适用于野外极其复杂场景的生成和漫游

## – 野外自然景物的模拟

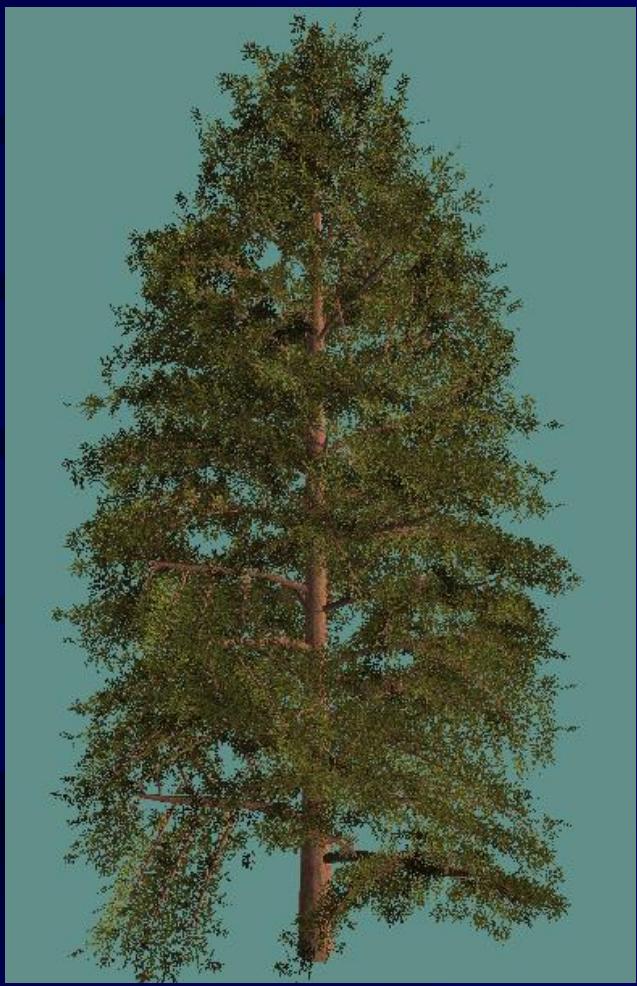
- 野外场景远远复杂于室内场景，绘制难度更大，方法更趋多样化
- 主要绘制山、水、云、树、草、火等等
- 绘制火的粒子系统（Particle System），基于生理模型的绘制植物的方法，绘制云的细胞自动机方法等



由清华大学自然景物平台生成的野外场景



日本Yoshinori Dobashi等人绘制的真实感云



Xfrog3.0生成的  
挪威云杉

# 与计算机网络技术的紧密结合

- 远程医疗与诊断
- 远程导航与维修
- 远程教育

# 计算机动画

- 计算机动画

- 计算机动画近十多年来取得了很大的发展，已渗透到人们生活的各个角落

- 商业广告、影视特技/片头、动画片
    - 教育、军事、飞行模拟等

- 分类

- 二维动画
      - 图象变形
      - 形状混合

- 三维动画
  - 关键帧动画
  - 变形物体的动画
  - 过程动画
  - 关节动画与人体动画



基于特征的图象变形

我国第一部利用计算机辅助摄制的动画片是《咪咪钓鱼》，1991年由北方工业大学和北京电视台合作制作，以二维动画为主。用386微机和C语言编程，利用数字化仪和摄像机产生关键帧，再由计算机在相邻两幅关键帧之间内插生成中间帧，并自动跟踪上色。多层画面叠加在一起，形成完整的画面。10分钟的片子，5人小组花了10个月时间。

# 用户接口

- 用户接口
  - 用户接口是人们使用计算机的第一观感。一个友好的图形化的用户界面能够大大提高软件的易用性
  - 图形学已经全面融入计算机的方方面面，很多软件几乎可以不看任何说明书，而根据它的图形、或动画界面的指示进行操作

- 目前几个大的软件公司都在研究下一代用户界面，开发面向主流应用的、自然、高效多通道的用户界面。研究多通道语义模型、多通道整合算法及其软件结构和界面范式是当前用户界面和接口方面研究的主流方向，而图形学在其中起主导作用。

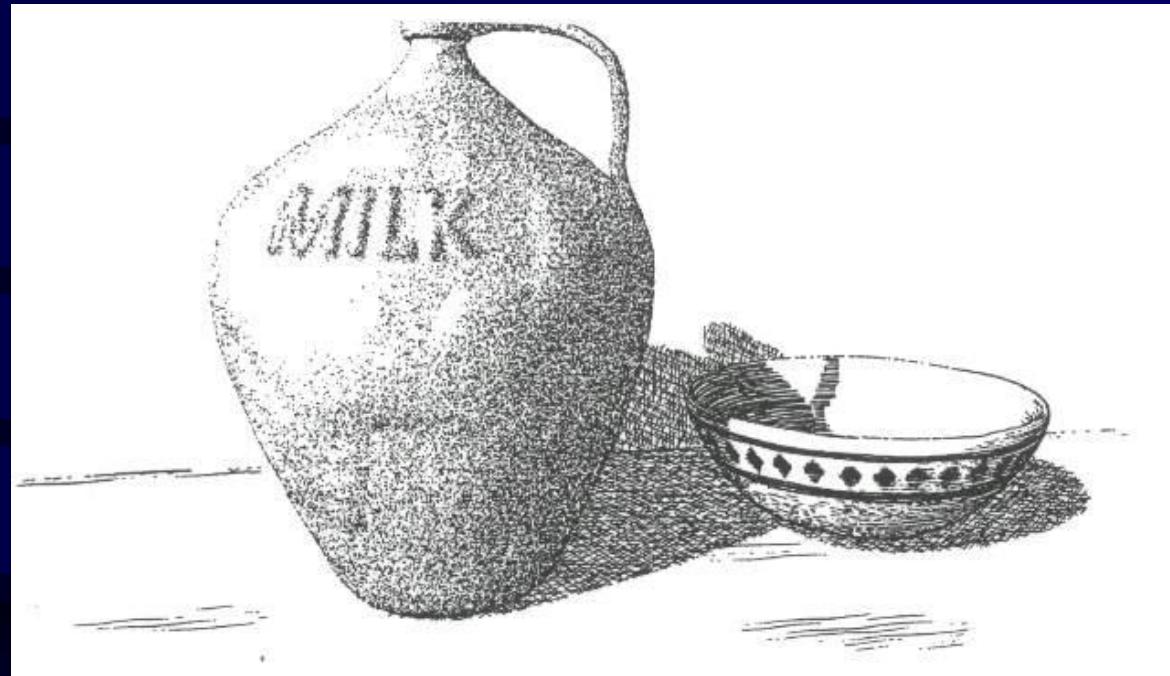
# 计算机艺术

- 计算机艺术
  - 用计算机软件从事艺术创作
    - 二维平面的画笔程序（如CorelDraw, Photoshop, PaintShop）
    - 图表绘制软件（如Visio）
    - 三维建模和渲染软件包（如3DMAX, Maya）、以及一些专门生成动画的软件（如Alias, Softimage）

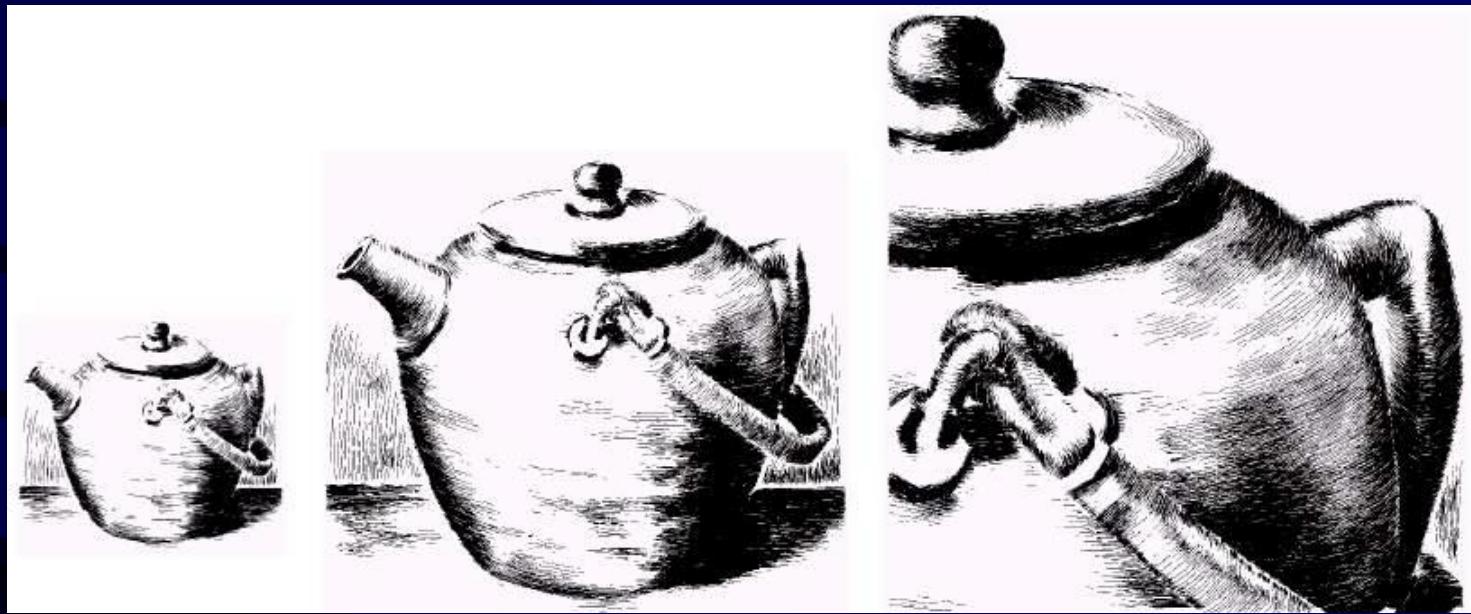
- 优点：
  - 提供多种风格的画笔画刷
  - 提供多种多样的纹理贴图，甚至能对图象进行雾化，变形等操作
  - 可以任意修改，取消败笔
- 不足：
  - 无法达到传统绘画中风格化的艺术效果
  - 很难得到有素描效果、油画效果的艺术品



- 非真实感绘制（NPR， Non-Photorealistic Rendering）
  - 用于模拟艺术效果，研究方法有别于真实感图形学
  - 钢笔素描的生成
    - 钢笔素描产生于中世纪，从19世纪开始成为一门艺术  
20世纪90年代开始研究用计算机模拟
  - 中国国画与书法的生成



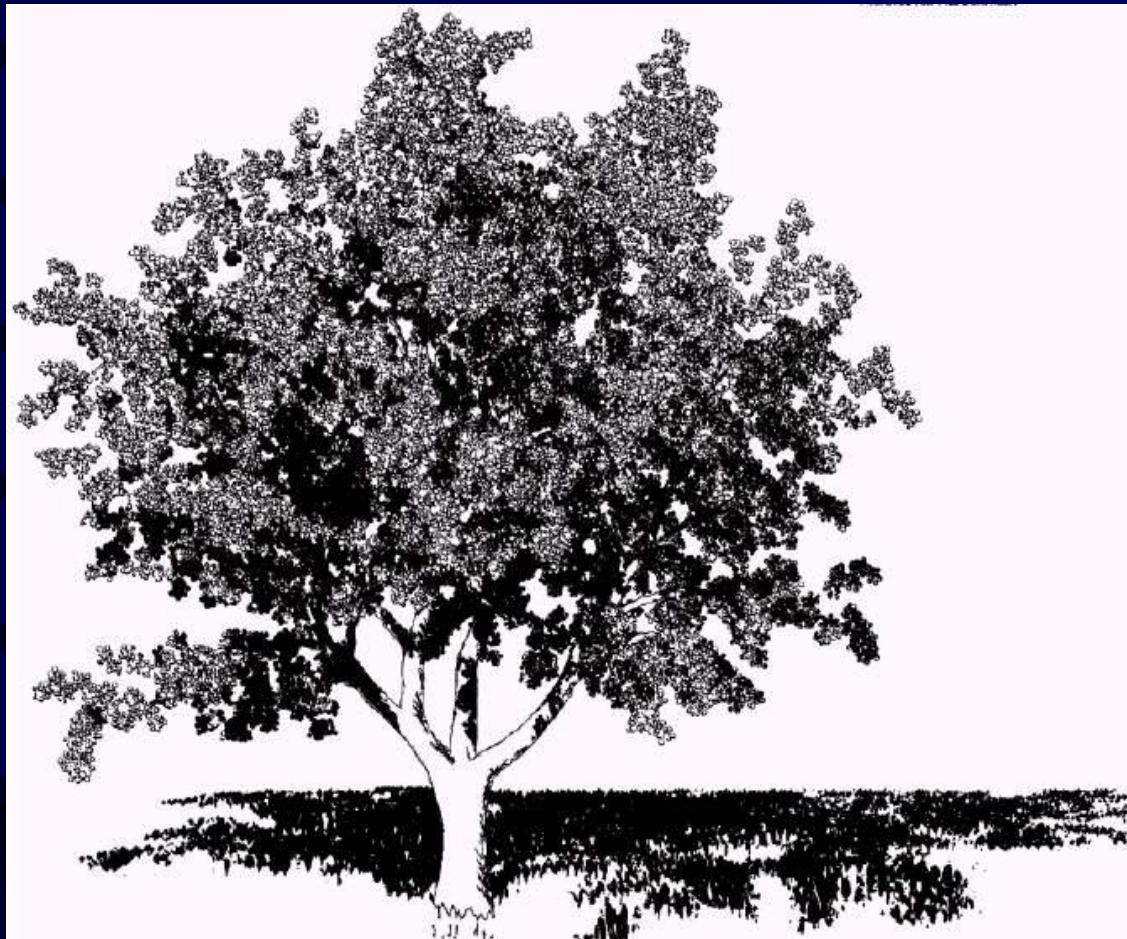
Georges Winkenbach  
L'Amour des Femmes  
(Sieggraph '96)



Salisbury绘制的茶壶(1760-1770)



Salisbury绘制的獾(引自《博物志》)



Oliver Deussen绘制的  
(Siggraph'2000)

# 概念与术语

图形

图形的表示

图形学的研究内容

图形学的应用

主要的研究动态

所熟悉的图形软件包

图形软件的标准

# 作业

- 查找资料：关于计算机图形学的相关内容，应用及前沿问题的研究，及其你对及计算机图形学的初步理解与看法。写成一篇论文。
- 发送到邮箱：  
[shilimin\\_syict@sina.com.cn](mailto:shilimin_syict@sina.com.cn)
- 作为选课依据。

# 第二章 计算机图形系统

## 2.1 计算机图形系统的组成

2.1.1图形系统组成

2.1.2图形系统的功能

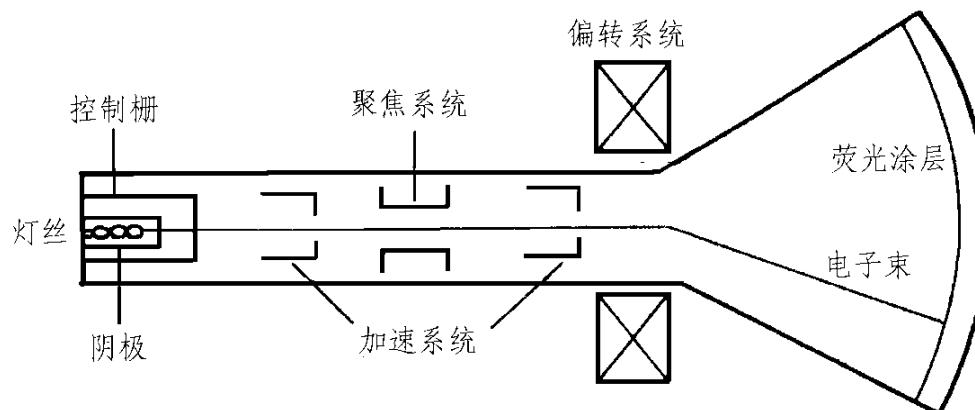
2.1.3 图形系统的分类

## 2.2 计算机图形显示器

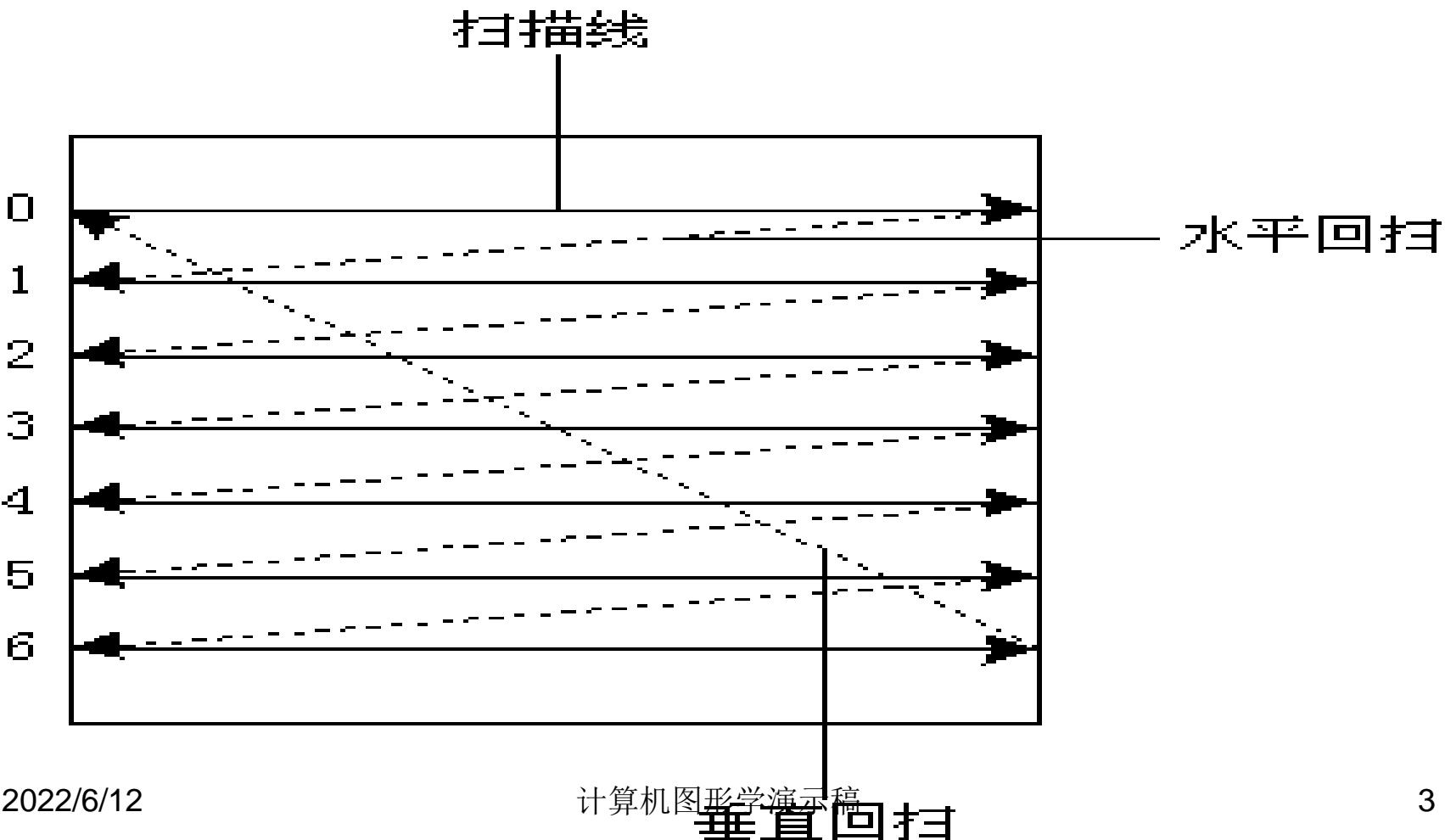
计算机图形系统一般使用视频显示器作为基本的输出设备。大部分视频监视器的操作是基于标准的阴极射线管（Cathode-Ray Tube, CRT）设计的。

### 2.2.1 CRT显示器工作原理

CRT主要由3部分组成：电子枪、偏转系统和荧光屏，其构造见下图。



要保持荧光屏上有稳定的图象就必须不断地发射电子束。**刷新**一次指电子束从上到下将荧光屏扫描一次，其扫描过程如下图所示。只有刷新频率高到一定值后，图象才能稳定显示。大约达到每秒60帧即60Hz时，人眼才能感觉到屏幕不闪烁，要使人眼觉得舒服，一般必须有85Hz以上的刷新频率。



一个CRT可以在水平和垂直方向上无重叠显示的最多点数称为**分辨率**（Resolution），通常简述为每个方向的总点数。这是衡量CRT的重要指标，它取决于所用荧光物质的类型以及聚焦和偏转系统。显然，点数愈多，分辨率愈高，显示的图形也就愈精确。分辨率与光点直径大小有关，但是不可能大于可寻址能力。典型的CRT分辨率如 $640\times 480$ 、 $1024\times 1024$ 、 $1280\times 1024$ 、 $2048\times 2048$ 等。

CRT显示器的另一性能指标是**纵横比**（Aspect Ratio）。它给出在屏幕两个方向生成同等长度的线段所需垂直点数对水平点数的比值。纵横比为 $3/4$ 意味着垂直线画三点的长度与水平线画四点的长度相同。

## 2.2.2 彩色阴极射线管

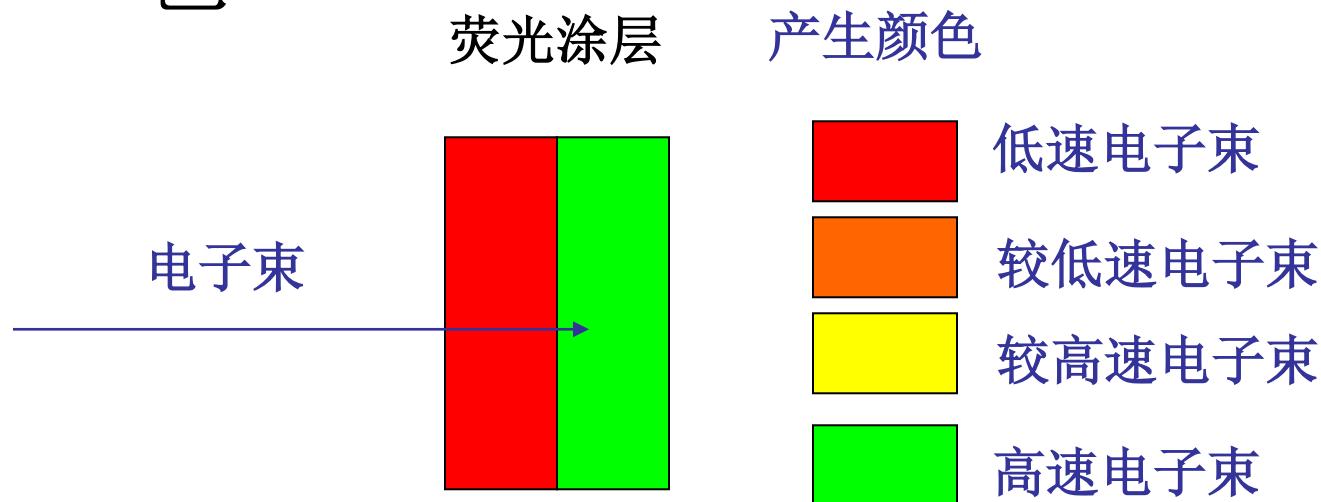
产生彩色的常用方法：

射线穿透法

影孔板法

## 2.2.2 彩色阴极射线管-射线穿透法

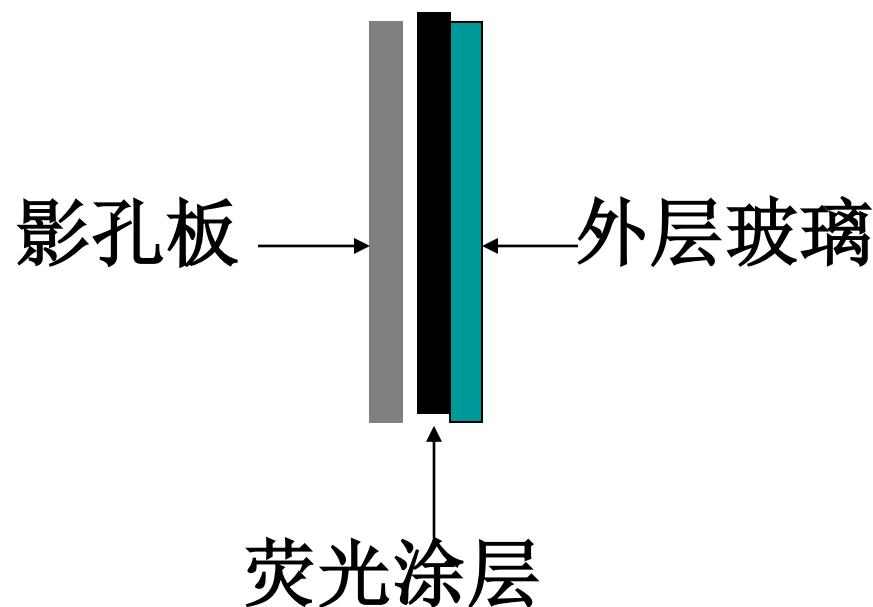
- 原理：两层荧光涂层，红色光和绿色光两种发光物质，电子束轰击穿透荧光层的深浅，决定所产生的颜色



## 2.1.2 彩色阴极射线管-影孔板法

### - 影孔板法

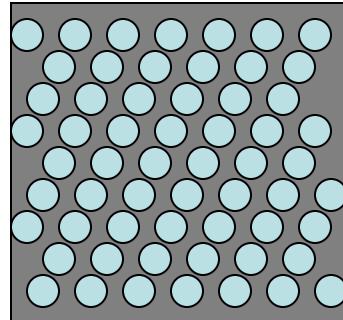
- 原理：影孔板被安装在荧光屏的内表面，用于精确定位像素的位置



## – 影孔板的类型

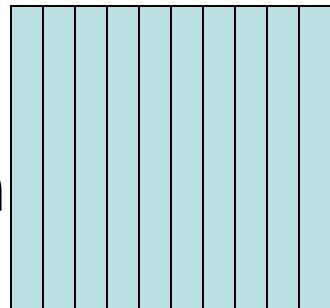
- 点状影孔板

代表：大多数球面与柱面显像管



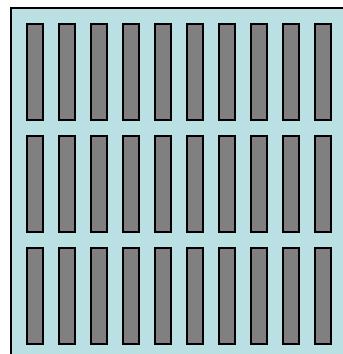
- 栅格式影孔板

代表：Sony的Trinitron与Mitsubishi的Diamondtron显像管



- 沟槽式影孔板

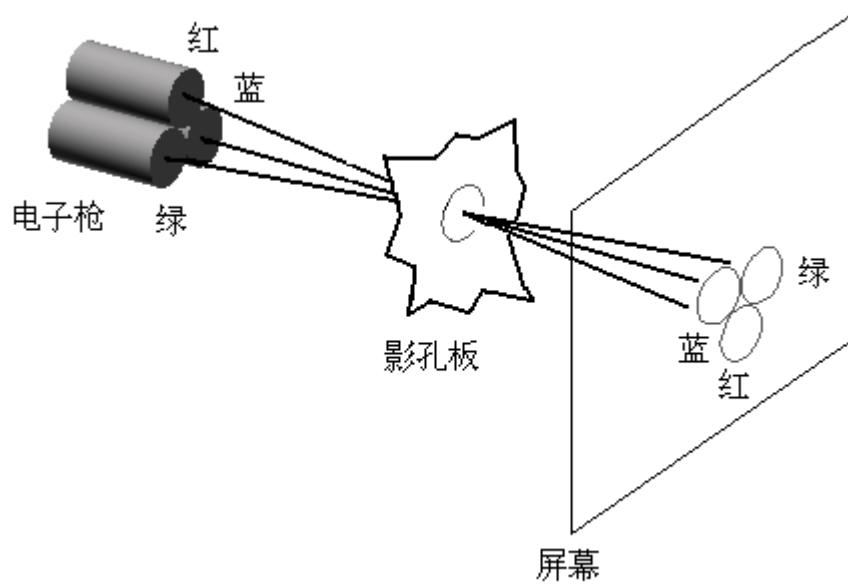
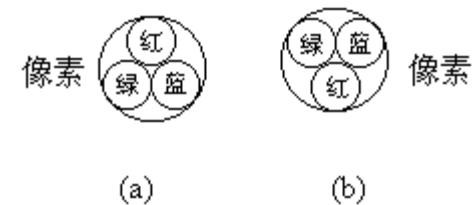
代表：LG的Flatron显像管



## 2.2.2 彩色阴极射线管-影孔板法

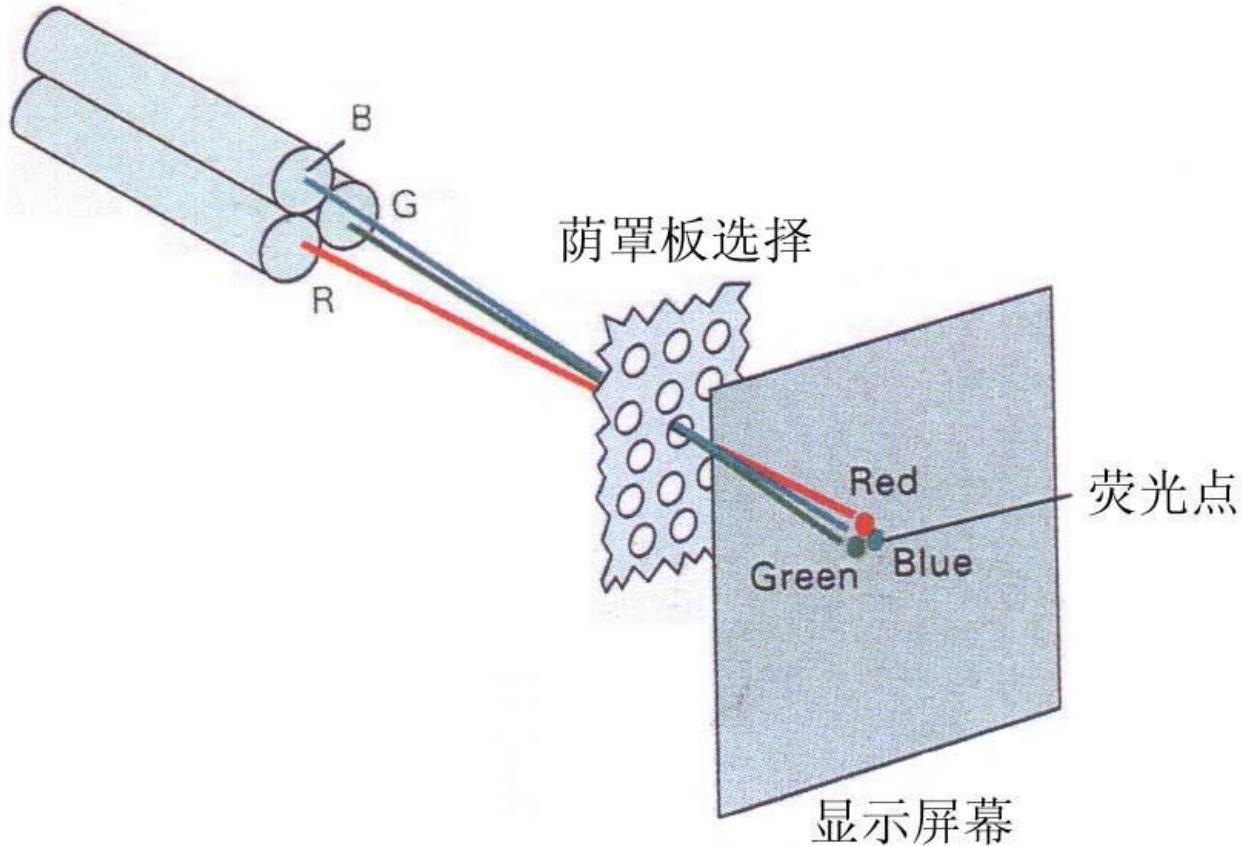
### - 点状影孔板工作原理

- 红、绿、兰三基色
- 三色荧光点（很小并充分靠近--> 像素）
- 三支电子枪



电子枪、影孔板中的一个小孔和荧光点呈一直线；  
每个小孔与一个像素（即三个荧光点）对应

电子枪



## 2.2.3 显示器的类型

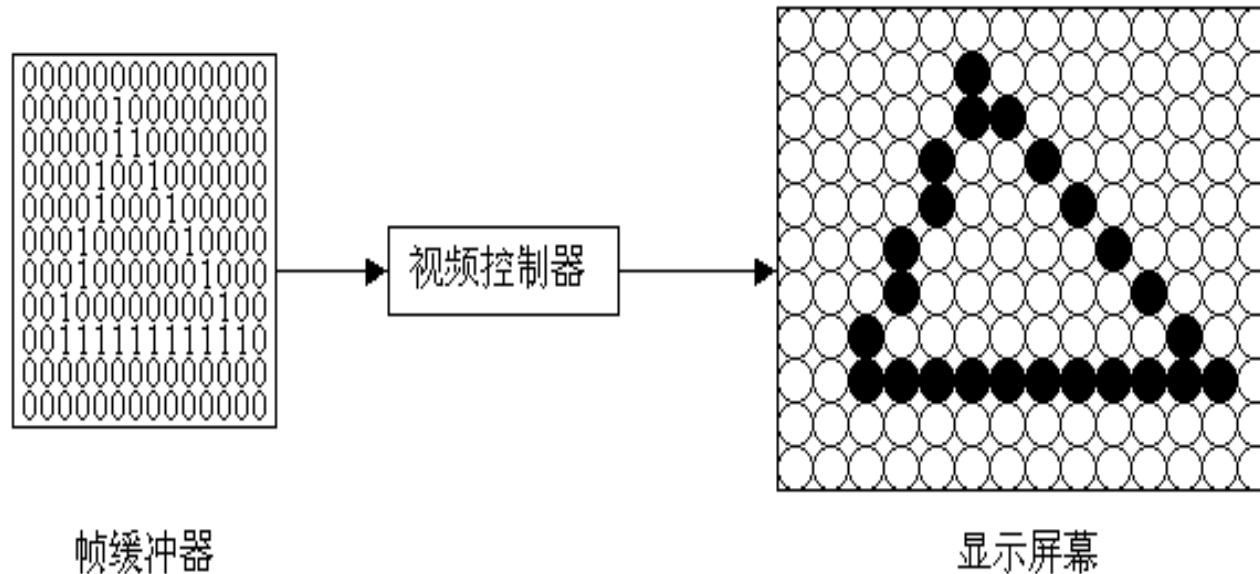
1. 随机扫描显示器
2. 存贮管显示器
3. 光栅扫描显示器

# 光栅扫描显示器

的CRT屏幕可分为横向和纵向双向扫描线，每一行又可分为N个小点。这样，整个屏幕就被分成 $M \times N$ 个小点，称为象素 (Pixel, Picture Element 的简写)。图形定义存于称为刷新缓冲器 (Refresh Buffer) 或帧缓冲器 (Frame Buffer) 的存储器中。

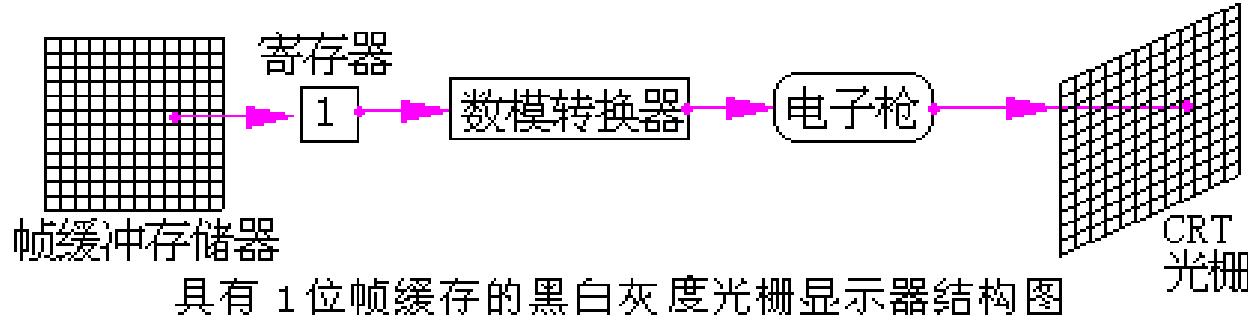
# 帧缓冲存储器

作用：存储屏幕上像素的颜色值  
简称帧缓冲器，俗称显存



- 缓存中单元数目与显示器上像素的数目相同，单元与像素一一对应，各单元的数值决定了其对应像素的颜色。
- ✓ 显示颜色的种类与帧缓存中每个单元的位数有关（图示帧缓冲器的每个单元只有一位）。

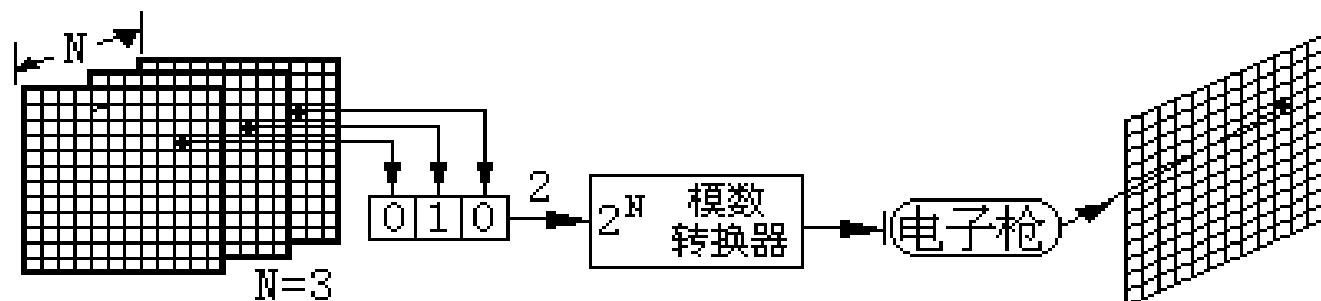
# 黑白光栅扫描显示器



- 黑白光栅显示器的逻辑框图如上：其中帧缓存是一块连续的计算机存储器。对于黑白单灰度显示器每一象素需要一位存储器，对一个 $1024 \times 1024$ 象素组成的黑白单灰度显示器所需要的最小缓存为 $2^{20}$ ，并在一个位面上。一个位面的缓存只能存储黑白图形，帧缓存是数字设备，光栅显示器是模拟设备，因而还需要数模转换器(DAC)。

# 彩色光栅扫描显示器

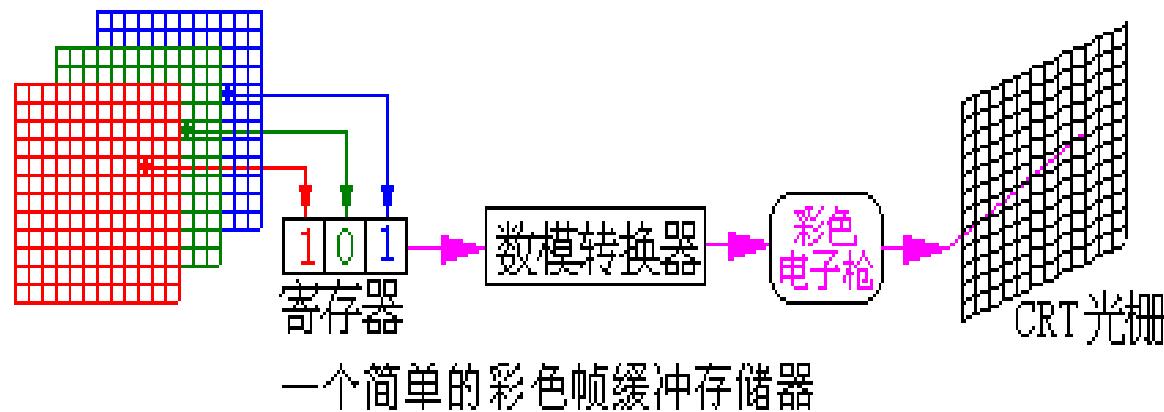
- 在光栅图形显示器中需要足够的位面和帧缓存结合起来才能反映图形的颜色和灰度等级。如下图是一个具有 $N$ 位面灰度等级的帧缓存。显示器上每个象素的亮度是由 $N$ 位面中对应的每个象素位置的内容控制的。该存储器中的二进制的数被翻译成灰度等级，范围是0到 $2^{N-1}$ 之间。



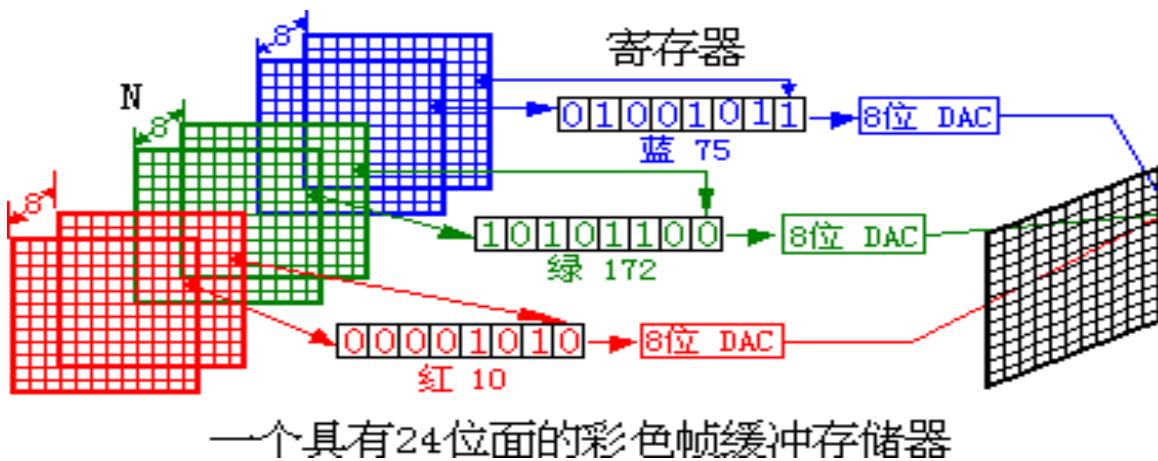
具有 $N$ 位帧缓存的黑白灰度光栅显示器结构图

# 彩色光栅扫描显示器

- 下图是彩色光栅显示器的逻辑图，对于红、绿、蓝三原色有三个位面的帧缓存和三个电子枪。



# 彩色光栅扫描显示器



- 每个颜色的电子枪可以通过增加帧缓存位面来提高颜色种类的灰度等级。如上图，每种原色电子枪有8个位位面的帧缓存和8位的数模转换器，每种原色可有256中灰度，三种原色的组合将是 $(2^8)^3=2^{24}$ 。



2022/6/12

计算机图形学演示稿

20

2022/6/12

计算机图形学演示稿

21

2022/6/12

计算机图形学演示稿

22



2022/6/12

计算机图形学演示稿

24

2022/6/12

计算机图形学演示稿

25

显示卡中VRAM(帧缓存) 同显示系统的关系:

用于显示器屏幕刷新的数据存储在显示卡的VRAM 中，因此相应于显示器屏幕的分辨率必须有足够的VRAM 存储显示数据。

例如：如果屏幕的分辨率是 $640 \times 480$ ，则如果是一个只能显示黑白图形的显示器，则需要VRAM的大小为：

$$(640 \times 480) / 8 = 38400 \text{ bytes} \approx 3.8 \text{ Kb}$$

如果是一个每个象素显示256个灰度级的图形显示器，则每个象素需要8位存储单元，需要VRAM的大小为：

$$640 \times 480 = 307200 \text{ bytes} \approx 307 \text{ Kb}$$

如果是一个彩色显示器，且每种基色均该显示256个亮度级，则每个象素需要24位存储单元，需要VRAM的大小为：

$$640 \times 480 \times 3 = 921600 \text{ bytes} \approx 921 \text{ Kb}$$

如果屏幕的分辨率是 $1024 \times 1024$ 的彩色显示器，且每种基色均该显示256个亮度级，需要VRAM的大小为：

$$1024 \times 1024 \times 3 = 3145728 \text{ bytes} \approx 3 \text{ Mb}$$



# 显存容量问题

- 分辨率M\*N、颜色个数K与显存容量V的关系

$$V \geq M \times N \times \lceil \log_2 K \rceil$$

# 显存容量问题

- 若存储器位长固定，则屏幕分辨率与同时可用的颜色种数成反比关系。
- 高分辨率和真彩要求有大的显存；

1024\*768真彩模式需要3M字节显存

- 解决方法：采用查色表(**Look-up Table**)

# 查色表 (LUT)

- 是一维线性表，其每一项的内容对应一种颜色，它的长度由帧缓存单元的位数决定
- 目的：在帧缓存单元的位数不增加的情况下，具有大范围内挑选颜色的能力。
- 颜色信息在帧缓存中的两种存放方式：
  - 颜色值直接存储在帧缓存中。
  - 把颜色码放在一个独立的表中，帧缓存存放的是颜色表中各项的索引值，索引色。
- 单色系统：查色表固化
- 彩显：可修改、创建查色表。

# 隔行扫描（Interlaced scan）工作原理

- 一帧完整的画面分成两场，即奇数场与偶数场
  - 场频： = 帧频 \* 2
- 
- 优点：
    - 降低了闪烁效应；
    - 只需逐行的一半时间即可显示一屏画面，降低了对扫描频率的要求，也降低了成本；
    - 帧缓存中数据量比逐行扫描少一半，降低了视频控制器存取帧缓存的速度及传输带宽的要求。

# 视频控制器（显示控制器）

- 作用：
  - 制图形的显示，建立帧缓存与屏幕像素之间的一一对应关系，负责按固定刷新频率和扫描顺序刷新屏幕图形

# 显示处理器

- 显示处理器
  - Display Processing Unit, 简称DPU
- 作用：
  - 代替CPU完成部分图形处理功能，扫描转换、几何变换、裁剪、光栅操作、纹理映射等等

- 光栅显示系统的特点
  - 优点：
    - 成本低
    - 易于绘制填充图形
    - 灰度和色彩丰富，图像逼真
    - 可以和电视机兼容
    - 刷新频率一定，与图形的复杂程度无关
  - 缺点：
    - 需要扫描转换
    - 扫描转换速度偏低，交互操作响应慢
    - 分辨率偏低，有阶梯效应

**CRT**固有的物理结构限制了它向更广的显示领域发展

屏幕的加大必然导致显象管的加长，显示器的体积必然要加大，在使用时候就会受到空间的限制

**CRT**显示器是利用电子枪发射电子束来产生图像，容易受电磁波干扰

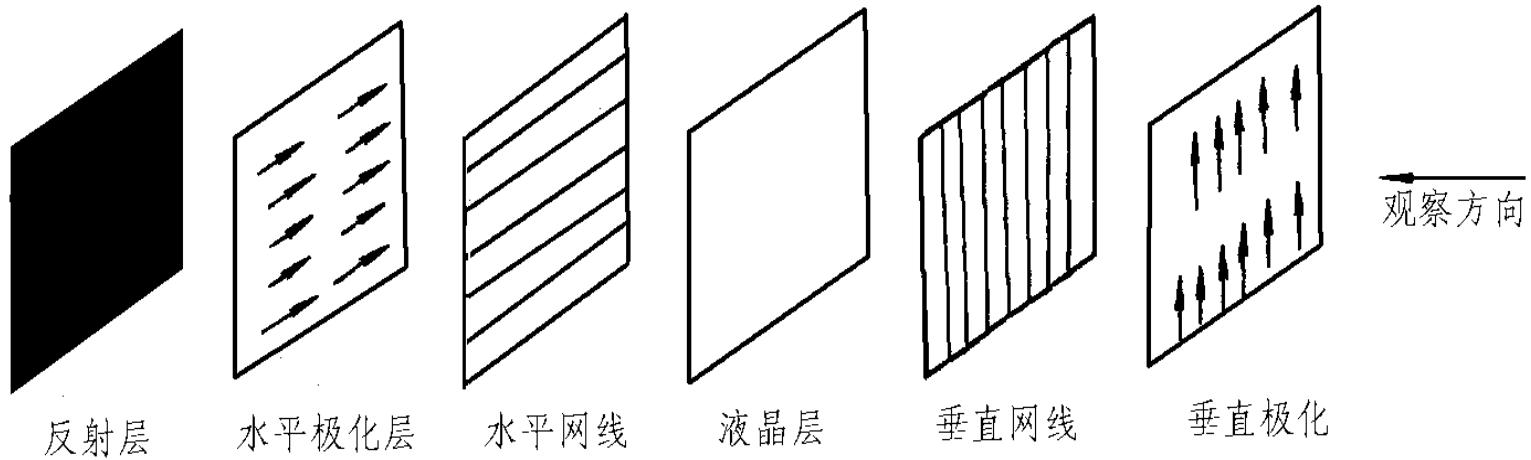
长期电磁辐射会对人们健康产生不良影响

## 2.2.4 平板显示器

目前，已经有不少其它类型的视频显示设备被使用。平板显示器（Flat—Panel Display）代表一类能比CRT减小体积、减轻重量并节省功耗的视频设备。

### 1. 液晶显示器

液晶显示器（Liquid—Crystal Display, LCD）生成图形的机理是通过能阻塞或传递光的液晶材料，传递来自周围的或内部光源的偏振光。液晶显示器由六层组成，如下图所示。



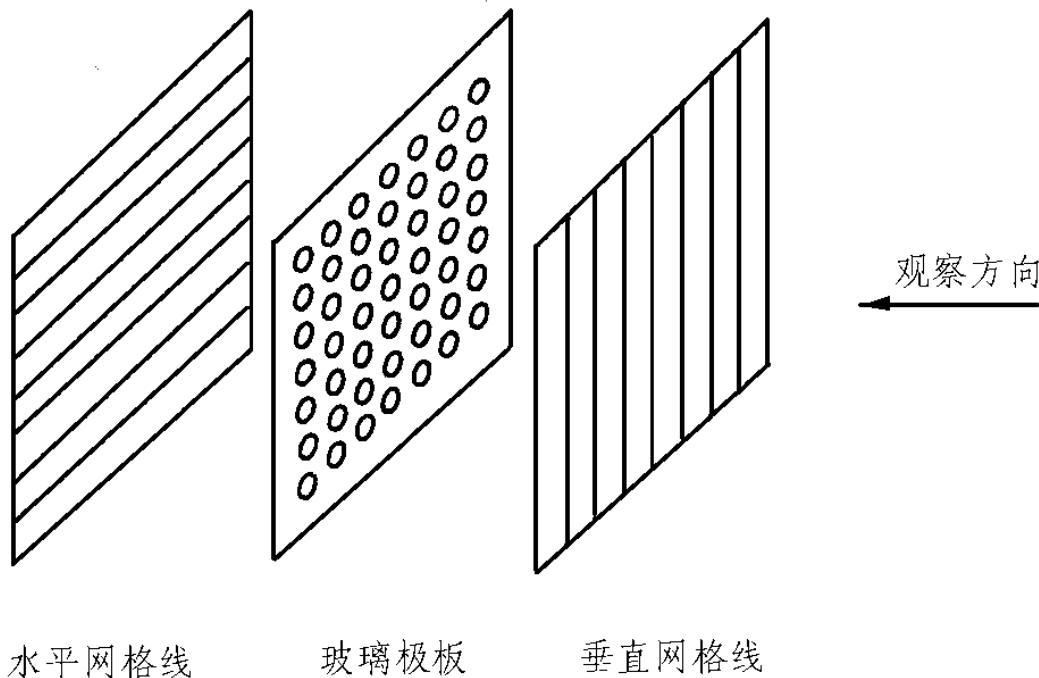
彩色液晶显示可用不同材料或染料，并在每个像素上放置三个薄膜晶体管。晶体管用来控制像素位置的电压，并阻止液晶单元慢性漏电。这些设备称为有源矩阵（Active-Matrix）显示器。

液晶显示器外观如右图



## 2. 等离子显示器

等离子体显示（Plasma Panel），其结构为用通常包括氖气的混合气体充入两块玻璃板之间的区域。一块玻璃板上放置一系列垂直导电带，而另一玻璃板上构造一组水平带，如下图所示。



在成对的水平和垂直导电带上施以点火电压，导致两导电带交叉点处的气体进入辉光放电的等离子区。图形的定义被存储在刷新缓冲器，点火电压以每秒60次的速率，用于刷新象素位置（导电带的交叉处），使用交变电流方法快速提供点火电压，可得到较亮的显示。等离子体显示技术适合于制造较大屏幕的显示器。

## 等离子体显示器图例



## 2.3 计算机图形输入设备

图形输入设备从逻辑上分为六类，即定位（Locator）、描绘（Stroke）、数值输入（Valuator）、选择（Choice）、拾取（Pick）及字符输入（String），也可称为六种逻辑设备。所谓逻辑设备，是指按逻辑功能定义的设备，并非具体的物理设备。一种逻辑设备对应于一种或一类特定的物理设备，而实际的物理设备可以完成一种或一种以上的逻辑设备功能。

常用的输入设备**键盘**和**鼠标器**也是图形输入设备。

**光笔**是一种可以直接在屏幕上绘图的设备，但目前已使用不多了。

数字化仪和图形输入板是常见的定位设备，其中全电子式坐标数字化仪由于精度高，使用方便，得到普遍应用。这种设备利用电磁感应原理：在台板的X方向上有许多水平方向的平行印刷线，Y方向上是垂直方向的平行印刷线。游标中装有一个线圈，当线圈中通有交流信号时，十字交叉丝的中心便产生一个电磁场，当游标在台板上运动时，台板下的印刷线上就会产生感应电流。印刷板周围的多路开关等线路可以检测出最大信号的位置，即十字叉线中心所在的位置，从而得到该点的坐标值。图形输入板（Tablet）工作原理相同于数字化仪，只是面积较小而已。

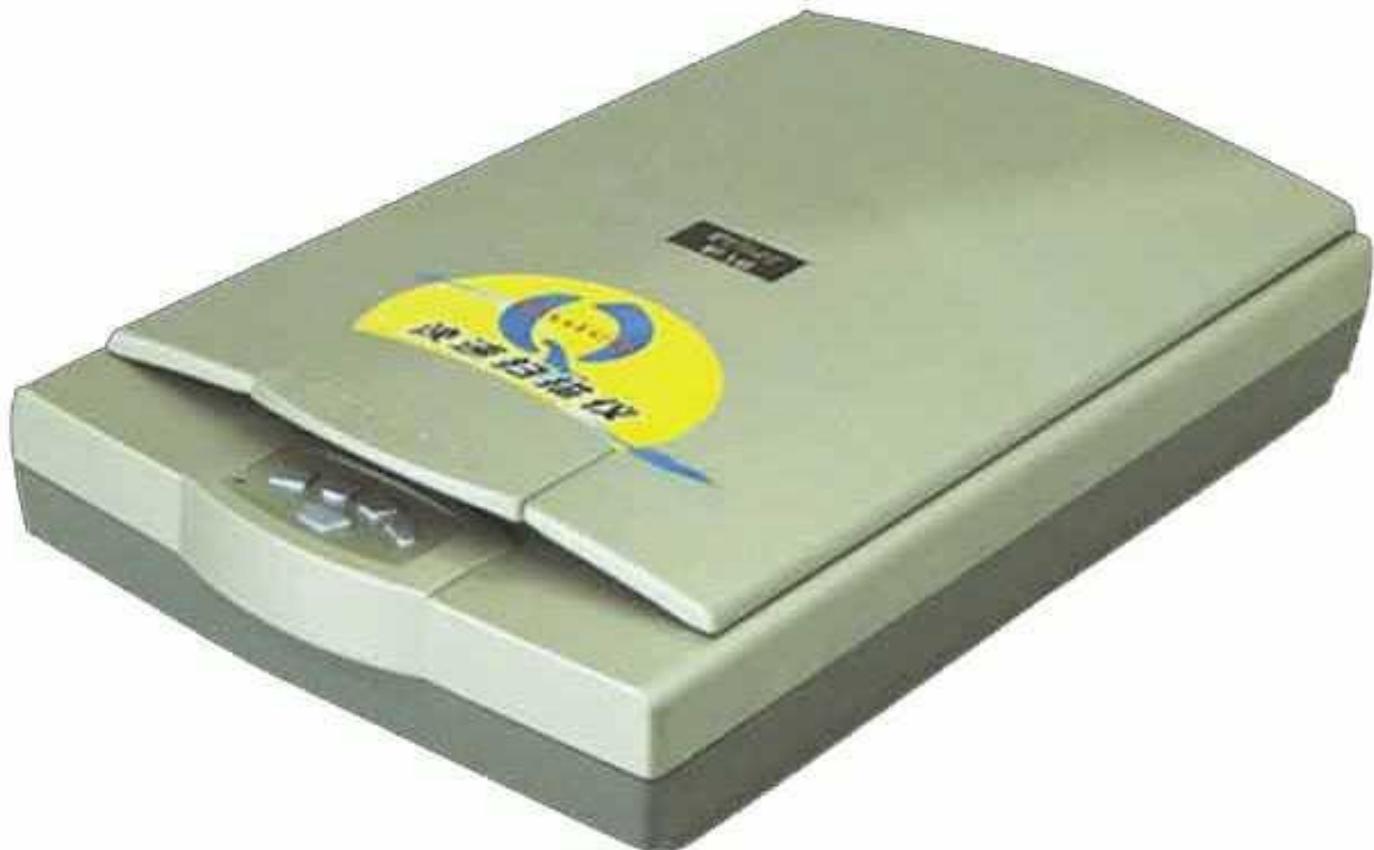
数字化仪时常用来拾取放在它上面的工程图上的大量点，经数字化后存储起来，以此作为图形输入一种手段。

**操纵杆，跟踪球**是将位移量转变成屏幕光标移动的输入设备。

**触摸屏**容许用手指触摸显示的物体或屏幕位置来实现选择，典型应用是对用图符或菜单表示的处理选项进行选择。目前广泛应用于公告查询系统。触摸屏的工作原理分电容、电阻和声波等。

**图象扫描仪**（Image Scanner）是直接把视图、图表、彩色和黑白照片扫描输入到计算机中，以像素信息进行存储表示的设备。扫描仪的幅面有A0,A1,A4等。扫描仪的分辨率是指在原稿的单位长度（英寸）上取样的点数，单位是dpi（dot per inch），常用分辨率为300dpi～1000dpi之间。扫描图形分辨率越高，所需的存储空间就越大。

# 清华紫光扫描仪



**数据手套**：(Data Glove) 通常用于虚拟环境中，可用来抓住“虚拟”对象。手套由一系列检测手和手指运动的传感器构成。发送天线和接收天线之间的电磁耦合，用来提供关于手的位置和方向的信息。发送和接收天线各自由一组三个相互垂直的线圈构成，形成三维笛卡儿坐标系统。来自手套的输入，可用来定位或操纵虚拟环境中的对象。

使用数据手套同计算机交互示图：



**声音系统：**（Voice—System）使用语音识别技术，把人们说的话转换成计算机能懂的数字代码。这样的代码能使其可以用于各种各样的应用程序,从口授文本变成字处理的文档、到说话控制计算机的功能。在计算机图形系统中可被用作接收声音命令的输入设备，可用于图形操作或输入数据等。

**摄像头：**可以将摄像直接输入计算机



## 2.4 计算机图形输出设备

图形输出设备主要包括显示器、绘图仪、打印机等。

**绘图仪：**主要有笔绘式、喷墨式和静电式三类。笔绘式绘图仪可分为平台式、滚筒式、平面电机式以及小型式四种。这是近些年来应用较多的图形输出设备。

**打印机：**根据打印机制式的不同，打印机有点阵式打印机和激光打印机两种。点阵式打印机又分为针打点阵打印机、静电点阵打印机、喷墨点阵打印机、热转换打印机等。

针打点阵打印机



# 喷墨打印机



# 大幅面打印机



## 2.5计算机图形软件

**图形软件的类型：**图形软件可以分为两类，基本图形软件或称支撑软件和应用图形软件。

基本图形软件包括解决图形设备与主机的通讯、接口等问题的基本输入、输出程序，又称为设备驱动程序；生成基本图元，建立图形数据结构，定义、修改和输出图形，以及对设备进行管理的基本程序软件。

应用图形软件是为解决某种具体应用问题而设计的图形软件，例如各种CAD系统。应用图形软件是在基本图形软件基础上设计完成的。

**图形软件标准化：**国际ISO组织和许多国家的标准化组织进行了合作，努力开发能被大家接受的计算机图形标准。在进行了相当可观的努力后，推出了图形核心系统GKS（Graphical Kernel System）。第二个图形软件标准是PHIGS（Programmer's Hierarchical Interactive Graphics System，程序员级分层结构交互图形系统），PHIGS此后也扩充为PHIGS+，提供PHIGS所没有的三维表面明暗处理能力。

## 通用的、与设备无关的图形标准

- GKS (Graphics Kernel System) (第一个官方标准, 1977)
- PHIGS (Programmer's Hierarchical Interactive Graphics system)

一些非官方图形软件，广泛应用于工业界，成为事实上的标准

- DirectX (MS)
- Xlib (X-Window 系统)
- OpenGL (SGI)
- Adobe 公司 Postscript

# 开放式的三维图形软件包OpenGL

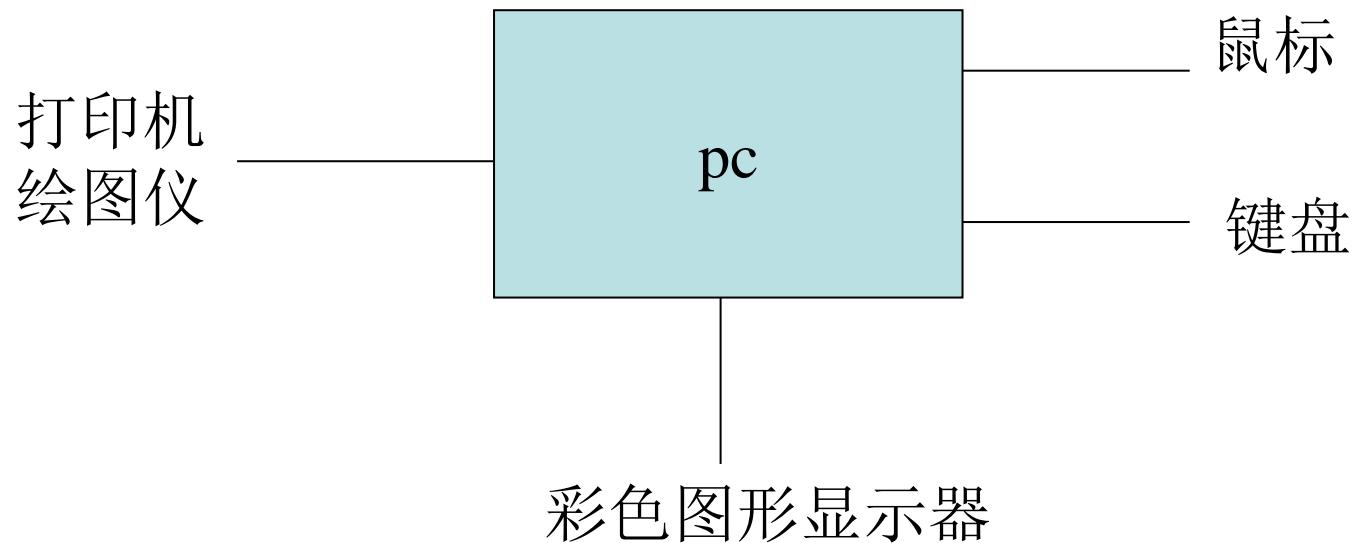
- OpenGL是近几年发展起来的一个性能卓越的三维图形标准，它是在SGI等多家世界闻名的计算机公司的倡导下，以SGI的GL三维图形库为基础制定的一个通用共享的开放式三维图形标准。
- 目前，包括Microsoft、SGI、IBM、SUN、HP等大公司都采用了OpenGL做为三维图形标准，许多软件厂商也纷纷以OpenGL为基础开发出自己的产品。
- 其中比较著名的产品包括：动画制作软件Softimage和3D Studio MAX、仿真软件Open Inventor、VR软件World Tool Kit、CAM软件ProEngineer、GIS软件ARC/INFO等等。

# OpenGL的优越性

- 独立于窗口系统和操作系统，以它为基础开发的应用程序可以十分方便地在各种平台间移植；
- 可与Visual C++紧密接口，便于实现机械手的有关计算和图形算法，可保证算法的正确性和可靠性；
- 使用简便，效率高。

# OpenGL图形库的功能

- 一共有100多个函数。其中核心函数有115个
- 除了提供基本的点、线、多边形的绘制函数外，还提供了复杂的三维物体（球、锥、多面体、茶壶等）以及复杂曲线和曲面（如Bezier、NURBS等曲线或曲面）绘制函数。
- 基本几何变换和投影变换。
- 颜色模式设置
- 光照和材质设置
- 纹理映射功能
- 位图显示和图象增强：反走样(Antialiasing)和雾(fog)的特殊图象效果处理
- 双缓存(Double Buffering)动画：双缓存即前台缓存和后台缓存，即后台缓存计算场景、生成画面，前台缓存显示后台缓存已画好的画面。



- 计算功能：形体设计、分析的算法程序库和描述形体的数据。包括点、线、面的表示及其求交，几何变换，光、色模型的建立和计算。
- 存储功能：在计算机的内存、外存中存储图形数据。
- 对话功能：是通过图形显示器进行人——机通讯。
- 输入功能：
- 输出功能：



# 第三章 基本光栅图形基础

\*\*\*\*\*

物体的形状和颜色可用象素矩阵或直线线段和多边形填充区域等基本几何结构来描述。点和直线段是最简单的几何成分，其它可供构造图形的输出图元有圆及其它圆锥曲线、二次曲面、样条曲线和曲面、多边形填充区域以及字符串等。而二维图形的生成是三维图形生成的基础，研究计算机生成图形需先从二维图形的生成开始。下面将讨论一些基本二维图元生成技术和算法，以光栅图形系统的扫描转换方法为基础。

# 本章内容

? 直线段的扫描转换算法

    DDA算法

    Bresenham画线算法

? 区域填充

    扫描线填充

    种子填充

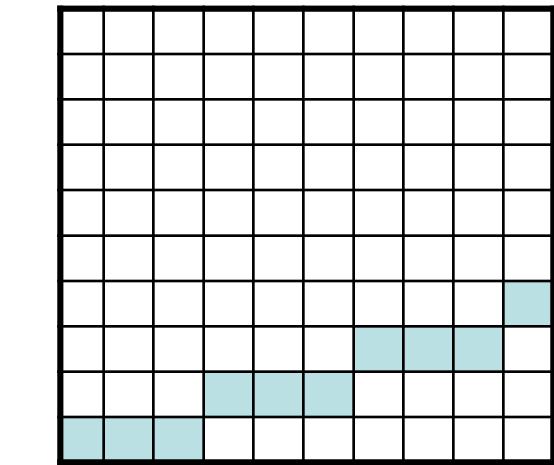
    边填充

# 直线段的扫描转换算法

- **直线的扫描转换:** 确定最佳逼近于该直线的一组象素，并且按扫描线顺序，对这些象素进行写操作。
- 常用算法：  
数值微分法（DDA）  
Bresenham算法。

直线是点的集合，在几何学中直线被定义为两个点之间的最短距离。也就是说一条直线是指所有在它上面的点的集合。直线是一维的，即它们具有长度但没有面积。直线可以向一个方向及其相反的方向无限伸长，这不是计算机图形学中所需要的，在图形学中研究的对象是直线段。已知线段的起点坐标 $(x_1, y_1)$ ，终点坐标 $(x_2, y_2)$ ，这两点就确定了一条线段。

一般来讲，任何图形输出设备都能准确地画出水平线X和垂直线Y，或对角线，但要画出一条准确斜线不是件容易的事。在光栅系统中，线段通过象素绘制，水平和垂直方向的台阶大小受象素的间隔限制。这就是说，必须在离散位置上对线段取样，并且在每个取样位置上决定距线段最近的象素，画一条直线实际上就计算出来一系列与该线靠近的象素。

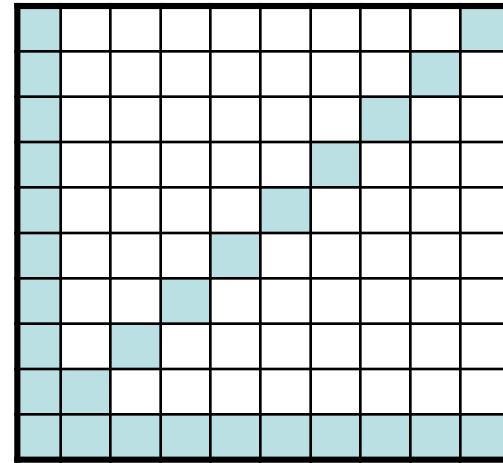


A

B

C

D



A

B

光栅化的直线

# 数值微分法 ( D D A )

假定直线的起点、终点分别为:  $(x_0, y_0)$ ,  
 $(x_1, y_1)$ , 且都为整数。

# 数值微分(DDA)法

- 基本思想

已知过端点  $P_0(x_0, y_0)$ ,  $P_1(x_1, y_1)$  的直线段  $L$

$$y = kx + b$$

直线斜率为

$$k = \frac{y_1 - y_0}{x_1 - x_0}$$

令  $x = x_0 \rightarrow x_1; x = x + stepx$

$$y = kx + b$$

$$\therefore (x, round(y))$$

这种方法直观，但效率太低，因为每一步需要一次浮点乘法和一次舍入运算。

# 数值微分(DDA)法

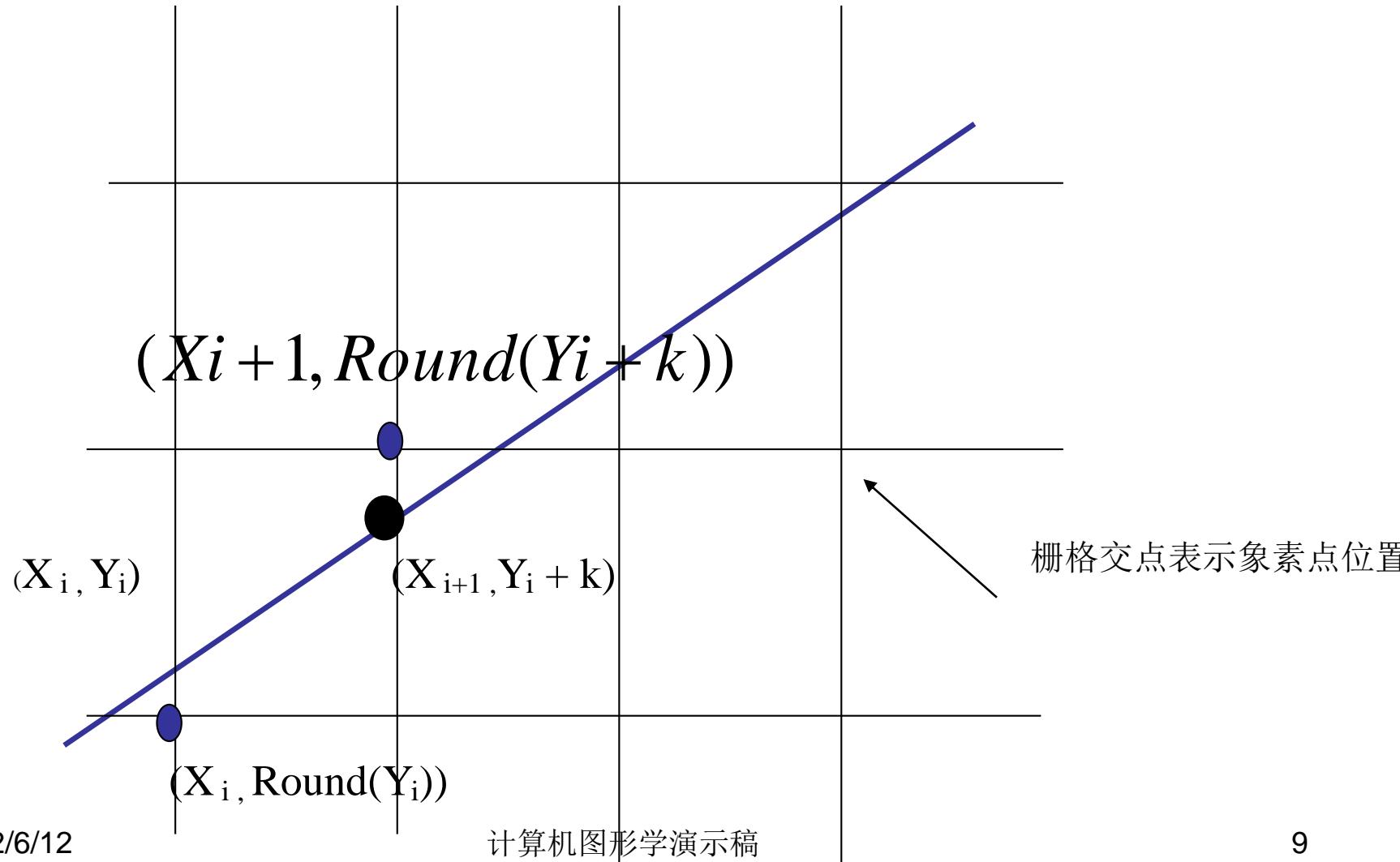
$$\text{计算 } y_{i+1} = kx_{i+1} + b$$

$$= kx_i + b + k\Delta x$$

$$= y_i + k\Delta x$$

$$\text{当 } \Delta x = 1; \quad y_{i+1} = y_i + k$$

- 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  $|k| \leq 1$  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



# 数值微分(DDA)法

- 增量算法：在一个迭代算法中，如果每一步的x、y值是用前一步的值加上一个增量来获得，则称为增量算法。
- DDA算法就是一个增量算法。

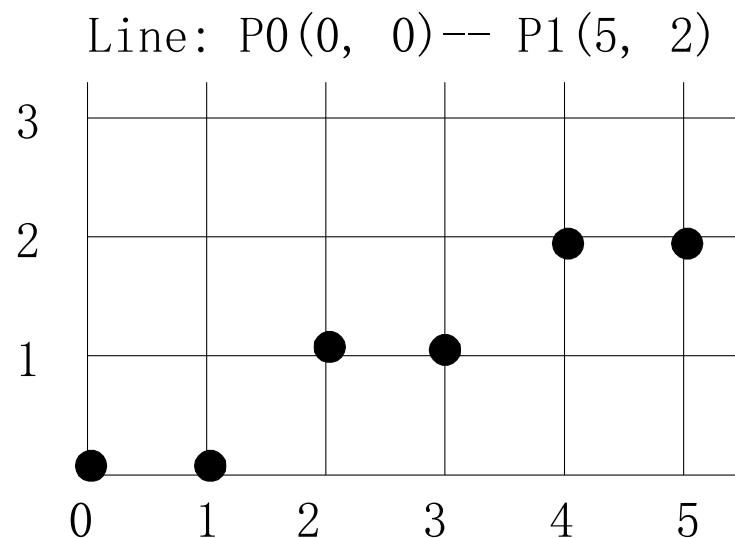
# 数值微分(DDA)法

```
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;
    }
}
```

# 数值微分(DDA)法

- 例：画直线段 $P_0(0,0)$ -- $P_1(5,2)$

x	$\text{int}(y+0.5)$	$y+0.5$
0	0	$0+0.5$
1	0	$0.4+0.5$
2	1	$0.8+0.5$
3	1	$1.2+0.5$
4	2	$1.6+0.5$
5	2	$2.0+0.5$

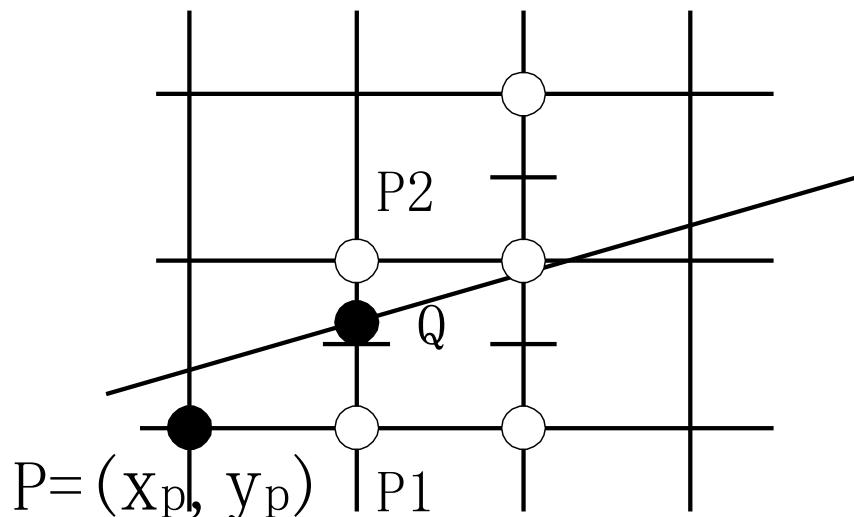


# 数值微分(DDA)法

- 缺点： 在此算法中，y、k必须是float，且每一步都必须对y进行舍入取整，不利于硬件实现。

# 中点画线法

- 原理:



假定直线斜率 $0 < K < 1$ , 且已确定点亮像素点 $P (X_p, Y_p)$ , 则下一个与直线最接近的像素只能是 $P_1$ 点或 $P_2$ 点。设 $M$ 为中点,  $Q$ 为交点

现需确定下一个点亮的像素  
◦

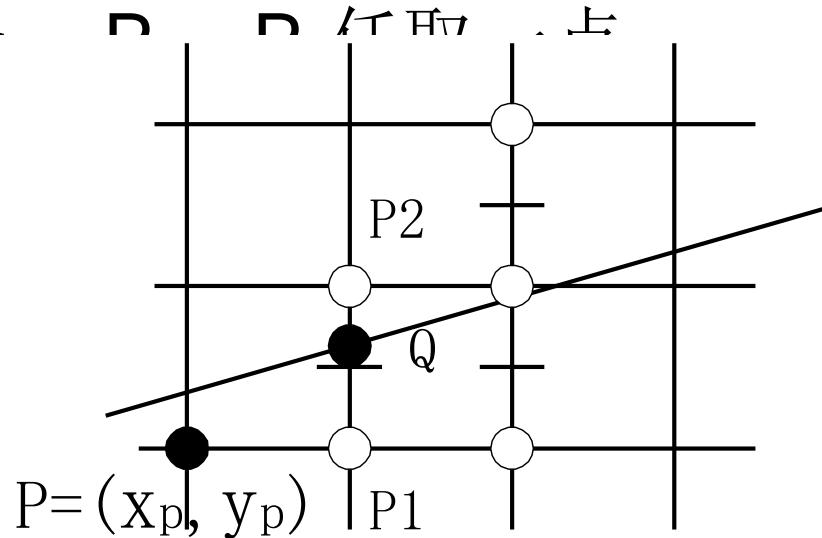






# 中点画线法

- 当M在Q的下方-> P<sub>2</sub>离直线更近更近->取P<sub>2</sub>。
- M在Q的上方-> P<sub>1</sub>离直线更近更近->取P<sub>1</sub>
- M与Q重合



- 问题：如何确定每次的步长？

# 中点画线法

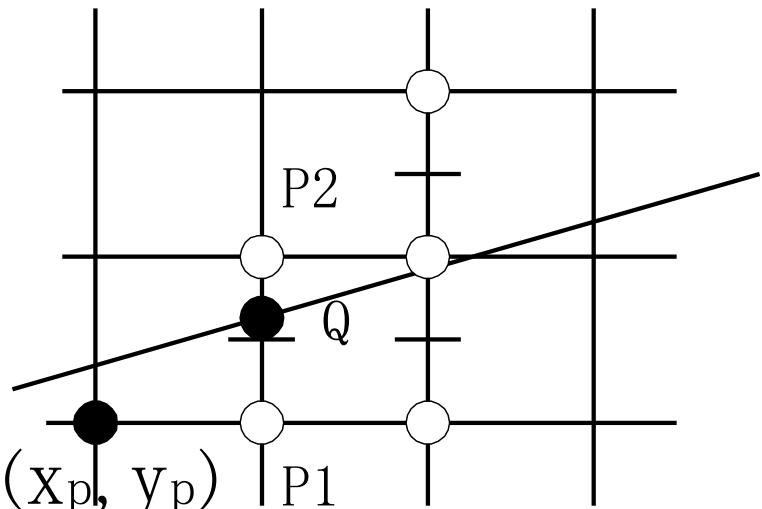
假设直线方程为:  $ax+by+c=0$

其中  $a=y_0-y_1$ ,  $b=x_1-x_0$ ,  $c=x_0y_1-x_1y_0$

由常识知:

$$\begin{cases} F(x, y) = 0 & \text{点在直线上面} \\ F(x, y) > 0 & \text{点在直线上方} \\ F(x, y) < 0 & \text{点在直线下方} \end{cases}$$

只需把  $M$  代入  $F(x, y)$ , 并检查它的符号。



# 中点画线法

构造判别式：

$$d = F(M) = F(x_p + 1, y_p + 0.5) \\ = a(x_p + 1) + b(y_p + 0.5) + c$$

当  $d < 0$ , M 在直线(Q点)下

方, 取右上方  $P_2$ ;

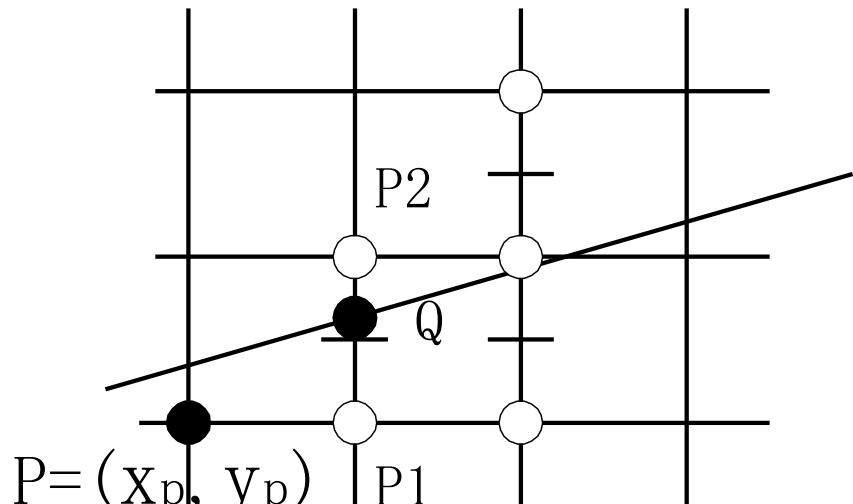
当  $d > 0$ , M 在直线(Q点)上

方, 取右方  $P_1$ ;

当  $d = 0$ , 选  $P_1$  或  $P_2$  均可,

约定取  $P_1$ ;

能否采用增量算法呢?



# 中点画线法

若  $d \geq 0 \rightarrow M$  在直线上方  $\rightarrow$  取  $P_1$ ;

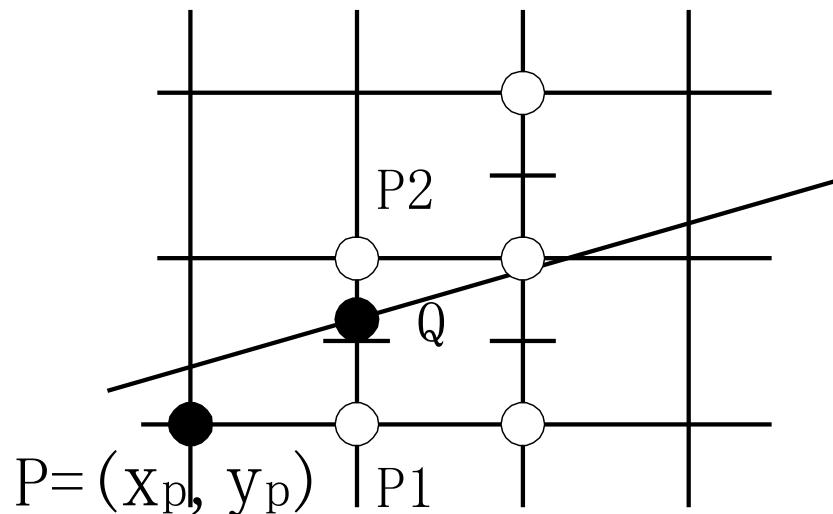
此时再下一个象素的判别式为

$$d_1 = F(x_p + 2,$$

$$y_p + 0.5) = a(x_p + 2) + b(y_p + 0.5) + c$$

$$= a(x_p + 1) + b(y_p + 0.5) + c + a = d + a;$$

增量为  $a$

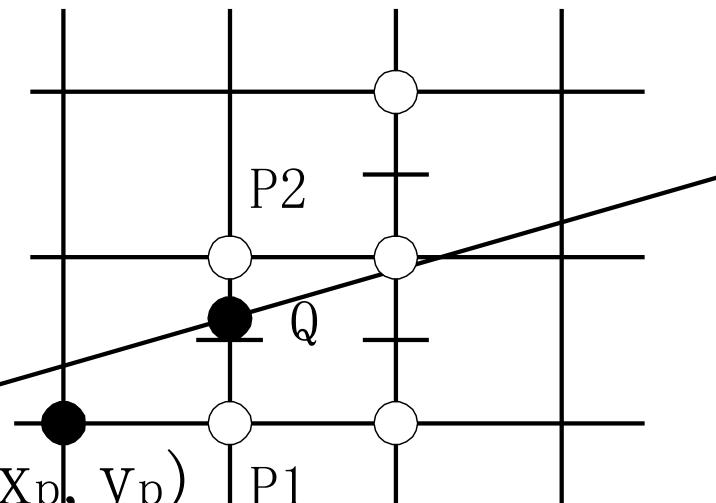


# 中点画线法

- 若 $d < 0 \rightarrow M$ 在直线下方 $\rightarrow$ 取 $P_2$ ;
- 此时再下一个象素的判别式为

$$\begin{aligned}d_2 &= F(x_p + 2, \\y_p + 1.5) = a(x_p + 2) + b(y_p + 1.5) + c \\&= a(x_p + 1) + b(y_p + 0.5) + c + a + b = d + a + b\end{aligned}$$

; 增量为 $a + b$



# 中点画线法

- 画线从 $(x_0, y_0)$ 开始,  $d$ 的初值

$$\begin{aligned}d_0 &= F(x_0+1, y_0+0.5) = a(x_0 + 1) + b(y_0 + 0.5) + c \\&= F(x_0, y_0) + a + 0.5b = a + 0.5b\end{aligned}$$

由于只用 $d$  的符号作判断, 为了只包含整数运算,  
可以用 $2d$ 代替 $d$ 来摆脱小数, 提高效率。

# 中点画线法

```
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 */
```

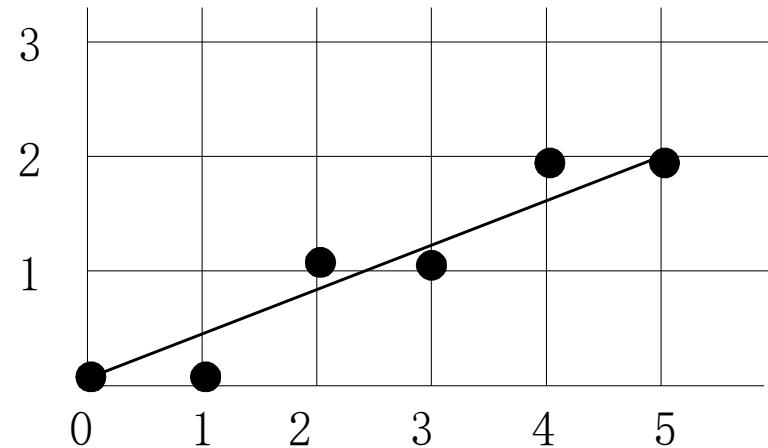
# 中点画线法

- 例：用中点画线法  $P_0(0,0)$   $P_1(5,2)$

$$a=y_0-y_1=-2 \quad b=x_1-x_0=5$$

$$d_0=2a+b=1 \quad d_1=2a=-4 \quad d_2=2(a+b)=6$$

i	$x_i$	$y_i$	d
1	0	0	1
2	1	0	-3
3	2	1	3
4	3	1	-1
5	4	2	5



# Bresenham画线算法

在直线生成的算法中Bresenham算法是最有效的算法之一。令  $k = \Delta y / \Delta x$ , 就  $0 \leq k \leq 1$  的情况来说明Bresenham算法。由DDA算法可知:

$$y_{i+1} = y_i + k \quad (1)$$

由于  $k$  不一定是整数, 由此式求出的  $y_i$  也不一定是整数, 因此要用坐标为  $(x_i, y_{ir})$  的象素来表示直线上的点, 其中  $y_{ir}$  表示最靠近  $y_i$  的整数。

# Bresenham画线算法

设图中 $x_i$ 列上已用 $(x_{i_r}, y_{i_r})$ 作为表示直线的点，又设B点是直线上的点，其坐标为 $(x_{i+1}, y_{i+1})$ ，显然下一个表示直线的点 $(x_{i+1}, y_{i+1,r})$ 只能从图中的C或者D点中去选。设A为CD边的中点。若B在A点上面则应取D点作为 $(x_{i+1}, y_{i+1,r})$ ，否则应取C点。

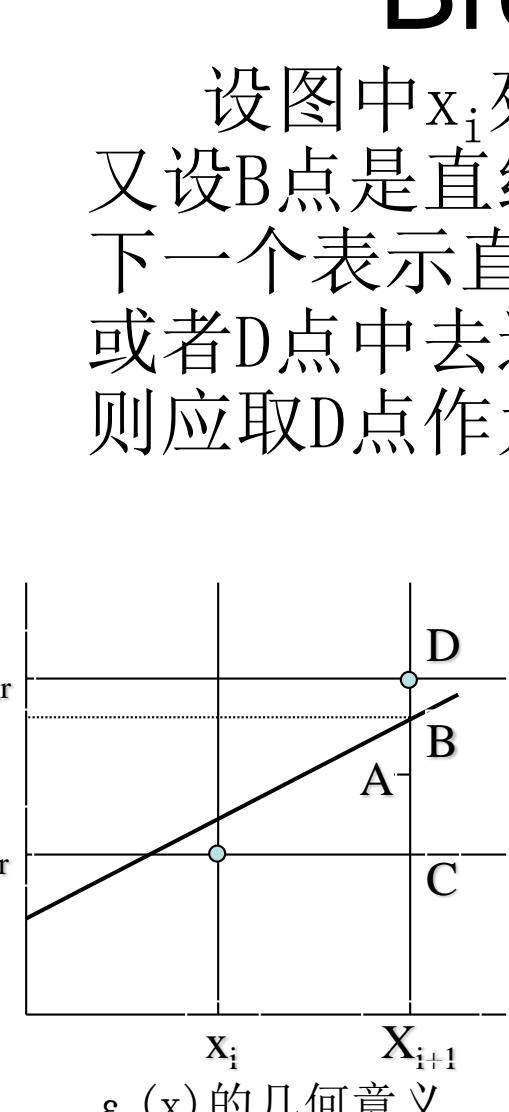
为能确定B在A点上面或下面，令

$$\varepsilon(x_{i+1}) = y_{i+1} - y_{i_r} - 0.5 \quad (2)$$

若B在A的下面，则有 $\varepsilon(x_{i+1}) < 0$ ，反之，则 $\varepsilon(x_{i+1}) > 0$ 。由图可知

$$(3) \quad y_{i+1,r} = y_{i_r} + 1, \text{ 若 } \varepsilon(x_{i+1}) \geq 0$$

$$y_{i+1,r} = y_{i_r}, \quad \text{若 } \varepsilon(x_{i+1}) \leq 0$$



$\varepsilon(x)$ 的几何意义

# Bresenham画线算法

由式 (2) 和式 (3) 可得到

$$\begin{aligned}\varepsilon(x_{i+2}) &= y_{i+2} - y_{i+1,r} - 0.5 \\ &= y_{i+1} + k - y_{i+1,r} - 0.5\end{aligned}\quad (4)$$

$$y_{i+1} - y_{ir} - 0.5 + k - 1, \text{ 当 } \varepsilon(x_{i+1}) \geq 0$$

$$y_{i+1} - y_{ir} - 0.5 + k, \quad \text{当 } \varepsilon(x_{i+1}) \leq 0$$

$$\varepsilon(x_{i+2}) = \varepsilon(x_{i+1}) + k - 1, \text{ 当 } \varepsilon(x_{i+1}) \geq 0$$

$$\varepsilon(x_{i+2}) = \varepsilon(x_{i+1}) + k, \quad \text{当 } \varepsilon(x_{i+1}) \leq 0$$

# Bresenham画线算法

程序如下： BresenhamLine(x0,y0,x1,y1,color)

```
int x0,y0,x1,y1,color;
{
    int x,y,dx,dy;
    float k,e; dx = x1-x0;
    dy = y1-y0;
    k = dy/dx;
    e = -0.5; x=x0; y=y0;
    for( i=0; i<=dx; i++)
    {
        drawpixel(x,y,color);
        x++;
        e=e+k;
        if(e >= 0)
            y++;
        e = e - 1;
    }
}
```

将 $e$ 乘以 $2\Delta x$ 记为 $E = 2\Delta x e$ , 则 $E$ 同 $e$ 有相同的符号, 取代 $e$ 判断 $E$ 的符号确定象素点的过程仍然正确。此时上述算法中各误差项的表示式做如下变动:

初始误差项:  $E_0 = 2\Delta x e_0$

$$= -\Delta x;$$

积累误差  $e_{k+1} = e_k + m$ 修改为:

$$\begin{aligned}E_{k+1} &= 2\Delta x e_{k+1} \\&= 2\Delta x(e_k + \Delta y/\Delta x) \\&= 2\Delta x e_k + 2\Delta y \\&= E_k + 2\Delta y;\end{aligned}$$

如果选取上面的象素点, 积累误差还要减去1, 修改为:

$$\begin{aligned}E_{k+1} &= 2\Delta x (e_{k+1} - 1) \\&= E_{k+1} - 2\Delta x\end{aligned}$$

程序如下： BresenhamLine(x0,y0,x1,y1,color)

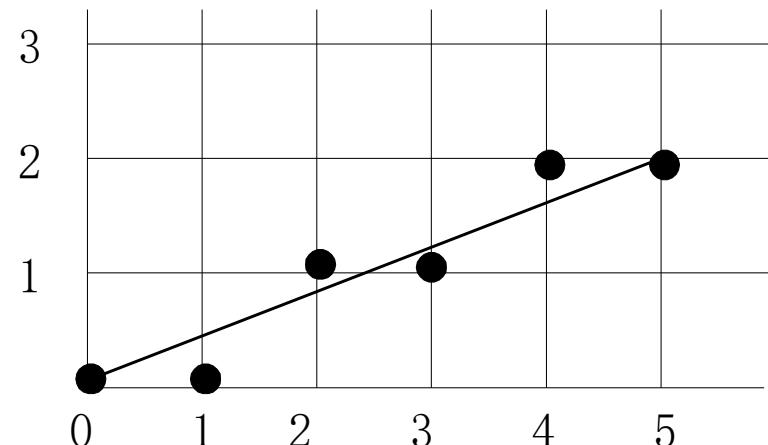
```
int x0,y0,x1,y1,color;
{
    int x,y,dx,dy;
    float k;
    int e;
    dx = x1-x0;
    dy = y1-y0;
    k = dy/dx;
    e = -dx;
    x=x0; y=y0;
    for( i=0; i<=dx; i++){
        drawpixel(x,y,color);
        x++;
        e=e+2*dy;
        if(e >= 0)
            {
                y++;  
                e = e - 2*dx;
            }
    }
}
```

# Bresenham画线法

- 例：用bresenhanm画线法  $P_0(0,0)$   $P_1(5,2)$

$$dy=y_1-y_0=2 \quad dx=x_1-x_0=5 \quad e_0=-dx=-5$$

i	$x_i$	$y_i$	e
1	0	0	-5
2	1	0	-1
3	2	1	3
4	3	1	-3
5	4	2	1



## 3.2 二次曲线绘制算法

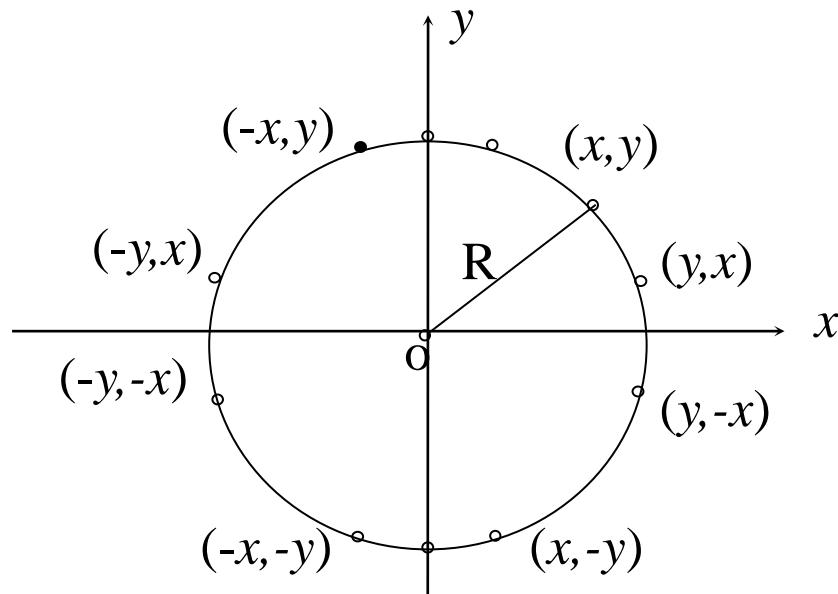
二次曲线是指那些能用二次函数

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

来表示的曲线，包括圆、椭圆、抛物线、双曲线等。由于图形输出设备的基本动作是显示像素点或者是画以步长为单位的直线段，所以，从图形显示器和绘图机上输出的图形，一般除了水平线和垂直线以外，其他的各种线段，包括直线和曲线，都是由很多的点和短直线构成的锯齿形线条组成的。也就是说，需要把曲线离散化，把它们分割成很多短直线段，用这些短直线段组成的折线来逼近曲线。

# 圆弧的扫描转换

- 圆的八对称性
  - 只考虑第二个八分圆
- 假设圆心在原点  
 $x^2+y^2=R^2$



# 圆弧的扫描转换

- 两种直接离散生成方法
  - 离散点
    - 开方运算
  - 离散角度
    - 三角函数运算
- 缺点:
  - 计算量大
  - 所画像素位置间的间距不一致

### 3.2.1 圆的参数方程生成算法

圆是图形中经常使用的元素，圆被定义为所有离一中心位置  $(x_c, y_c)$  距离为给定值R的点集，其函数方程为：

$$(x - x_c)^2 + (y - y_c)^2 = R^2$$

利用这个方程，我们可以沿X轴从 $x_c - R$  到 $x_c + R$  以单位步长计算对应的y值来得到圆周上每点的位置，但这并非是生成圆的好方法。

## 圆的方程绘制方法C++实现

```
void 圆的方程绘制(HDC hdc)
{
    double xc=300,yc=200,R=150;
    double x,y;

    y=yc;

    for(x=xc-R;x<=450;x++)
    {
        y=sqrt(R*R-(x-xc)*(x-xc))+yc;
        SetPixel(hdc,x,y,RGB(0,0,0));

        y=-sqrt(R*R-(x-xc)*(x-xc))+yc;
        SetPixel(hdc,x,y,RGB(0,0,0));
    }

    Sleep(50);
}
```

## 圆的方程绘制方法演示

这一方法包括乘法和平方根运算，计算量较大，所画像素位置间的间距也是不一致的。下面介绍两种生成速度较快的算法。先介绍圆的参数生成方法。

假定圆心在 $(x_c, y_c)$ 点，将圆用参数方程表示：

$$\begin{aligned} X &= x_c + R \cos \theta \\ Y &= y_c + R \sin \theta \end{aligned} \quad 0 \leq \theta \leq 2\pi$$

将 $\theta$ 离散化

$$\begin{aligned} x_k &= x_c + R \cos(2\pi \frac{k}{n}) \\ y_k &= y_c + R \sin(2\pi \frac{k}{n}) \end{aligned} \quad 0 \leq k \leq n$$

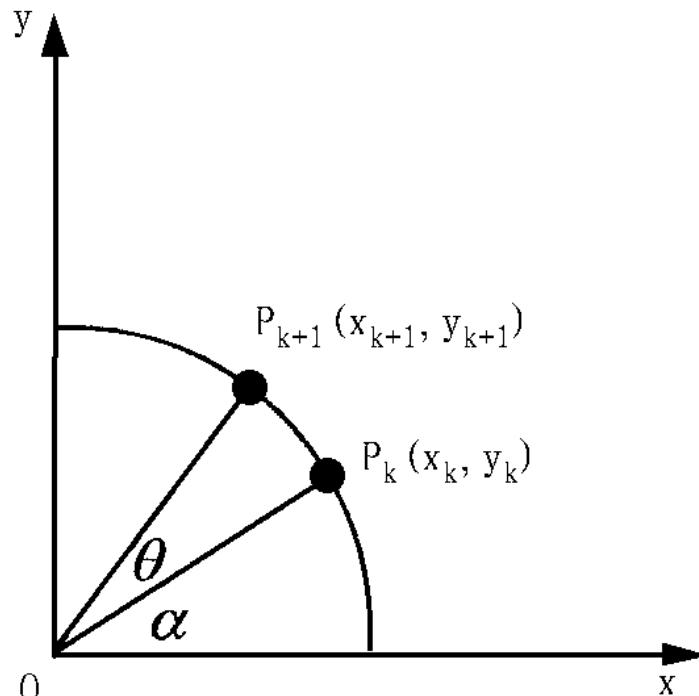
使用上述离散化方程，可以得到如下算法：

```
begin
    for k = 0 to n
        x = x_c + Rcos(2π·k/n)
        y = y_c + Rsin(2π·k/n)
        putpixel(x, y)
    next k
endfor
end
```

使用上述方法可沿圆周等距点绘制出圆来。算法中， $n$ 取的值越大，计算的点越多，但执行时间越长。此算法的缺点是含有三角函数，计算量大。

为了避免三角函数运算，考虑下图，如果已经计算出圆上一点  $(x_k, y_k)$ ，则增加一个角度  $\theta$  后，下一点  $(x_{k+1}, y_{k+1})$  的坐标值可以用上一个点表示出。

$$\begin{aligned}
 x_{k+1} &= R\cos(\theta + \alpha) \\
 &= R (\cos \alpha \cos \theta - \sin \alpha \sin \theta) \\
 &= R \cdot \cos \alpha \cos \theta - R \sin \alpha \sin \theta \\
 &= x_k \cos \theta - y_k \sin \theta \\
 y_{k+1} &= R\sin(\theta + \alpha) \\
 &= R(\sin \alpha \cos \theta + \cos \alpha \sin \theta) \\
 &= R \cos \alpha \cos \theta - R \sin \alpha \sin \theta \\
 &= y_k \cos \theta + x_k \sin \theta
 \end{aligned}$$



当 $\theta$  足够小时有:

$$\cos \theta \approx 1$$

$$\sin \theta \approx \theta$$

如是方程式可以改写为:

$$x_{k+1} = x_k - y_k \theta$$

$$y_{k+1} = y_k + x_k \theta$$

习惯上用 $\varepsilon$  表示一个较小的量, 用它替代 $\theta$ , 式子改写为:

$$x_{k+1} = x_k - y_k \varepsilon$$

$$y_{k+1} = y_k + x_k \varepsilon$$

利用前式，圆的参数方程生成算法可以描述如下：

```
begin
    x = xc + R
    y = yc
    θ= 0
    for θ≤2π
        putpixel(x, y)
        x= x - yε
        y= y + xε
        θ= θ + ε
    endfor
end
```

此圆的生成算法中仍包含浮点数和乘法运算，影响速度。

上述算法中， $\varepsilon$ 选取的值越小，计算的点越多，但执行时间越长。为了在光栅系统上得到连续的边界，可选取 $\varepsilon = 1 / R$ ，这样绘制的象素位置大约为一个单位间隔。此算法也被称为**DDA**圆的生成算法。

## 圆的参数绘制方法C++实现

```
void ApplicationProceesing(HDC hdc)
{
```

```
    double xc=300,yc=200,R=150;
    double x,y,theta,delta;
```

```
    x=xc+R;
```

```
    y=yc;
```

```
    delta=1.0/R;
```

```
    for(theta=0;theta<=2*3.1416;theta=theta+delta)
    {
```

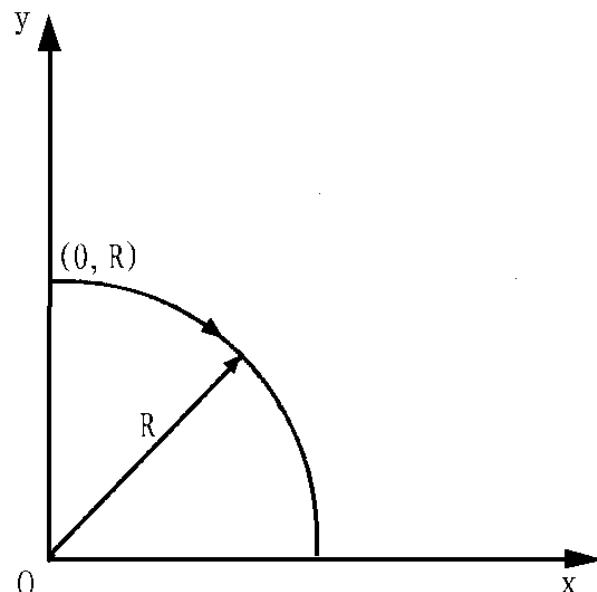
```
        SetPixel(hdc,x,y,RGB(0,0,0));
        x=x-(y-yc)*delta;
```

```
        y=y+(x-xc)*delta;
```

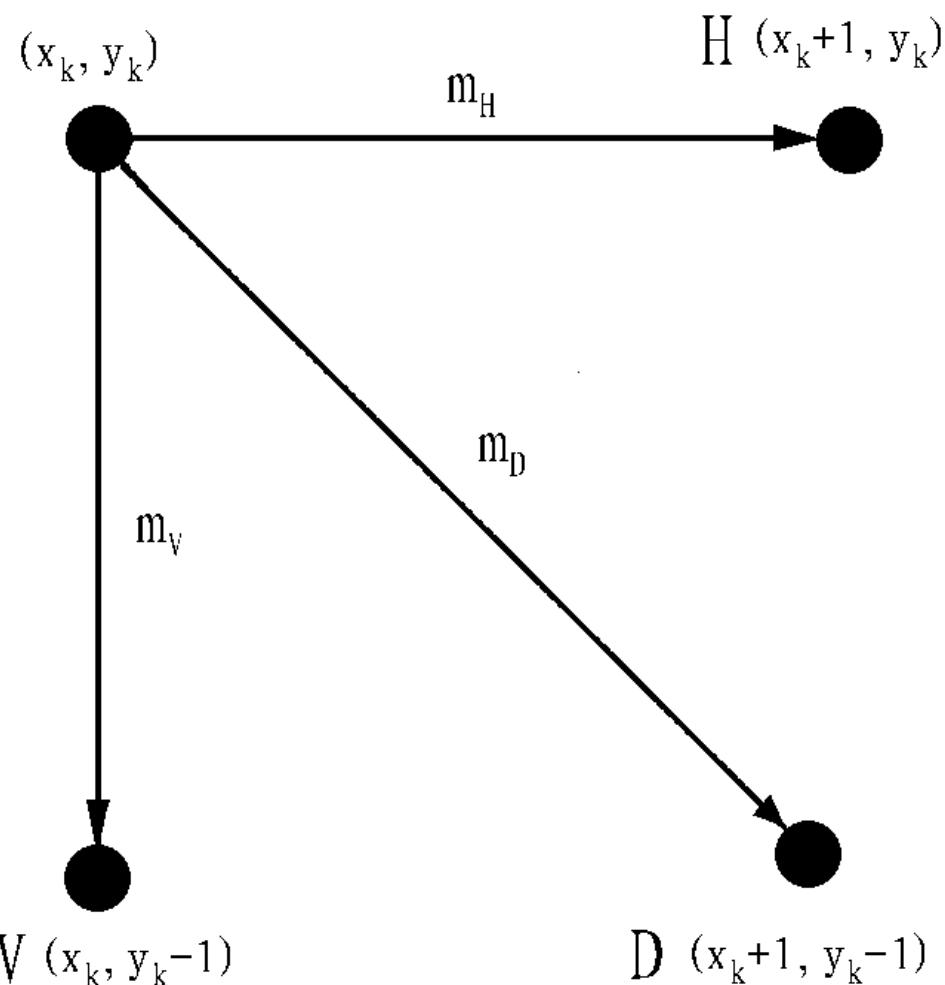
## 圆的参数绘制方法演示

### 3.2.3 圆的Bresenham生成算法

圆的生成更有效的算法是Bresenham画圆算法。为了推导Bresenham圆的生成算法，考虑以坐标原点为圆心第一象限内的四分之一圆，如下图所示，可以看到，如果算法以点( $x = 0, y = R$ )为起点按顺时针方向生成圆时，在第一象限内 $y$ 是 $x$ 的单调递减函数。

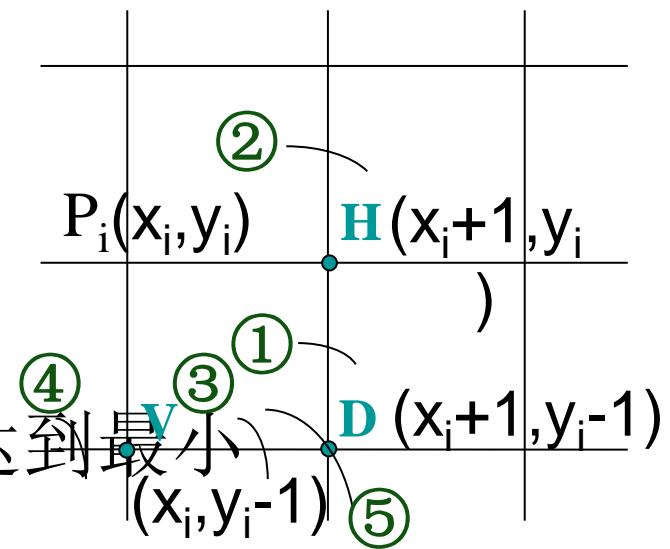


从圆上任意一点出发，按顺时针方向生成圆时，为了最佳逼近该圆，对于下一象素的取法只有三种可能的选择，即右方象素、右下角象素和下方象素，并分别记为H，D和V，如右图所示。要在这三者中决定一象素使其与圆的距离平方达到最小。



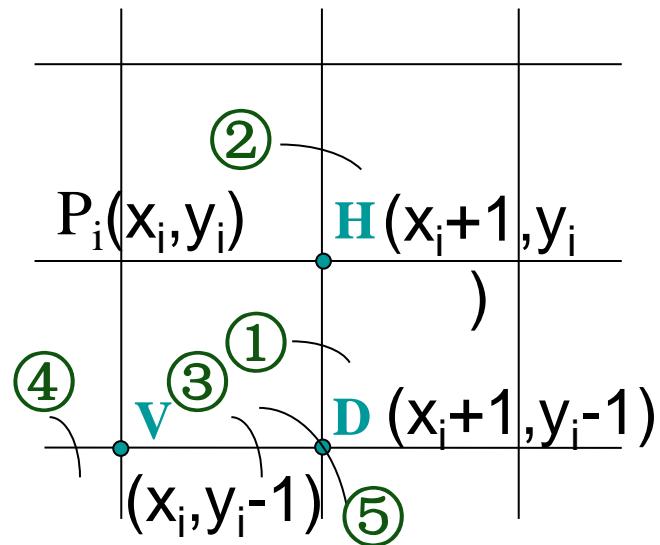
# Bresenham画圆算法 (1/7)

- 顺时针画第一四分圆，下一步选择哪个点？
- 基本思想：
  - 通过比较像素与圆的距离平方来避免开方运算
- 下一像素有3种可能的选择
  - $m_H = |(x_i+1)^2 + y_i^2 - R^2|$
  - $m_D = |(x_i+1)^2 + (y_i-1)^2 - R^2|$
  - $m_V = |x_i^2 + (y_i-1)^2 - R^2|$
- 选择像素的原则
  - 使其与实际圆弧的距离平方达到最小



# Bresenham画圆算法 (2/7)

- 圆弧与点 $(x_i, y_i)$ 附近光栅网格的相交关系有5种
- 右下角像素 $D(x_i, y_i)$ 与实际圆弧的近似程度
  - $\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$
  - 当 $\Delta_i < 0$ 时， $D$ 在圆内，①②
  - 当 $\Delta_i > 0$ 时， $D$ 在圆外，③④
  - 当 $\Delta_i = 0$ 时， $D$ 在圆上，⑤



# Bresenham画圆算法 (3/7)

- 当 $\Delta i < 0$ 时， D在圆内， ①②
- 情形①， 选 $m_H$ ，  $m_D$  中最小者
- $d = m_H - m_D$

$$= |(x_i+1)^2 + y_i^2 - R^2| - |(x_i+1)^2 + (y_i-1)^2 - R^2|$$

$$= (x_i+1)^2 + y_i^2 - R^2 + (x_i+1)^2 + (y_i-1)^2 - R^2$$

$$= 2(\Delta i + y_i) - 1$$

- 若 $d < 0$ ， 则选H

- 若 $d > 0$ ， 则选D

- 若 $d = 0$ ， 则选H



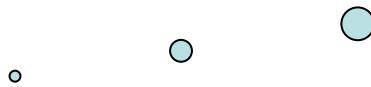
情形②也  
适用

# Bresenham画圆算法 (4/7)

- 当 $\Delta i > 0$ 时， D在圆外， ③④
- 情形③， 选 $m_v$ ,  $m_D$  中最小者
- $d' = m_D - m_V$

$$\begin{aligned}d' &= |(x_i+1)^2 + (y_i-1)^2 - R^2| - |x_i^2 + (y_i-1)^2 - R^2| \\&= (x_i+1)^2 + (y_i-1)^2 - R^2 + x_i^2 + (y_i-1)^2 - R^2 \\&= 2(x_i+1) - 2\end{aligned}$$

- 若 $d' < 0$ , 则选D
- 若 $d' > 0$ , 则选V
- 若 $d' = 0$ , 则选D



情形④也  
适用

# Bresenham画圆算法 (5/7)

- 当 $\Delta i=0$ 时， D在圆上， ⑤
- 按d判别， 有 $d>0$ ， 应选D
- 按d'判别， 有 $d'<0$ ， 应选D

# Bresenham画圆算法 (6/7)

- 当 $\Delta i < 0$ 时，
  - 若 $d \leq 0$ , 选H
  - 若 $d > 0$ , 选D
- 当 $\Delta i > 0$ 时，
  - 若 $d' \leq 0$ , 选D
  - 若 $d' > 0$ , 选V
- 当 $\Delta i = 0$ 时, 选D

下面给出Bresenbam画圆算法描述（第一象限四分之一圆）：

begin

$x_k = 0$

$y_k = R$

$\Delta_k = 2(1 - R) \quad (\Delta_k = (x_k + 1) + (y_k - 1) - R = 1 + (R - 1) = 2(1 - R))$

    Limit = 0

    1 putpixel( $x_k, y_k$ )

        if  $y_k \leqslant \text{Limit}$  then 4

            if  $\Delta_k < 0$  then 2

            if  $\Delta_k > 0$  then 3

            if  $\Delta_k = 0$  then 20

    2  $\delta = 2\Delta_k + 2y_k - 1$

        if  $\delta \leqslant 0$  then 10

        if  $\delta > 0$  then 20

    3  $\delta = 2\Delta_k - 2x_k - 1$

        if  $\delta \leqslant 0$  then 20

        if  $\delta > 0$  then 30

    10  $x_k = x_k + 1$

$\Delta_k = \Delta_k + 2x_k + 1$

        goto 1

    20  $x_k = x_k + 1$

$y_k = y_k - 1$

$\Delta_k = \Delta_k + 2x_k - 2y_k + 2$

        goto 1

    30  $y_k = y_k - 1$

$\Delta_k = \Delta_k - 2y_k + 1$

        goto 1

    4 finish

end

如果圆心不在原点， 设圆心坐标为  $(x_c, y_c)$ ， 只需  
要改变初始化值：

$$x_k = x_c$$

$$y_k = y_c + R$$

$$\Delta_k = (x_c + 1)^2 + (y_c + R - 1)^2 - R^2$$

$$= x_c^2 + 2x_c + 1 + y_c^2 + 2y_c(R - 1) - 2R + 1$$

由圆的对称性， 很容易将上述方法转换到其它三个象限中。

## Bresenham圆的绘制算法例子：

绘制以坐标原点为圆心，半径8  
的圆在第一象限的部分。

$$x_0=0$$

$$y_0=8$$

$$R=8$$

$$\Delta_k = (x_k+1)^2 + (y_k-1)^2 - R^2$$

当  $\Delta_k < 0$  时

$$\delta_{HD} = 2(\Delta_k + y_k) - 1$$

$\delta_{HD} \leq 0$  时取 H  $(x_k+1, y_k)$  点

$$\Delta_k = \Delta_k + 2x_{k+1} + 1$$

$\delta_{HD} > 0$  时取 D  $(x_k+1, y_k-1)$  点

$$\Delta_k = \Delta_k + 2x_{k+1} - 2y_{k+1} + 2$$

当  $\Delta_k > 0$  时

$$\delta_{DV} = 2(\Delta_k - x_k) - 1$$

$\delta_{DV} \leq 0$  时取 D  $(x_k+1, y_k-1)$  点

$$\Delta_k = \Delta_k + 2x_{k+1} - 2y_{k+1} + 2$$

$\delta_{DV} > 0$  时取 V  $(x_k, y_k-1)$  点

$$\Delta_k = \Delta_k - 2y_{k+1} + 2$$

当  $\Delta_k = 0$  时取 D  $(x_k+1, y_k-1)$  点

$$\Delta_k = \Delta_k + 2x_{k+1} - 2y_{k+1} + 2$$

### 3.2.2 其它二次曲线的绘制

由于各种其它二次曲线都有类似于圆的参数方程，因此其生成算法可以仿照圆的参数方程DDA生成算法设计。例如椭圆的参数方程为

$$\begin{aligned}x &= x_c + R_x \cdot \cos \theta \\y &= y_c + R_y \cdot \sin \theta\end{aligned}\quad 0 \leq \theta \leq 2\pi$$

只有 $R_x$ 和 $R_y$ 两个半轴常数不同于圆的单一半径 $R$ ，但它们并不影响算法设计。几乎可以不加改变地使用圆的DDA算法过程。

## 区域填充

一个区域是指一组相邻而又相连的象素，且具有同样的属性。根据边或轮廓线的描述，生成实区域的过程称为区域填充。

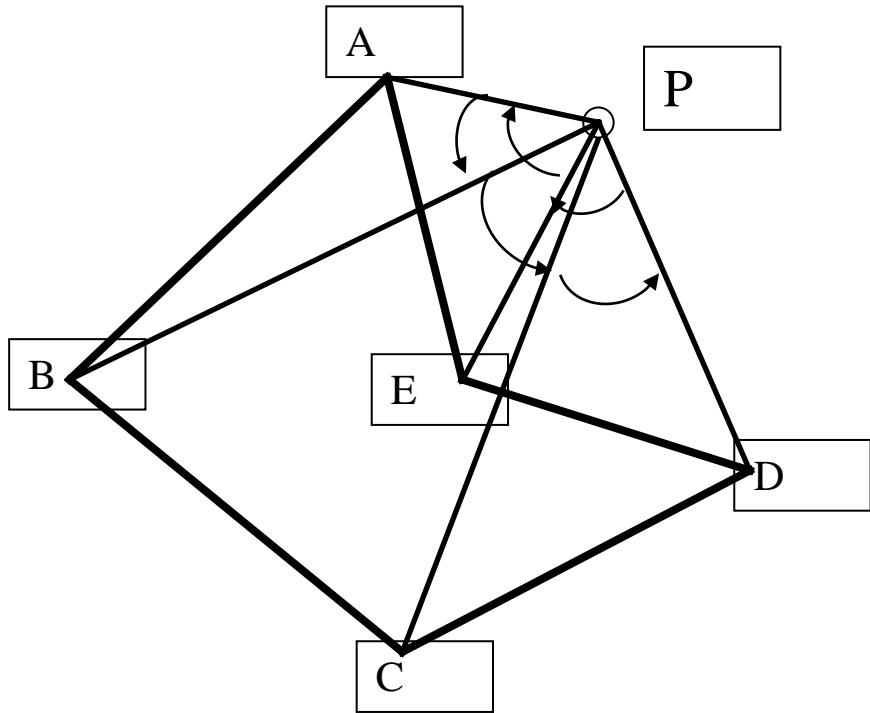
# 实区域填充算法

- 解决的主要问题是什么？
  - 确定待填充的象素，即检查光栅的每一像素是否位于多边形区域内
- 图案填充还有一个什么象素填什么颜色的问题
- 曲线围成的区域，可用多边形逼近

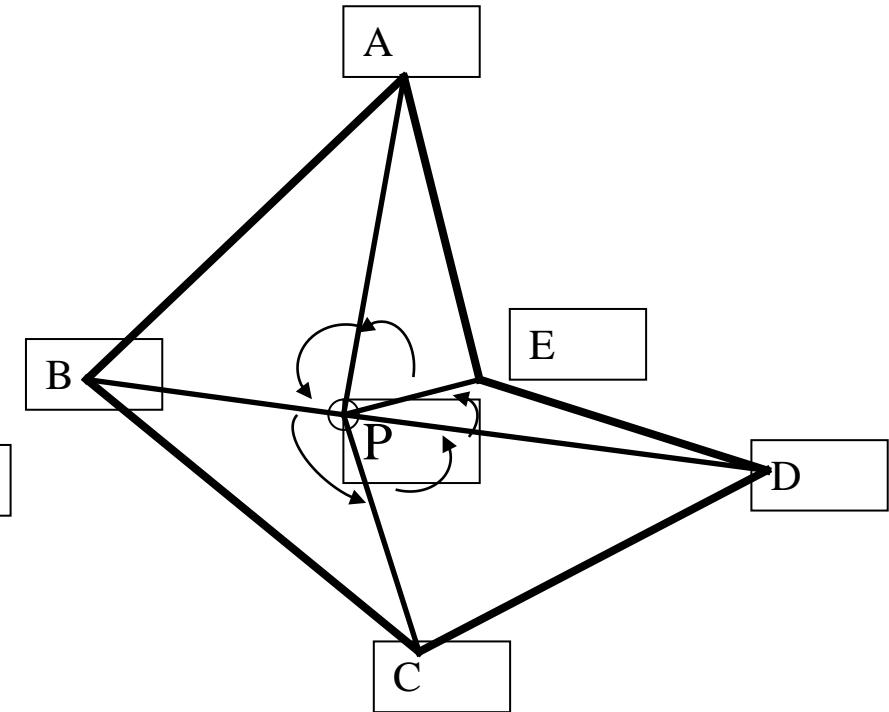
# 点在多边形内的包含性检验

- 检验夹角之和
- 射线法检验交点数

# 检验夹角之和



若夹角和为0，则点p在多边形外

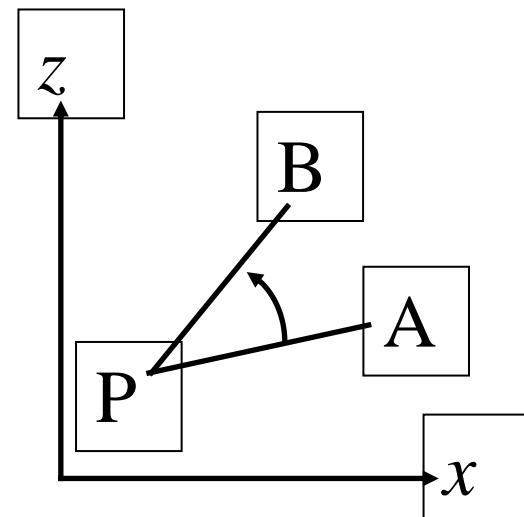
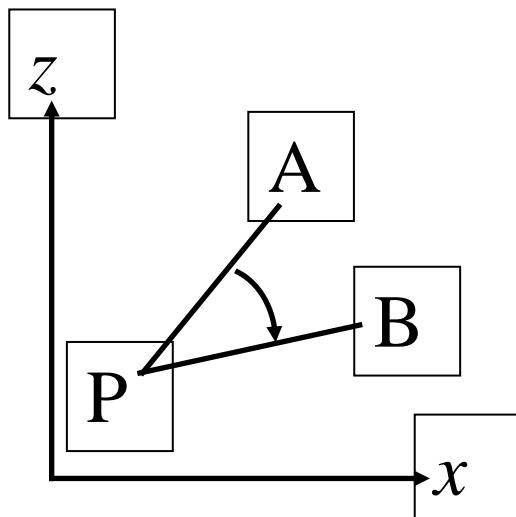


若夹角和为 $360^\circ$ ，则点p在多边形内

# 夹角如何计算？

- 大小：利用余弦定理
- 方向：令

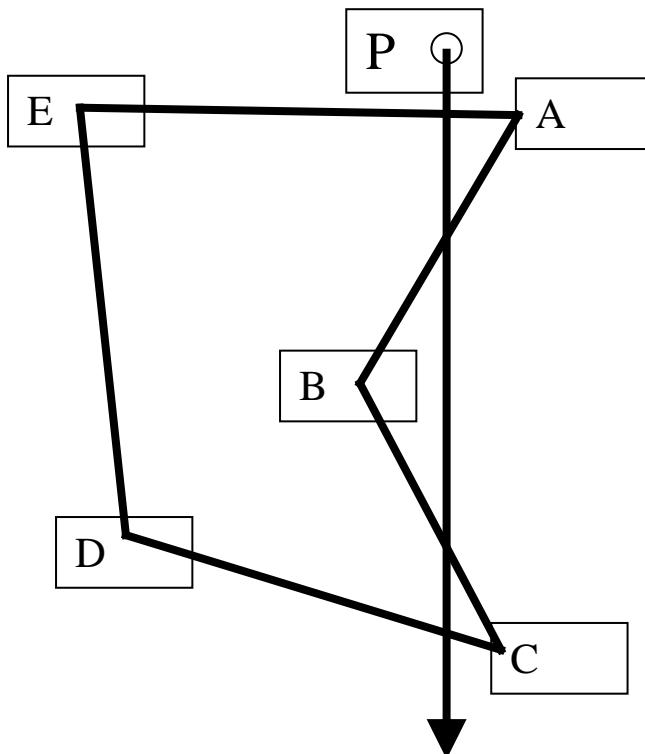
$$T = \begin{vmatrix} x_A - x_P & z_A - z_P \\ x_B - x_P & z_B - z_P \end{vmatrix} = (x_A - x_P)(z_B - z_P) - (x_B - x_P)(z_A - z_P)$$



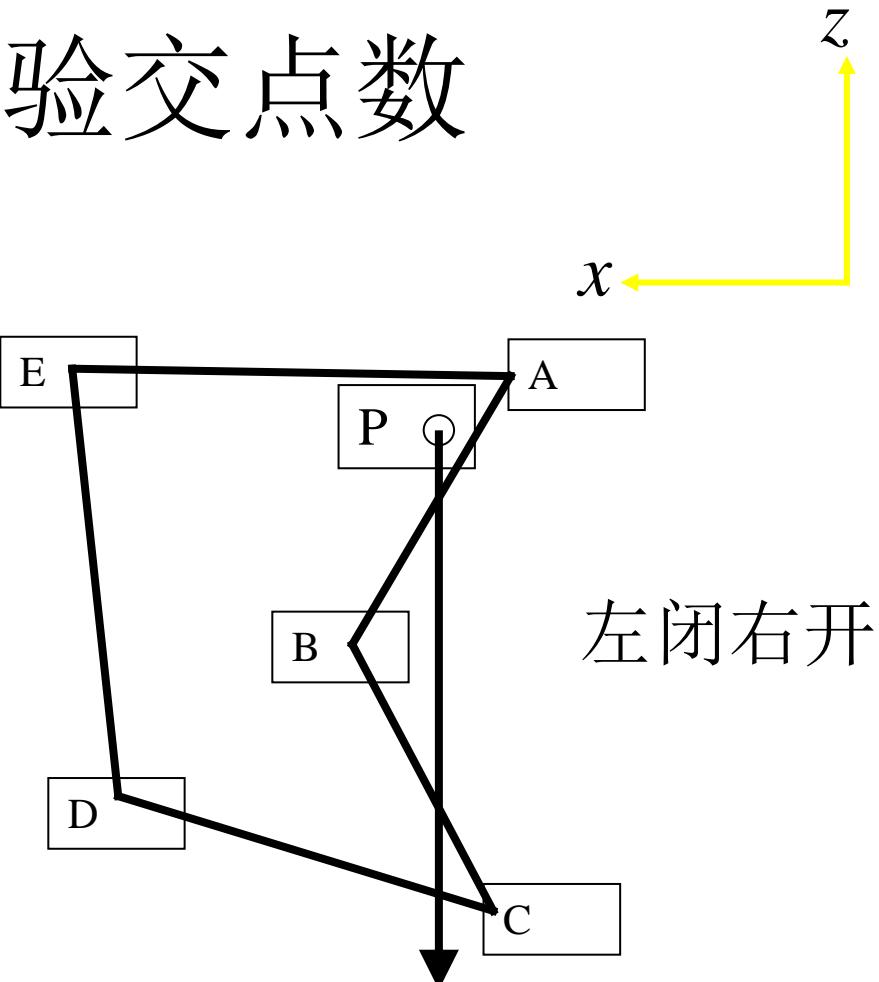
当  $T < 0$  时，  $AP$  斜率  $>$   $BP$  斜率，为顺时针角

当  $T > 0$  时，  $AP$  斜率  $<$   $BP$  斜率，为逆时针角

# 射线法检验交点数



交点数=偶数（包括0）  
点在多边形之外



左闭右开  
交点数=奇数  
点在多边形之内

# 实区域填充算法分类

- 扫描线填充算法
  - 扫描线顺序
- 种子填充算法
  - 内部一个点出发

种子填充算法首先假定封闭轮廓线内某点是已知的，然后算法开始搜索与种子点相邻且位于轮廓线内的点。种子填充算法只适用于光栅扫描设备。

扫描转换填充算法则是按扫描线的顺序确定某一点是否位于多边形或轮廓线范围之内。这些算法一般从轮廓线的顶部开始进行到它的底部。

区域填充的边界可以是直线也可以是曲线。

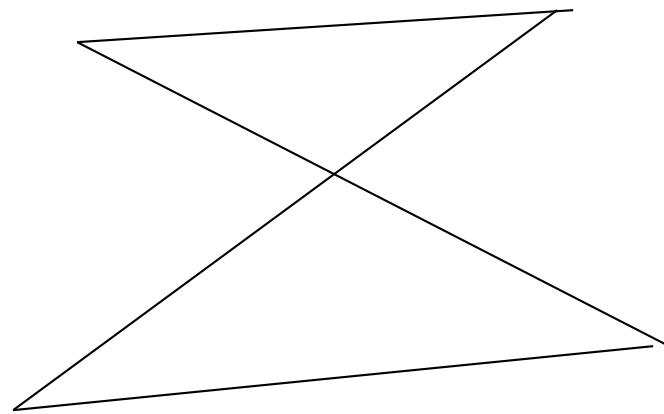
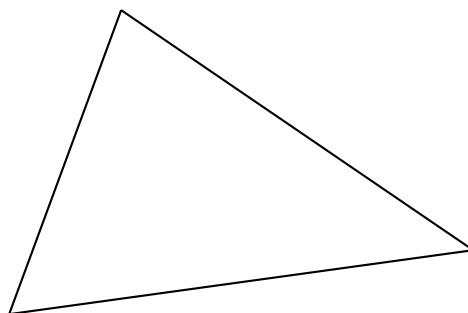
- 多边形的表示方法
  - 顶点表示
  - 点阵表示
- 顶点表示：用多边形顶点的序列来刻划多边形。直观、几何意义强、占内存少；不能直接用于面着色。
- 点阵表示：用位于多边形内的象素的集合来刻划多边形。失去了许多重要的几何信息；便于运用帧缓冲存储器表示图形，易于面着色。

# 多边形的扫描转换

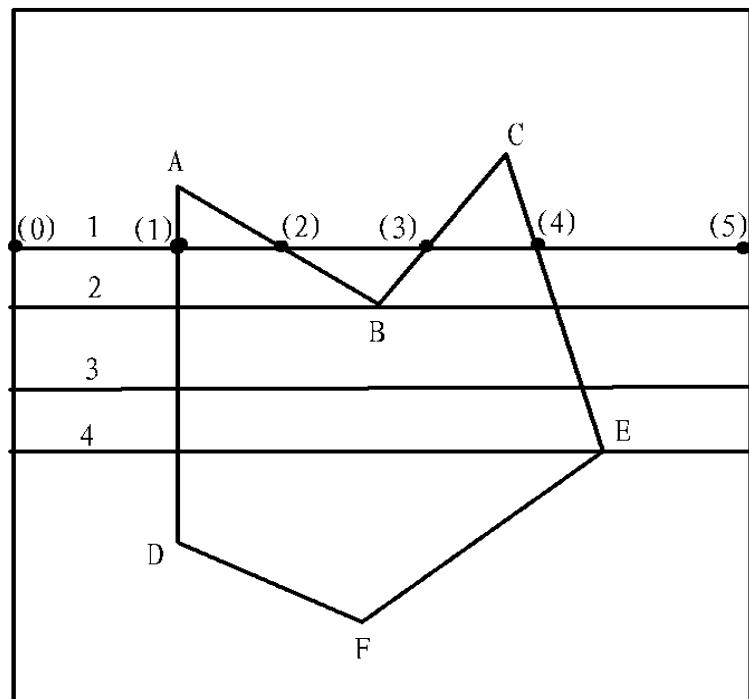
- **多边形的扫描转换：**把多边形的顶点表示转换为点阵表示，也就是从多边形的给定边界出发，求出位于其内部的各个象素，并给帧缓冲器内的各个对应元素设置相应的灰度和颜色，通常称这种转换为多边形的扫描转换。  
。
- **几种方法：**逐点判断法；扫描线算法；边填充法；栅栏填充法；边界标志法。

# 扫描线算法

- 扫描线算法
  - 目标：利用相邻像素之间的连贯性，提高算法效率
  - 处理对象：非自交多边形（边与边之间除了顶点外无其它交点）

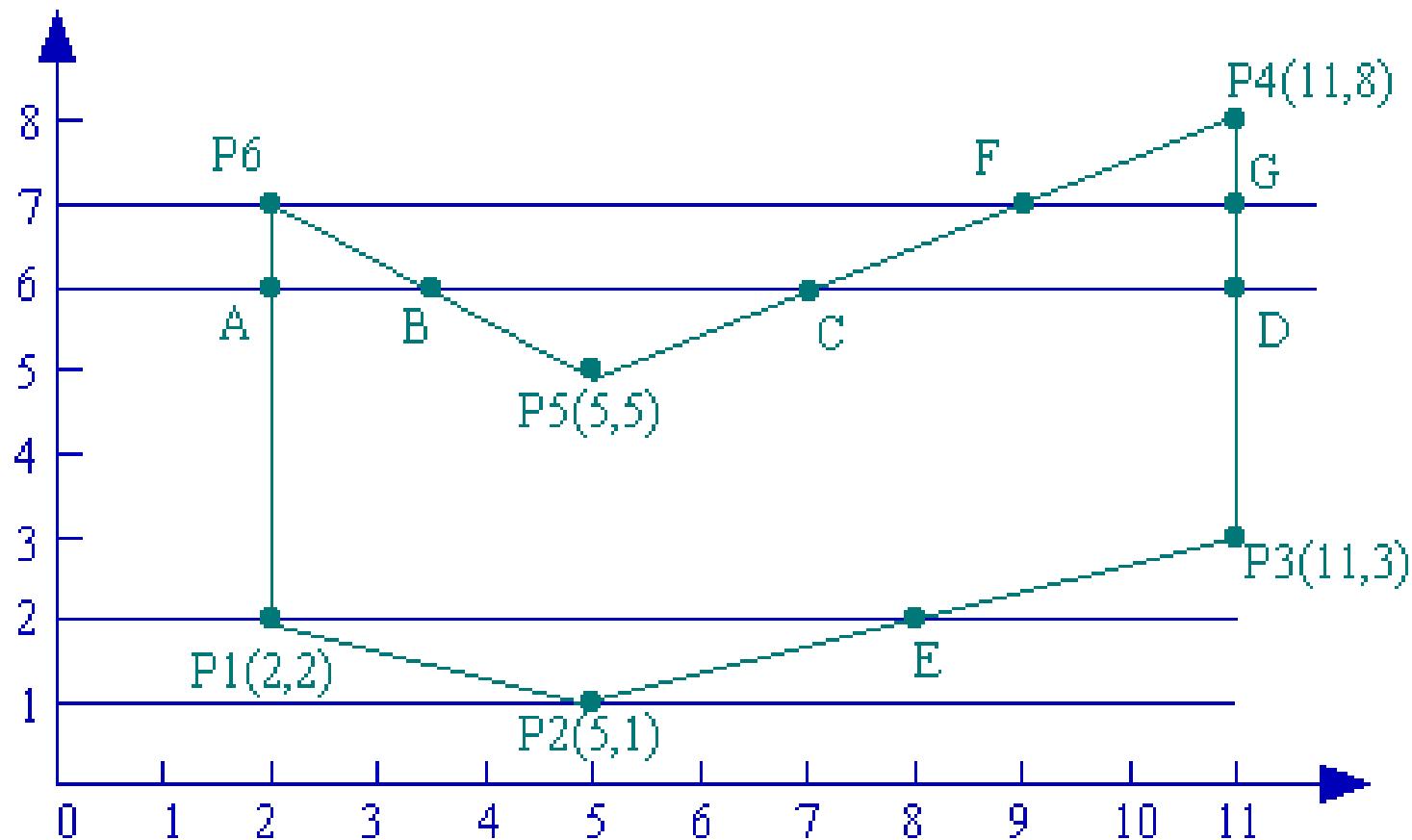


右图所示是一简单多边形，各条扫描线与多边形有不同的交点个数，交点将扫描线分成了不同的区域。求出扫描线与多边形的交点后，将各条扫描线上的交点按X方向从小到大排序后两两配对。每对交点所确定的区间均取填充的光强或色彩，其它区间则取背景光强或色彩，即可完成填充。

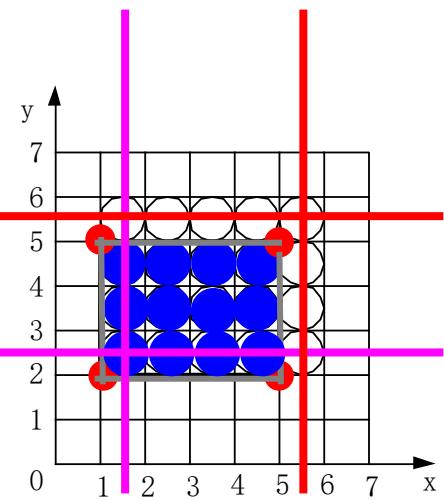
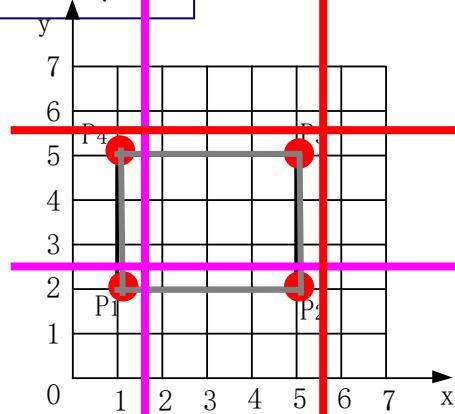
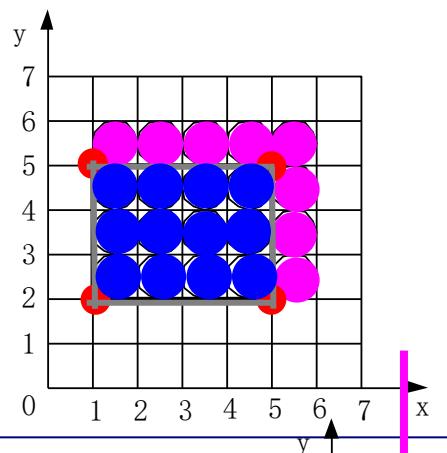
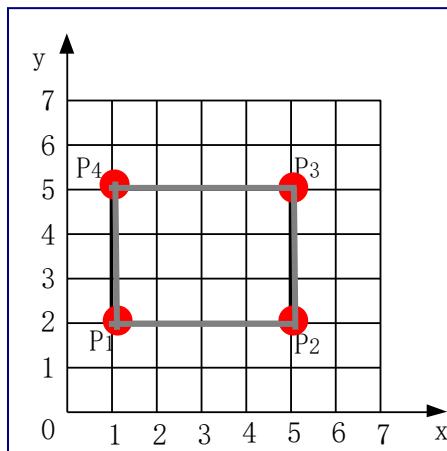


## 步骤：

- (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) 着色：把相交区间内的象素置成多边形颜色，把相交区间外的象素置成背景色。



- 填充扩大化问题



# 扫描线算法

- 交点的取整规则
  - 要求：使生成的像素全部位于多边形之内
    - 用于线画图元扫描转换的四舍五入原则导致部分像素位于多边形之外，从而不可用
  - 假定非水平边与扫描线 $y=e$ 相交，交点的横坐标为 $x$ ，规则如下

# 扫描线算法

## ● 规则1：

X为小数，即交点落于扫描线上两个相邻像素之间

- (a) 交点位于左边之上，向右取整
- (b) 交点位于右边之上，向左取整

# 扫描线算法

## ● 规则2：

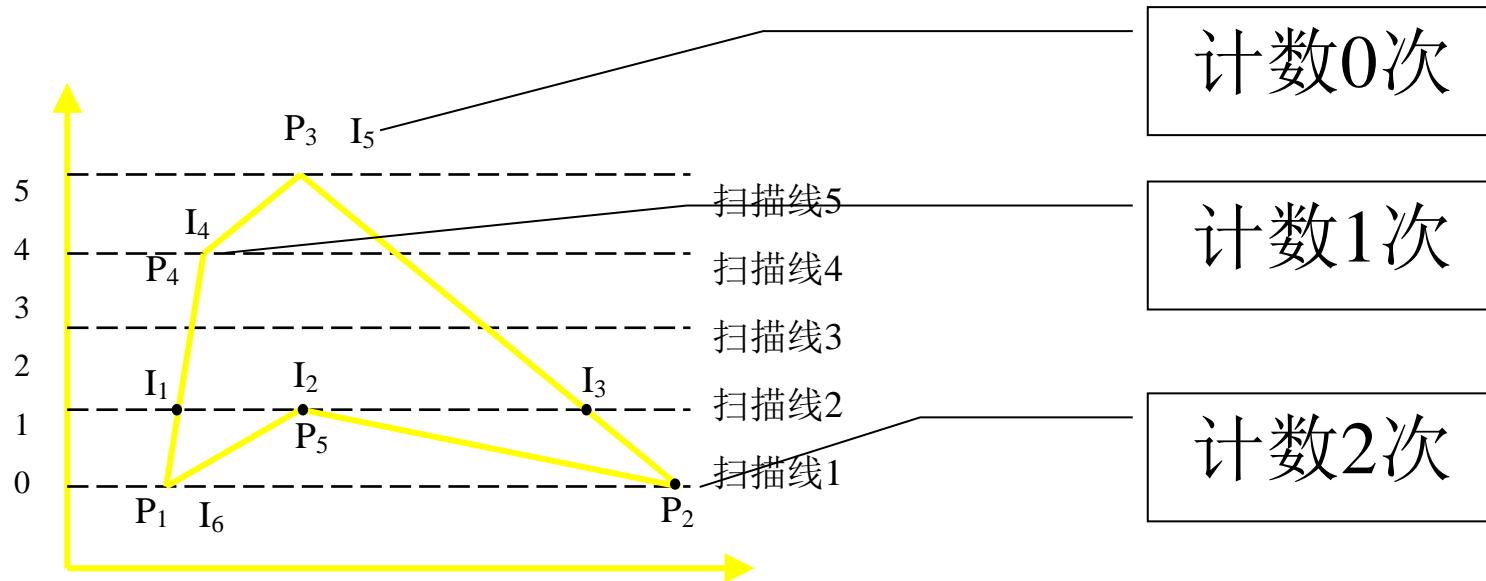
边界上象素的取舍问题，避免填充扩大化。

## ● 解决方法：

边界象素：规定落在右上边界的象素不予填充。

具体实现时，只要对扫描线与多边形的相交区间左闭右开

- 顶点交点的计数问题



- 检查交于该顶点的两条边的另外两个端点的y值大于该顶点y值的个数

# 扫描线算法

## ● 规则3：

扫描线与多边形的顶点相交时，交点的取舍，保证交点正确配对。

## ● 解决方法：

检查两相邻边在扫描线的哪一侧。

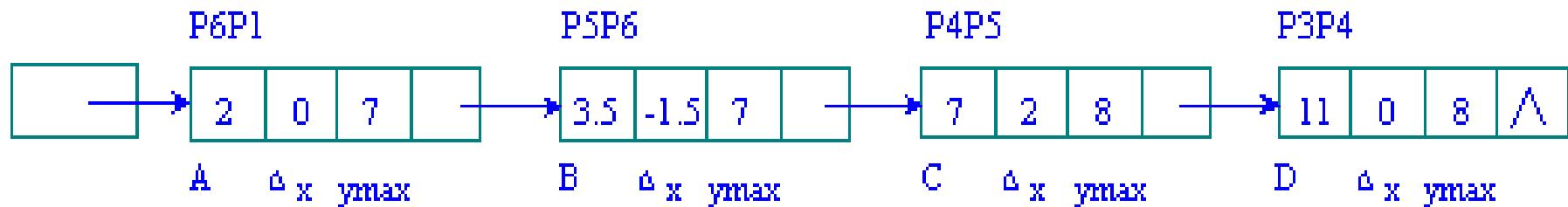
只要检查顶点的两条边的另外两个端点的Y值，两个Y值中大于交点Y值的个数是0, 1, 2, 来决定取0, 1, 2个交点。  
◦

# 算法

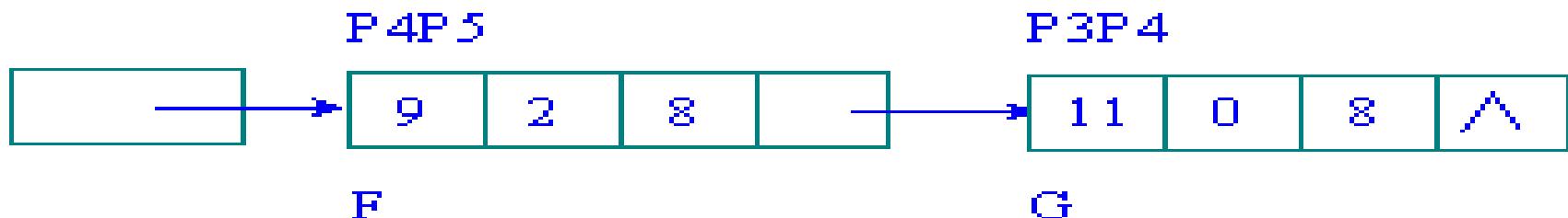
- 影响一般扫描线填充算法效率的因素？
- 求交和排序
- 把多边形所有边放在一个表中，按顺序取出，分别计算与每条扫描线的交点？
- 如何提高效率？
- 目标是简化交点计算
- 建立每条扫描线的活性边表
- 何谓活性边？

# 数据结构的实现

- 为了提高效率，在处理一条扫描线时，仅对与它相交的多边形的边进行求交运算。我们把与当前扫描线相交的边称为活性边，并把它们按与扫描线交点 $x$ 坐标递增的顺序存放在一个链表中，称此链表为**活性边表(AET)**。



(a) 扫描线6的活性边表



(b) 扫描线7的活性边表

- 活性边表的建立
- 结点信息

$x$ : 当前扫描线与边的交点

$\Delta x$ : 从当前扫描线到下一条扫描线之间的 $x$ 增量

$y_{max}$ : 边所交的最高扫描线号

- 活性边表的更新

新边插入

旧边删除

$$\Delta x = 1/k$$



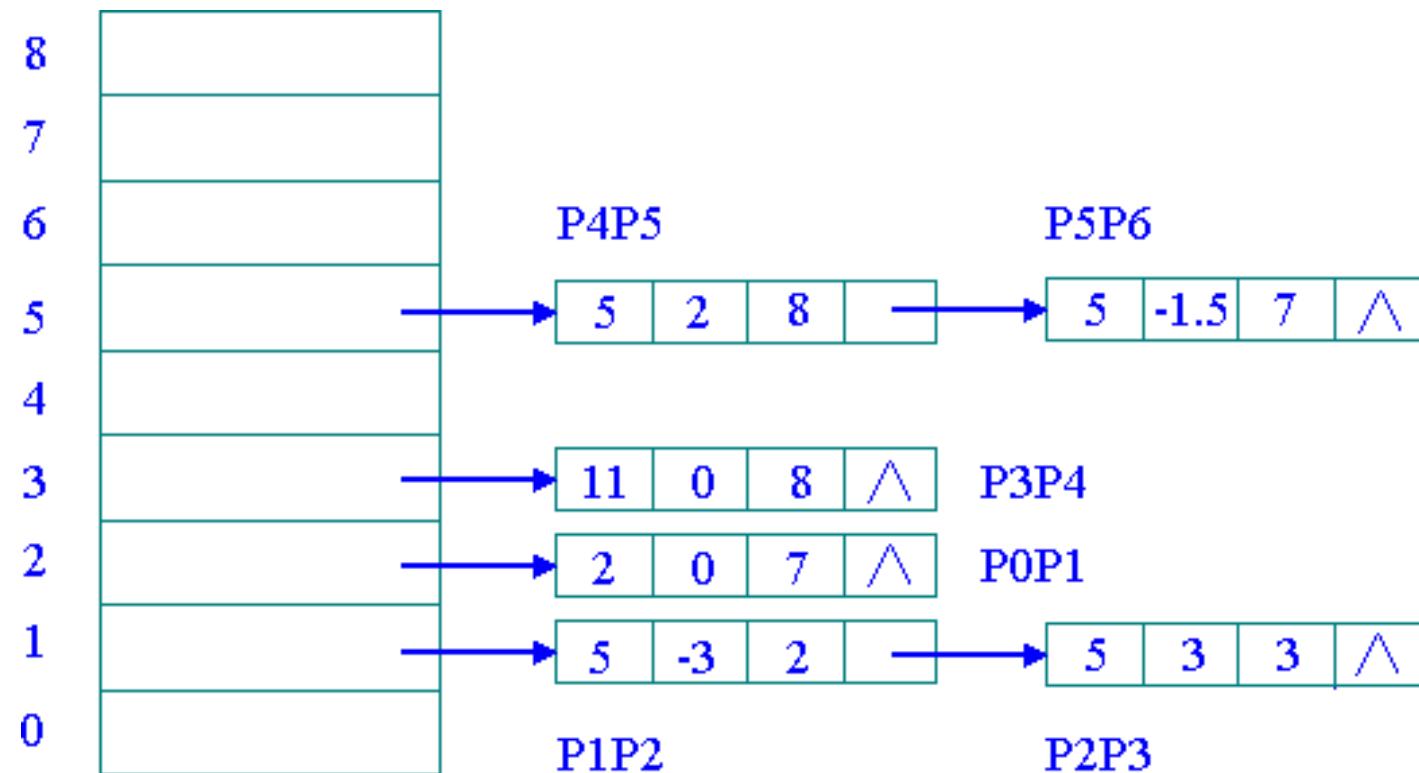
假定当前扫描线与多边形某一条边的交点的横坐标为 $x$ , 则下一条扫描线与该边的交点不必要重计算, 只要加一个增量 $\Delta x$ 即可, 下面, 我们推导这个结论。

设该边的直线方程为:  $ax+by+c=0$ ,  
 $y=y_i$ 时,  $x=x_i$ ; 则当 $y=y_{i+1}$ 时,  
 $X_{i+1}=x_i-b/a$

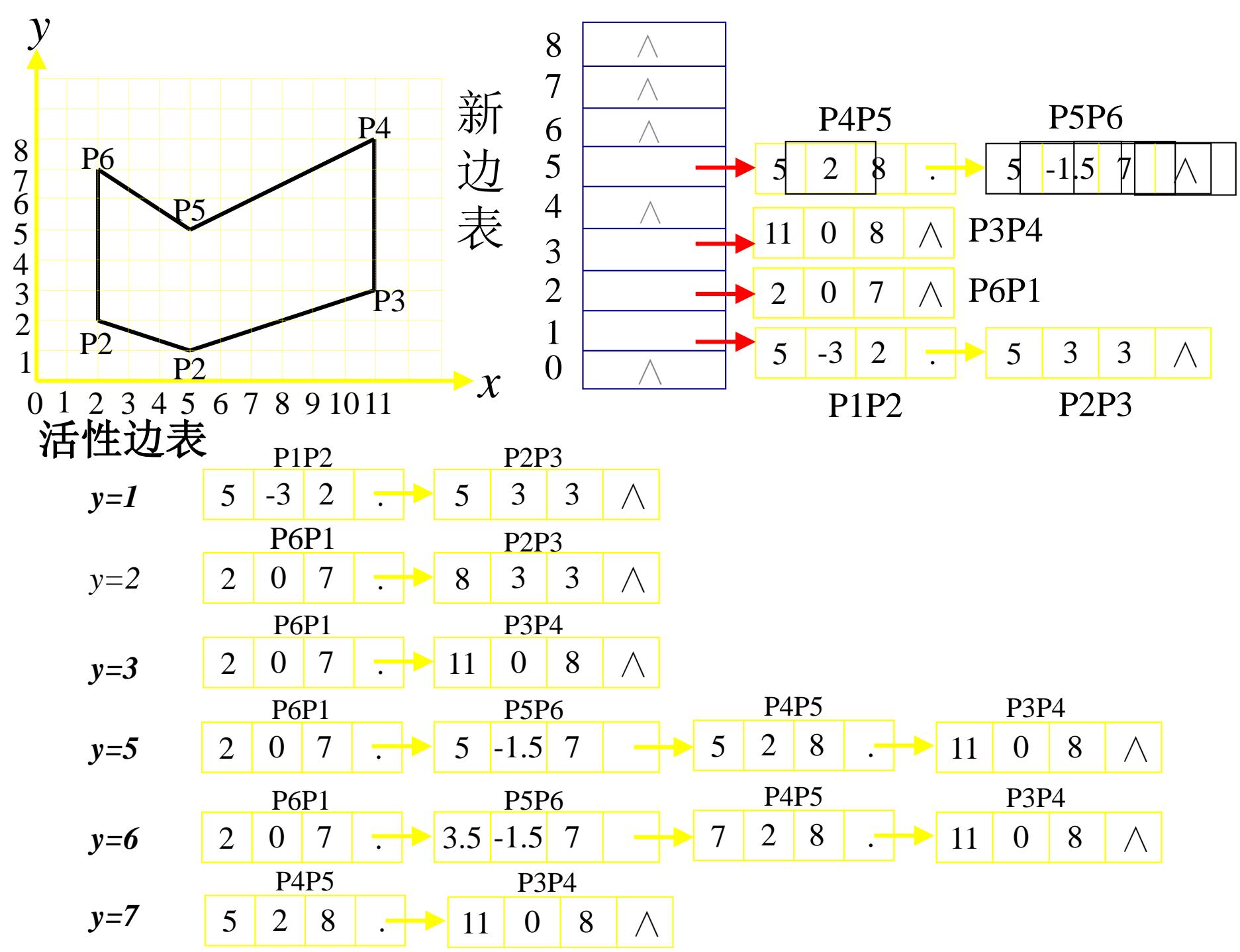
$$\Delta x = -a/b$$

另外使用增量法计算时, 我们需要知道一条边何时不再与下一条扫描线相交, 以便及时把它从活性边表中删除出去。综上所述, 活性边表的结点应为对应边保存如下内容: 第1项存当前扫描线与边的交点坐标 $x$ 值; 第2项存从当前扫描线到下一条扫描线间 $x$ 的增量 $Dx$ ; 第3项存该边所交的最高扫描线号 $ymax$ 。

为了方便活性边表的建立与更新，我们为每一条扫描线建立一个新边表（NET），存放在该扫描线第一次出现的边。也就是说，若某边的较低端点为 $y_{min}$ ，则该边就放在扫描线 $y_{min}$ 的新边表中。



上图所示各条扫描线的新边表NET



算法过程：

```
void polyfill (polygon, color)
```

```
int color;多边形 polygon;
```

```
{ for (各条扫描线i )
```

```
{ 初始化新边表头指针NET [i];
```

```
    把y min = i 的边放进边表NET [i];
```

```
}
```

```
y = 最低扫描线号;
```

```
初始化活性边表AET为空;
```

```
for (各条扫描线i )
```

```
{ 把新边表NET[i]中的边结点用插入排序法插入AET表，使之按x坐标递增顺序排列；
```

```
    遍历AET表，把配对交点区间(左闭右开)上的象素(x, y)，用drawpixel (x, y, color) 改写象素颜色值；
```

```
    遍历AET表，把y max= i 的结点从AET表中删除，并把y max > i结点的x值递增D x;
```

```
    若允许多边形的边自相交，则用冒泡排序法对AET表重新排序；
```

```
}
```

```
} /* polyfill */
```

# 扫描线算法

- 优点：
  - 对每个像素只访问一次
  - 与设备无关

## ● 缺点：

- 数据结构复杂
- 只适合软件实现

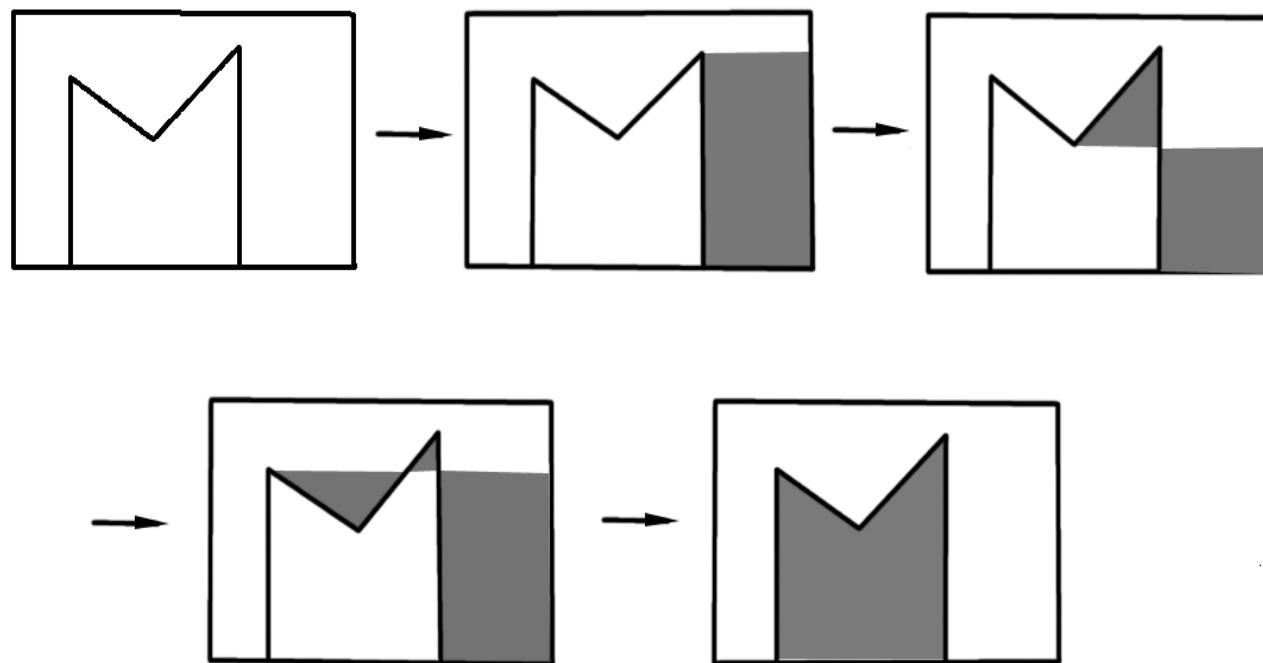
## 边填充法

另一种扫描转换方法即所谓的边填充算法。其基本思想是对每条扫描线和每条多边形的交点 ( $x_k, y_k$ )，将该扫描线上交点右方的所有象素取补。对多边形的每条边作此处理，就可以完成多边形区域填充。

边填充算法描述如下：

1. 取多边形的一条边；
  2. 求出每一条扫描线与该边的交点坐标 ( $x_k, y_k$ )；
  3. 将  $(x_k, y_k)$  右边的全部象素取补；
  4. 还有没处理的多边形边时转1，否则结束
-

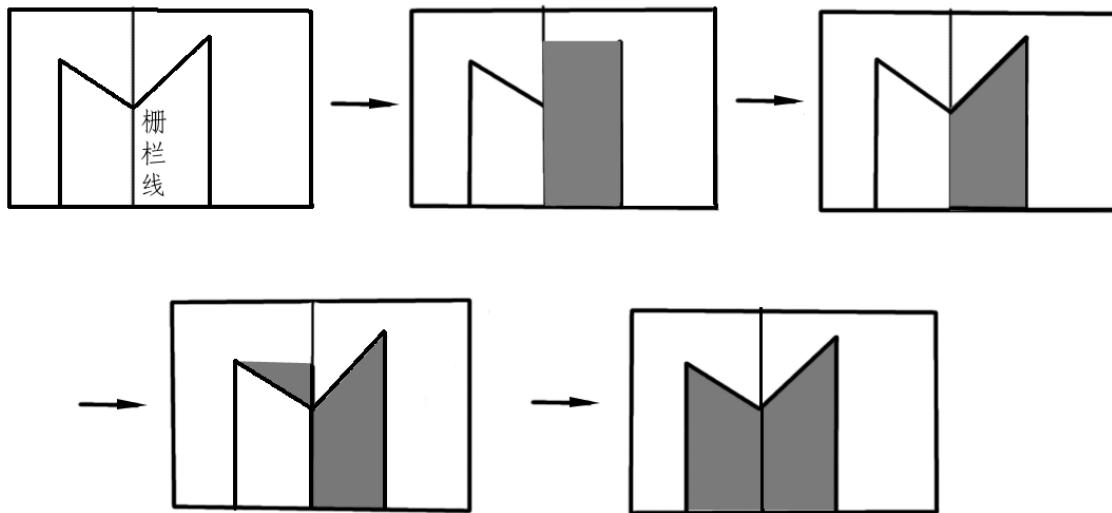
下图示出了边填充算法的实现过程。



# 边填充算法 (Edge Fill Algorithm)

- **优点:**
  - 最适合于有帧缓存的显示器
  - 可按任意顺序处理多边形的边
  - 仅访问与该边有交点的扫描线上右方的像素，算法简单
- **缺点:**
  - 对复杂图形，每一像素可能被访问多次，输入/输出量大
  - 图形输出不能与扫描同步进行，只有全部画完才能打印

# 栅栏填充算法 (Fence Fill Algorithm)



- 引入栅栏的目的？

## 种子填充算法\*\*

以上讨论的填充多边形的算法都是按扫描线顺序进行的。种子填充算法则来用完全不同的方法。

**种子填充算法**假设在多边形或区域内部至少有一个象素是已知的。然后设法找到区域内所有其它象素，并对它们进行填充。

- 区域指已经表示成点阵形式的填充图形，它是象素的集合。

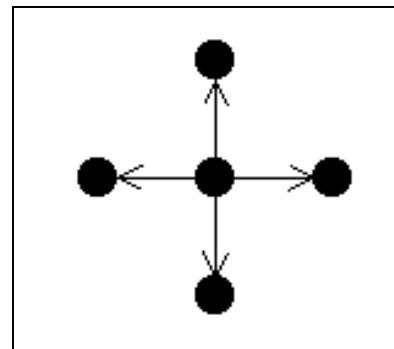
- 表示方法：内点表示、边界表示
- 内点表示
  - 枚举出区域内部的所有像素
  - 内部的所有像素着同一个颜色
  - 边界像素着与内部像素不同的颜色
- 边界表示
  - 枚举出边界上所有的像素
  - 边界上的所有像素着同一颜色
  - 内部像素着与边界像素不同的颜色

区域填充要求区域是连通的

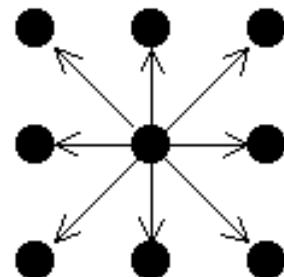
- 连通性

4连通、8连通

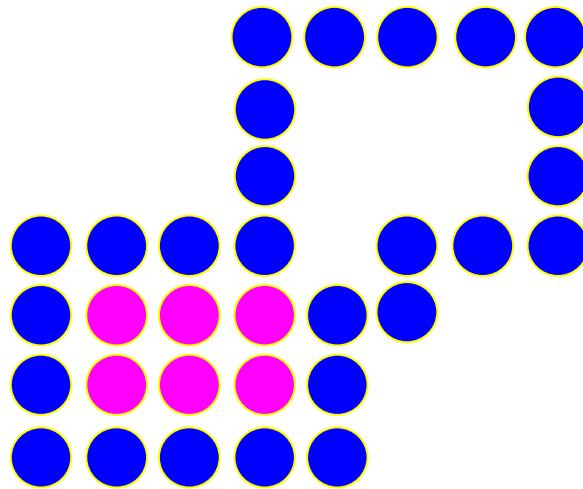
- 4连通：



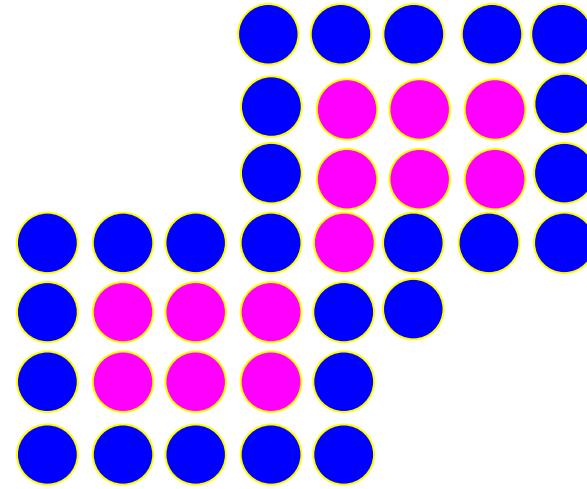
- 8连通



# 区域连通方式对填充结果的影响



4连通区域边界填  
充算法的填充结果



8连通区域边界填  
充算法的填充结果

# 简单的种子填充算法 (4连通边界)

对4连接算法，应用边界定义区域，可使用堆栈以建立简单的种子填充算法。使用的堆栈的种子填充算法如下：

- 1 种子象素压入堆栈；
- 2 当堆栈非空时做
  - (1) 栈顶象素出栈；
  - (2) 将出栈象素置成填充颜色；
  - (3) 检查每个与当前象素邻接的4连接象素，若其中有象素不为边界且没有设置成填充颜色，将该象素压入堆栈；
  - (4) 转2

# 算法可用伪语言描述如下：

算法: seed(x, y) 作为种子

BV (BoundaryValue) 边界值

NV (newValue) 填充值

begin

    pixel(x, y)=seed(x, y)

    push pixel(x, y)

    while (stack not empty)

        pop pixel(x, y)

        if pixel(x, y)<>NV

            then putpixel(x, y, NV)

            pixel(x, y)=NV

        endif

        if pixel(x+1, y)<>NV and pixel(x+1, y)<>BV

            then push pixel(x+1, y)

        endif

        if pixel(x, y+1)<>NV and pixel(x, y+1)<>BV

            then push pixel(x, y+1)

        endif

        if pixel(x-1, y)<>NV and pixel(x-1, y)<>BV

            then push pixel(x-1, y)

        endif

        if pixel(x, y-1)<>NV and pixel(x, y-1)<>BV

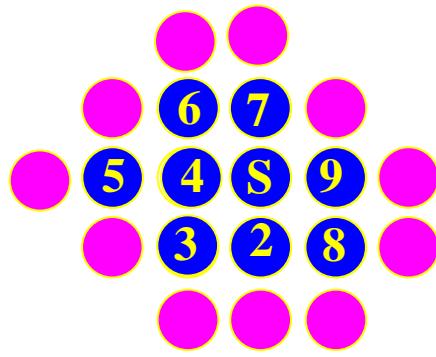
            then push pixel(x, y-1)

        endif

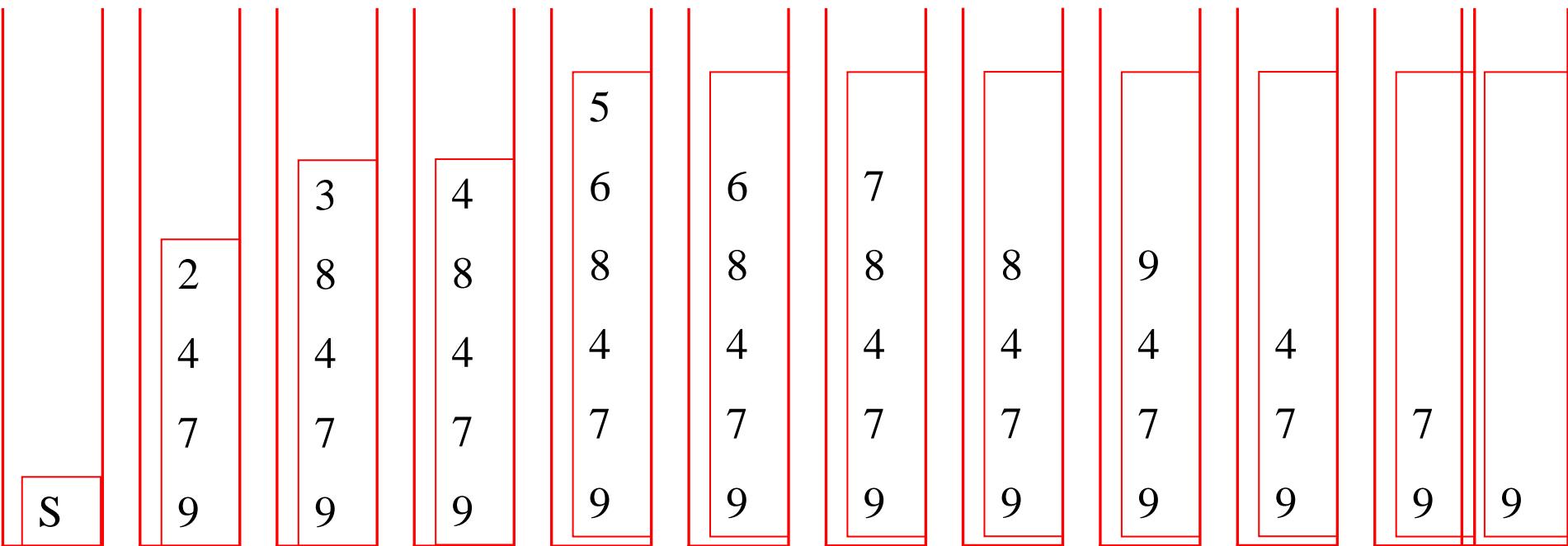
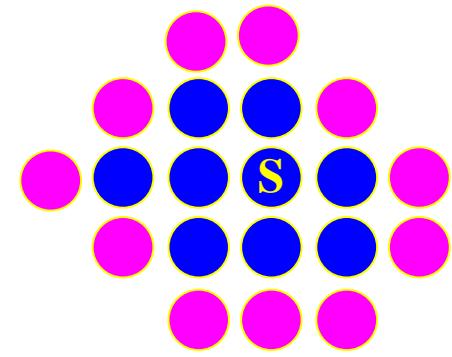
    endwhile

end

# 填充算法演示



缺点?



#### 4-connected boundary-fill

```
void BoundaryFill4(int x,int y,int fill,int
boundary)
{
    int current;
    current = getpixel(x, y);
    if ((current != boundary) && (current != fill))
    {
        putpixel(x, y, fill);
        BoundaryFill4(x+1, y, fill, boundary);
        BoundaryFill4(x-1, y, fill, boundary);
        BoundaryFill4(x, y+1, fill, boundary);
        BoundaryFill4(x, y-1, fill, boundary);
    }
}
```

#### 4-connected boundary-fill

```
void FloodFill4(int x,int y,int fillColor,int
oldColor)
{
    int current;
    current = getpixel(x, y);
    if (current == oldColor)
    {
        putpixel(x, y, fillColor);
        BoundaryFill4(x+1, y, fillColor, oldColor);
        BoundaryFill4(x-1, y, fillColor, oldColor);
        BoundaryFill4(x, y+1, fillColor, oldColor);
        BoundaryFill4(x, y-1, fillColor, oldColor);
    }
}
```

上面的算法是一种深度优先搜索算法，采用堆栈实现。也可以改为广度优先搜索，采用队列实现。

# 多边形扫描转换与区域填充方法比较

- 联系：都是光栅图形面着色，用于真实感图形显示。可相互转换。
- 多边形的扫描转换转化为区域填充问题：当给定多边形内一点为种子点，并用*Bresenham*或*DDA*算法将多边形的边界表示成八连通区域后，则多边形的扫描转换转化为区域填充。
- 区域填充转化为多边形的扫描转换；若已知给定多边形的顶点，则区域填充转化为多边形的扫描转换。

# 多边形扫描转换与区域填充 方法比较

- 不同点：

1. 基本思想不同：前者是顶点表示转换成点阵表示，后者只改变区域内填充颜色，没有改变表示方法。
2. 对边界的要求不同
  - 前者只要求扫描线与多边形边界交点个数为偶数。后者：区域封闭，防止递归填充跨界。
3. 基本的条件不同
  - 前者：从边界顶点信息出发。
  - 后者：区域内种子点。

# 扫描线种子填充算法

- 利用扫描线的连贯性
- 减少递归次数

## 扫描线种子填充算法

种子填充算法的缺点是可能把太多的象素压入堆栈，有些象素甚至会入栈多次，降低算法的效率，存储空间需求大。解决的一种方法是对每一条扫描线实行种子算法。在任意不间断的扫描线象素段中，只取一个种子象素。象素段是指区域内相邻象素在水平方向的组合，它的两端以具有边界值的象素为界，其中间不包括具有新值的象素。

对于区域内的每一象素段，我们可以只保留其最右（或左）端的象素作为种子象素。因此，区域中每一个未被填充的部分，至少有一个象素段是保持在栈里的。扫描线种子填充算法适用于边界定义的区域。区域可以是凸的，也可以是凹的，还可以包含一个或多个孔。

定义的区域。区域可以是凸的，也可以是凹的，还可以包含一个或多个孔。

算法叙述如下：

- 1 种子象素入栈；
- 2 当堆栈非空时做
  - (1) 栈顶象素出栈；

(2) 沿扫描线对出栈象素的左右象素进行填充，直到遇到边界象素为止，即每出栈一个象素，便对包含该象素的整个区间进行填充；

(3) 上述区间内最左最右的象素分别记为 $x_{Left}$ 、 $x_{Right}$ ；

(4) 在区间 $[x_{Left}, x_{Right}]$ 中检查与当前扫描线相邻的上下两条扫描线的有关象素是否全为边界象素或为已填充的象素，若存在非边界未填充的象素，则把每一区间的最右象素取作种子象素入栈；

(5) 转2

此算法可以有效地解决简单种子填充算法存在的堆栈可能过深的问题。

## 扫描线种子填充算法主要实现子程序实例

```
void push(int x, int y)
{
    points[top][0]=x;
    points[top][1]=y;
    top++;
}
?
void pop()
{
    x=points[top-1][0];
    y=points[top-1][1];
    top--;
}
?
void Scanline_seed_fill_draw(HDC hdc)
{
    int i;
    COLORREF BoundColor;
    POINT triangle[]={300, 275, 280, 325, 320, 325, 300, 275} ;
    POINT fourline[]={250, 340, 300, 375, 350, 340, 300, 350, 250, 340} ;
    BoundColor=RGB(255, 0, 0);
    SetColor(BoundColor, hdc) ;
```

```

Ellipse(hdc, 180, 200, 420, 400) ;
Ellipse(hdc, 230, 230, 270, 310) ;
Ellipse(hdc, 330, 230, 370, 310) ;
Ellipse(hdc, 245, 264, 255, 296) ;
Ellipse(hdc, 345, 264, 355, 296) ;
for(i=0;i<3;i++)
    line(hdc, triangle[i], triangle[i+1]) ;
for(i=0;i<4;i++)
    line(hdc, fourline[i], fourline[i+1]) ;
Scanline_seed_fill(hdc, 300, 300, RGB(0, 0, 255), BoundColor) ;
Scanline_seed_fill(hdc, 250, 280, RGB(0, 255, 0), BoundColor) ;
Scanline_seed_fill(hdc, 350, 280, RGB(0, 255, 0), BoundColor) ;
Scanline_seed_fill(hdc, 300, 360, RGB(0, 255, 255), BoundColor) ;
Scanline_seed_fill(hdc, 235, 270, RGB(0, 0, 0), BoundColor) ;
Scanline_seed_fill(hdc, 335, 270, RGB(0, 0, 0), BoundColor) ;
}

void Scanline_seed_fill(HDC hdc, int seed_x, int seed_y, COLORREF FillColor, COLORREF
BoundColor)
{
    int active_x, active_y, Left_x, Right_x, flag, Nextspan_x;
    COLORREF rgbColor;
    push(seed_x, seed_y);
    while(top!=bottom)
    {
        pop();
        SetPixel(hdc, x, y, FillColor);
}

```

```

//填充扫描线右半部分
active_x=x+1;
while(GetPixel(hdc, active_x, y)!=BoundColor)
{
    SetPixel(hdc, active_x, y, FillColor);
    active_x++;
    Sleep(1);
}
Right_x=active_x-1;
//填充扫描线左半部分
active_x=x-1;
while(GetPixel(hdc, active_x, y)!=BoundColor)
{
    SetPixel(hdc, active_x, y, FillColor);
    active_x--;
    Sleep(1);
}
Left_x=active_x+1;
active_x=Left_x;
//确定下一条扫描线的种子
y=y+1;
while(active_x<Right_x)
{
    flag=0;
    while((GetPixel(hdc, active_x, y)!=FillColor) &&
          (GetPixel(hdc, active_x, y)!= BoundColor) && (active_x<Right_x))
    {

```

```

if(flag==0)
    flag=1;
    active_x++;
}
if(flag==1)
{
    if((active_x==Right_x)&&
        (GetPixel(hdc, active_x, y) !=FillColor) && (GetPixel(hdc, active_x, y) !=BoundColor))
        push(active_x, y);
    else
        push(active_x-1, y);
    flag==0;
}
Nextspan_x=active_x;
while(((GetPixel(hdc, active_x, y)==BoundColor) || (GetPixel(hdc, active_x, y)
            ==FillColor))&&(active_x<Right_x))
    active_x++;
if(Nextspan_x==active_x)
    active_x++;
}
active_x=Left_x;

//确定上一条扫描线的种子
y=y-2;

```

```

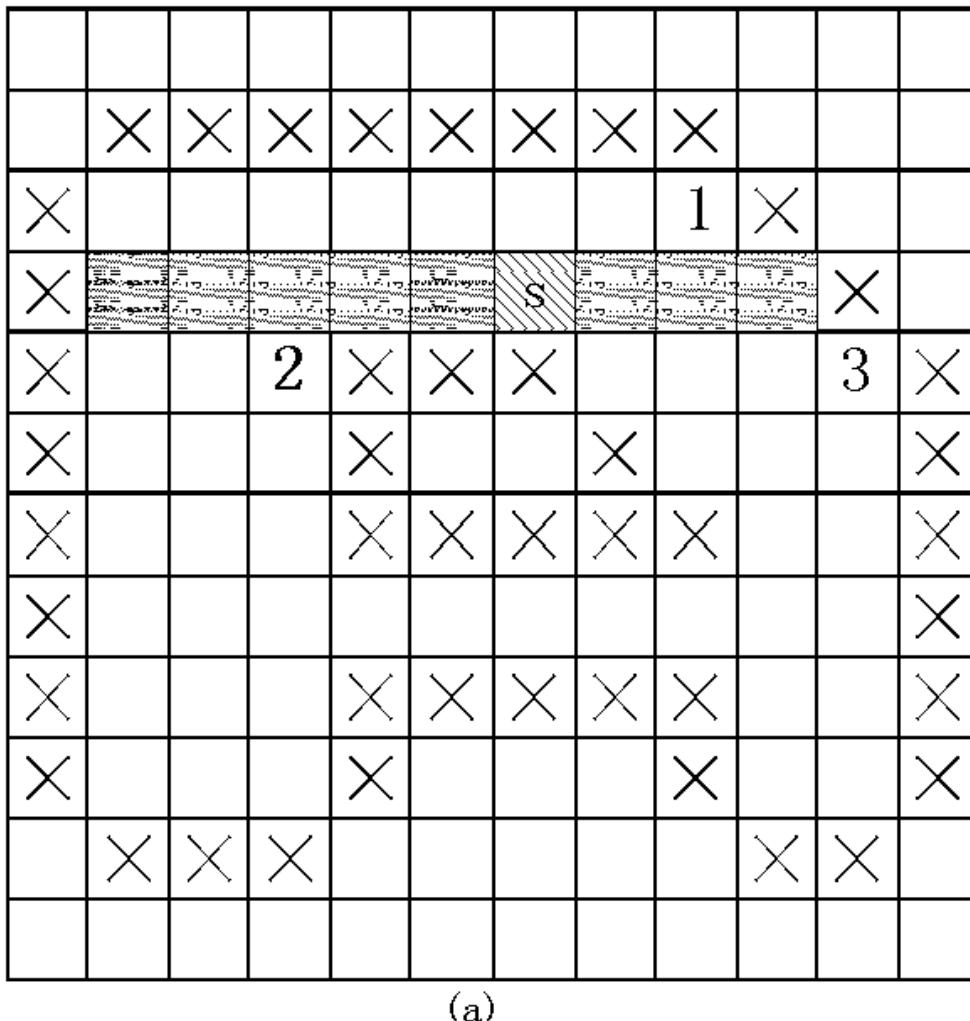
while(active_x<Right_x)
{
    flag=0;
    while((GetPixel(hdc, active_x, y)!=BoundColor)&&(GetPixel(hdc, active_x, y)!=
    FillColor)&&(active_x<Right_x))
    {
        if (flag==0)
            flag=1;
        active_x++;
    }
    if(flag==1)
    {
        if((active_x==Right_x)&&
           (GetPixel(hdc, active_x, y)!=FillColor) && (GetPixel(hdc, active_x, y)!=BoundColor))
            push(active_x, y);
        else
            push(active_x-1, y);
        flag==0;
    }
    Nextspan_x=active_x;
    while(((GetPixel(hdc, active_x, y)==BoundColor) ||
           (GetPixel(hdc, active_x, y)==FillColor))&&(active_x<Right_x))
        active_x++;
    if(Nextspan_x==active_x)
        active_x++;
}
}
}

```

## 扫描线种子填充算法演示

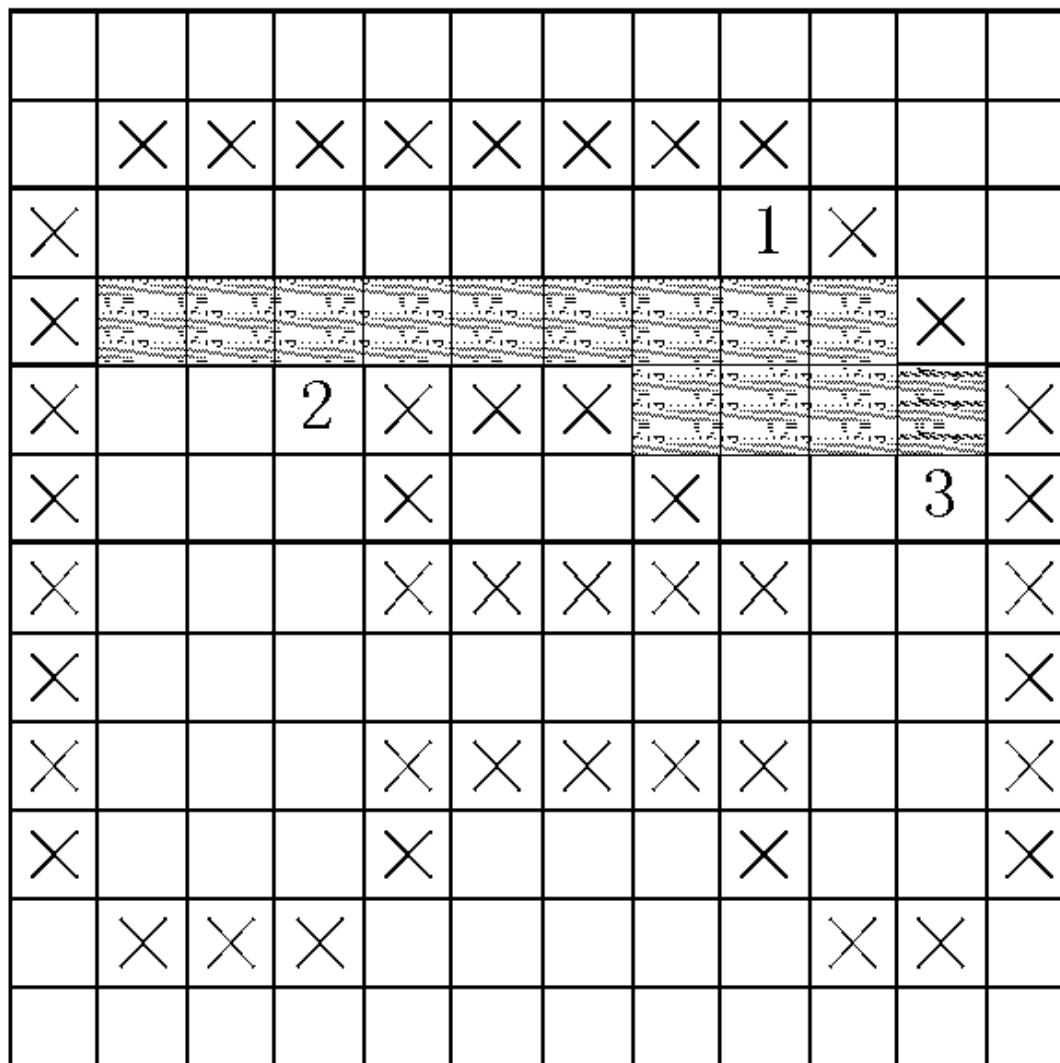
# 扫描线算法分析（举例分析 ）

- 该算法也可以填充有孔区域。



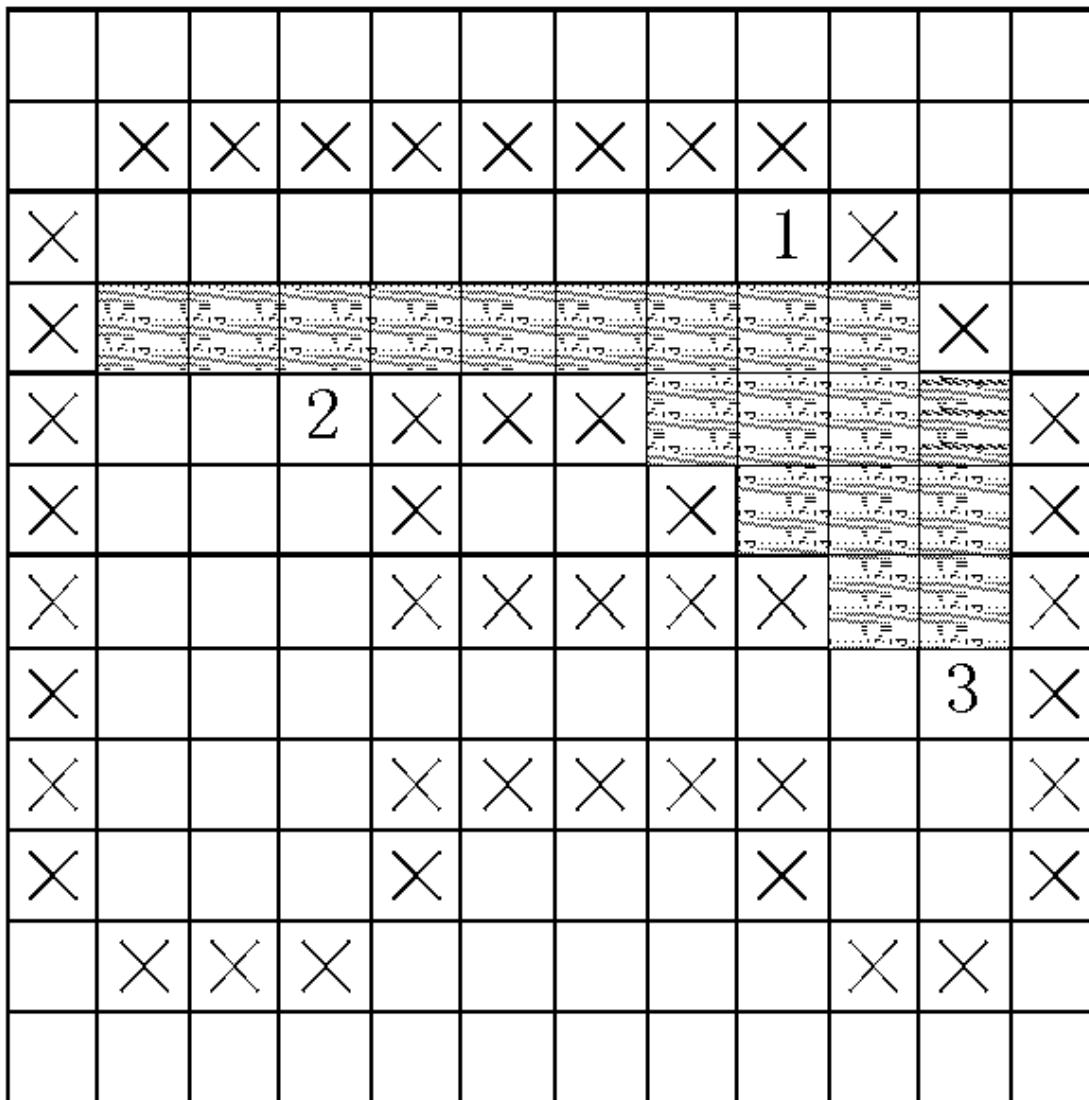
像素中的序  
号标指它所  
在区段位于  
堆栈中的位  
置

# 扫描线算法分析（举例分析 ）



(b)

# 扫描线算法分析（举例分析 ）



(c)

# 扫描线算法分析（举例分析 ）

(d)

# 二维光栅图形的混淆与反混淆

- 混淆现象
- 反混淆方法

# 混淆(antialiasing)

图形的锯齿状： 图形信号连续，光栅显示系统中，离散表示。

用离散量(像素)表示连续的量(图形)而引起的失真，叫混淆或叫走样(aliasing)

光栅图形混淆：

- 阶梯状边界；
- 图形细节失真；
- 狹小图形遗失： 动画序列中时隐时现，产生闪烁。

# 走样现象举例

## □ 走样现象：

- 光栅图形产生的阶梯形
- 图形的细节或纹理绘制失真
- 图形中包含相对微小的物体时，这些物体在静态图形中容易被丢弃或忽略，在动画序列中时隐时现，产生闪烁

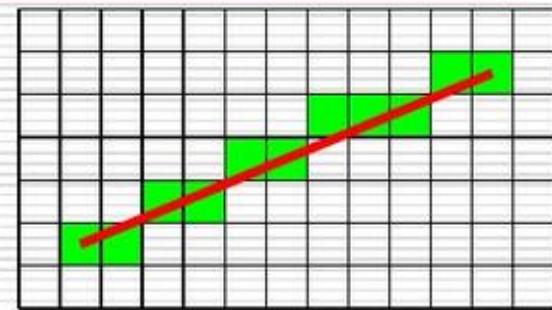
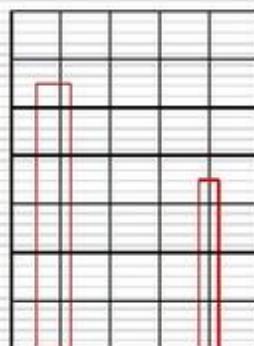
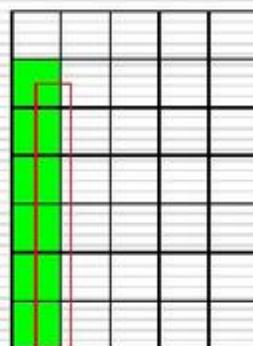


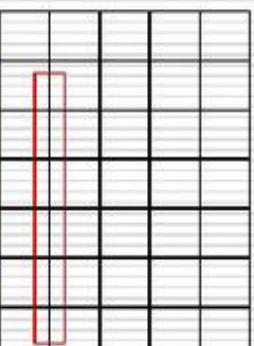
图5-48 绘制直线时的反走样现象



(1) 需显示的矩形 (2) 显示结果



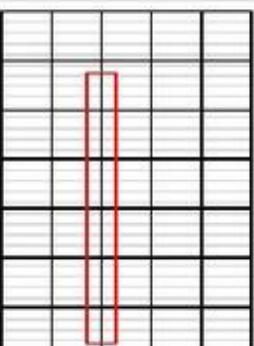
(3) 需显示的矩形 (4) 显示结果



(5) 需显示的矩形 (6) 显示结果



(7) 需显示的矩形 (8) 显示结果



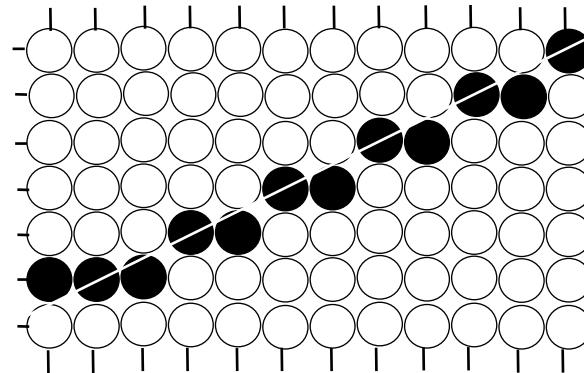
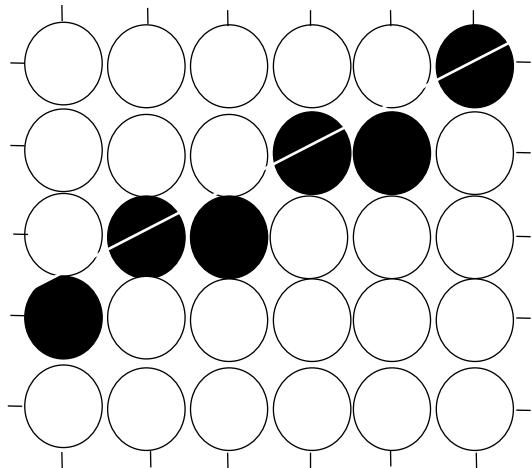
(9) 需显示的矩形 (10) 显示结果

# 图形反走样技术 (antialiasing)

- 1. 从硬件角度提高分辨率
  - 高分辨率显示器
- 显示器点距减少一倍
- 帧缓存容量增加到原来的4倍
- 输带宽提高4倍
- 扫描转换花4倍时间
- 代价高

# 提高分辨率

- 把显示器分辨率提高一倍，
  - 直线经过两倍的象素， 锯齿也增加一倍，
  - 但同时每个阶梯的宽度也減小了一倍，
  - 所以显示出的直线段看起来就平直光滑了一些。



# 图形反走样技术 (antialiasing)

- 2. 从软件角度替高分辨率
  - 高分辨率计算，低分辨率显示
  - 像素细分技术，相当于后置滤波

1	1
1	1

算术  
平均

1	2	1
2	4	2
1	2	1

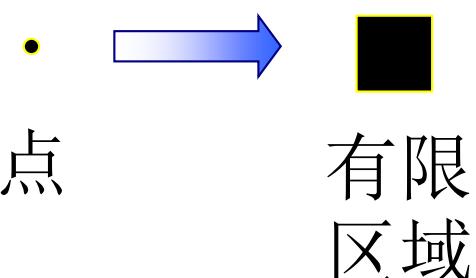
加权  
平均

- 只能减轻，不能消除

# 图形反走样技术（antialiasing）

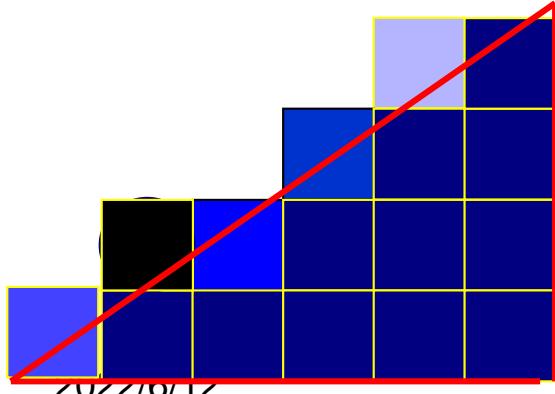
- 3. 区域采样技术
  - 改变边或直线的外观，模糊淡化阶梯
  - 相当于图像的前置滤波

直线有宽度

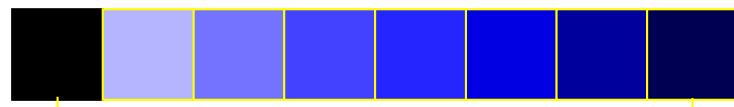


# 图形反走样技术（antialiasing）

根据相交的面积值决定像素显示的亮度级别



8级灰度



$0 \leq \text{面积} \leq 1/8$

$7/8 \leq \text{面积} \leq 1$

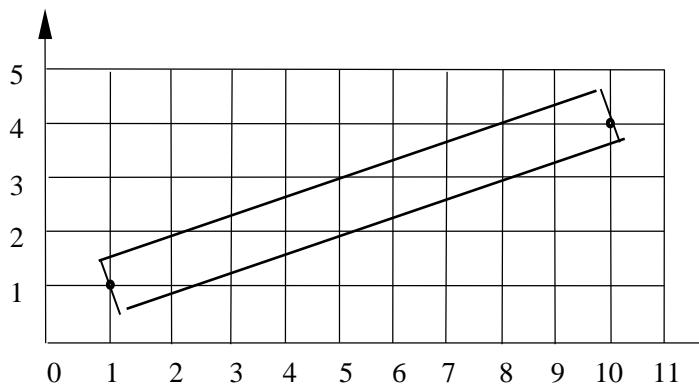
# 简单区域取样

- 解决方法：改变直线段模型，由此产生算法
- 方法步骤：
  - 1、将直线段看作具有一定宽度的狭长矩形；
  - 2、当直线段与某象素有交时，求出两者相交区域的面积；
  - 3、根据相交区域的面积，确定该象素的亮度值

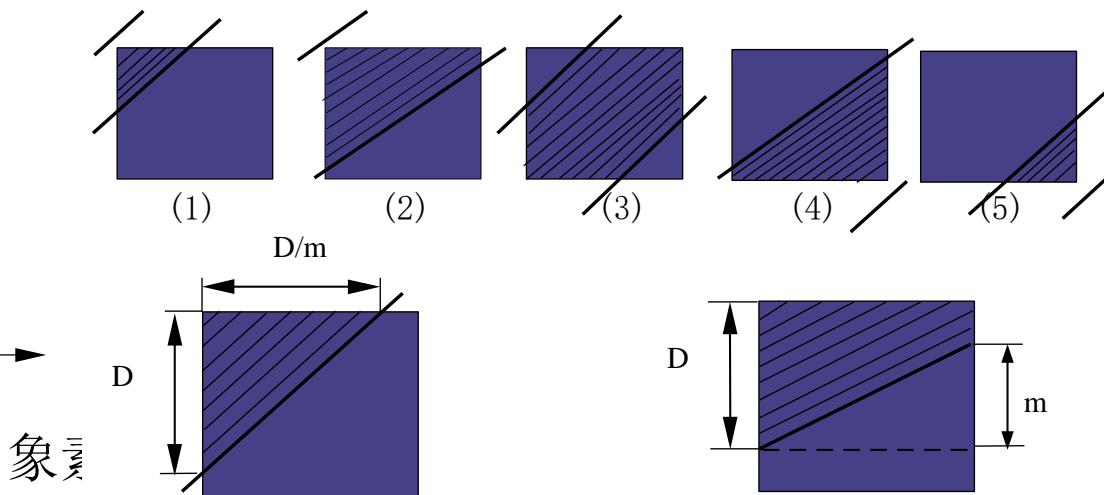
# 简单区域取样

- 基本思想：

- 每个象素是一个具有一定面积的小区域，将直线段看作具有一定宽度的狭长矩形。当直线段与象素有交时，求出两者相交区域的面积，然后根据相交区域面积的大小确定该象素的亮度值。

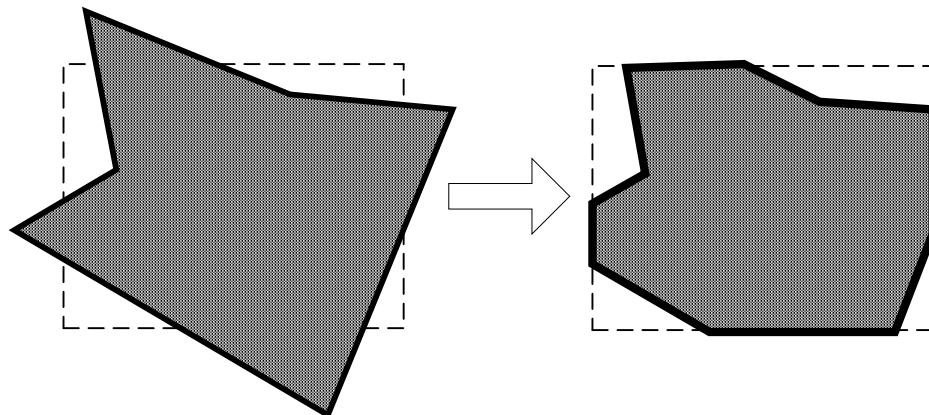


有宽度的线条轮廓



# 第四章 图形裁剪

在用户坐标系中定义的图形往往是大而复杂的，而输出设备如显示屏幕的尺寸及其分辨率却是有限的，为了能够清晰地观察某一部分或对其进行某些绘图操作，就需要将所关心的这一局部区域的图形从整个图形中区分出来，这个区分指定区域内和区域外的图形过程称为**裁剪**，所指定的区域称为**裁剪窗口**。



裁剪通常是对用户坐标系中窗口边界进行裁剪，然后把窗口内的部分映射到视区中，也可以首先将用户坐标系的图形映射到设备坐标系或规范化设备坐标系中，然后用视区边界裁剪。

下面假定裁剪是针对用户坐标系中窗口边界进行的，裁剪完成后，再把窗口内图形映射到视区。所以裁剪的目的是显示可见点和可见部分，删除视区外的部分。

## 4.1 窗口区和视图区

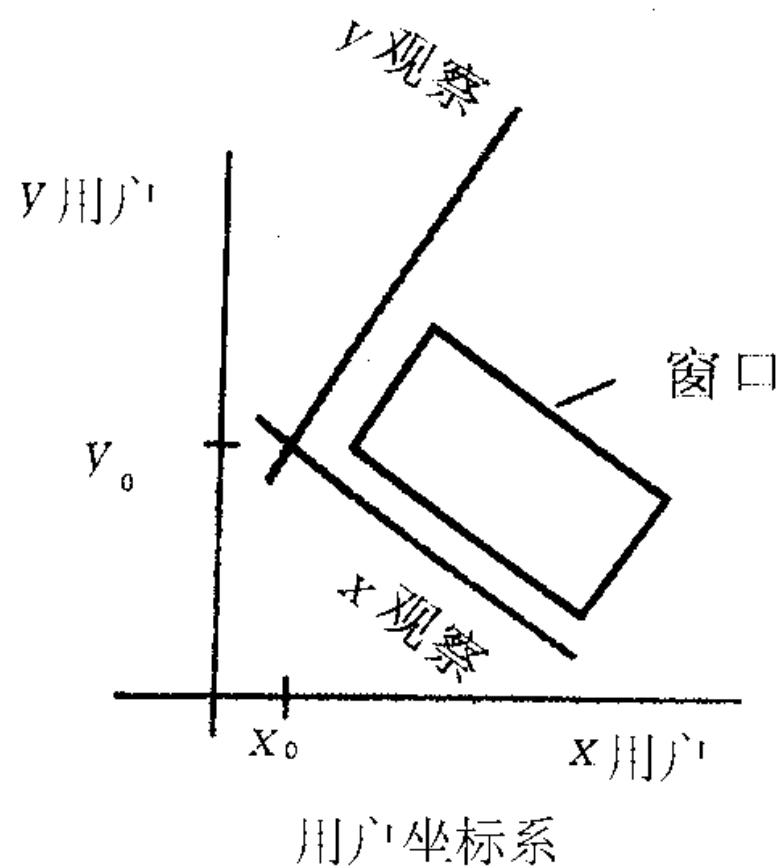
### 4.1.1 坐标系

#### 1. 用户坐标系 (World Coordinates)

又称为世界坐标系、完全坐标系等，它可以是用户用来定义设计对象的各种标准坐标系，例直角坐标、极坐标、球坐标、对数坐标等。用户坐标系所拥有的区域范围从理论上说是连续的、无限的。

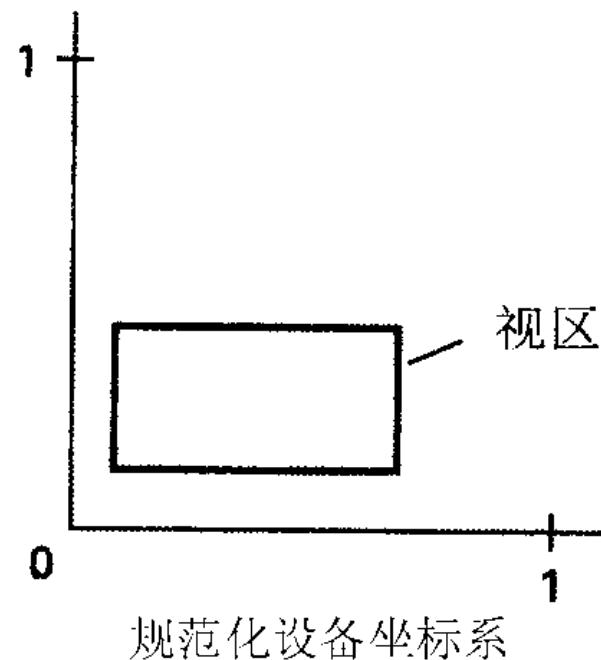
#### 2. 观察坐标系 (Viewing Coordinates)

在用户坐标中设置观察坐标系，在观察坐标系中定义一个观察窗口。观察坐标系用来任意设置矩形窗口的方向。一旦建立了观察参考系，就可以将用户坐标系下的描述变换到观察坐标系下。



### 3. 规范化设备坐标系 (Normalized Device Coordinates)

在规范化坐标系下（取值范围从0到1）定义视区，将观察坐标系下的场景描述映射到规范坐标系下。



## 4. 设备坐标系 (Device Coordinates)

图形输出时需要一定的设备，如绘图仪、显示器等，使用的是设备坐标系。设备坐标系一般为二维坐标，如屏幕坐标。它的范围有限，单位一般为整数。设备坐标一般采用无量纲方式，可以转换为有量纲坐标。一旦场景变换到规范化坐标系下的单位正方形中，以后该单位面积只需要简单地映射到具体输出设备的显示区。给出合适的设备驱动程序，就可以使用不同的输出设备。

## 4.1.2 窗口区和视图区

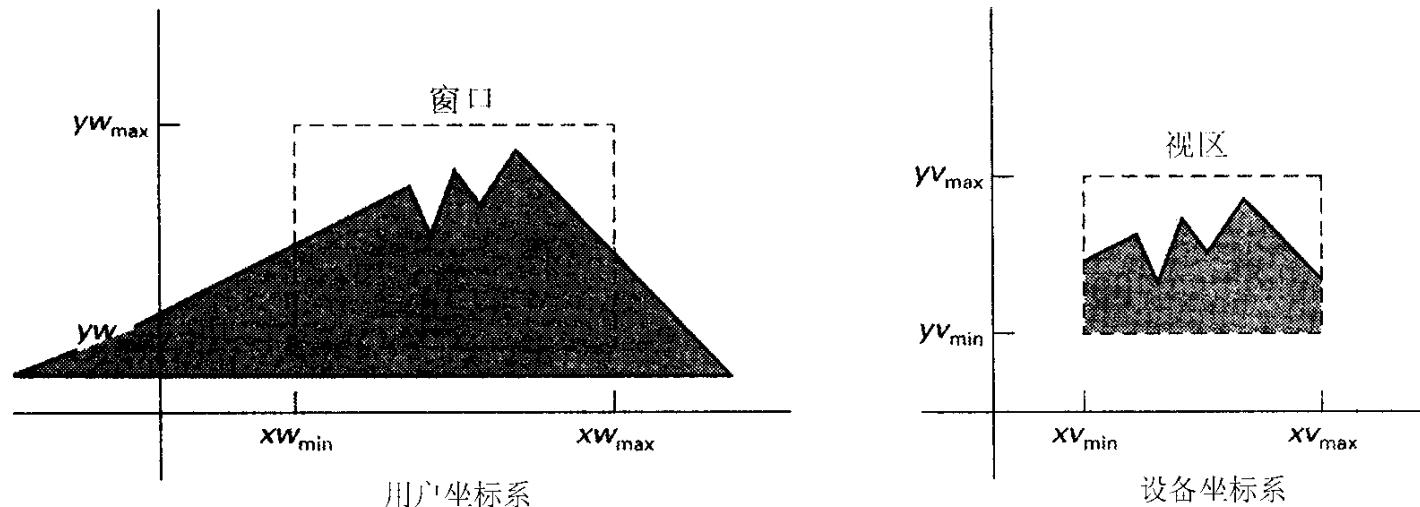
用户可以在用户坐标系中指定感兴趣的任意区域，把这部分区域内的图形输出到屏幕上，这个指定区域称为窗口区。窗口区一般是矩形区域，可以用左下角和右上角两点坐标来定义其大小和位置。定义窗口的目的是要选取图形中需要观察的那一部分图形。在计算机图形学术语中，窗口最初是指要观察的图形区域，但是目前窗口也用于窗口管理系统。在本章中，我们将窗口理解为用户坐标系中要显示的区域。

图形设备上用来输出图形的最大区域称之为屏幕域，它是有限的整数域，大小随具体设备而异。任何小于或等于屏幕域的区域都可定义为视图区。视图区由用户在屏幕域中用设备坐标定义，一般也定义成矩形，由其左下角和右上角两点坐标来定义。所以，用户可以利用窗口来选择需要观察那一部分图形，而利用视图区来指定这一部分图形在屏幕上显示的位置。标准的窗口区和视图区一般都是矩形，其各边分别与坐标轴平行。

用户定义的图形从窗口区到视图区的输出过程如下：

从应用程序得到图形的用户坐标 (WC-World Coordinates) → 对窗口区进行裁剪 (WC) → 窗口区到视图区的规格化变换 (NDC-Normalized Device Coordinate) → 视图区从规格化设备系到设备坐标系的变换 (DC-Device Coordinate) → 在图形设备上输出图形。

下图是从用户坐标系取图形在设备坐标系下显示的示意图：

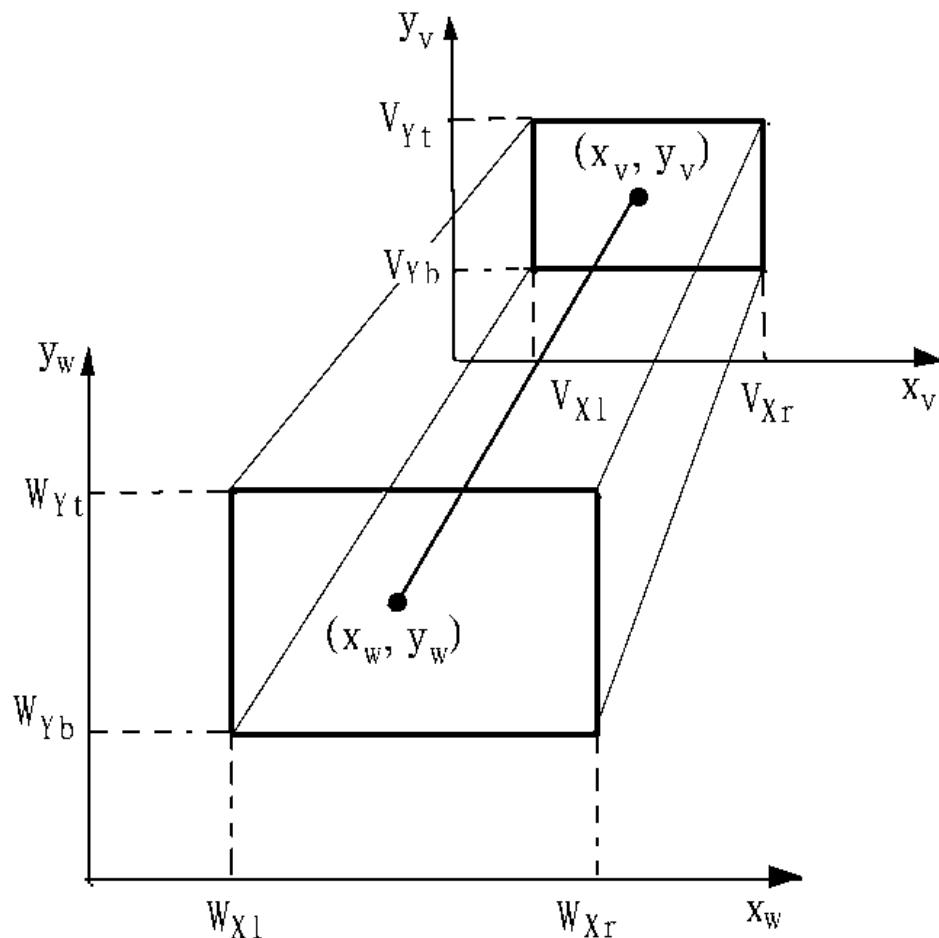


### 4.1.3 窗口区和视图区之间的坐标变换

由于窗口和视图是在不同坐标系中定义的，因此，在把窗口中图形信息转换到视图区之前，必须进行坐标变换，即把用户坐标系的坐标值转化为设备坐标系的坐标值。

设在用户坐标系下，矩形窗口左下角点坐标为 $(W_{x1}, W_{yb})$ ，右上角点坐标为 $(W_{xr}, W_{yt})$ 。屏幕中视图区的两个角点在设备坐标系下分别为 $(V_{x1}, V_{yb})$ 和 $(V_{xr}, V_{yt})$ 。则在窗口中的点 $(x_w, y_w)$ 对应视图区中的点 $(x_v, y_v)$ ，其变换公式为：

下图为示图：



记：

$$a = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}}$$

$$b = V_{xl} - \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} \cdot W_{xl}$$

$$c = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}}$$

$$d = V_{yb} - \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} \cdot W_{yb}$$

公式可以化简为：

写成矩阵形式为：

$$\begin{bmatrix} x_v & y_v & 1 \end{bmatrix} = \begin{bmatrix} X_w & Y_w & 1 \end{bmatrix} \times \begin{bmatrix} a & 0 & 0 \\ 0 & c & 0 \\ b & d & 1 \end{bmatrix}$$

上述变换是二维变换中比例变换和平移变换的组合变换。通过变换，可以实现将用户坐标系中窗口区中任意一点转换成设备坐标系中视图区中一点，从而可以把实际图形转换到具体输出设备的显示区。

## 4.2 直线段裁剪

直线段是组成其它图形的基础，其它任何图形，一般都能用不同直线段或近似为直线段组合形成。直线段裁剪是其他裁剪的基础，下面介绍几种基本的线段裁剪方法。

Cohen-Sutherland算法  
梁友栋—barskey算法

#### 4.2.1 点的裁剪

裁剪算法中最基本的也是最简单的是点的裁剪。设裁剪窗口是一个标准矩形，窗口左下角点坐标为( $W_{x1}, W_{yb}$ )，右上角点坐标为( $W_{xr}, W_{yt}$ )，若点P(x, y)满足下列不等式：

$$W_{x1} \leq x \leq W_{xr}$$

$$W_{yb} \leq y \leq W_{yt}$$

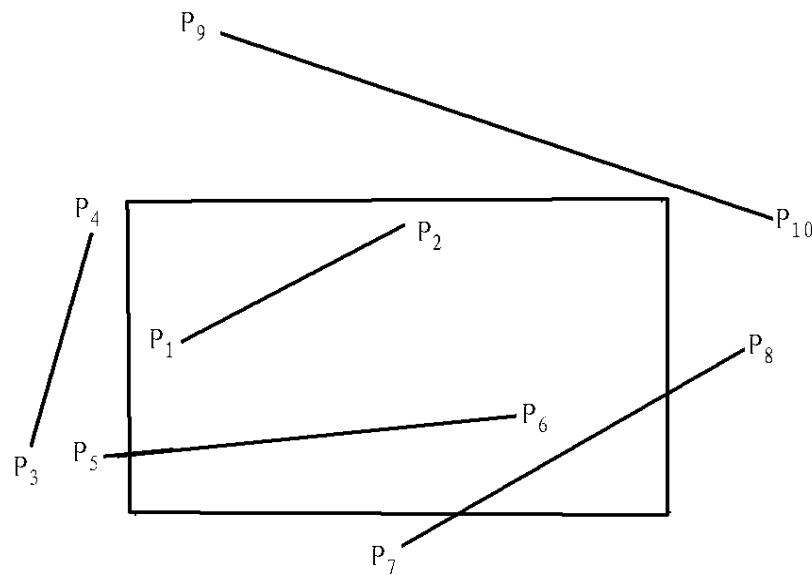
则该点在窗口内，其中等号表示点位于窗口边界上。这样的点属于可见点，应予保留，如果这四个不等式中有任何一个不满足，则该点在窗口外，应被裁剪掉。

## 4.2.2 编码裁剪算法

这是一个最早最流行的线段裁剪算法，是由Cohen和Sutherland提出的，也称为Cohen-Sutherland线段裁剪算法。该算法通过初始测试来减少需要计算的交点数目，从而加快线段裁剪算法的速度。

### 1. 线段同窗口的关系

在许多情形中，大多数线段不是在裁剪窗口之内就是在裁剪窗口之外，因此需要迅速决定是否接受或抛弃线段。对于任意一条直线段，它相对于裁剪窗口的可能关系如右图所示。



- (1) 若一线段两端点均位于裁剪窗口内，则该线段也位于窗口内且可见，如图中的线段 $P_1P_2$ ；
- (2) 当线段两端点均在窗口之外，且位于裁剪窗口的同一侧时，该线段必全部在窗口之外，从而是不可见的，如图中的线段 $P_3P_4$ 。
- (3) 但当线段两端点在窗口之外，而不位于裁剪窗口的同一侧时，该线段可能完全在窗口的外面，如图中的线段 $P_9P_{10}$ ，也可能不全部在窗口的外面，如图中的线段 $P_7P_8$ 。
- (4) 若线段的两端点一个位于窗口之内，一个位于窗口之外，则该线段部分可见，如图中的线段 $P_5P_6$ 。

对于上述(1)(2)两种情况，可以通过判断一线段两端点位置直接决定是否接受或抛弃线段。而对于(3)(4)两种情况，则需要进一步计算线段与裁剪窗口的交点，得到需要保留的部分线段。

## 2 编码方法

根据上面的分析，把包含窗口的平面区域沿窗口的四条边线分成九个区域，如下图所示。用四位二进制码来标识线段的端点位于九个区域中的哪一个区域内。四位编码规则如下：

第一位置1：

线段端点位于窗口左侧；

第二位置1：

线段端点位于窗口右侧；

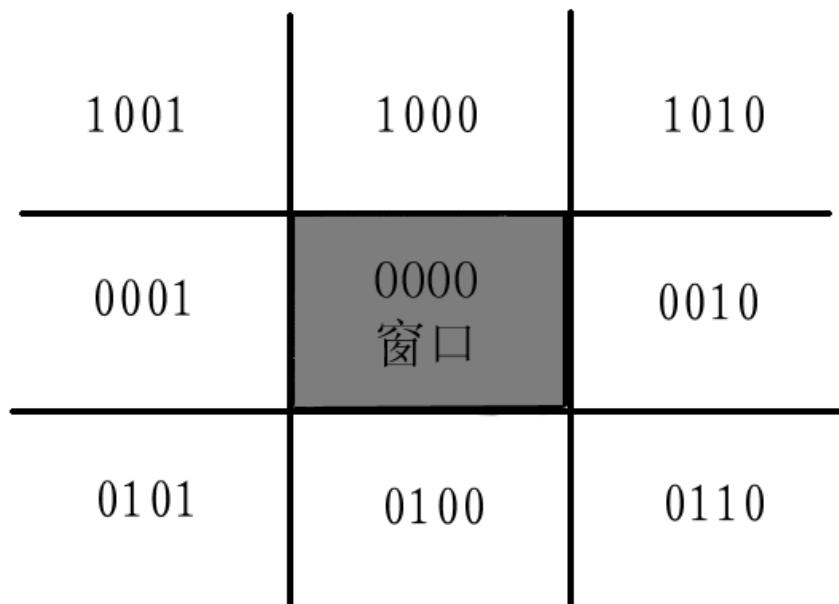
第三位置1：

线段端点位于窗口下面；

第四位置1：

线段端点位于窗口上面；

否则相应位置零。



由编码规则可知，若线段完全在裁剪窗口内，则线段两端点编码均为0000。当线段两端点均在窗口之外，且位于裁剪窗口的同一侧时，则两端点编码必有一位同时为1。

一旦给所有的线段端点建立了编码，就可以快速判断哪条线段完全在裁剪窗口内，哪条线段完全在窗口外。其方法如下：

(1) 线段两端点的编码逐位取逻辑“或”，若结果为零，则该线段必为完全可见，应保留。

(2) 线段两端点的编码逐位取逻辑“与”，若结果非零，则该线段必为完全不可见，因而可立即抛弃。

通过上面两步判定得知完全可见线段和完全不可见线段后，剩下线段两端点编码逻辑“或”不为零，或两端点编码的逻辑“与”均为零，这些线段可能部分可见，也可能完全不可见。

对于不能判定完全可见和完全不可见的线段，需要对线段进行再分割，即找到与窗口一个边框的交点，根据交点位置，也赋予4位代码，并对分割后的线段进行检查，或者接受，或者舍弃，或者再次进行分割。重复这一过程，直到全部线段均被舍弃或被接受为止。线段与窗口边求交次序的选择是任意的。但是无论哪种次序，对于有些线段的裁剪，可能不得不重复4次，计算与4条窗口边的交点。

下面介绍交点的求法。设线段两端点坐标分别为  $P_1(x_1, y_1)$  和  $P_2(x_2, y_2)$ ，通过  $P_1(x_1, y_1)$  和  $P_2(x_2, y_2)$  两点的直线方程为：

$$y = m(x - x_1) + y_1 \quad \text{或} \quad x = \frac{1}{m}(y - y_1) + x_1$$

其中

利用两点式直线方程，线段与窗口四条边线的交点坐标，可分别确定如下：

同窗口左边线的交点：
$$\begin{cases} x = W_{xl} \\ y = m(W_{xl} - x_1) + y_1 \end{cases}$$

同窗口右边线的交点：
$$\begin{cases} x = W_{rl} \\ y = m(W_{rl} - x_1) + y_1 \end{cases}$$

同窗口上边线的交点：
$$\begin{cases} y = W_{yt} \\ x = \frac{1}{m}(W_{yt} - y_1) + x_1 \end{cases}$$

同窗口下边线的交点：
$$\begin{cases} y = W_{yb} \\ x = \frac{1}{m}(W_{yb} - y_1) + x_1 \end{cases}$$

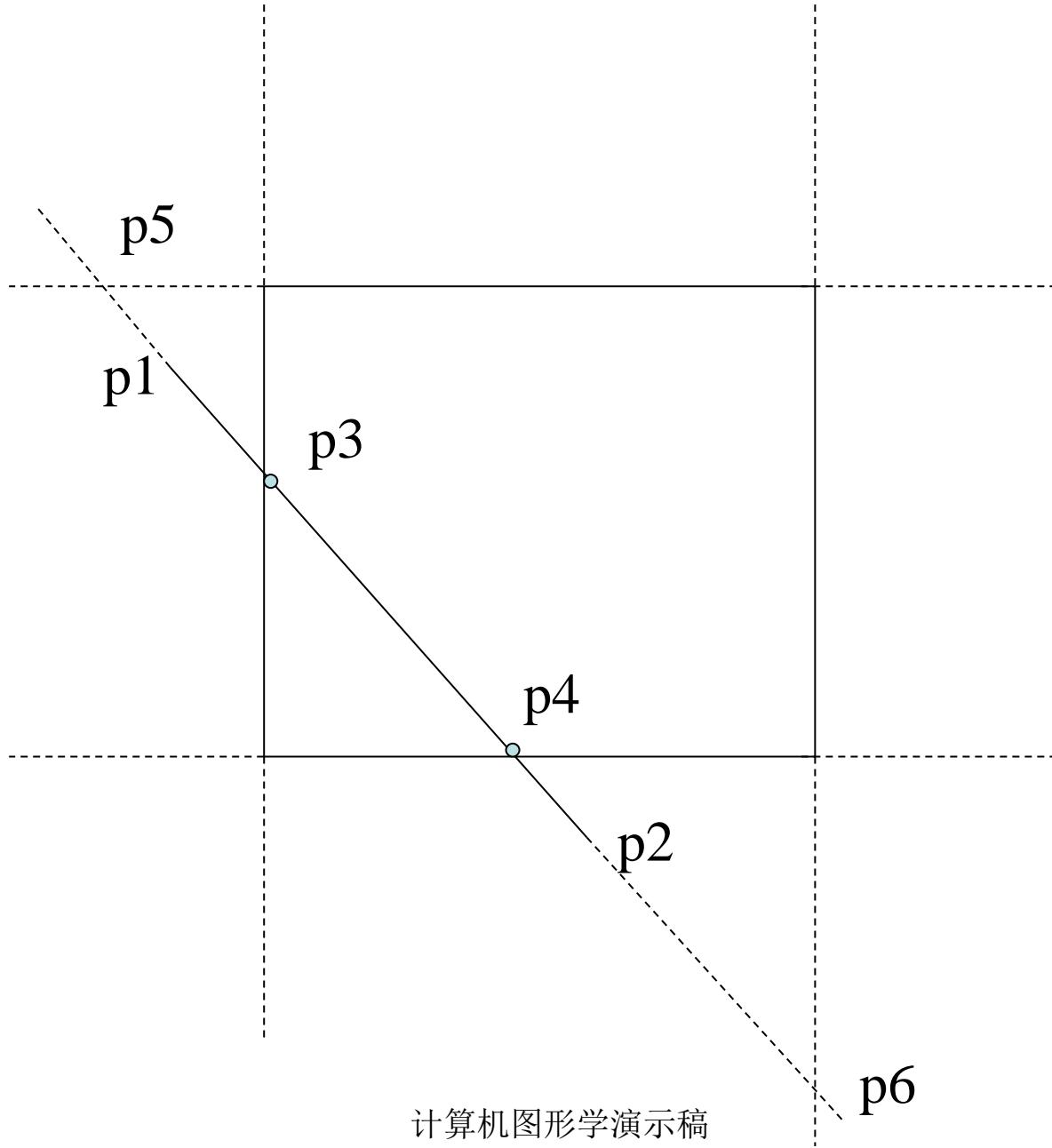
在上面的求交点计算过程中，需要考虑对某些特殊情形的处理。若直线的斜率m为无穷大，则直线平行于窗口的左边和右边，故仅需检查直线与上、下两边的交点。同样，若直线斜率为零，则它平行于窗口的上、下两边，这时仅需检查直线与左、右两边的交点。

#### 4.2.3. 参数化线段裁剪算法

更快的线段裁剪算法基于线段的参数化方程的分析，它是由梁友栋和Barsky提出的，也称为梁友栋–Barsky线段裁剪算法。

设线段两端点坐标分别为 $P_1(x_1, y_1)$  和 $P_2(x_2, y_2)$ ，则其参数化直线方程可写成下列形式：

$$\begin{cases} x = x_1 + u(x_2 - x_1) = x_1 + u\Delta x \\ y = y_1 + u(y_2 - y_1) = y_1 + u\Delta y \end{cases} \quad 0 \leq u \leq 1$$



坐标(x, y)表示直线上两端点之间的任一点。当u=0时，得点P<sub>1</sub>，当u=1时，得点P<sub>2</sub>。线段的裁剪条件可以由下面的不等式表示：

$$W_{x1} \leq x_1 + u \Delta x \leq W_{xr}$$

$$W_{yb} \leq y_1 + u \Delta y \leq W_{yt}$$

这四个不等式可以表示为：

$$up_k \leq q_k \quad k=1, 2, 3, 4$$

其中，参数p, q定义为：

$$p_1 = -\Delta x, \quad q_1 = x_1 - W_{x1}$$

$$p_2 = \Delta x, \quad q_2 = W_{xr} - x_1$$

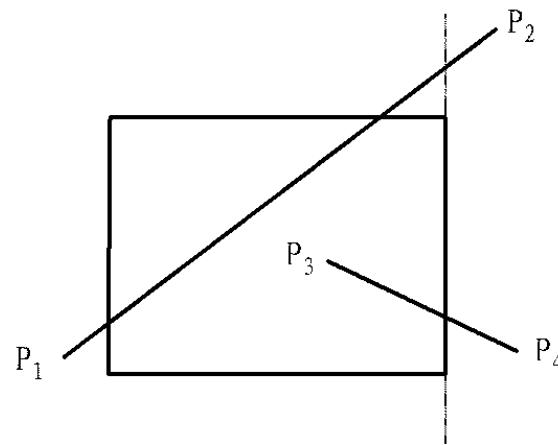
$$p_3 = -\Delta y, \quad q_3 = y_1 - W_{yb}$$

$$p_4 = \Delta y, \quad q_4 = W_{yt} - y_1$$

下标k=1, 2, 3, 4分别对应裁剪窗口的左、右、下、上四条边界线。

如果线段平行于裁剪窗口的某两边界，则必有相应的  $p_k = 0$ ，如果还满足  $q_k < 0$ ，则线段的端点位于窗口外部，即线段在窗口外，应该舍弃。如果  $q_k \geq 0$ ，线段在窗口内外需要继续判断。

当  $p_k < 0$  时，直线是从裁剪窗口第  $k$  条边界线的外部延伸到内部。例如当  $p_1 < 0$  时，则  $x_2 > x_1$ ，直线必然从裁剪窗口的左边界线的外部进入内部，如下图中的线段  $P_1P_2$ 。当  $p_k > 0$  时，直线是从裁剪窗口第  $k$  条边界线的内部延伸到外部。例如  $p_2 > 0$  时，则  $x_2 > x_1$ ，直线必然从裁剪窗口的右边界线的内部进入外部，如下图中的线段  $P_3P_4$ 。



当 $p_k$ 不等于零时，可以计算出线段与第k条裁剪窗口边界线的交点参数：

$$r_k = \frac{q_k}{p_k}$$

根据定义，对于每条线段， $p_k$ 中必有两个小于零，而另两个大于零。对于小于零的 $p_k$ ，直线同第k条裁剪窗口边线是从外到内相遇的，此时如果线段同第k条裁剪窗口边界线有交点的话，是参数u从0变大时遇到的，这时计算出相应的 $r_k$ 值，取0和各个 $r_k$ 值之中的最大值记为 $u_1$ 。与此相反，对于大于零的 $p_k$ ，计算出相应的 $r_k$ 值，取1和各个 $r_k$ 值之中的最小值记为 $u_2$ 。两个参数 $u_1$ 和 $u_2$ 定义了在裁剪窗口内的线段部分。如果 $u_1 > u_2$ ，则线段完全落在裁剪窗口之外，应被舍弃。否则被裁剪线段可见部分的端点由参数 $u_1$ 和 $u_2$ 计算出来。

## 梁友栋线的裁剪算法例子

设:  $W_{xl}=2$ ;  $W_{xr}=4$ ;  $W_{yb}=2$ ;  $W_{yt}=4$

被裁剪线段的两端点:  $(1, 2)$ ,  $(5, 3)$

计算:  $\Delta x=5-1=4$

$$\Delta y=3-2=1$$

$$p_1=-4, q_1=-1$$

$$p_2=4, q_2=3$$

$$p_3=-1, q_3=0$$

$$p_4=1, q_4=2$$

$p_1, p_3$ 小于0, 决定 $u_1$ , 取0与 $r_1$ 和 $r_3$ 中的大者,  $u_1=1/4$   
 $p_2, p_4$ 大于0, 决定 $u_2$ , 取1与 $r_2$ 和 $r_4$ 中的小者,  $u_2=3/4$   
两交点由 $u_1, u_2$ 决定:

计算: (1)  $x = 1 + \frac{1}{4} \times 4 = 2$

$$y = 2 + \frac{1}{4} \times 1 = 2\frac{1}{4}$$

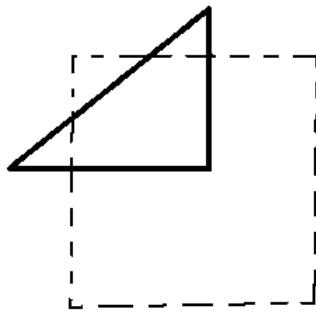
(2)  $x = 1 + \frac{3}{4} \times 4 = 4$

$$y = 2 + \frac{3}{4} \times 1 = 2\frac{3}{4}$$

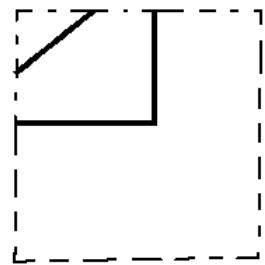
通常梁友栋-Barsky算法比Cohen-Sutherland算法更有效，因为需要计算的交点数目减少了。一次计算就可以确定出线段的可见性及可见部分。这两种线段裁剪算法都可以扩展为三维线段裁剪算法。

## 4.3 多边形裁剪

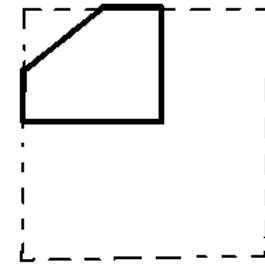
尽管多边形是由线段组成的，但却不能简单地将线段裁剪方法用于多边形裁剪。这是因为，在线段裁剪中，是把一条线段的两个端点孤立地加以考虑的，而多边形是由一些有序的线段组成的，裁剪后的多边形仍应保持原多边形各边的连接顺序。还有，一个完整的封闭多边形经裁剪后一般不再是封闭的，需要用裁剪窗口边界适当部分来形成一个或多个封闭区域。所以，多边形裁剪后的输出应该是定义裁剪后的多边形边界的顶点序列。下图示出了一个多边形裁剪的例子。图(a)中是矩形裁剪窗口和需要裁剪的三角形，图(b)中是不正确的非封闭裁剪结果，图(c)是正确的封闭裁剪结果。



(a) 待裁剪图形



(b) 不正确的结果

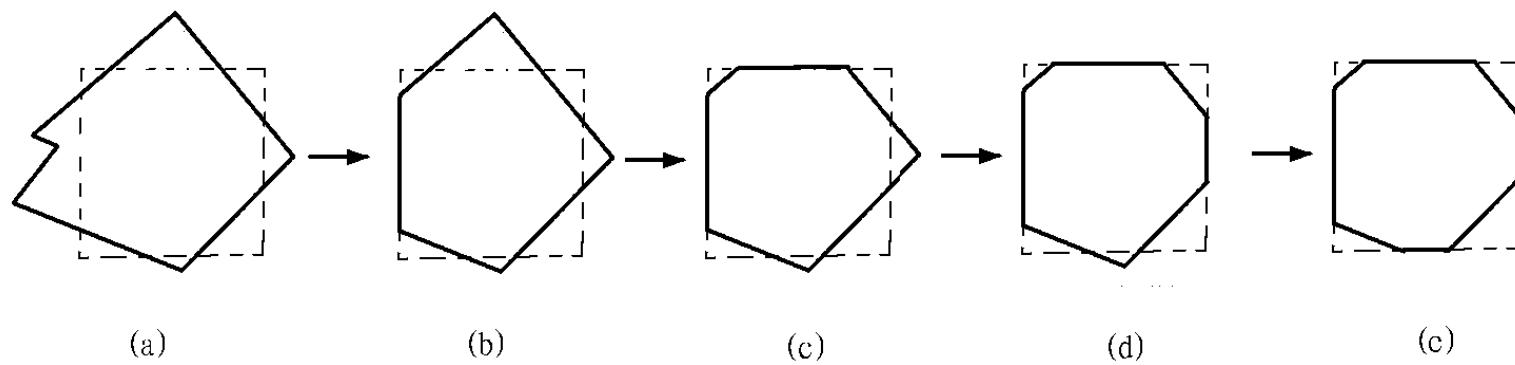


(c) 正确的结果

多边形裁剪方法很多，下面介绍两种常用的多边形裁剪算法。

#### 4.3.1 逐边裁剪法

这个算法是由Sutherland和Hodgman提出来的，也称为Sutherland-Hoeman多边形裁剪方法。对于矩形裁剪窗口的一条边界线，如果称窗口区域所在的一侧为内侧，另一侧为外侧，则方法的具体做法是：每次用裁剪窗口的一条边界对要裁剪的多边形进行裁剪，把落在此边界外侧的多边形部分去掉，只保留内侧部分，形成一个新的多边形，并把它作为下一次待裁剪的多边形。若依次用裁剪窗口的4条边界对要裁剪的原始多边形进行裁剪，则最后形成了裁剪出来的多边形。下图说明了这个过程。

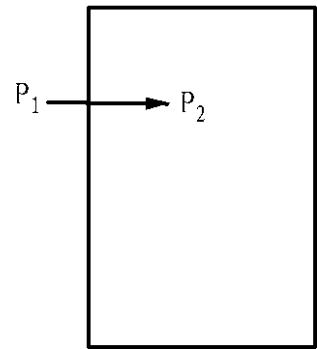


在剪取过程中，实际是多边形的每一边与窗口的一边界进行比较，从而确定它们的位置关系。多边形是用顶点表示的，相邻的一对顶点构成一条边。具体实现时首先把待裁剪多边形各顶点按照一定方向有次序地组成顶点序列。然后用窗口的一条边界裁剪多边形，产生新的顶点序列。

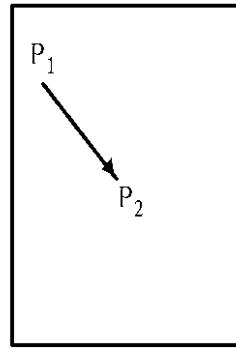
当多边形顶点序列中一条边的起点和终点被一窗口边界裁剪时，会遇到边与窗口的四种情况之一，做如下处理：

- ① 如果起点在窗口边界外侧而终点在窗口边界内侧，则将多边形的该边与窗口边界的交点和终点都加到输出顶点表中；
- ② 如果两顶点都在窗口边界内侧，则只有终点加入输出顶点表中；
- ③ 如果起点在窗口边界内侧而终点在外侧，则只将与窗口边界的交点加到输出顶点表中；
- ④ 如果两个点都在窗口边界外侧，输出表中不增加任何点。

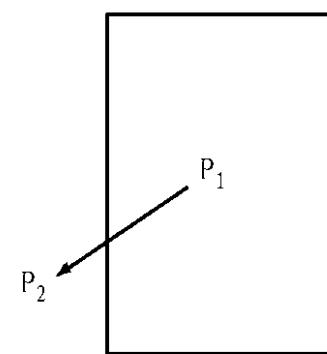
边与窗口的四种可能的关系如下图所示：



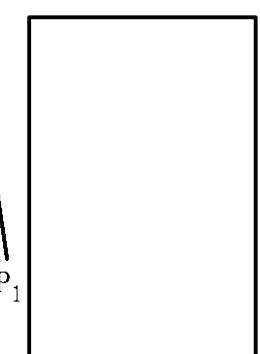
外 → 内



内 → 内



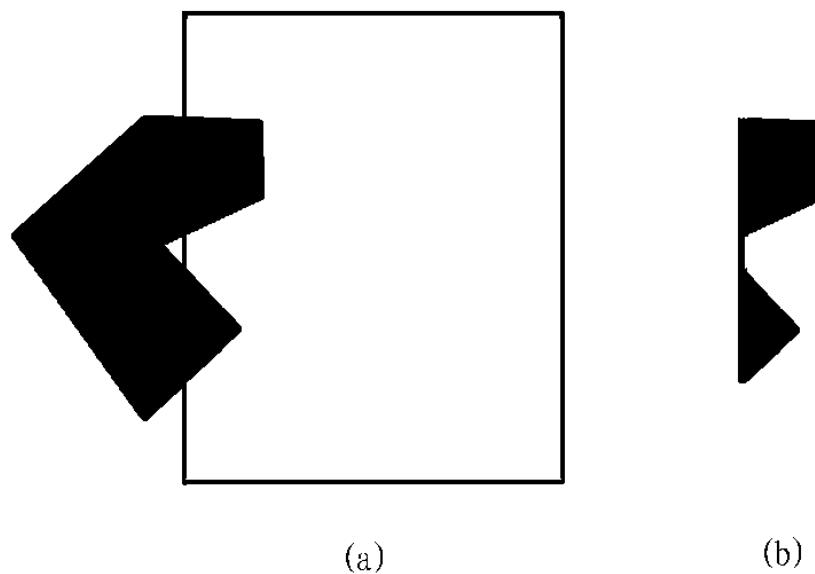
内 → 外



外 → 外

按照上述处理方法，窗口的一条裁剪边界处理完所有顶点后，其输出是一个新的封闭多边形顶点序列表，用于窗口的下一条边界继续裁剪。所有的窗口边界都裁剪完后，得到的是裁剪后的多边形顶点序列，它自然也是封闭的。

凸多边形可以用Sutherland-Hodgeman算法获得正确的裁剪结果，但是对凹多边形的裁剪，可能出现如下图所示多余的连线。这种情况在裁剪后的多边形有两个或者多个分离部分的时候出现。因为只有一个输出顶点表，所以表中最后一个顶点总是连着第一个顶点。如下面所述的另一算法可以消除这一问题。

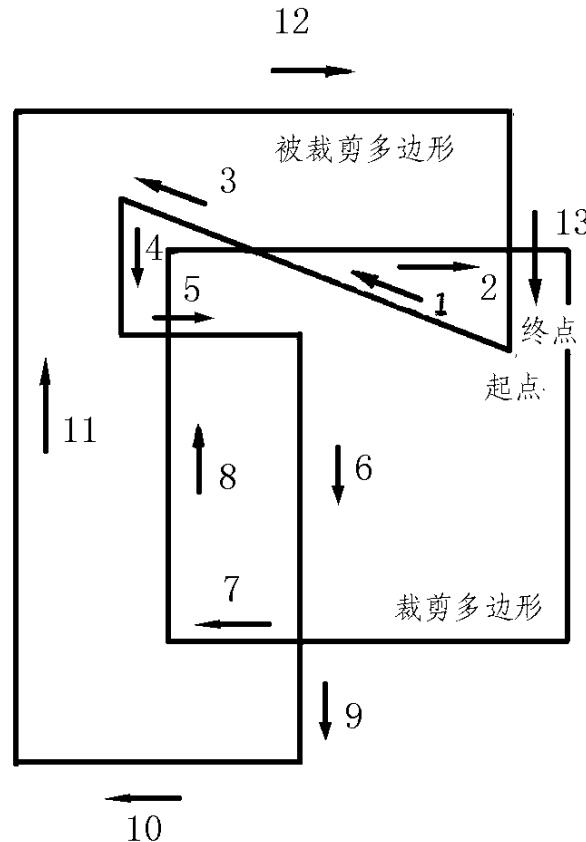


### 4.3.2 双边裁剪法

此算法是由Weiler和Atherton提出来的。算法的基本做法是：有时沿着多边形边的方向来处理顶点，有时沿着窗口的边界方向来处理，从而避免产生多余的连线。设被裁剪多边形和裁剪窗口都按顺时针确定排列方向，因此，沿多边形的一条边前进，其右边为多边形的内部。算法首先沿多边形的任一点出发，跟踪检测多边形的每一条线段，当线段与裁剪窗口边界相交时：

- ①如果线段起点在窗口外部而终点在窗口内部，则求出交点，输出线段可见部分，继续沿多边形方向往下处理；
- ②如果线段起点在窗口内部而终点在窗口外部，则求出交点，输出线段可见部分。从此交点开始，沿着窗口边界方向往前检测，找到一个多边形与窗口边界的新交点后，输出由前交点到此新交点之间窗口边界上的线段；
- ③返回到前交点，再沿着多边形方向往下处理，直到处理完多边形的每一条边，回到起点为止。

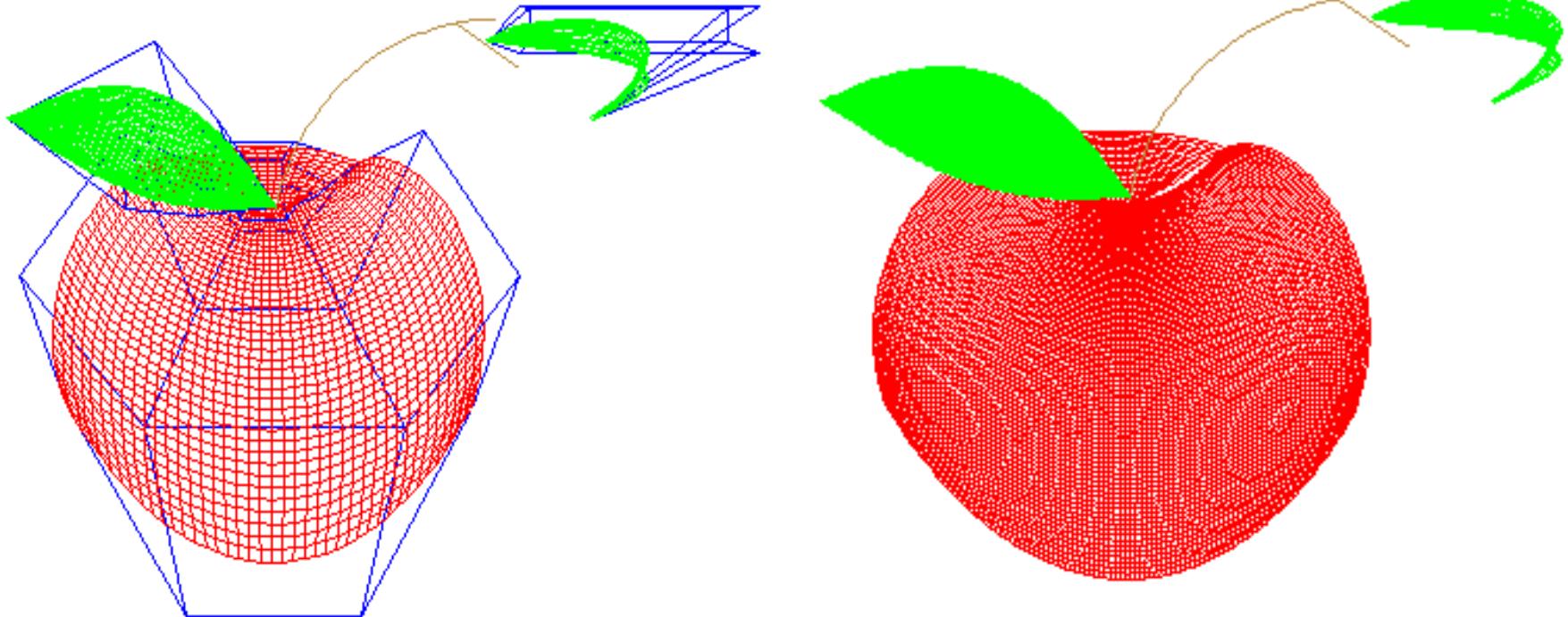
下图说明了双边裁剪算法的执行过程。

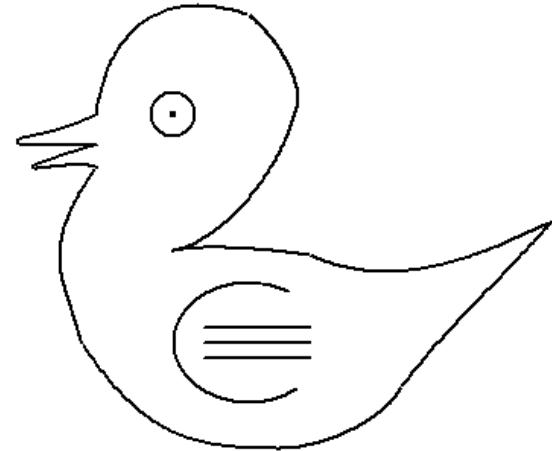
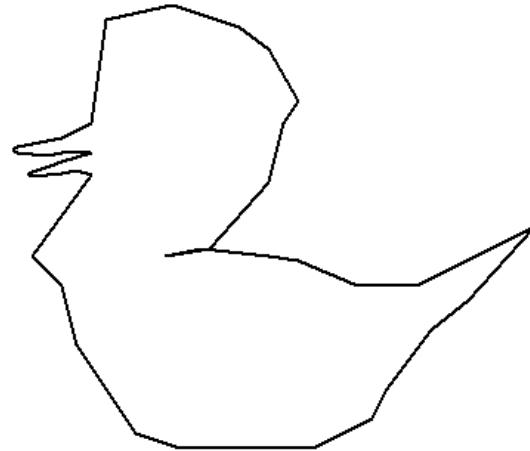
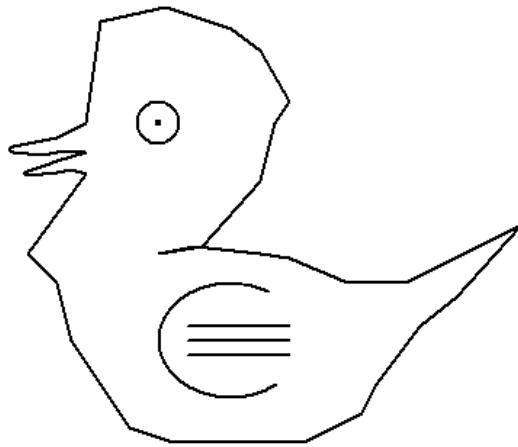
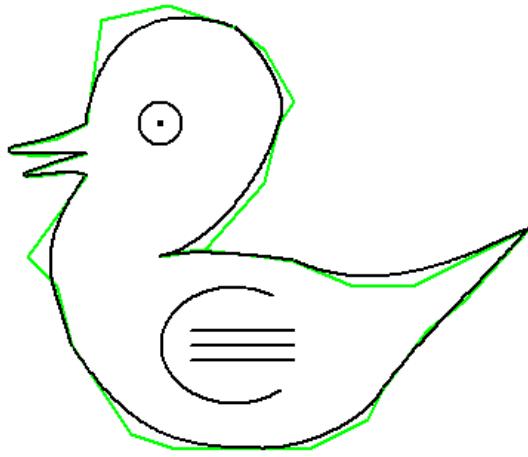


Weiler-Atherton算法可适用于任何凸的或凹的多边形裁剪，不会产生多余连线。

# 第五章 自由曲线

自由曲线和曲面是指那些形状比较复杂、不能用初等解析函数直接表示出来的曲线和曲面。汽车车身、飞机机翼和轮船船体等的曲线和曲面均属于这一类。一般情况下，它们需要利用插值或逼近的方法，对型值点进行拟合，得到拟合曲线和曲面。





## 5.1 曲线的参数表示及连续性

### 5.1.1 曲线的参数表示

如果用 $u$ 表示参数，二维空间自由曲线的参数方程可以记为：

$$x = x(u), y = y(u) \quad u \in [0, 1]$$

二维空间曲线上一点的参数表示为：

$$P(u) = [x(u), y(u)]$$

三维空间自由曲线的参数方程表示为：

$$x = x(u), y = y(u), z = z(u); \quad u \in [0, 1]$$

曲线上一点的参数表示为：

$$P(u) = [x(u), y(u), z(u)]$$

## 5.1.2 插值、逼近和拟合

给出一组有序的型值点列，根据应用的要求来得到一条光滑曲线，通常采用两种不同的方法，即插值方法和逼近方法。

插值方法要求生成的曲线通过每个给定的型值点。曲线插值方法有多项式插值，分段多项式插值，样条函数插值等。

逼近方法要求生成的曲线靠近每个型值点，但不一定要求通过每个点。逼近方法有最小二乘法，Bezier方法，B样条方法等。

用插值或逼近来构造曲线的方法通称为曲线拟合方法。

### 5.1.3 参数连续性条件

0阶导数连续性，记作 $C^0$ 连续，是指曲线相连。即第一个曲线段在 $u=1$ 处的 $x, y, z$ 值与第二个曲线段在 $u=0$ 处的 $x, y, z$ 值相等。

一阶导数连续性，记作 $C^1$ 连续，指两个相邻曲线段在交点处有一阶导数相同。

二阶导数连续性，记作 $C^2$ 连续，指两个相邻曲线段在交点处有一阶和二阶导数相同。高阶参数连续性可类似定义。

0阶几何连续性，记为 $G^0$ 连续，与0阶导数连续性相同。即两个曲线段在公共点处有相同的坐标。

一阶几何连续性，记为 $G^1$ 连续，指一阶导数在两个相邻段的交点处成比例，而大小不一定相等。

二阶几何连续性，记为 $G^2$ 连续，指两个曲线段在相交处其一阶和二阶导数均成比例。 $G^2$ 连续下，两个曲线段在交点处的曲率相等。

## 5.1.4 参数样条曲线

### 1. 样条曲线

在计算机图形学中，术语**样条曲线**指由多项式曲线段连接而成的曲线，在每段的边界处满足特定连续条件。样条用来设计曲线和曲面形状，典型的**CAD**应用包括汽车、飞机和航天飞机表面设计以及船壳设计。

### 2. 参数样条表示

在计算机图形学应用中使用几种不同的样条描述。每种描述是一个带有某特定边界条件多项式的特殊类型。

例如空间一条曲线用三次参数方程可以表示如下：

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z \quad u \in [0, 1]$$

或

$$P(u) = au^3 + bu^2 + cu + d \quad u \in [0, 1]$$

如果曲线的边界条件设定为端点处满足给定坐标值  $P(0)$  和  $P(1)$ ，同时端点处的导数也满足给定值  $P'(0)$  和  $P'(1)$ 。这四个边界条件对决定上式中方程的系数是充分条件。例如已知  $x(0)$ 、 $x(1)$ 、 $x'(0)$  和  $x'(1)$ ，则  $a_x$ 、 $b_x$ 、 $c_x$  和  $d_x$  就可以求出。解出各个系数后的上式就是一种确定的三次参数样条表示式。

## 5.2 三次样条插值曲线

实际上，通常使用的是三次样条曲线。这是因为三次多项式曲线是能使曲线段的端点通过特定的点，并能使曲线段在连接处保持位置和斜率连续性的最低阶次的多项式。与更高次多项式相比，三次多项式只需较少的计算和存储且较稳定，而更低次多项式又难以用来描述复杂形状的曲线。

如果想使用三次样条获得一条通过各个型值点的连续曲线，需要利用三次样条分段插值得到通过每个型值点的分段三次样条曲线。对 $n+1$ 个型值点，分段插值时段与段之间要建立合适的边界条件，既能使各段之间平滑连续，又可建立起足够的方程数，求出所有的系数。

### 5.2.1 Hermite 样条插值曲线

Hermite 样条插值（以法国数学家 Charles Hermite 命名）使用型值点和型值点处的一阶导数建立边界条件。设  $P_k$  和  $P_{k+1}$  为第  $K$  个和第  $K+1$  个型值点，Hermite 样条插值边界条件规定为：

$$P(0) = P_k$$

$$P(1) = P_{k+1}$$

$$P'(0) = D_k$$

$$P'(1) = D_{k+1}$$

其中， $D_k$  和  $D_{k+1}$  分别为  $P_k$  和  $P_{k+1}$  处的一阶导数。

将参数方程写成矩阵形式为：

$$p(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

将边界条件  $P(0) = P_k$  和  $P(1) = P_{k+1}$  代入方程得：

$$P_k = d$$

$$P_{k+1} = a + b + c + d$$

一阶导数为：

$$p'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

将边界条件  $p'(0) = D_k$  和  $p'(1) = D_{k+1}$  代入方程得：

$$D_k = c$$

$$D_{k+1} = 3a + 2b + c$$

由边界条件构成的4个方程联立：

$$P_k = d$$

$$P_{k+1} = a + b + c + d$$

$$D_k = c$$

$$D_{k+1} = 3a + 2b + c$$

写成矩阵的形式为：

$$\begin{bmatrix} P_k \\ P_{k+1} \\ D_k \\ D_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

解此方程得：

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ D_k \\ D_{k+1} \end{bmatrix}$$
$$= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ D_k \\ D_{k+1} \end{bmatrix}$$

$$M_h = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

称为Hermite矩阵，插值样条参数方程可以写成

$$P(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot M_h \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ D_k \\ D_{k+1} \end{bmatrix}$$

将上式展开写成代数形式为：

$$\begin{aligned} P(u) &= P_k(2u^3 - 3u^2 + 1) + P_{k+1}(-2u^3 + 3u^2) \\ &\quad + D_k(u^3 - 2u^2 + u) + D_{k+1}(u^3 - u^2) \\ &= P_k H_0(u) + P_{k+1} H_1(u) + D_k H_2(u) + D_{k+1} H_3(u) \end{aligned}$$

其中

$$H_0(u) = 2u^3 - 3u^2 + 1$$

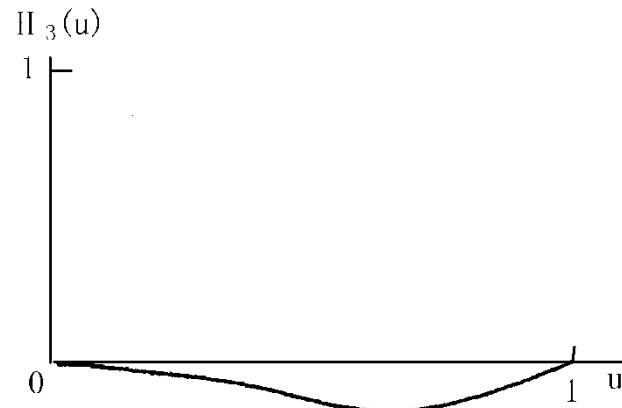
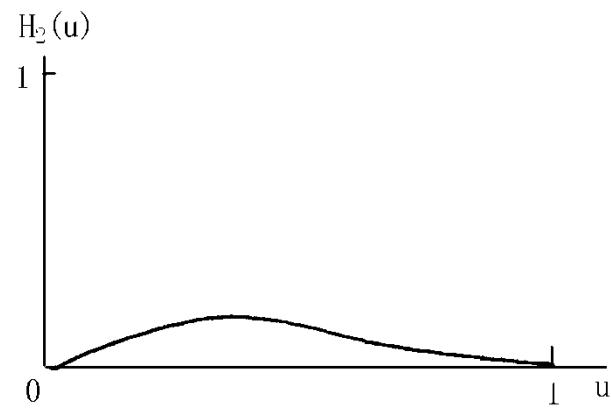
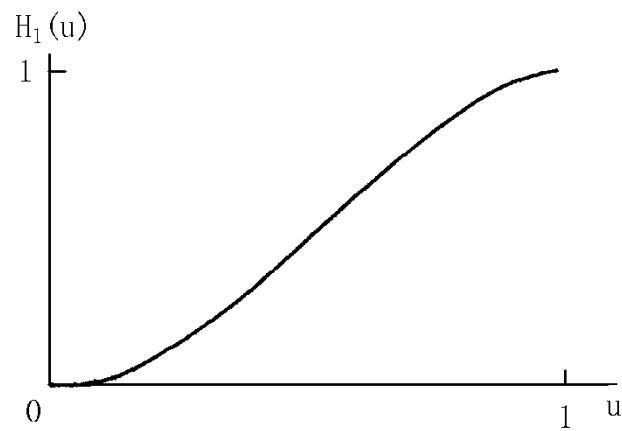
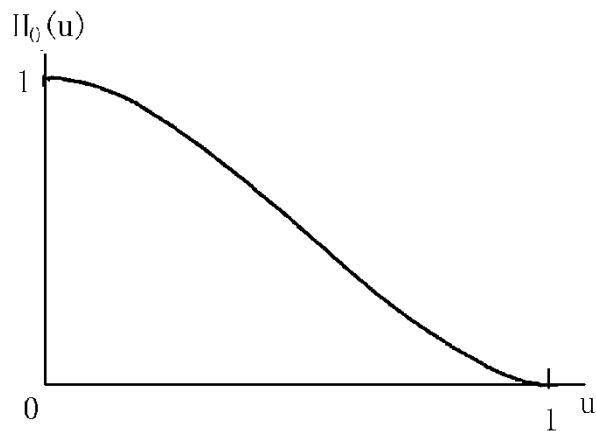
$$H_1(u) = -2u^3 + 3u^2$$

$$H_2(u) = u^3 - 2u^2 + u$$

$$H_3(u) = u^3 - u^2$$

称为Hermite样条调和函数，因为它们调和了边界约束值，使在整个参数范围内产生曲线的坐标值。调和函数仅与参数u有关，而与初始条件无关，且调和函数对于空间的三个坐标分量(x, y, z)是相同的。

下图表示出Hermite样条曲线的调和函数随参数u变化的曲线



还可将方程整理成如下形式：

$$\begin{aligned} P(u) = & (2P_k - 2P_{k+1} + D_k + D_{k+1}) u^3 \\ & + (-3P_k + 3P_{k+1} - 2D_k - D_{k+1}) u^2 + D_k u + P_k \end{aligned}$$

写成坐标分量形式则如下式：

$$\begin{aligned} x(u) = & (2x_k - 2x_{k+1} + x_k' + x_{k+1}') u^3 \\ & + (-3x_k + 3x_{k+1} - 2x_k' - x_{k+1}') u^2 + x_k' u + x_k \end{aligned}$$

$$\begin{aligned} y(u) = & (2y_k - 2y_{k+1} + y_k' + y_{k+1}') u^3 \\ & + (-3y_k + 3y_{k+1} - 2y_k' - y_{k+1}') u^2 + y_k' u + y_k \end{aligned}$$

$$\begin{aligned} z(u) = & (2z_k - 2z_{k+1} + z_k' + z_{k+1}') u^3 \\ & + (-3z_k + 3z_{k+1} - 2z_k' - z_{k+1}') u^2 + z_k' u + z_k \end{aligned}$$

如果是平面曲线，则只有x和y分量。

例：给定9个型值点，其中起始点和终止点是同一个点，从而其特征多边形是一个首尾相接的封闭多边形，具体坐标位置如下：

(100, 300) , (120, 200) , (220, 200) ,  
(270, 100) , (370, 100) , (420, 200) ,  
(420, 300) , (220, 280) , (100, 300)

假定各点处的一阶导数数值如下：

(70, -70) , (70, -70) , (70, -70) , (70, -70) ,  
(70, 70) , (70, 70) , (-70, 70) , (-70, 70) ,  
(70, -70)

用Hermite插值方法绘制曲线。

## Hermit三次曲线算法主要实现子程序实例

```
void HermitCurve()
{
    int i;
    int arry1[9][2]={100,300, 220,200,120,200,,270,100,370,100,420,200,420,300,220,280,100,300};
    int arry2[9][2]={70,-70,70,-70,70,-70,70,-70,70,70,70,70,-70,70,-70,70,70,-70};
    for(i=0;i<8;i++)
    {
        line(arry1[i][0],arry1[i][1],arry1[i+1][0],arry1[i+1][1]);
        Hermit3(hdc,arry1,arry2,i,100);
    }
}

void Hermit3(int arry1[2][2],int arry2[2][2],int n,int steps)
{
    int i,x,y,k1,k2,k3,k4,m1,m2,m3,m4;
    float a0,a1,a2,a3,b0,b1,b2,b3,dt,t,t2,t3;
    k1=arry1[n][0];
    k2=arry1[n+1][0];
    k3=arry2[n][0];
    k4=arry2[n+1][0];
    m1=arry1[n][1];
    m2=arry1[n+1][1];
    m3=arry2[n][1];
    m4=arry2[n+1][1];
```

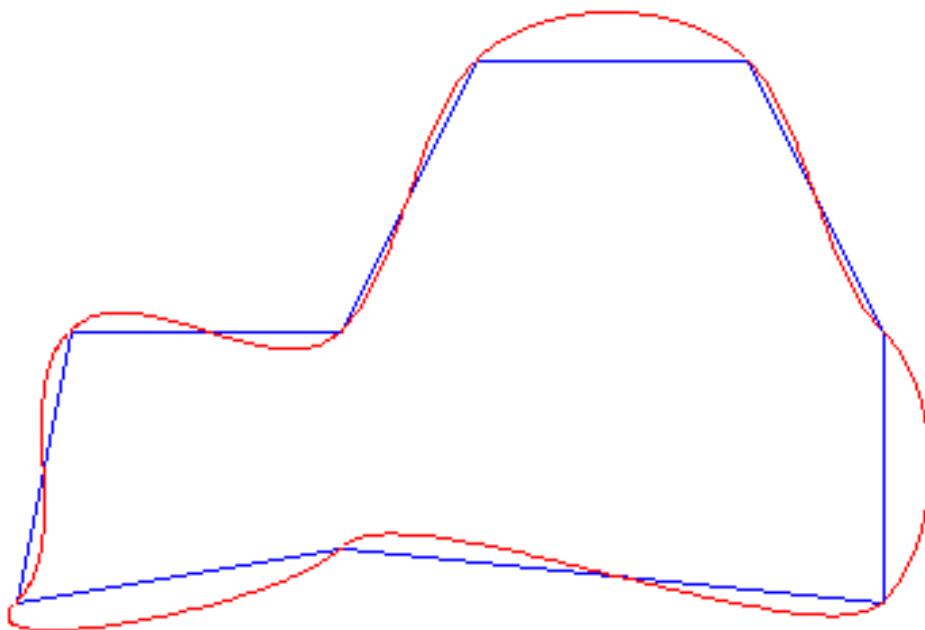
```
a0 = k1;  
a1 = k3;  
a2 = -3*k1+3*k2-2*k3-k4;  
a3 = 2*k1-2*k2+k3+k4;  
b0 = m1;  
b1 = m3;  
b2 = -3*m1+3*m2-2*m3-m4;  
b3 = 2*m1-2*m2+m3+m4;
```

```
dt = 1.0/steps;
```

```
for(i=1;i<steps;i++)  
{  
    t = i*dt;  
    t2 = t*t;  
    t3 = t*t2;  
    x = a0+a1*t+a2*t2+a3*t3;  
    y = b0+b1*t+b2*t2+b3*t3;
```

```
    LineTo(x,y);  
    Sleep(5);  
}  
}
```

### Hermit三次曲线



### Hermit三次曲线绘制演示

Hermite样条曲线比较简单，易于理解，但要求确定每个型值点处的一阶导数作为初始条件，这是很不方便的，有时甚至是难于实现的。更好的做法是不需要输入曲线斜率值或其它几何信息就能生成样条曲线。

## 5.2.2 Cardinal 样条曲线

象Hermite样条曲线一样，Cardinal样条曲线也是插值分段三次曲线，且边界条件也是限定每段曲线端点处的一阶导数。与Hermite样条曲线的区别是，在Cardinal样条曲线中端点处的一阶导数值是由两个相邻型值点坐标来计算的。

设相邻的四个型值点分别记为 $P_{k-1}$ 、 $P_k$ 、 $P_{k+1}$ 、 $P_{k+2}$ ，Cardinal样条插值方法规定 $P_k$ 、 $P_{k+1}$ 两型值点间插值多项式的边界条件为：

$$P(0) = P_k$$

$$P(1) = P_{k+1}$$

$$P'(0) = (1 - t)(P_{k+1} - P_{k-1})/2$$

$$P'(1) = (1 - t)(P_{k+2} - P_k)/2$$

其中 $t$ 为一可调参数，称为张力参数，可以控制Cardinal样条曲线型值点间的松紧程度。

记 $S=(1-t)/2$ , 用类似Hermite曲线样条中的方法, 将Cardinal边界条件代入参数方程, 可以得到矩阵表达式:

$$\begin{aligned}
 P(u) &= [u^3 \quad u^2 \quad u \quad 1] \cdot \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{k-1} \\ P_k \\ P_{k+1} \\ P_{k+2} \end{bmatrix} \\
 &= [u^3 \quad u^2 \quad u \quad 1] \cdot M_c \cdot \begin{bmatrix} P_{k-1} \\ P_k \\ P_{k+1} \\ P_{k+2} \end{bmatrix}
 \end{aligned}$$

其中

$$M_c = \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

称为Cardinal矩阵。

将式子展开写成代数形式为：

$$\begin{aligned} P(u) = & P_{k-1}(-su^3 + 2su^2 - su) + P_k((2 - s)u^3 \\ & + (s - 3)u^2 + 1) + P_{k+1}((s - 2)u^3 \\ & + (3 - 2s)u^2 + su) + P_{k+2}(su^3 - su^2) \end{aligned}$$

其中

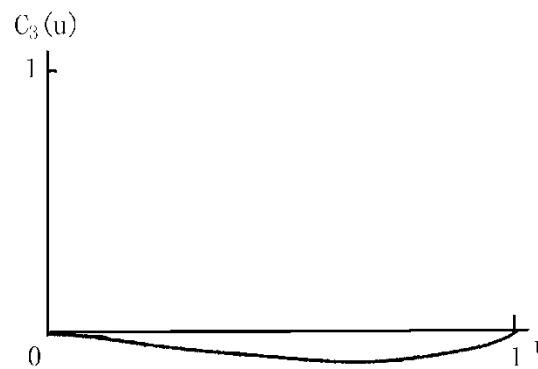
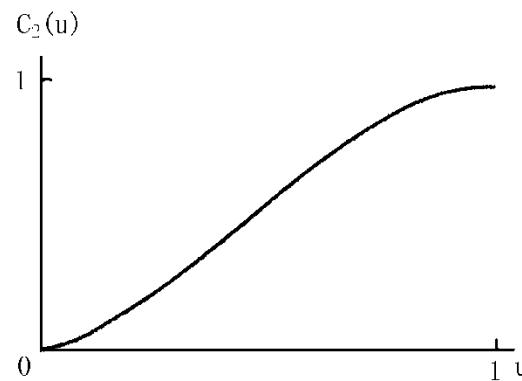
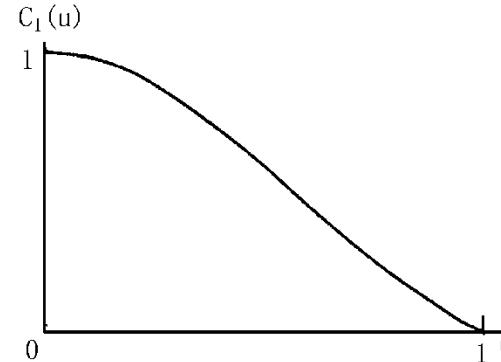
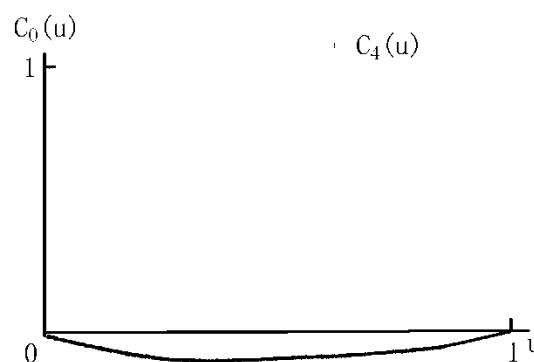
$$C_0(u) = -su^3 + 2su^2 - su$$

$$C_1(u) = (2 - s)u^3 + (s - 3)u^2 + 1$$

$$C_2(u) = (s - 2)u^3 + (3 - 2s)u^2 + su$$

$$C_3(u) = su^3 - su^2$$

称为Cardinal样条调和函数。下图是Cardinal样条调和函数的曲线图。



还可将方程整理成如下形式：

$$\begin{aligned} P(u) = & (-sP_{k-1} + (2-s)P_k + (s-2)P_{k+1} + sP_{k+2}) u^3 \\ & + (2sP_{k-1} + (s-3)P_k + (3-2s)P_{k+1} - sP_{k+2}) u^2 \\ & + (-sP_{k-1} + sP_{k+1}) u + P_k \end{aligned}$$

将方程式写成坐标分量的形式为：

$$\begin{aligned} x(u) = & (-sx_{k-1} + (2-s)x_k + (s-2)x_{k+1} + sx_{k+2}) u^3 \\ & + (2sx_{k-1} + (s-3)x_k + (3-2s)x_{k+1} - sx_{k+2}) u^2 \\ & + (-sx_{k-1} + sx_{k+1}) u + x_k \\ y(u) = & (-sy_{k-1} + (2-s)y_k + (s-2)y_{k+1} + sy_{k+2}) u^3 \\ & + (2sy_{k-1} + (s-3)y_k + (3-2s)y_{k+1} - sy_{k+2}) u^2 \\ & + (-sy_{k-1} + sy_{k+1}) u + y_k \\ z(u) = & (-sz_{k-1} + (2-s)z_k + (s-2)z_{k+1} + sz_{k+2}) u^3 \\ & + (2sz_{k-1} + (s-3)z_k + (3-2s)z_{k+1} - sz_{k+2}) u^2 \\ & + (-sz_{k-1} + sz_{k+1}) u + z_k \end{aligned}$$

可以看出，一个Cardinal样条曲线完全由四个连续的型值点给出。中间两个型值点是曲线段端点，另外二个点用来辅助计算端点斜率。只要给出一组型值点的坐标值，就可以分段计算出Cardinal样条曲线，并组合成一整条三次样条曲线。

例：取 $t=0$ , 则 $s=1$ ，上面的平面代数方程变为：

$$\begin{aligned}x(u) &= (-x_{k-1} + x_k - x_{k+1} + x_{k+2}) u^3 + (2x_{k-1} - 2x_k + x_{k+1} - x_{k+2}) u^2 \\&\quad + (-x_{k-1} + x_{k+1}) u + x_k \\y(u) &= (-y_{k-1} + y_k - y_{k+1} + y_{k+2}) u^3 + (2y_{k-1} - 2y_k + y_{k+1} - y_{k+2}) u^2 \\&\quad + (-y_{k-1} + y_{k+1}) u + y_k\end{aligned}$$

设在平面上给定的9个型值点坐标分别为：(100, 300), (120, 200), (220, 200), (270, 100), (370, 100), (420, 200), (420, 300), (220, 280), (100, 300)  
起始点和终止点相重。画出其曲线。

## Cardinal三次曲线算法主要实现子程序实例

```
void CardinalCurve()
{
    int i;
    int arry[9][2]={100, 300, 120, 200, 220, 200, 270, 100, 370, 100, 420, 200, 420, 300, 220, 280, 100, 300} ;
    for(i=0;i<8;i++)
    {
        line(arry[i][0],arry[i][1],arry[i+1][0],arry[i+1][1]);
        Cardinal3(arry, i, 100);
    }
}

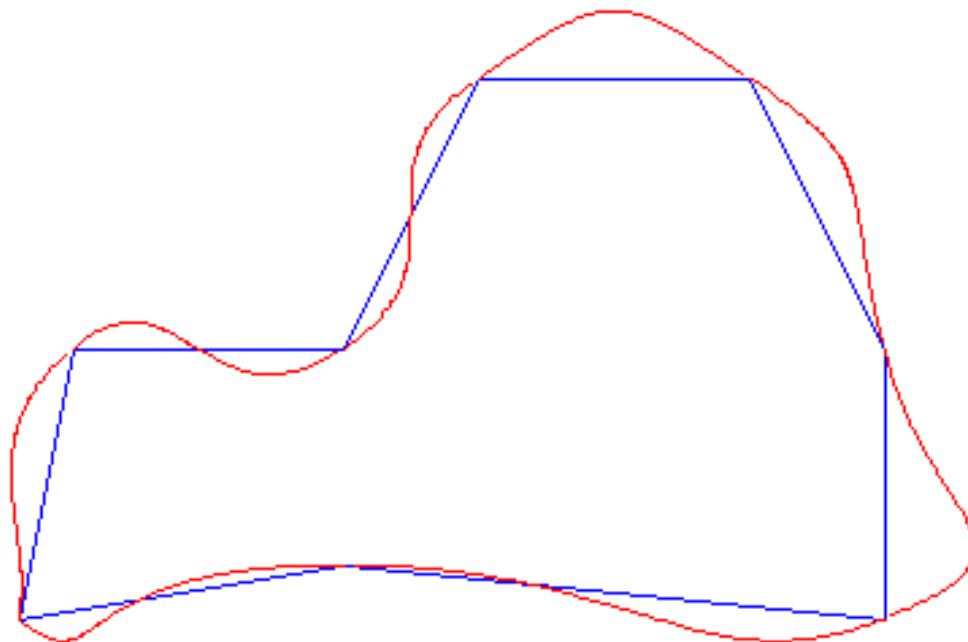
void Cardinal3(int arry[4][2], int n, int steps)
{
    int i, x, y, k1, k2, k3, k4, m1, m2, m3, m4;
    float a0, a1, a2, a3, b0, b1, b2, b3, dt, t, t2, t3;
    if (n==0)
        k1=arry[8][0];
    else
        k1=arry[n-1][0];
    k2=arry[n][0];
    k3=arry[n+1][0];
```

```
if (n<7)
    k4=arry[n+2][0];
else
    k4=arry[1][0];
if (n==0)
    m1=arry[8][1];
else
    m1=arry[n-1][1];
m2=arry[n][1];
m3=arry[n+1][1];
if (n<7)
    m4=arry[n+2][1];
else
    m4=arry[1][1];
MoveTo(arry[n][0],arry[n][1]);
a0 = k2;
a1 = -k1+k3;
a2 = 2*k1-2*k2+k3-k4;
a3 = -k1+k2-k3+k4;
b0 = m2;
b1 = -m1+m3;
b2 = 2*m1-2*m2+m3-m4;
b3 = -m1+m2-m3+m4;
dt = 1.0/steps;
```

```
for(i=1;i<steps;i++)
{
    t = i*dt;
    t2 = t*t;
    t3 = t*t2;
    x = a0+a1*t+a2*t2+a3*t3;
    y = b0+b1*t+b2*t2+b3*t3;

    LineTo(x, y);
    Sleep(5);
}
}
```

**Cardinal**三次曲线



## Cardinal三次曲线绘制演示

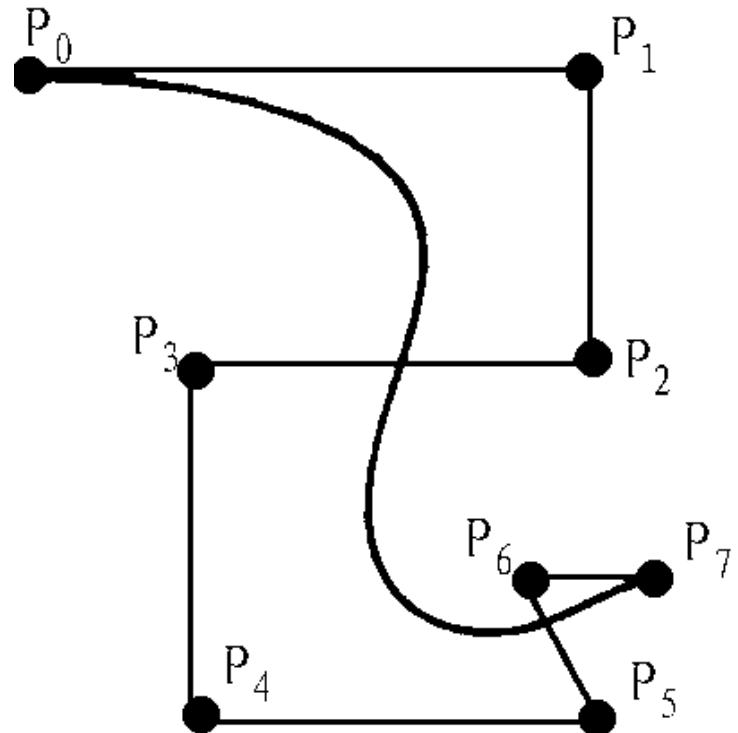
## 5.3 Bezier 曲线

前面讨论过的三次参数样条曲线通过给定的型值点，属于样条插值曲线，适合于已知曲线上的某些点而生成曲线的情形。但在外形设计时，初时给出的型值点有时并不精确，由给定的型值点生成的样条曲线并不能满足性能或美观的要求，需要加以修改。但多数插值样条曲线作为外形设计工具不能直观地表示出应该如何控制和修改曲线的形状，缺少灵活性和直观性。法国雷诺汽车公司工程师P.E.Bezier在1962年提出了一种新的参数曲线表示方法，称为Bezier曲线。这种方法的特点是所输入型值点与生成曲线之间的关系明确，能比较方便地通过修改输入参数来改变曲线的形状和阶次。

### 5.3.1 Bezier 曲线的定义及特性

Bezier曲线是由一组多边折线定义的，在多边折线的各项点中，只有第一点和最后一点在曲线上，第一条和最后一条折线分别表示出曲线在起点和终点处切线方向。曲线的形状趋向于多边折线的形状，因此，多边折线又称为**特征多边形**，其顶点称为**控制点**。

## Bezier曲线及其特征多边形图例：



## 1. 数学表达式

Bezier曲线次数严格依赖于确定该段曲线的控制点个数，通常由( $n + 1$ )个顶点定义一个n次多项式，曲线上各点参数方程式为：

$$P(u) = \sum_{k=0}^n P_k B_{k,n}(u) \quad u \in [0, 1]$$

其中， $P_k$ 为特征多边形第k个顶点的坐标值 $(x_k, y_k, z_k)$ ，而基函数 $B_{k,n}(u)$ 的定义如下

$$B_{k,u} = C_n^k u^k (1-u)^{n-k} \quad (k = 0, 1, \dots, n)$$

函数 $B_{k,n}(u)$ 称为Bernstein多项式，其中

$$C_n^k = \frac{n!}{k!(n-k)!} \quad \text{为组合公式。}$$

式中参数u的取值范围为[0,1],n是多项式次数，也是曲线次数。

## 2. Bezier曲线的性质

(1) 曲线的起点和终点同特征多边形的起点和终点重合

对Bernstein多项式有：

当  $u=0$  时，只有  $k=0$  的项不为 0，其它项都为  $u^k = 0^k = 0$ ，因此

$$P(0) = \frac{n!}{0!(n-0)!} \cdot 0^0 \cdot (1-0)^n \cdot P_0 = P_0$$

其中规定： $0! = 1$ ,  $0^0 = 1$ 。

当  $u=1$  时，只有  $k=n$  的项不为 0，其它项为  $(1-u)^{n-k} = 0$ ，因此

$$P(1) = \frac{n!}{n!(n-n)!} \cdot 1^n \cdot (1-1)^0 \cdot P_n = P_n$$

## (2) 一阶导数

对参数u求导得：

$$P'(u) = n \sum_{k=1}^n (P_k - P_{k-1}) B_{k-1, n-1}(u)$$

在起始点  $u = 0$ ,  $B_{0, n-1}(0) = 1$ , 其余项均为0, 故有:

$$P'(0) = n(P_1 - P_0)$$

在终止点  $u = 1$ ,  $B_{n-1, n-1}(1) = 1$ , 其余项均为0, 故有:

$$P'(1) = n(P_n - P_{n-1})$$

即Bezier曲线在端点处的一阶导数只同相近的两个控制点有关, 其方向相同于两点的连线方向。

### (3) 二阶导数

式(4-14)对参数u求二阶导数可得：

$$P''(u) = n(n-1) \sum_{k=0}^n (P_{k+2} - 2P_{k+1} + P_k) B_{k,n-1}(u)$$

在起始点u= 0处的二阶导数为：

$$P''(0) = n(n-1) (P_2 - 2P_1 + P_0)$$

在终止点u= 1处的二阶导数为：

$$P''(1) = n(n-1) (P_n - 2P_{n-1} + P_{n-2})$$

即Bezier曲线在端点处的二阶导数只同相近的三个控制点有关。

#### (4) 凸包性

Bezier曲线的另一个重要性质是它落在特征多边形顶点所形成的凸包内。即当特征多边形为凸时，Bezier曲线也是凸的；当特征多边形有凹有凸时，其曲线的凸凹形状与之对应。Bezier曲线的凸包性质保证了多项式曲线随控制点平稳前进而不会振荡。

#### (5) 几何不变性

由Bezier曲线的数学定义式知，曲线的形状由特征多边形的顶点 $P_k$  ( $k = 0, 1, \dots, n$ ) 唯一确定，与坐标系的选取无关，这就是几何不变性。

### 5.3.2 三次Bezier 曲线

一般地说，可以用任何数目的控制点拟合出一条 Bezier 曲线，但这需要计算更高次的多项式。复杂曲线可以由一些较低次数的 Bezier 曲线段连接而成，较小的曲线段连接也便于更好地控制小区域内的曲线形状，最常使用的是三次 Bezier 曲线。

三次 Bezier 曲线由四个控制点  $P_0$ 、 $P_1$ 、 $P_2$ 、 $P_3$  定义：

$$P(u) = \sum_{k=0}^3 P_k B_{k,3}(u)$$

展开后的表达式为：

$$\begin{aligned}P(u) &= (-u^3 + 3u^2 - 3u + 1)P_0 + (3u^3 - 6u^2 + 3u)P_1 \\&\quad + (-3u^3 + 3u^2)P_2 + u^3P_3 \\&= B_{0,3}(u)P_0 + B_{1,3}(u)P_1 + B_{2,3}(u)P_2 + B_{3,3}(u)P_3\end{aligned}$$

其中

$$B_{0,3}(u) = -u^3 + 3u^2 - 3u + 1$$

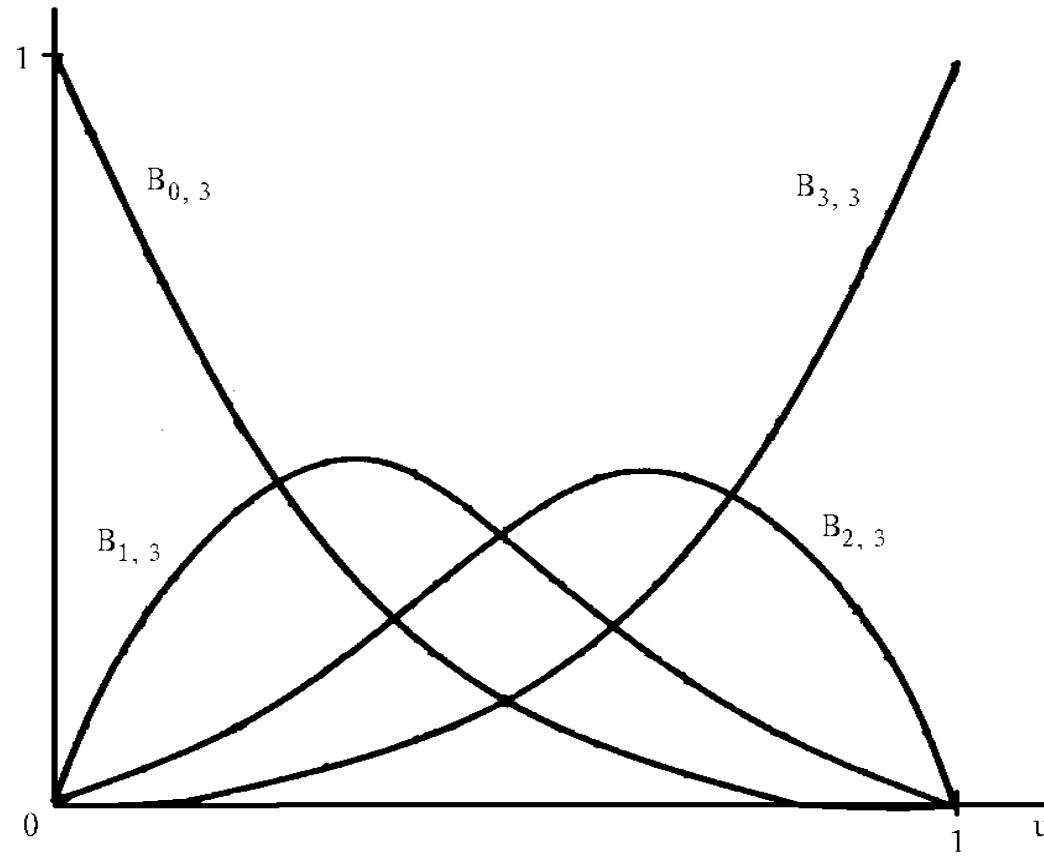
$$B_{1,3}(u) = 3u^3 - 6u^2 + 3u$$

$$B_{2,3}(u) = -3u^3 + 3u^2$$

$$B_{3,3}(u) = u^3$$

称为三次Bezier曲线的调和函数，下图表示出调和函数的四条曲线。这四条曲线形成了三次Bezier曲线的一组基，任何三次Bezier曲线都是这四条曲线的线性组合。

三次Bezier曲线的调和函数：



三次Bezier曲线函数式用矩阵形式表示为：

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_{be} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$
$$M_{be} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

是三次Bezier系数矩阵。

三次Bezier曲线的在端点处的一阶导数为：

$$P'(0) = 3(P_1 - P_0)$$

$$P'(1) = 3(P_3 - P_2)$$

二阶导数为：

$$P''(0) = 6(P_2 - 2P_1 + P_0)$$

$$P''(1) = 6(P_3 - 2P_2 + P_1)$$

三次Bezier曲线函数式分别写成坐标分量的形式如下：

$$\begin{aligned}x(u) = & (-u^3 + 3u^2 - 3u + 1)x_0 + (3u^3 - 6u^2 + 3u)x_1 \\& + (-3u^3 + 3u^2)x_2 + u^3x_3\end{aligned}$$

$$\begin{aligned}y(u) = & (-u^3 + 3u^2 - 3u + 1)y_0 + (3u^3 - 6u^2 + 3u)y_1 \\& + (-3u^3 + 3u^2)y_2 + u^3y_3\end{aligned}$$

$$\begin{aligned}z(u) = & (-u^3 + 3u^2 - 3u + 1)z_0 + (3u^3 - 6u^2 + 3u)z_1 \\& + (-3u^3 + 3u^2)z_2 + u^3z_3\end{aligned}$$

实际生成曲线时，按问题的要求取一合适的步长，控制u从0到1变化，求出一系列(x, y)坐标点，将其用小线段顺序连接起来，就可以得到一条Bezier曲线。

对于二维平面的情况，只有x, y坐标分量，可以给出四点三次Bezier曲线如下的算法描述：

begin

  x=x<sub>0</sub>

  y=y<sub>0</sub>

  moveto (x, y)

  for u= 0 to 1 step Δu

$$x = B_{0,3}(u)x_0 + B_{1,3}(u)x_1 + B_{2,3}(u)x_2 + B_{3,3}(u)x_3$$

$$y = B_{0,3}(u)y_0 + B_{1,3}(u)y_1 + B_{2,3}(u)y_2 + B_{3,3}(u)y_3$$

  lineto (x, y)

  endfor

end

三次Bezier曲线例子：

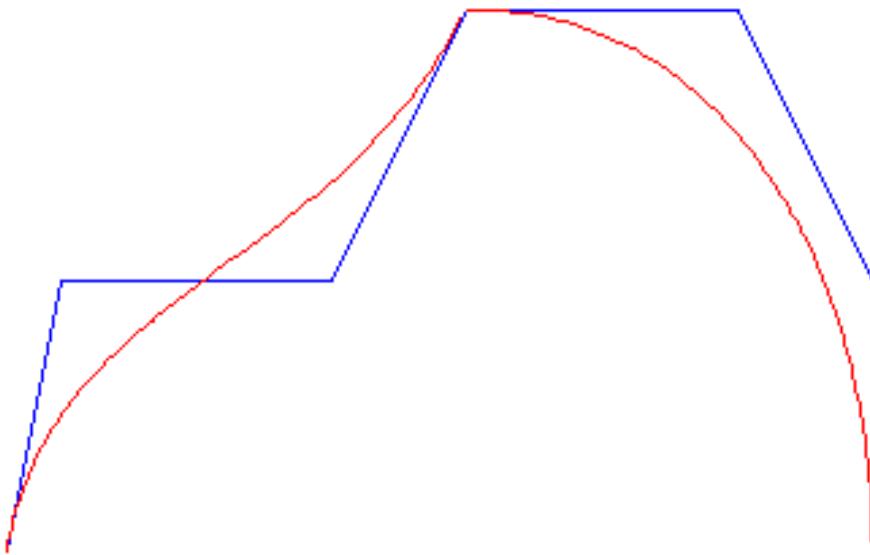
设在平面上给定的7个控制点坐标分别为：

(100, 300) , (120, 200) , (220, 200) ,

(270, 100) , (370, 100) , (420, 200) ,

(420, 300) 。画出其曲线。

## Bezier三次曲线



## 三次Bezier曲线绘制演示

### 5.3.3 Bezier 曲线的光滑连接

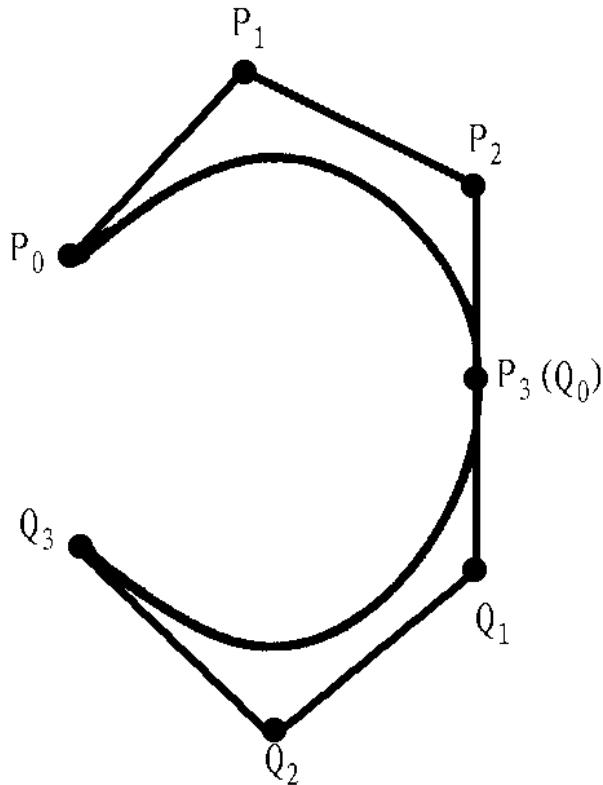
复杂曲线可以由一些较低次数的Bezier曲线段连接而成，工程上通常使用分段三次Bezier曲线来描述。将分段的三次Bezier曲线连接起来构成三次Bezier曲线，其关键问题是保证连接处具有连续性。

设有两段三次Bezier曲线，其中一段曲线由控制点 $P_0$ 、 $P_1$ 、 $P_2$ 、 $P_3$ 生成，另一条曲线由控制点 $Q_0$ 、 $Q_1$ 、 $Q_2$ 、 $Q_3$ 生成， $P_3$ ( $Q_0$ )是两段曲线的公共控制点，如下图所示。如果两段曲线要达到光滑连接，需要一阶导数连续，甚至二阶导数连续。对于一阶导数连续，由前面所推出的公式，第一段曲线终点处的导数为：

$$P'(1) = 3(P_3 - P_2)$$

第二段曲线起点处的导数为：

$$Q'(0) = 3(Q_0 - Q_1)$$



两段Bezier曲线光滑连接的条件示意图

一阶导数要连续，则应有  $P'(1) = Q'(0)$ ，即：

$$P_3 - P_2 = Q_1 - Q_0$$

也即要求  $P_2 P_3(Q_0) Q_1$  三点共线，而且  $P_3(Q_0)$  为中点，是它们的公切线。

由公式(4-28)，第一段曲线终点处的二阶导数为：

$$P''(1) = 6(P_3 - 2P_2 + P_1)$$

第二段曲线起点处的二阶导数为：

$$Q''(0) = 6(Q_2 - 2Q_1 + Q_0)$$

要达到二阶导数连续，则应有  $P''(1) = Q''(0)$ ，即：

$$P_3 - 2P_2 + P_1 = Q_2 - 2Q_1 + Q_0$$

整理后可得：

$Q_2 - P_1 = 2(Q_1 - P_2)$ ，即连线  $Q_2P_1$  和  $Q_1P_2$  要平行，且  $Q_2P_1$  的长度为  $Q_1P_2$  的两倍。

Bezier样条曲线为外形设计提供了灵活直观的方法，但对  $(n+1)$  个控制点，需要  $n$  阶 Bernstein 多项式，当  $n$  较大时，特征多边形对曲线控制减弱，曲线修改和使用都不便。如果使用低次多项式分段实现，光滑连接所需要的条件要求比较高。另外，如果改变任一个控制点位置，整个曲线都受到影响，缺乏对曲线形状进行局部修改的灵活性。

## 5.4 B样条曲线

1974年，Gordon与Riesenfeld等人拓宽了Bezier曲线，使用B样条函数代替Bernstein多项式函数。B样条(Basic Spline, B-spline)曲线除保持了Bezier曲线的直观性和凸包性等优点之外，多项式次数也独立于控制点数目，而且B样条曲线允许局部调整。由于以上原因，B样条曲线得到越来越广泛的应用。

### 5.4.1 B样条的定义

B样条曲线分为均匀B样条曲线(Uniform B-spline)和一般非均匀B样条曲线(General Non-uniform B-spline)，这里，我们只学习均匀B样条曲线。

B样条曲线是由若干样条曲线段光滑连接而成，首先定义B样条曲线段。设给定 $n+1$ 个控制点，用 $P_k$ 表示 $(k=0, 1, \dots, n)$ ， $n$ 次B样条曲线段的参数表达式为：

$$P(u) = \sum_{k=0}^n P_k F_{k,n}(u) \quad u \in [0, 1]$$

与Bezier曲线类似，依次用线段连接 $P_k$ 中相邻两个控制点所得折线多边形称为B样条特征多边形。式中

$$F_{k,n}(u) = \frac{1}{n!} \sum_{j=0}^{n-k} (-1)^j C_{n+1}^j (u + n - k - j)^n$$

其中  $C_{n+1}^j = (n+1)!/(j!(n+1-j)!)$

$$u \in [0, 1], \quad k = 0, 1, \dots, n.$$

$F_{k,n}(u)$ 称为B样条基函数，它是由 $k$ 从0到 $n$ 共 $(n+1)$ 个函数组成。

B样条曲线不同于Bezier曲线整体生成，它是分段生成连接起来的，B样条曲线段之间是自然连接的。

给定控制点  $P_k$  ( $k = 0, 1, \dots, n, \dots, n+m+1$ ) (即至少  $n+1$  个控制点)，则  $n$  次B样条整体曲线表达式为：

$$P_{k,n}(u) = \sum_{k=0}^n P_{i+k} F_{k,n}(u) \quad u \in [0, 1], i=0, 1, \dots, m$$

即，对于  $(n+m+1)$  个控制点，使用  $n$  次B样条函数，生成曲线时需要  $(m+1)$  次计算。各段B样条曲线能够自动光滑连接形成一整条B样条曲线，曲线的整体称为  $n$  次B样条曲线。当  $m=0$  时，需要 1 次计算。

## 5.4.2 三次B样条曲线

工程上最常使用的是三次B样条曲线。对三次B样条曲线函数式为：

$$P(u) = \sum_{k=0}^3 P_k F_{k,3}(u) \quad u \in [0, 1]$$

B样条函数的表达式为：

$$F_{k,3}(u) = \frac{1}{3!} \sum_{j=0}^{3-k} (-1)^j C_{3+1}^j (u + 3 - k - j)^3$$

展开有：

$$F_{0,3}(u) = (-u^3 + 3u^2 - 3u + 1) / 6;$$

$$F_{1,3}(u) = (3u^3 - 6u^2 + 4) / 6;$$

$$F_{2,3}(u) = (-3u^3 + 3u^2 + 3u + 1) / 6;$$

$$F_{3,3}(u) = u^3 / 6.$$

如果给定四个控制点，使用三次B样条函数计算一次就可以得到B样条曲线。将三次B样条函数式用矩阵形式表示为：

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_{bs} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$
$$M_{bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

是三次B样条系数矩阵。

将(4-34)分别写成坐标分量的形式如下：

$$\begin{aligned}x(u) = & (-u^3/6 + u^2/2 - u/2 + 1/6)x_0 + (u^3/2 - u^2 + 2/3)x_1 \\& + (-u^3/2 + u^2/2 + u/2 + 1/6)x_2 + u^3x_3/6\end{aligned}$$

$$\begin{aligned}y(u) = & (-u^3/6 + u^2/2 - u/2 + 1/6)y_0 + (u^3/2 - u^2 + 2/3)y_1 \\& + (-u^3/2 + u^2/2 + u/2 + 1/6)y_2 + u^3y_3/6\end{aligned}$$

$$\begin{aligned}z(u) = & (-u^3/6 + u^2/2 - u/2 + 1/6)z_0 + (u^3/2 - u^2 + 2/3)z_1 \\& + (-u^3/2 + u^2/2 + u/2 + 1/6)z_2 + u^3z_3/6\end{aligned}$$

当u从0到1变化时，曲线将在P<sub>0</sub>到P<sub>3</sub>之间顺序地连续形成。

如果给定控制点P<sub>k</sub>(k=0, 1, ..., n; n≥3)，使用三次B样条函数生成整体B样条曲线需要计算(n-2)次。第一次计算使用0~3四个控制点生成第一段B样条曲线，然后向前移动一个控制点，使用1~4四个控制点计算生成第二段B样条曲线，两段B样条曲线会自然形成平滑连接，

用三次B样条函数生成二维B样条曲线的算法描述如下：

begin

$$x = F_{0,3}(0)x_0 + F_{1,3}(0)x_1 + F_{2,3}(0)x_2 + F_{3,3}(0)x_3$$

$$y = F_{0,3}(0)y_0 + F_{1,3}(0)y_1 + F_{2,3}(0)y_2 + F_{3,3}(0)y_3$$

moveto (x, y)

for k = 0 to n-3

    for u = 0 to 1 step  $\Delta u$

$$x = F_{0,3}(u)x_k + F_{1,3}(u)x_{k+1} + F_{2,3}(u)x_{k+2} + F_{3,3}(u)x_{k+3}$$

$$y = F_{0,3}(u)y_k + F_{1,3}(u)y_{k+1} + F_{2,3}(u)y_{k+2} + F_{3,3}(u)y_{k+3}$$

    lineto (x, y)

endfor

endfor

end

## 5. 4. 3 B样条曲线的性质

### 1. 端点性质

B样条曲线也是逼近曲线，一般情况下，曲线不经过控制点。分别取 $u=0$ 及 $u=1$ 得曲线端点值为：

$$P(0) = (P_0 + 4P_1 + P_2)$$

$$P(1) = (P_1 + 4P_2 + P_3)$$

也就是说，起始控制点和终止控制点都不在曲线上。而且，三次B样条曲线，其起点只与前三个控制点有关，终点只与后三个控制点有关。实际上，B样条曲线都具有这种控制点的邻近影响性，这正是B样条曲线局部可调整性好的原因。

### 2. 连续性

B样条曲线段之间是自行光滑连续的。而且，n次B样条曲线具有 $n-1$ 阶导数的连续性。

### 3. 局部性和扩展性

在三次B样条曲线中，每个B样条曲线段受四个控制点影响，改变一个控制点的位置，最多影响四个曲线段。因而，通过改变控制点的位置就可对B样条曲线进行局部修改，这是一个非常重要的性质。同时，B样条曲线在端点处的一阶和二阶导数也具有只受邻近控制点影响的性质。

由于B样条曲线可以由曲线段自然连续生成，如果增加一个控制点，就相应地增加了一段B样条曲线。此时，原有的B样条曲线不受影响，而且新增的曲线段与原曲线的连接处具有一阶、二阶导数连续的特性，这一点是由B样条曲线本身的性质所保证的，不需要附加任何条件，因而要对原有的B样条曲线加以扩展是很方便的。

三次B样条曲线例子：

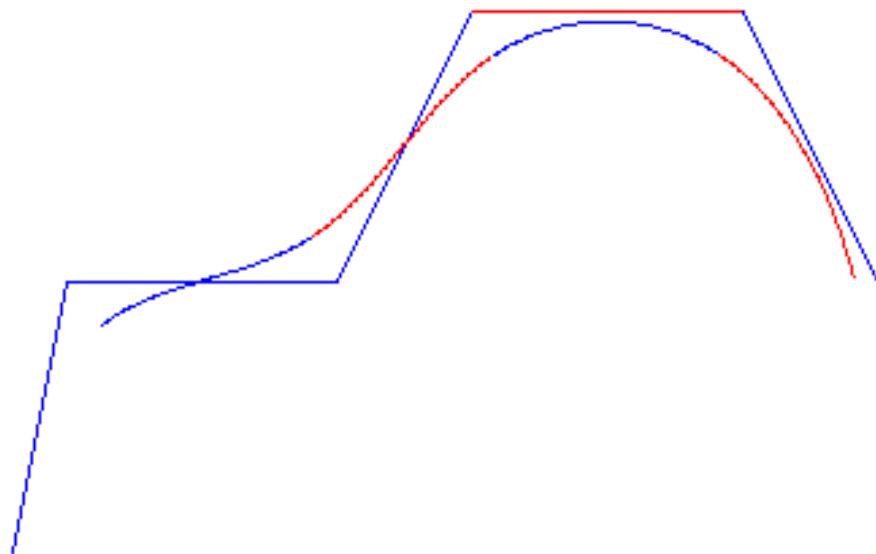
设在平面上给定的7个控制点坐标分别为：

(100, 300) , (120, 200) , (220, 200) ,

(270, 100) , (370, 100) , (420, 200) ,

(420, 300) 。画出其曲线。

## B样条三次曲线

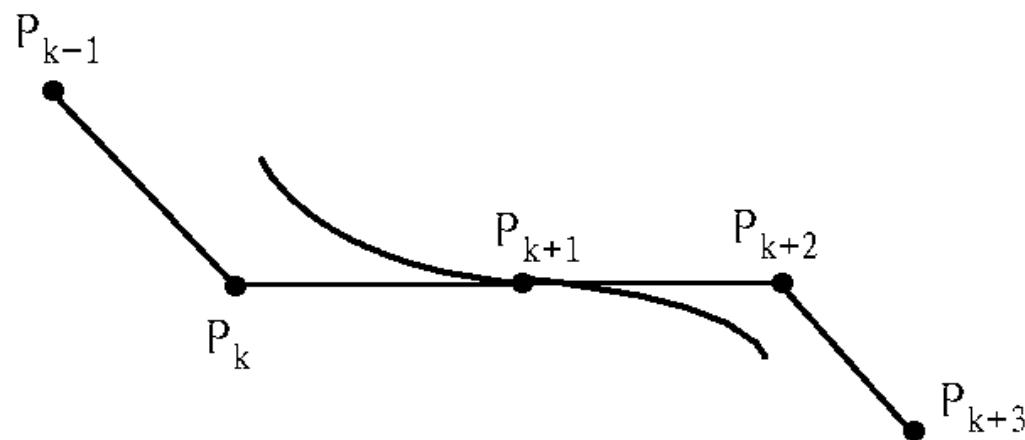


## 三次B样条曲线绘制演示1

## 4. 三次B样条的几种特殊情况

### (1) 三个连续的控制点共线

如果三个连续的控制点共线连成一段直线，则曲线将过直线上的一点，且在此点处，曲线直线化。可以用这样的点构成曲线的拐点，如下图所示。



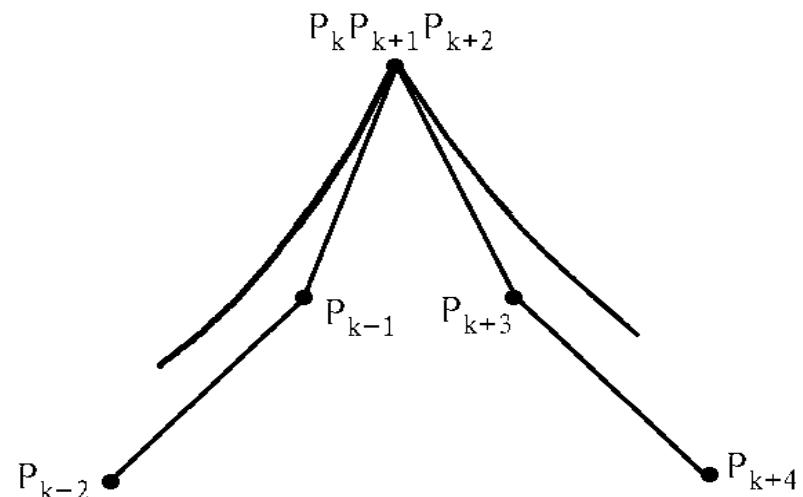
## (2) 四个连续的控制点共线

四个连续的控制点共线时，曲线变为直线，直线的长度小于四点构成的直线，如下图所示：



## (3) 三个连续的控制点重合

当三个连续的控制点重合时，形成尖点，如右图所示。



通过上面的讨论，可以总结控制三次B样条曲线几何形态的一些方法，归纳如下：

1) 为在曲线内嵌入一段直线，应用四个顶点共线的技巧。

2) 为使曲线和特征多边形相切，应用三顶点共线或两顶点重合的技术。

3) 为使曲线在某一项点处形成尖角，可在该处使三个顶点相重合。

4) 改变一个顶点，将影响相邻四段曲线的形状。

5) 用三重顶点或二重顶点控制曲线的端点。用三重顶点时，曲线通过端点，但开始段B样条曲线是一小段直线；用二重顶点时，曲线不通过端点，而在多边形首边上靠近二重顶点的某一点开始。

使用重点绘制通过起点和终点的三次B样条曲线例子：

设在平面上给定的11个控制点坐标分别为：

(100, 300) , (100, 300) , (100, 300) ,

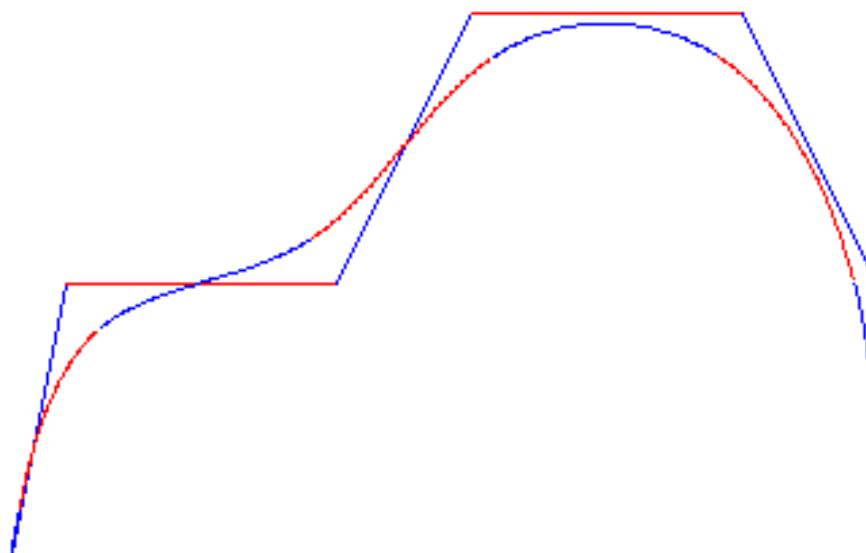
(120, 200) , (220, 200) , (270, 100) ,

(370, 100) , (420, 200) ,

(420, 300) , (420, 300) , (420, 300) 。

画出其曲线。

B样条三次曲线-起点处和终点处三点相重



## 三次B样条曲线绘制演示2

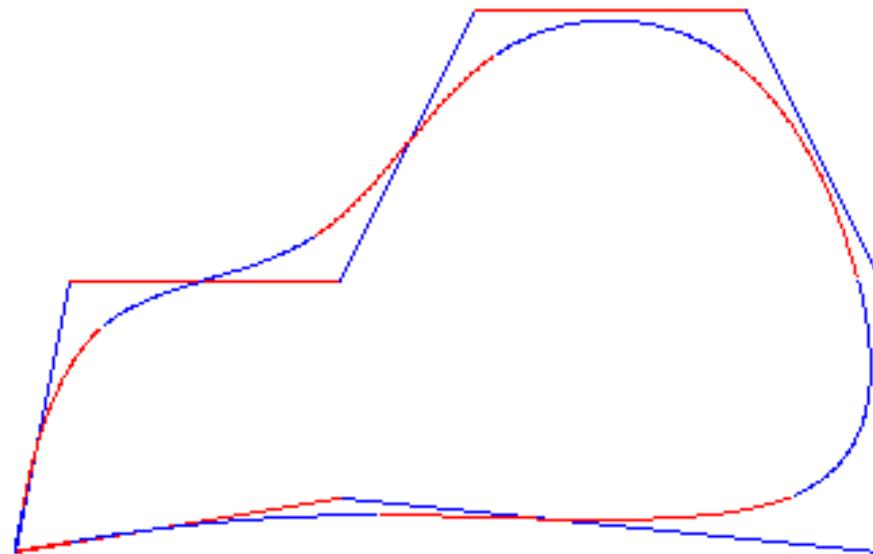
使用重点绘制封闭的三次B样条曲线例子：

设在平面上给定的15个控制点坐标分别为：

(100, 300) , (100, 300) , (100, 300) ,  
(120, 200) , (220, 200) , (270, 100) ,  
(370, 100) , (420, 200) , (420, 300) , (220, 280) ,  
(100, 300) , (100, 300) , (100, 300) 。

画出其曲线。

B样条三次曲线-起点和终点相重构成封闭曲线



### 三次B样条曲线绘制演示3

## 5.5 曲面

一些工程实际中应用的复杂自由曲面，如飞机、船舶、汽车等几何外形的描述，传统上是用人工作图法完成的。由于需要大量的试画和反复修正工作，以保证整个曲面光顺，所以非常繁琐而又费时。可以用样条的方法来设计与描述曲面，由计算机、绘图仪及图形显示器去完成绘制工作。

### 5.5.1 空间曲面的参数表示

前面讨论了一条自由曲线可以由一系列的曲线段连接而成，与此类似，一自由曲面也可以由一系列的曲面片拼合而成。因此，曲面片是曲面的基础，

首先来讨论曲面片的数学表示形式及其性质。

一个曲面片是以曲线为边界的点的集合，这些点的坐标( $x, y, z$ )均可用双参数的单值函数表示如下：

$$x = x(u, w), y = y(u, w), z = z(u, w); \quad u, w \in [0, 1]$$

曲面上任一点的参数表示为：

$$P(u, w) = [x(u, w), y(u, w), z(u, w)]$$

如果用三次参数方程来表示曲面片，可以表示成如下形式：

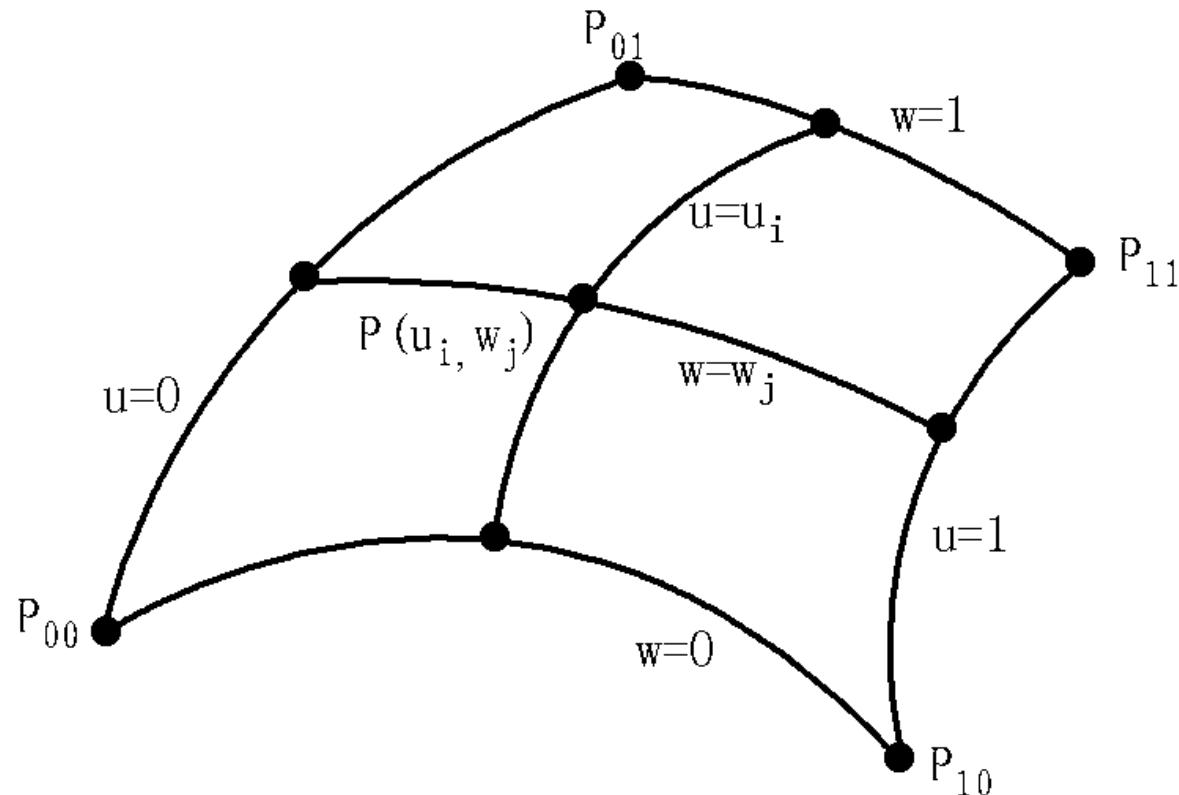
$$\begin{aligned}
 P(u, w) = & a_{33}u^3w^3 + a_{32}u^3w^2 + a_{31}u^3w + a_{30}u^3 \\
 & + a_{23}u^2w^3 + a_{22}u^2w^2 + a_{21}u^2w + a_{20}u^2 \\
 & + a_{13}uw^3 + a_{12}uw^2 + a_{11}uw + a_{10}u \\
 & + a_{03}w^3 + a_{02}w^2 + a_{01}w + a_{00}
 \end{aligned}$$

或

$$P(u, w) = \sum_{i=0}^3 \sum_{j=0}^3 a_{ij} u^i w^j \quad u, w \in [0, 1]$$

式中u,w为参数。此参数方程共有16个系数，每一系数都有3个独立的坐标分量，因而总共有48个自由度。上式所描述的曲面片也称为双三次曲面片。

下图所示为双三次曲面片的一例。



实际上，一个双三次曲面片正是由参数空间相互正交的两组曲线集组成的，这两组曲线集分别由参数 $u$ 及 $w$ 来定义。一组曲线包括 $u=0$ 及 $u=1$ 这两条边界曲线及无穷多条由 $u=u_i$ 决定的中间曲线。与此相似，另一组曲线包括 $w=0$ 及 $w=1$ 这两条边界曲线以及无穷多条由 $w=w_j$ 决定的中间曲线。

### 5.5.2 Coons曲面

在讨论Hermite样条插值曲线时，我们知道了它是使用两个端点的坐标值及端点处的导数来决定一条曲线段。与此类似，Coons曲面是使用曲面片角点和角点处的偏导数来决定曲面。用 $P_{00}^u$ 表示在角点 $u=0, w=0$ 点处对 $u$ 的偏导数，即：

角点处的其它偏导数与此类似地表示。记：

$$[C] = \begin{bmatrix} P_{00} & P_{01} & P_{00}^w & P_{01}^w \\ P_{10} & P_{11} & P_{10}^w & P_{11}^w \\ P_{00}^u & P_{01}^u & P_{00}^{uw} & P_{01}^{uw} \\ P_{10}^u & P_{11}^u & P_{10}^{uw} & P_{11}^{uw} \end{bmatrix}$$

称为角点信息矩阵。

Hermite样条曲线是利用Hermite样条调和函数对边界条件调和而生成，而Coons曲面是使用Hermite样条调和函数对角点信息矩阵进行调合生成曲面。Coons双三次曲面的数学表达式如下：

$$\begin{aligned} P(u, w) &= [H(u)] [C] [H(w)]^T \\ &= [U] [M_h] [C] [M_h]^T [W]^T \end{aligned}$$

其中

$$M_h = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

为Hermite矩阵。

$[U] = [u^3 \ u^2 \ u \ 1]$ ,  $[W] = [w^3 \ w^2 \ w \ 1]$  为两个参数  $u, w$  的矩阵向量。

角点信息矩阵[C]可分成四组，左上角一组可以代表四个角点的位置坐标，右上角和左下角分别代表边界曲线在四个角点处的两组切线向量。右下角一组则为角点处的混合偏导，也称为扭矢量。整个曲面就是由四个角点的这四组十六个信息来控制的。其中前三组信息完全决定了四条边界曲线的位置和形状。第四组角点扭矢量则与边界形状没有关系，但它却影响边界曲线上中间各点的切线向量，从而影响整个曲面片的形状。

双三次Coons曲面的主要缺点是必须给定矩阵[C]中的16个向量，才能唯一确定曲面片的位置和形状，而要给定扭矢量是相当困难的，因而使用起来不太方便。另外，两个曲面片之间的光滑连接也需要两个角点信息矩阵中相应偏导和混合偏导满足一定的条件。

### 4.5.3 Bezier 曲面

Bezier曲面是由Bezier曲线拓广而来，它也是以Bernstein函数作为基函数，可以构造由空间网格的顶点位置来控制的曲面。

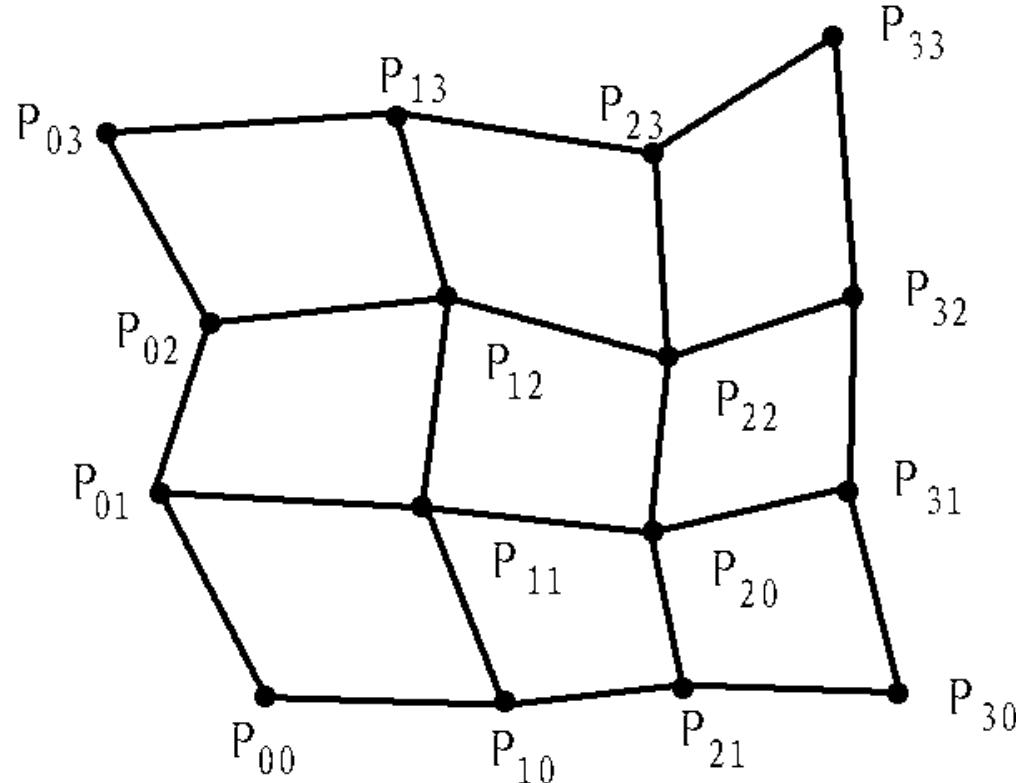
#### 1. Bezier曲面的数学表示式

给定  $(n+1) \times (m + 1)$  个空间点  $P_{ij}$  ( $i=0,1,\dots,n$  ;  $j=0,1,\dots,m$ )，Bezier曲面的数学表达式如下：

$$P(u, w) = \sum_{i=0}^n \sum_{j=0}^m P_{ij} B_{i,n}(u) B_{j,m}(w) \quad u, w \in [0, 1]$$

上述公式所描述的曲面称为 $n \times m$ 次Bezier曲面。 $P_{ij}$ 是 $P(u, w)$ 的控制顶点, $B_{i,n}(u)$ 和 $B_{j,m}(w)$ 为Bernstein基函数。

依次用线段连接点列  $P_{ij}$  ( $i=0,1,\dots,n$ ;  $j=0,1,\dots,m$ ) 中相邻两点所形成的空间网格称为特征网格，如下图所示。



## 2. 双三次Bezier曲面

当  $n=m=3$  时，得到双三次Bezier曲面。给定  $P_{ij}$  ( $i=0, 1, 2, 3$ ;  $j=0, 1, 2, 3$ ) 16个控制点，双三次Bezier曲面片的表示式为：

$$\begin{aligned} P(u, w) &= \sum_{i=0}^3 \sum_{j=0}^3 P_{ij} B_{i,3}(u) B_{j,3}(w) \\ &= [B_{0,3}(u) \quad B_{1,3}(u) \quad B_{2,3}(u) \quad B_{3,3}(u)] \times \\ &\quad \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \times \begin{bmatrix} B_{0,3}(w) \\ B_{1,3}(w) \\ B_{2,3}(w) \\ B_{3,3}(w) \end{bmatrix} \end{aligned}$$

$$\begin{aligned}
 &= [B(u)] [P] [B(w)]^T \\
 &= [U] [M_{be}] [P] [M_{be}]^T [W]^T
 \end{aligned}$$

式中  $[U] = [u^3 \ u^2 \ u \ 1]$ ,  $[W] = [w^3 \ w^2 \ w \ 1]$  为两个参数  $u, w$  的矩阵向量。而

$$M_{be} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

是三次Bezier系数矩阵。

Bezier曲面是由Bezier曲线交织而成的曲面。曲面生成时可以通过固定w, 变化u得到一族Bezier曲线; 固定u, 变化w得到另一簇Bezier曲线。Bezier曲面与Bezier曲线具有相同的性质, 不同曲面片之间的拼接需要满足一定的条件。对于C<sup>0</sup>连接性只要边界上的控制点匹配就可获得, 而C<sup>1</sup>和C<sup>2</sup>连续性的条件类似于我们在前面讨论过的Bezier曲线光滑连接时的条件要求。

下面，将式  $P(u, w) = [U] [M_{be}] [P] [M_{be}]^T [W]^T$   
 展开成代数形式如下：

$$P(u, w) = [U] [M_{be}] [P] [M_{be}]^T [W]^T$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

$$\times \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

$$= [-u^3 + 3u^2 - 3u + 1, 3u^3 - 6u^2 + 3u, -3u^3 + 3u^2, u^3]$$

$$\times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \times \begin{bmatrix} -w^3 + 3w^2 - 3w + 1 \\ 3w^3 - 6w^2 + 3w \\ -3w^3 + 3w^2 \\ w^3 \end{bmatrix}$$

$$= [ (-u^3 + 3u^2 - 3u + 1)P_{00} + (3u^3 - 6u^2 + 3u)P_{10} + (-3u^3 + 3u^2)P_{20} + u^3 P_{30}, \\ (-u^3 + 3u^2 - 3u + 1)P_{01} + (3u^3 - 6u^2 + 3u)P_{11} + (-3u^3 + 3u^2)P_{21} + u^3 P_{31}, \\ (-u^3 + 3u^2 - 3u + 1)P_{02} + (3u^3 - 6u^2 + 3u)P_{12} + (-3u^3 + 3u^2)P_{22} + u^3 P_{32}, \\ (-u^3 + 3u^2 - 3u + 1)P_{03} + (3u^3 - 6u^2 + 3u)P_{13} + (-3u^3 + 3u^2)P_{23} + u^3 P_{33}, ]$$

$$\begin{aligned}
&= ((-u^3+3u^2-3u+1)P_{00} + (3u^3-6u^2+3u)P_{10} + (-3u^3+3u^2)P_{20} + u^3P_{30}) (-w^3+3w^2-3w+1) \\
&+ ((-u^3+3u^2-3u+1)P_{01} + (3u^3-6u^2+3u)P_{11} + (-3u^3+3u^2)P_{21} + u^3P_{31}) (3w^3-6w^2+3w) \\
&+ ((-u^3+3u^2-3u+1)P_{02} + (3u^3-6u^2+3u)P_{12} + (-3u^3+3u^2)P_{22} + u^3P_{32}) (-3w^3+3w^2) \\
&+ ((-u^3+3u^2-3u+1)P_{03} + (3u^3-6u^2+3u)P_{13} + (-3u^3+3u^2)P_{23} + u^3P_{33}) (w^3)
\end{aligned}$$

写成分量坐标的形式为：

$$\begin{aligned}
x(u, w) &= ((-u^3+3u^2-3u+1)x_{00} + (3u^3-6u^2+3u)x_{10} + (-3u^3+3u^2)x_{20} + u^3x_{30}) (-w^3+3w^2-3w+1) \\
&+ ((-u^3+3u^2-3u+1)x_{01} + (3u^3-6u^2+3u)x_{11} + (-3u^3+3u^2)x_{21} + u^3x_{31}) (3w^3-6w^2+3w) \\
&+ ((-u^3+3u^2-3u+1)x_{02} + (3u^3-6u^2+3u)x_{12} + (-3u^3+3u^2)x_{22} + u^3x_{32}) (-3w^3+3w^2) \\
&+ ((-u^3+3u^2-3u+1)x_{03} + (3u^3-6u^2+3u)x_{13} + (-3u^3+3u^2)x_{23} + u^3x_{33}) (w^3)
\end{aligned}$$

$$\begin{aligned}
y(u, w) &= ((-u^3+3u^2-3u+1)y_{00} + (3u^3-6u^2+3u)y_{10} + (-3u^3+3u^2)y_{20} + u^3y_{30}) (-w^3+3w^2-3w+1) \\
&+ ((-u^3+3u^2-3u+1)y_{01} + (3u^3-6u^2+3u)y_{11} + (-3u^3+3u^2)y_{21} + u^3y_{31}) (3w^3-6w^2+3w) \\
&+ ((-u^3+3u^2-3u+1)y_{02} + (3u^3-6u^2+3u)y_{12} + (-3u^3+3u^2)y_{22} + u^3y_{32}) (-3w^3+3w^2) \\
&+ ((-u^3+3u^2-3u+1)y_{03} + (3u^3-6u^2+3u)y_{13} + (-3u^3+3u^2)y_{23} + u^3y_{33}) (w^3)
\end{aligned}$$

$$\begin{aligned}
z(u, w) &= ((-u^3+3u^2-3u+1)z_{00} + (3u^3-6u^2+3u)z_{10} + (-3u^3+3u^2)z_{20} + u^3z_{30}) (-w^3+3w^2-3w+1) \\
&+ ((-u^3+3u^2-3u+1)z_{01} + (3u^3-6u^2+3u)z_{11} + (-3u^3+3u^2)z_{21} + u^3z_{31}) (3w^3-6w^2+3w) \\
&+ ((-u^3+3u^2-3u+1)z_{02} + (3u^3-6u^2+3u)z_{12} + (-3u^3+3u^2)z_{22} + u^3z_{32}) (-3w^3+3w^2) \\
&+ ((-u^3+3u^2-3u+1)z_{03} + (3u^3-6u^2+3u)z_{13} + (-3u^3+3u^2)z_{23} + u^3z_{33}) (w^3)
\end{aligned}$$

当是平面曲面时，只有x和y坐标分量。

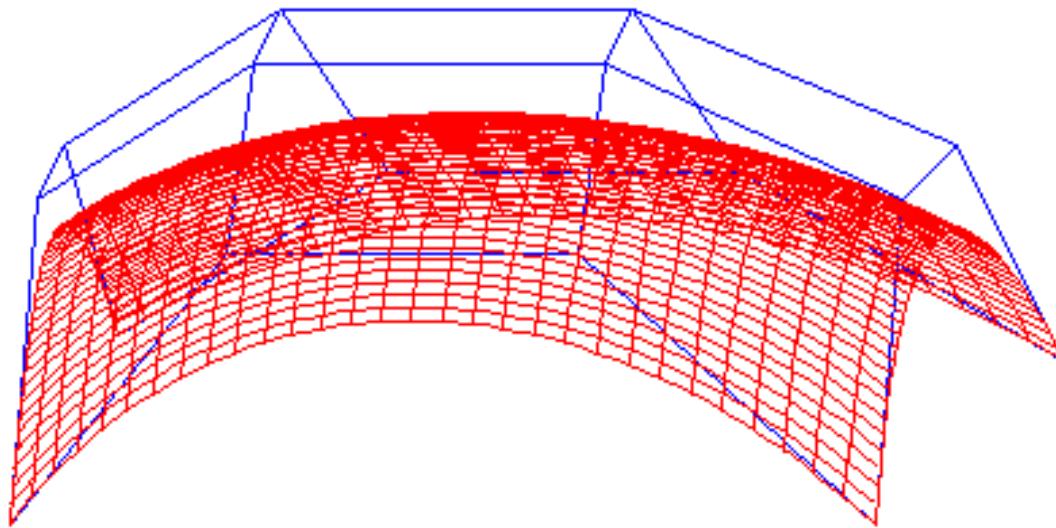
下面举几个例子：

例1：给定16个控制点坐标如下：

(100, 300), (110, 180), (120, 160), (140, 230),  
(180, 200), (190, 130), (200, 110), (240, 170),  
(310, 200), (320, 130), (330, 110), (370, 170),  
(420, 300), (430, 180), (450, 160), (490, 240)。

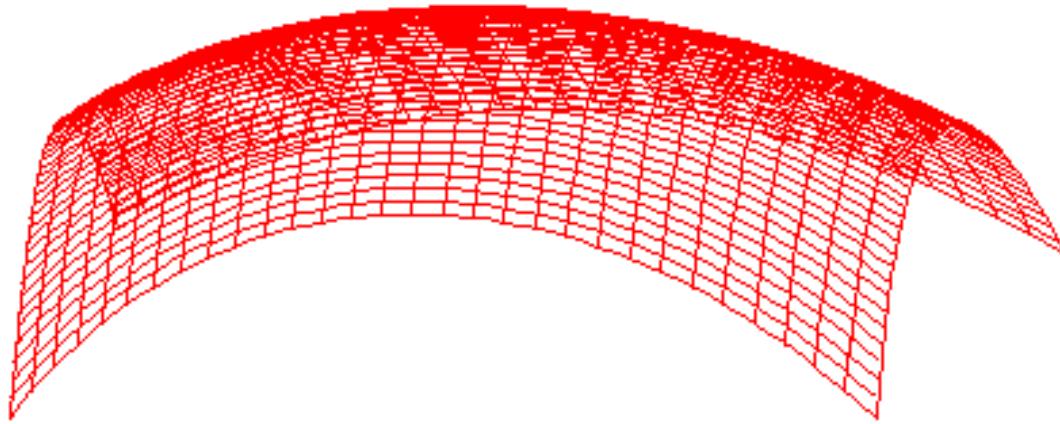
绘制三次Bezier曲面。

16个点构成一凸特征多边形生成一片Bezier曲面



## 三次Bezier曲面绘制演示1

16个点构成一凸特征多边形生成一片Bezier曲面



## 不绘制特征网格曲面演示2

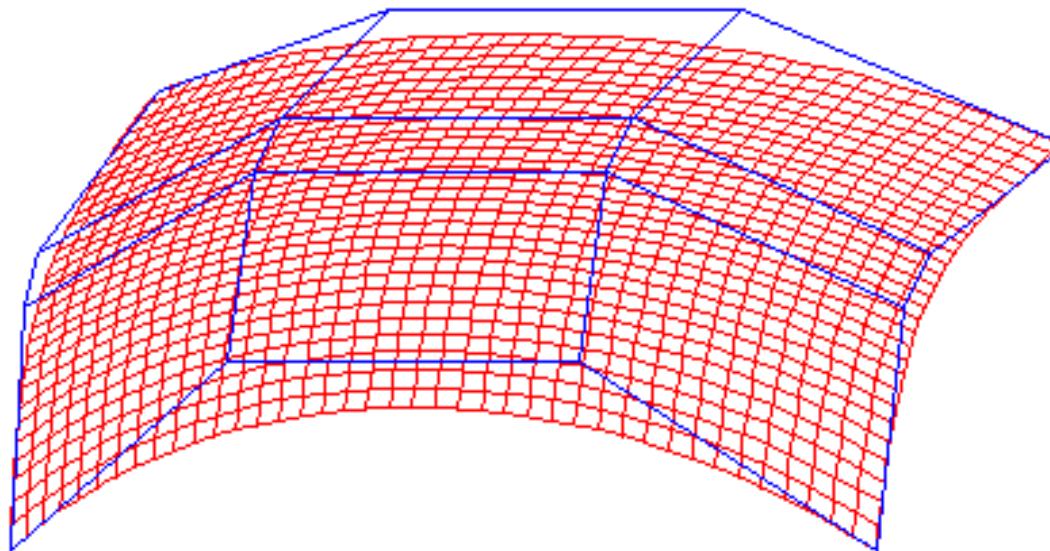
例2：给定16个控制点坐标如下：

(100, 270), (105, 180), (110, 160), (155, 100),  
(180, 200), (190, 130), (200, 110), (240, 70),  
(310, 200), (320, 130), (330, 110), (370, 70),  
(420, 270), (430, 180), (440, 160), (490, 120)。

绘制三次Bezier曲面。

### 三次Bezier曲面曲面绘制演示3

16个点构成一凸特征多边形生成一片Bezier曲面



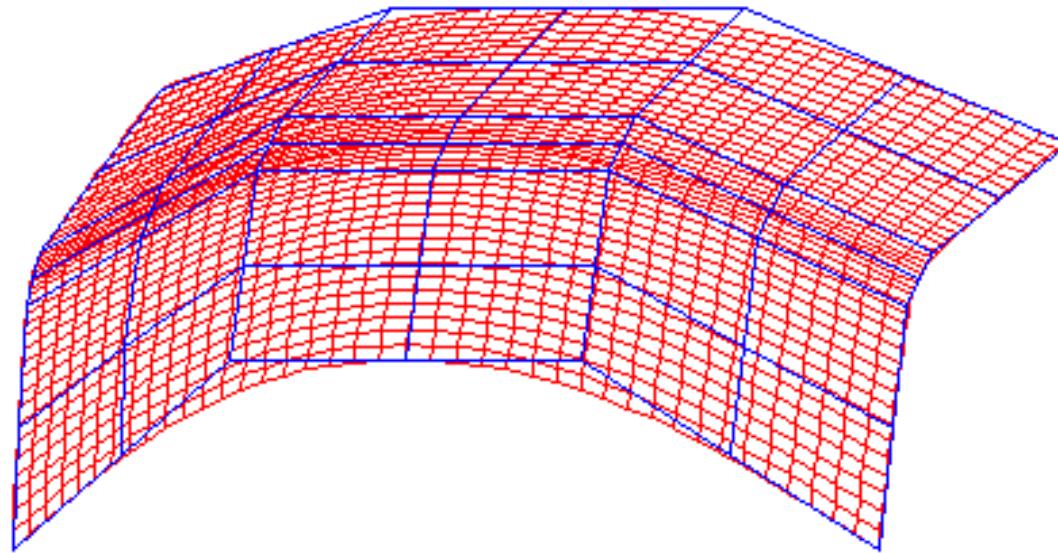
### 三次Bezier曲面曲面绘制演示3

例3：给定 $7 \times 7 = 49$ 个点构成一凸特征多边形，其中相邻3点共线且中间点在中点处，生成一片Bezier曲面。

49个给定点的坐标如下：

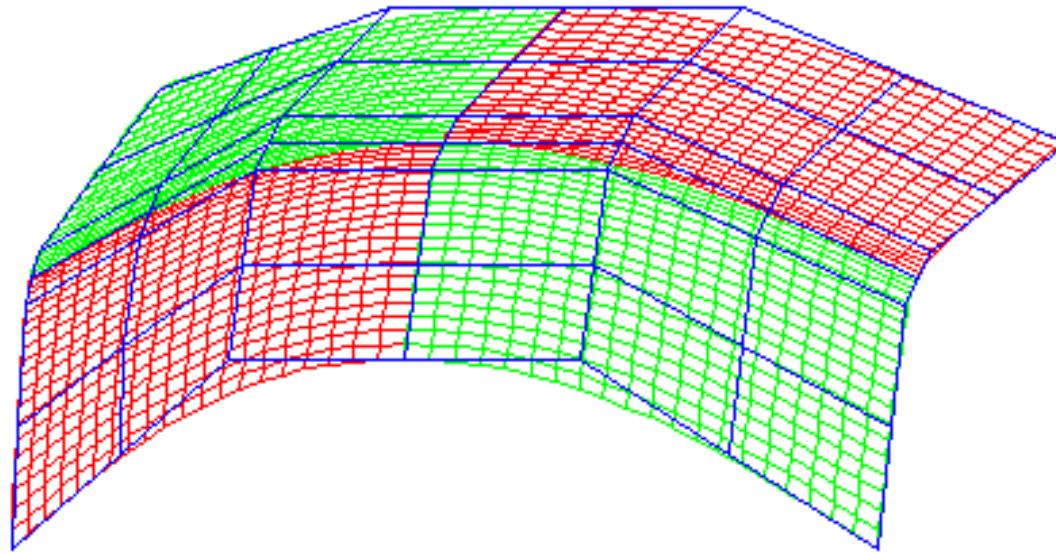
(100, 270), (102, 225), (105, 180), (107, 170), (110, 160), (132, 130), (155, 100),  
(140, 235), (141, 195), (147, 155), (151, 145), (155, 135), (176, 110), (197, 85),  
(180, 200), (185, 165), (190, 130), (195, 120), (200, 110), (220, 90), (240, 70),  
(245, 200), (250, 165), (255, 130), (260, 120), (265, 110), (285, 90), (305, 70),  
(310, 200), (315, 165), (320, 130), (325, 120), (330, 110), (350, 90), (370, 70),  
(365, 235), (370, 195), (375, 155), (380, 145), (385, 135), (407, 115), (430, 95),  
(420, 270), (425, 225), (430, 180), (435, 170), (440, 160), (465, 140), (490, 120)

$7 \times 7 = 49$ 个点构成一凸特征多边形其中相邻3点共线且中间点在中点处,生成一片 Bezier 曲面



## 三次Bezier曲面曲面绘制演示4

$7 \times 7 = 49$ 个点构成一凸特征多边形其中相邻3点共线且中间点在中点处,生成一片 Bezier 曲面



四个分曲面构成整个曲面演示5

## 5.5.4 B样条曲面

B样条曲面是B样条曲线的拓广。

### 1. B样条曲面的数学表示式

给定  $(n+1) \times (m+1)$  个空间点  $P_{ij}$  ( $i=0, 1, \dots, n$ ;  
 $j=0, 1, \dots, m$ )， B样条曲面的数学表达式如下：

$$P(u, w) = \sum_{i=0}^n \sum_{j=0}^m P_{ij} F_{i,n}(u) F_{j,m}(w) \quad u, w \in [0, 1]$$

$P_{ij}$  是  $P(u, w)$  的控制顶点,  $F_{i,n}(u)$  和  $F_{j,m}(w)$  为B样条基函数。如果  $n=m=3$ , 则由  $4 \times 4$  个顶点构成特征网格, 其相应的曲面片称为双三次B样条曲面片。

## 2. 双三次B 样条曲面

双三次B 样条曲面应用最广， 其表示式为：

$$\begin{aligned} P(u, w) &= \sum_{i=0}^3 \sum_{j=0}^3 P_{ij} F_{i,3}(u) F_{j,3}(w) \\ &= [F_{0,3}(u) \quad F_{1,3}(u) \quad F_{2,3}(u) \quad F_{3,3}(u)] \times \\ &\quad \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \times \begin{bmatrix} F_{0,3}(w) \\ F_{1,3}(w) \\ F_{2,3}(w) \\ F_{3,3}(w) \end{bmatrix} \end{aligned}$$

$$\begin{aligned}
 &= [F(u)] [P] [F(w)]^T \\
 &= [U] [M_{bs}] [P] [M_{bs}]^T [W]^T
 \end{aligned}$$

式中  $[U] = [u^3 \ u^2 \ u \ 1]$ ,  $[W] = [w^3 \ w^2 \ w \ 1]$  为两个参数  $u, w$  的矩阵向量。而

$$M_{bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

是三次B样条系数矩阵。

B样条曲面与B样条曲线具有相同的性质。双三次B样条曲面片四个角点不在特征网格的角点上。如果将网格向外扩展，曲面也相应延伸，而且由于三次B样条基函数是二阶连续的，所以双三次B样条曲面也达到二阶连续。

下面，将式  $P(u, w) = [U] [M_{bs}] [P] [M_{bs}]^T [W]^T$   
 展开成代数形式如下：

$$P(u, w) = [U] [M_{bs}] [P] [M_{bs}]^T [W]^T$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \times \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

$$\times \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

$$= 1/36 \times [-u^3 + 3u^2 - 3u + 1, 3u^3 - 6u^2 + 4, -3u^3 + 3u^2 + 3u + 1, u^3]$$

$$\times \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \times \begin{bmatrix} -w^3 + 3w^2 - 3w + 1 \\ 3w^3 - 6w^2 + 4 \\ -3w^3 + 3w^2 + 3w + 1 \\ w^3 \end{bmatrix}$$

$$= 1/36 \times [ (-u^3 + 3u^2 - 3u + 1)P_{00} + (3u^3 - 6u^2 + 4)P_{10} + (-3u^3 + 3u^2 + 3u + 1)P_{20} + u^3 P_{30}, \\ (-u^3 + 3u^2 - 3u + 1)P_{01} + (3u^3 - 6u^2 + 4)P_{11} + (-3u^3 + 3u^2 + 3u + 1)P_{21} + u^3 P_{31}, \\ (-u^3 + 3u^2 - 3u + 1)P_{02} + (3u^3 - 6u^2 + 4)P_{12} + (-3u^3 + 3u^2 + 3u + 1)P_{22} + u^3 P_{32}, \\ (-u^3 + 3u^2 - 3u + 1)P_{03} + (3u^3 - 6u^2 + 4)P_{13} + (-3u^3 + 3u^2 + 3u + 1)P_{23} + u^3 P_{33}, ]$$

$$\begin{aligned}
&= 1/36 \times ((((-u^3+3u^2-3u+1)P_{00} + (3u^3-6u^2+4)P_{10} + (-3u^3+3u^2+3u+1)P_{20} + u^3P_{30}) (-w^3+3w^2-3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)P_{01} + (3u^3-6u^2+4)P_{11} + (-3u^3+3u^2+3u+1)P_{21} + u^3P_{31}) (3w^3-6w^2+4) \\
&\quad + ((-u^3+3u^2-3u+1)P_{02} + (3u^3-6u^2+4)P_{12} + (-3u^3+3u^2+3u+1)P_{22} + u^3P_{32}) (-3w^3+3w^2+3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)P_{03} + (3u^3-6u^2+4)P_{13} + (-3u^3+3u^2+3u+1)P_{23} + u^3P_{33}) (w^3))
\end{aligned}$$

写成分量坐标的形式为：

$$\begin{aligned}
x(u, w) = & 1/36 \times ((((-u^3+3u^2-3u+1)x_{00} + (3u^3-6u^2+4)x_{10} + (-3u^3+3u^2+3u+1)x_{20} + u^3x_{30}) (-w^3+3w^2-3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)x_{01} + (3u^3-6u^2+4)x_{11} + (-3u^3+3u^2+3u+1)x_{21} + u^3x_{31}) (3w^3-6w^2+4) \\
&\quad + ((-u^3+3u^2-3u+1)x_{02} + (3u^3-6u^2+4)x_{12} + (-3u^3+3u^2+3u+1)x_{22} + u^3x_{32}) (-3w^3+3w^2+3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)x_{03} + (3u^3-6u^2+4)x_{13} + (-3u^3+3u^2+3u+1)x_{23} + u^3x_{33}) (w^3))
\end{aligned}$$

$$\begin{aligned}
y(u, w) = & 1/36 \times ((((-u^3+3u^2-3u+1)y_{00} + (3u^3-6u^2+4)y_{10} + (-3u^3+3u^2+3u+1)y_{20} + u^3y_{30}) (-w^3+3w^2-3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)y_{01} + (3u^3-6u^2+4)y_{11} + (-3u^3+3u^2+3u+1)y_{21} + u^3y_{31}) (3w^3-6w^2+4) \\
&\quad + ((-u^3+3u^2-3u+1)y_{02} + (3u^3-6u^2+4)y_{12} + (-3u^3+3u^2+3u+1)y_{22} + u^3y_{32}) (-3w^3+3w^2+3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)y_{03} + (3u^3-6u^2+4)y_{13} + (-3u^3+3u^2+3u+1)y_{23} + u^3y_{33}) (w^3))
\end{aligned}$$

$$\begin{aligned}
z(u, w) = & 1/36 \times ((((-u^3+3u^2-3u+1)z_{00} + (3u^3-6u^2+4)z_{10} + (-3u^3+3u^2+3u+1)z_{20} + u^3z_{30}) (-w^3+3w^2-3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)z_{01} + (3u^3-6u^2+4)z_{11} + (-3u^3+3u^2+3u+1)z_{21} + u^3z_{31}) (3w^3-6w^2+4) \\
&\quad + ((-u^3+3u^2-3u+1)z_{02} + (3u^3-6u^2+4)z_{12} + (-3u^3+3u^2+3u+1)z_{22} + u^3z_{32}) (-3w^3+3w^2+3w+1) \\
&\quad + ((-u^3+3u^2-3u+1)z_{03} + (3u^3-6u^2+4)z_{13} + (-3u^3+3u^2+3u+1)z_{23} + u^3z_{33}) (w^3))
\end{aligned}$$

当是平面曲面时，只有x和y坐标分量。

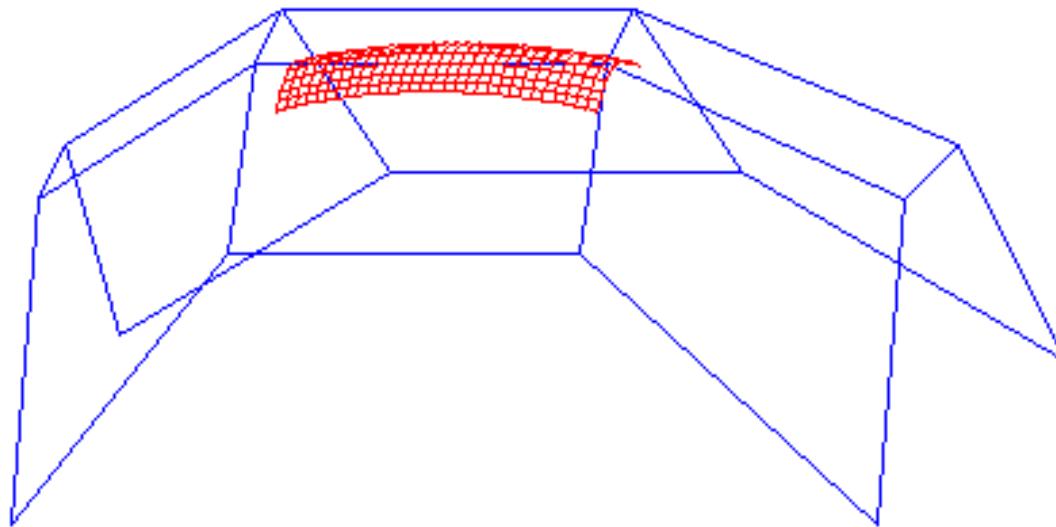
下面举几个例子：

例1：给定16个控制点坐标如下：

(100, 300), (110, 180), (120, 160), (140, 230),  
(180, 200), (190, 130), (200, 110), (240, 170),  
(310, 200), (320, 130), (330, 110), (370, 170),  
(420, 300), (430, 180), (450, 160), (490, 240)。

绘制三次B样条曲面。

16个点构成一凸特征多边形生成一片B样条曲面



## 三次B样条曲面绘制演示1

16个点构成一凸特征多边形生成一片B样条曲面



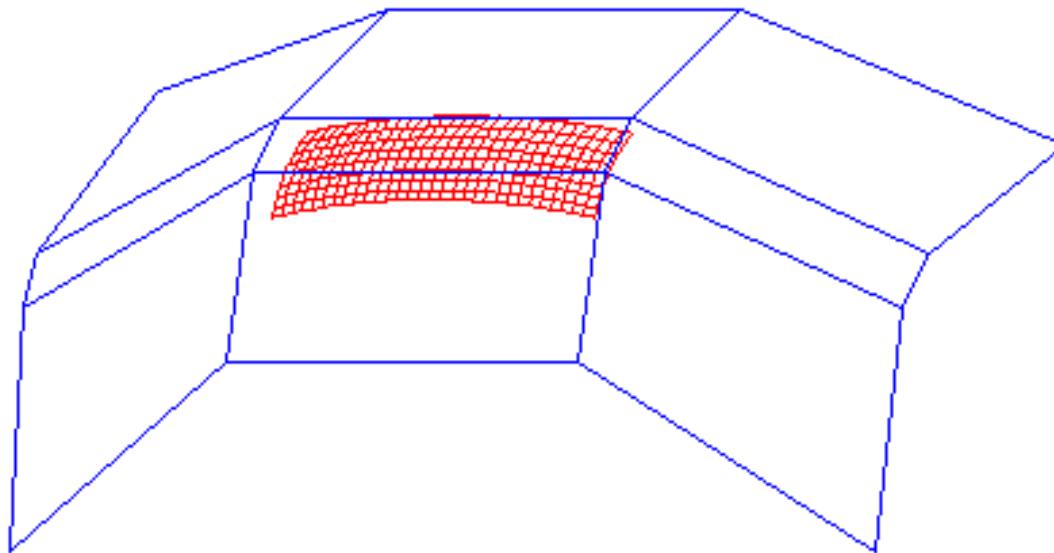
## 无特征网格的曲面绘制演示2

例2：给定16个控制点坐标如下：

(100, 270), (105, 180), (110, 160), (155, 100),  
(180, 200), (190, 130), (200, 110), (240, 70),  
(310, 200), (320, 130), (330, 110), (370, 70),  
(420, 270), (430, 180), (440, 160), (490, 120)。

绘制三次B样条曲面。

16个点构成一凸特征多边形生成一片B样条曲面



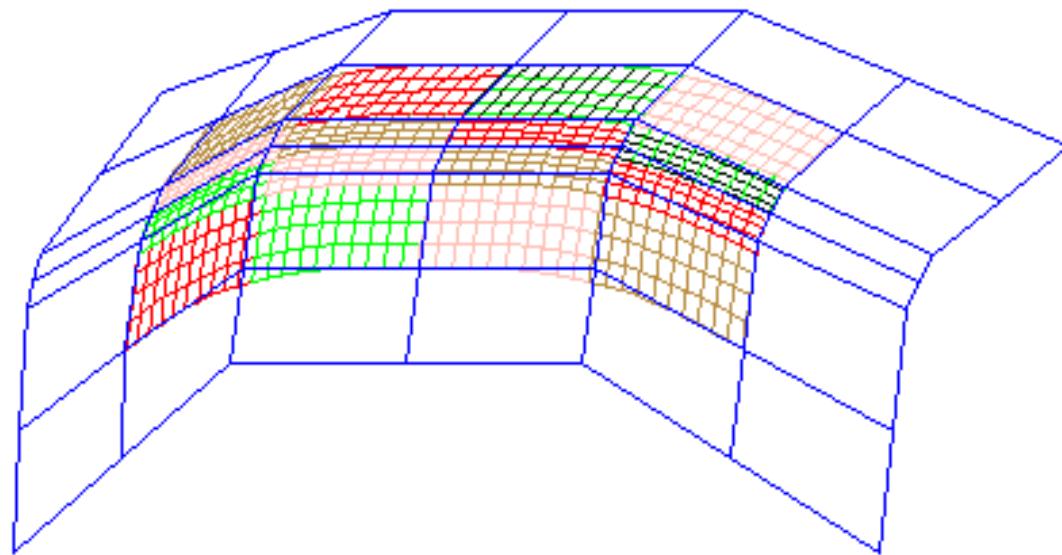
### 三次B样条曲面绘制演示3

例3：给定 $7 \times 7 = 49$ 个点构成一凸特征多边形，其中相邻3点共线且中间点在中点处，生成一片B样条曲面。

49个给定点的坐标如下：

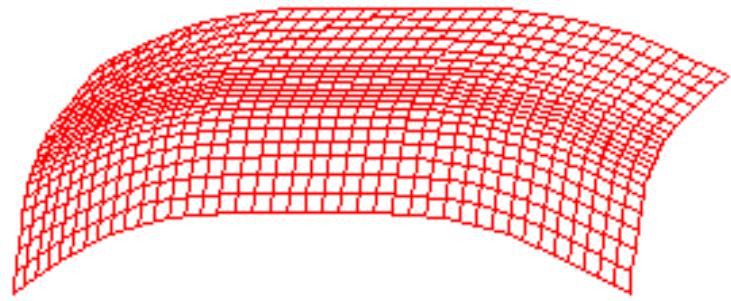
(100, 270), (102, 225), (105, 180), (107, 170), (110, 160), (132, 130), (155, 100),  
(140, 235), (141, 195), (147, 155), (151, 145), (155, 135), (176, 110), (197, 85),  
(180, 200), (185, 165), (190, 130), (195, 120), (200, 110), (220, 90), (240, 70),  
(245, 200), (250, 165), (255, 130), (260, 120), (265, 110), (285, 90), (305, 70),  
(310, 200), (315, 165), (320, 130), (325, 120), (330, 110), (350, 90), (370, 70),  
(365, 235), (370, 195), (375, 155), (380, 145), (385, 135), (407, 115), (430, 95),  
(420, 270), (425, 225), (430, 180), (435, 170), (440, 160), (465, 140), (490, 120)

$7 \times 7 = 49$ 个点构成一凸特征多边形其中相邻3点共线且中间点在中点处，生成一片B样条曲面，起点过中间点



## 三次B样条曲面绘制演示4

$7 \times 7 = 49$ 个点构成一凸特征多边形其中相邻3点共线且中间点在中点处，生成一片B样条曲面，起点过中间点



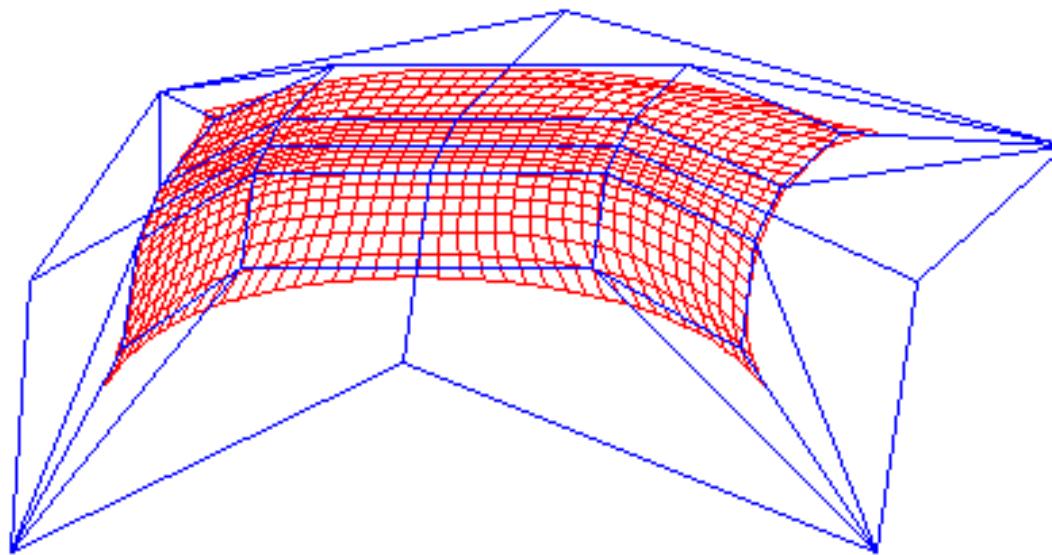
## 无特征网格的曲面绘制演示5

例4：给定 $7 \times 7 = 49$ 个点构成一凸特征多边形，其中边界上相邻3点相重，生成一片B样条曲面。

49个给定点的坐标如下：

(100, 270), (100, 270), (100, 270), (107, 170), (155, 100), (155, 100), (155, 100),  
(100, 270), (141, 195), (147, 155), (151, 145), (155, 135), (176, 110), (155, 100),  
(100, 270), (185, 165), (190, 130), (195, 120), (200, 110), (220, 90), (155, 100),  
(245, 200), (250, 165), (255, 130), (260, 120), (265, 110), (285, 90), (305, 70),  
(420, 270), (315, 165), (320, 130), (325, 120), (330, 110), (350, 90), (490, 120),  
(420, 270), (370, 195), (375, 155), (380, 145), (385, 135), (407, 115), (490, 120),  
(420, 270), (420, 270), (420, 270), (435, 170), (490, 120), (490, 120), (490, 120)

$7 \times 7 = 49$ 个点构成一凸特征多边形,特征多边形顶点处3点相重并不能使B样条曲面过顶点!



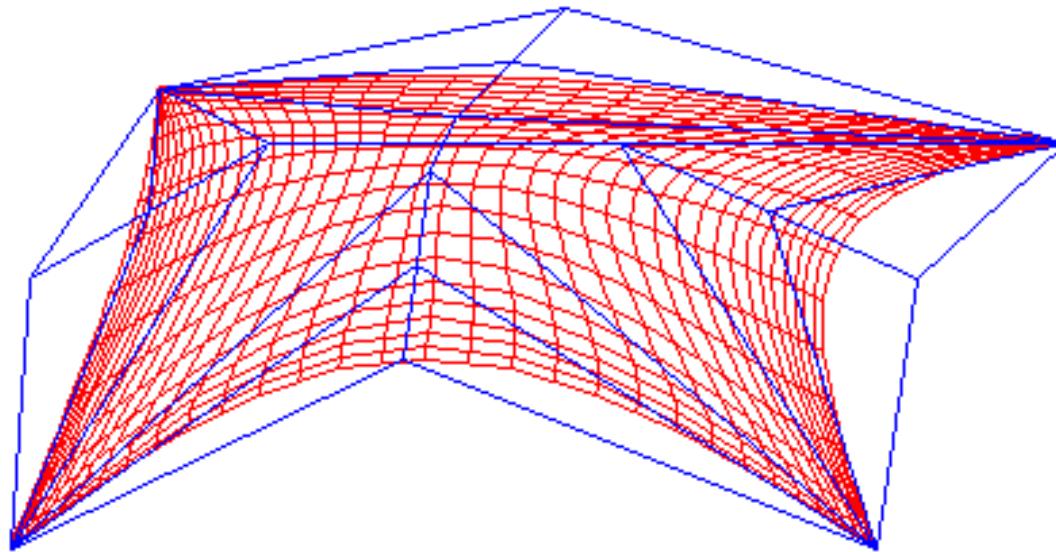
## 三次B样条曲面绘制演示6

例4：给定 $7 \times 7 = 49$ 个点构成一凸特征多边形，其中边界顶点处相邻9点相重，生成一片B样条曲面。

49个给定点的坐标如下：

(100, 270), (100, 270), (100, 270), (107, 170), (155, 100), (155, 100), (155, 100),  
(100, 270), (100, 270), (100, 270), (151, 145), (155, 100), (155, 100), (155, 100),  
(100, 270), (100, 270), (100, 270), (195, 120), (155, 100), (155, 100), (155, 100),  
(245, 200), (250, 165), (255, 130), (260, 120), (265, 110), (285, 90), (305, 70),  
(420, 270), (420, 270), (420, 270), (325, 120), (490, 120), (490, 120), (490, 120),  
(420, 270), (420, 270), (420, 270), (380, 145), (490, 120), (490, 120), (490, 120),  
(420, 270), (420, 270), (420, 270), (435, 170), (490, 120), (465, 140), (490, 120)。

$7 \times 7 = 49$ 个点构成一凸特征多边形,特征多边形顶点处9点相同,生成一片过顶点的B样条曲面



## 三次B样条曲面绘制演示7

# 第六章 图形变换

在计算机绘图应用中，经常要进行从一个几何图形到另一个几何图形的变换，例如，将图形向某一方向平移一段距离；将图形旋转一定的角度；或将图形放大或缩小等等，这种变换过程称为几何变换。图形的几何变换是计算机绘图中极为重要的一个组成部分，利用图形变换还可以实现二维图形和三维图形之间转换，甚至还可以把静态图形变为动态图形，从而实现景物画面的动态显示。

# 主要介绍

- 二维几何变换

## 6.1 二维图形变换

### 6.1.1 二维图形几何变换的基本原理

二维平面图形的几何变换是指在不改变图形连线次序的情况下，对一个平面点集进行的线性变换。实际上，由于一个二维图形可以分解成点、直线、曲线。把曲线离散化，它可以用一串短直线段来逼近，而每一条直线段均由两点所决定，这样，二维平面图形不论是由直线段组成，还是由曲线段组成，都可以用它的轮廓线上顺序排列的平面点集来描述。因此可以说，对图形作几何变换，其实质是对点的几何变换，通过讨论点的几何变换，就可以理解图形几何变换的原理。

例如，如果要对下图中的四边形ABCD进行平移变换，只需要对四个顶点A、B、C、D做平移变换，连接平移后的四个顶点即可得到四边形平移变换的结果。

对二维图形进行几何变换有五种基本变换形式，它们是：**平移、旋转、比例、对称和错切**，这些图形变换的规则可以用函数来表示。有两种不同的变换形式：**一种是图形不动，而坐标系变动**，即变换前与变换后的图形是针对不同坐标而言的，称之为坐标模式变换；**另一种是坐标系不动，而图形改变**，即变换前与变换后的坐标值是针对同一坐标系而言的，称之为图形模式变换。实际应用中，后一种图形变换更有实际意义，下面讨论的图形变换是属于后一种变换。

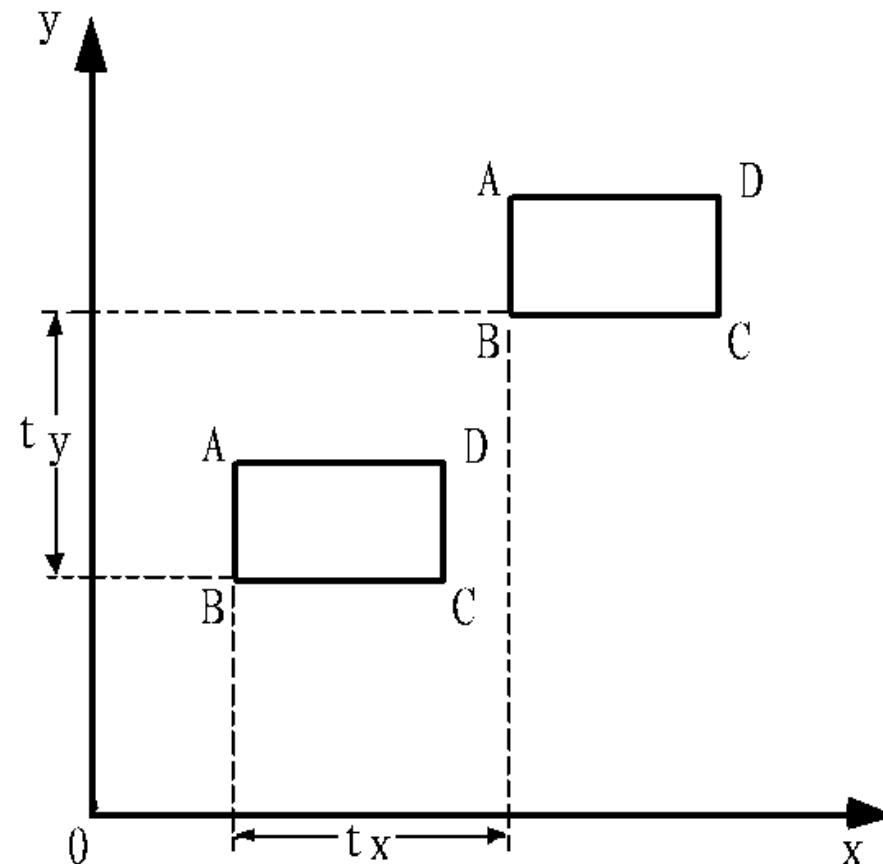
## 6. 1. 2 平移变换

平移变换是指将图形从一个坐标位置移到另一个坐标位置的重定位变换。已知一点的原始坐标是  $P(x, y)$ ，加上一个沿X, Y方向的平移量  $t_x$  和  $t_y$ ，平移此点到新坐标  $(x + t_x, y + t_y)$ ，则新坐标的表达式为：

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

如果对一图形的每个点都进行上述变换，即可得到该图形的平移变换。实际上，线段是通过对其两端点进行平移变换，多边形的平移是平移每个顶点的坐标位置，曲线可以通过平移定义曲线的坐标点位置，用平移过的坐标点重构曲线路径来实现。

- 平移变换只改变图形的位置，不改变图形的大小和形状。下图是一平移变换的例子。



可以用矩阵形式来表示二维平移变换方程。图形变换通常使用齐次坐标矩阵来表示。平移变换方程的齐次坐标矩阵表示式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

其中

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

称为变换矩阵。

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

沿X、Y坐标轴方向各平移1个绘图单位。

# 齐次坐标

所谓齐次坐标表示法就是由 $n+1$ 维向量表示一个 $n$ 维向量。如 $n$ 维向量 $(P_1, P_2, \dots, P_n)$ 表示为 $(hP_1, hP_2, \dots, hP_n, h)$ ，其中 $h$ 称为哑坐标。

1、 $h$ 可以取不同的值，所以同一点的齐次坐标不是唯一的。

如普通坐标系下的点 $(2,3)$ 变换为齐次坐标可以是 $(1,1.5,0.5)(4,6,2)(6,9,3)$ 等等。

2、普通坐标与齐次坐标的关系为“一对多”  
由普通坐标 $\times h \rightarrow$ 齐次坐标

由齐次坐标 $\div h \rightarrow$ 普通坐标

3、当 $h=1$ 时产生的齐次坐标称为“规格化坐标”，因为前 $n$ 个坐标就是普通坐标系下的 $n$ 维坐标。

# 齐次坐标

(x, y) 点对应的齐次坐标  $(x_h, y_h, h)$

$$x_h = hx, y_h = hy, h \neq 0$$

(x, y) 点对应的齐次坐标为三维空间的一条直线

$$\begin{cases} x_h = hx \\ y_h = hy \\ z_h = h \end{cases}$$

# 齐次坐标的作用

1. 将各种变换用阶数统一的矩阵来表示。提供了用矩阵运算把二维、三维甚至高维空间上的一个点从一个坐标系变换到另一坐标系的有效方法。
2. 便于表示无穷远点。

例如：  $(x \times h, y \times h, h)$ ，令  $h$  等于 0

3. 齐次坐标变换矩阵形式把直线变换成直线段，平面变换成平面，多边形变换成多边形，多面体变换成多面体。
4. 变换具有统一表示形式的优点
  - 便于变换合成
  - 便于硬件实现



### 6.1.3 比例变换

一个图形中的坐标点  $P(x,y)$  若在 X 轴方向变化一个比例系数  $s_x$ , 在 Y 轴方向变化一个比例系数  $s_y$ , 则新坐标点  $P'(x',y')$  的表达式为:

$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \end{cases}$$

这一变换称为相对于坐标原点的比例变换， $s_x$  和  $s_y$  分别表示点  $P(x, y)$  沿 X 轴方向和 Y 轴方向相对坐标原点的比例变换系数。比例变换改变图形的大小。变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

以坐标原点为放缩参照点

当  $S_x=S_y=1$  时： 恒等比例变换

当  $S_x=S_y>1$  时： 沿  $x,y$  方向等比例放大。

当  $S_x=S_y<1$  时： 沿  $x,y$  方向等比例缩小

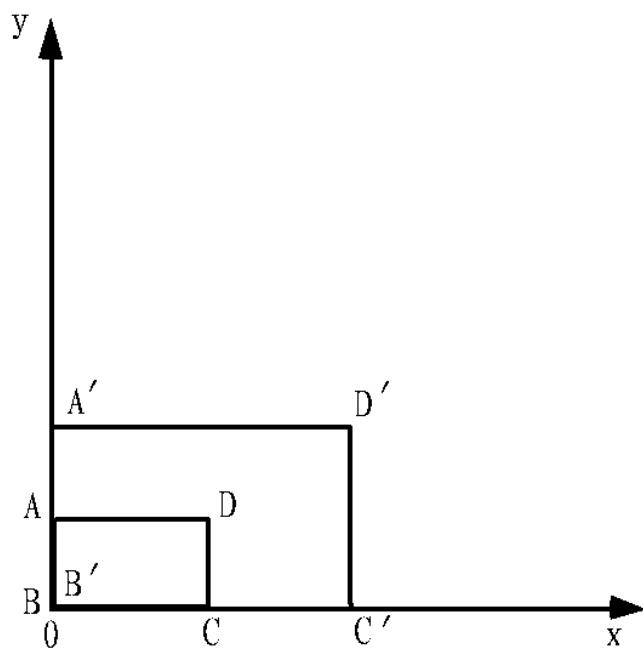
当  $S_x \neq S_y$  时： 沿  $x,y$  方向作非均匀的比例变换，  
图形变形。

$$T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

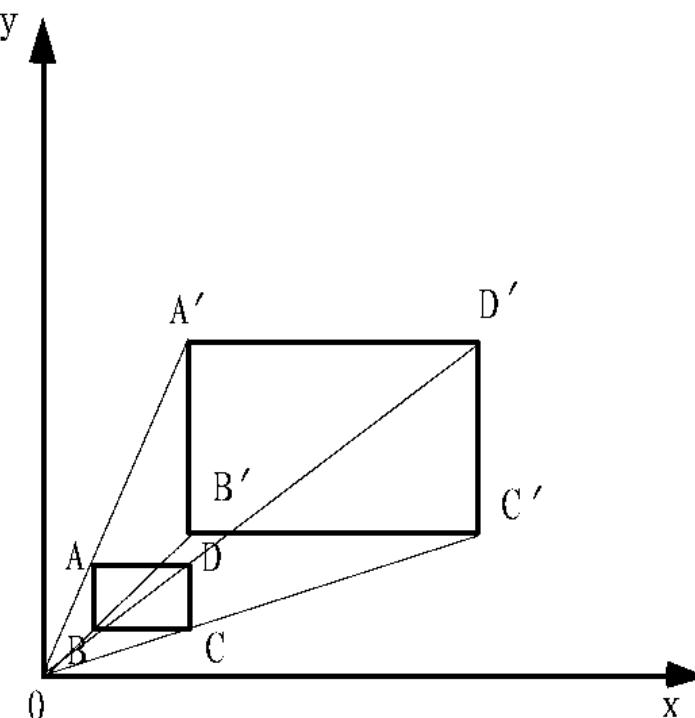
图形沿x轴、y轴方向放大2倍

下图是一图形比例变换的例子：

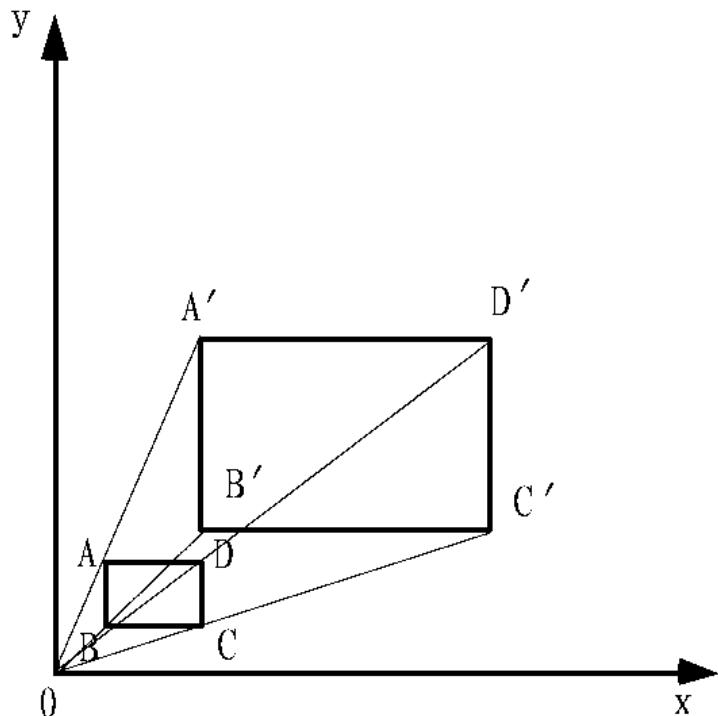
中心在原点的放大变换



中心不在原点的放大变换



可以这样考虑，先平移坐标原点  $(0,0)$  到  $(x_c, y_c)$ ，然后进行比例变换，变换后再将坐标原点移回到  $(0,0)$ 。三个过程的结果就是相对于点  $(x_c, y_c)$  的比例变换。三个过程的变换矩阵分别是：



$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

$$T = T_1 T_2 T_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

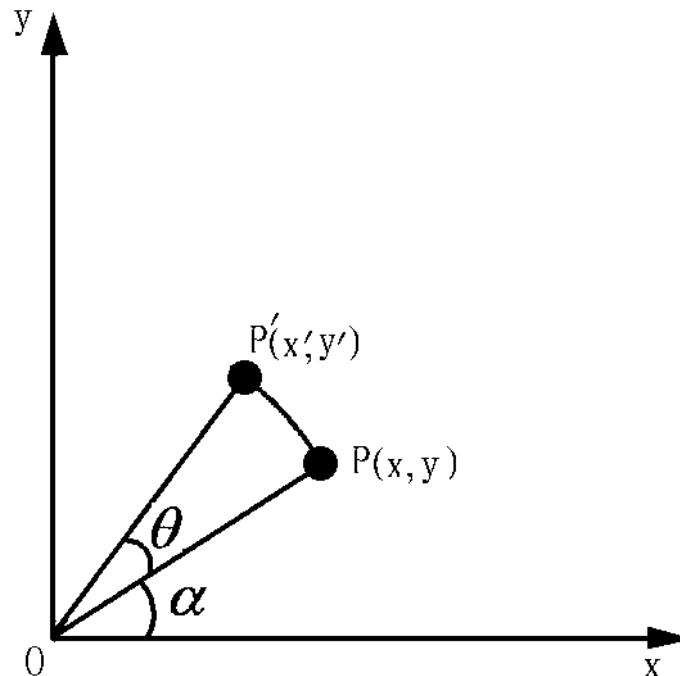
$$= \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ x_c(1-s_x) & y_c(1-s_y) & 1 \end{bmatrix}$$

## 6.1.4 旋转变换

若图形中的坐标点  $P(x, y)$  绕坐标原点逆时针旋转一个角度  $\theta$ , 则新坐标点  $P(x', y')$  的表达式为:

$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{cases}$$

公式的推导可参考右图



变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵

$$T = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

上面是点P(x, y)以坐标原点为中心的旋转变换，还可以任意点P<sub>c</sub>(x<sub>c</sub>, y<sub>c</sub>) 为中心做旋转变换。其变换公式为：

此公式的推导过程可以这样考虑，先平移坐标原点(0,0)到(x<sub>c</sub>,y<sub>c</sub>)，然后进行旋转变换，变换后再将坐标原点移回到(0,0)。三个过程的结果就是以点(x<sub>c</sub>,y<sub>c</sub>)为中心的旋转变换。

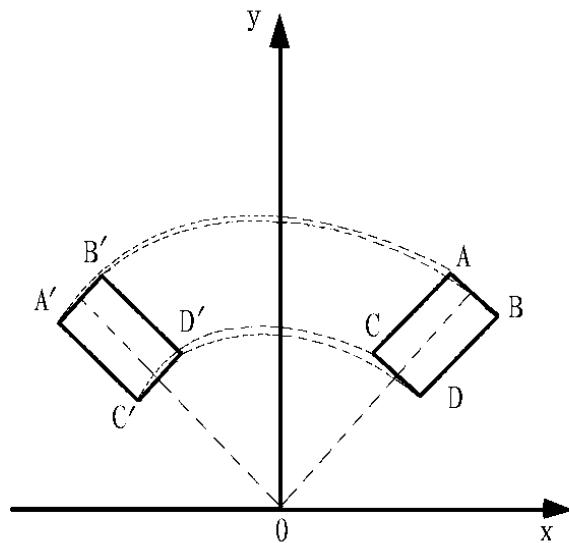
写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_c(1-\cos\theta) + y_c \sin\theta & y_c(1-\cos\theta) - x_c \sin\theta & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ x_c(1-\cos \theta) + y_c \sin \theta & y_c(1-\cos \theta) - x_c \sin \theta & 1 \end{bmatrix}$$

旋转变换只能改变图形的方位，而图形的大小和形状不变。旋转变换的几何表示见下图。



## 6.1.5 对称变换

对称变换是产生图形镜象的一种变换，也称镜象变换或反射变换。将图形绕对称轴旋转就可以生成镜象图形。

- 对称于X轴
- 对称于Y轴
- 对称于原点
- 对称平行于X轴的直线
- 对称平行于Y轴的直线
- 对称任意一点
- 对称任意一轴

# 1. 对称于X轴

当变换对称于X轴时，则坐标点 $P(x,y)$ 经对称变换后，新坐标点 $P'(x',y')$ 的表达式为：

$$\begin{cases} x' = x \\ y' = -y \end{cases}$$

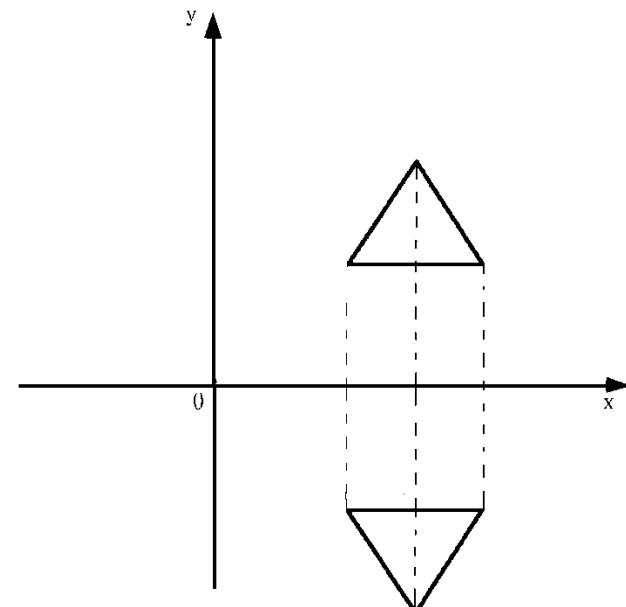
变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

对称X轴变换的几何表示见右图



## 2. 对称于Y轴

当变换对称于Y轴时，则坐标点P(x, y)经对称变换后，新坐标点P'(x', y')的表达式为：

$$\begin{cases} x' = -x \\ y' = y \end{cases}$$

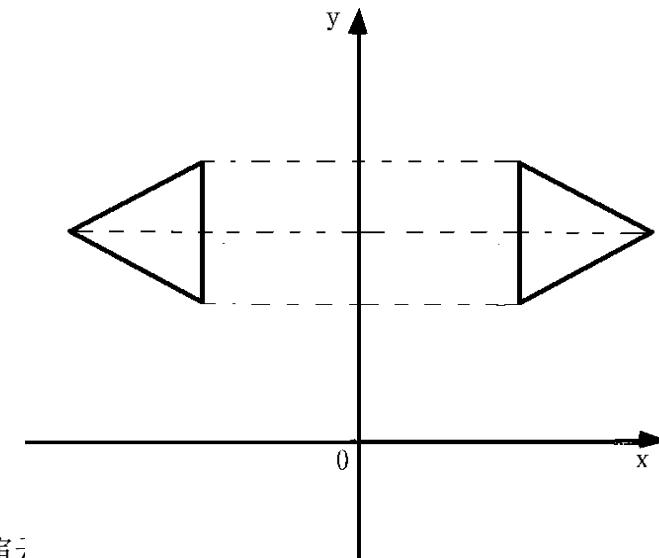
变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

对称Y轴变换的几何表示见右图



### 3. 对称于原点

当图形对X轴和Y轴都进行对称变换时，即得相对于坐标原点的对称变换。这一变换前后点坐标之间的关系为：

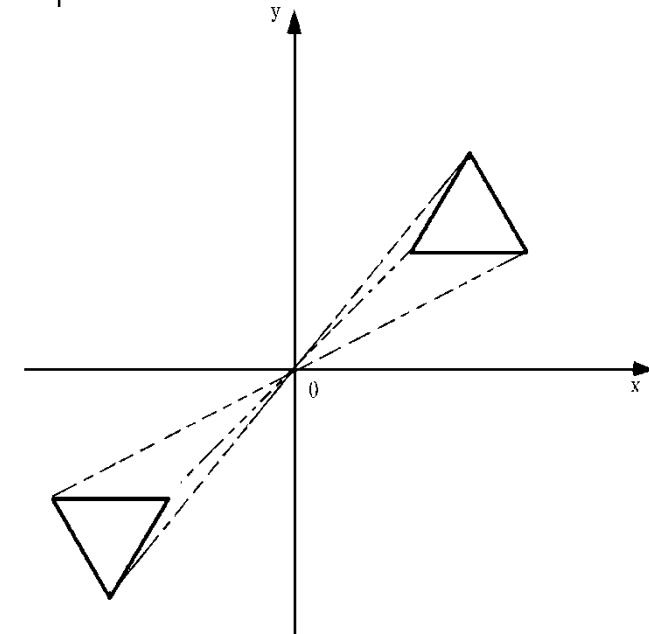
$$\begin{cases} x' = -x \\ y' = -y \end{cases}$$

写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



#### 4. 对称平行于X轴的直线

当对称轴是平行于X轴的直线 $y = y_c$ 时，变换前后点的坐标之间的关系为：

$$\begin{cases} x' = x \\ y' = -(y - y_c) + y_c = -y + 2y_c \end{cases}$$

变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 2y_c & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 2y_c & 1 \end{bmatrix}$$

## 5. 对称平行于Y轴的直线

当对称轴是平行于Y轴的直线 $x = x_c$ 时，变换前后点的坐标之间的关系为：

$$\begin{cases} x' = -(x - x_c) + x_c = -x + 2x_c \\ y' = y \end{cases}$$

变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 2x_c & 0 & 1 \end{bmatrix}$$

$$\text{其中变换矩阵: } T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 2x_c & 0 & 1 \end{bmatrix}$$

## 6. 对称于任一点 $(x_c, y_c)$ 的变换

对称于任一点  $(x_c, y_c)$  的变换，实际上可以看做分别相对于直线轴  $x = x_c$  和直线轴  $y = y_c$  的两次对称变换，因此其变换公式是两者的综合：

$$\begin{cases} x' = -x + 2x_c \\ y' = -y + 2y_c \end{cases}$$

变换方程写成齐次坐标矩阵形式为：

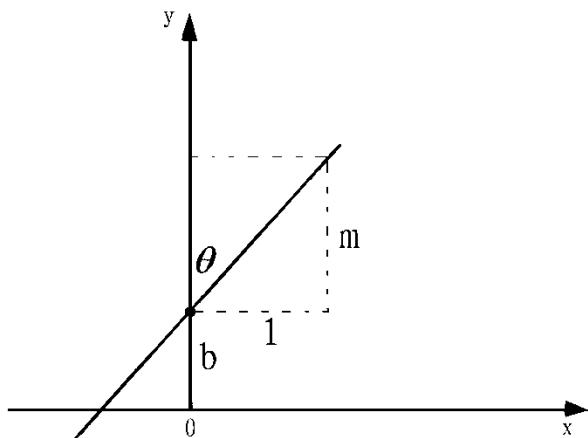
$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 2x_c & 2y_c & 1 \end{bmatrix}$$

其中变换矩阵： $T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 2x_c & 2y_c & 1 \end{bmatrix}$

## 7. 对称于任一轴的变换

关于XY平面内任一直线 $y = mx + b$ 为对称轴的变换，可以分解为平移、旋转、对称于坐标轴等变换的组合。首先平移直线经过坐标原点，而后将直线绕坐标原点旋转至同某一坐标轴重合，做对称于坐标轴的变换，最后反向旋转和反向平移将直线置回原处。

如下图所示，平移直线经过坐标原点需要在Y轴方向上移动距离 $b$ ，然后将直线绕坐标原点旋转至同Y轴重合，设旋转角度为 $\theta$ ，两步的变换矩阵分别为：



$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -b & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

做对称于Y轴的对称变换，其变换矩阵为： $T_3 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

最后反向旋转和反向平移将直线置回原处，其变换矩阵分别为：

$$T_4 = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) & 0 \\ -\sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

所以，对称于任一轴 $y = mx + b$ 的变换矩阵为：

$$T = T_1 T_2 T_3 T_4 T_5 = \begin{bmatrix} \sin^2 \theta - \cos^2 \theta & 2\sin \theta \cos \theta & 0 \\ 2\sin \theta \cos \theta & \cos^2 \theta - \sin^2 \theta & 0 \\ -2b \sin \theta \cos \theta & -b(\cos^2 \theta - \sin^2 \theta) + b & 1 \end{bmatrix}$$

变换矩阵中的和需要用已知量表示出来。当m为直线斜率，b为截距时有：

$$\cos(90^\circ - \theta) = \frac{1}{\sqrt{1+m^2}} = \cos 90^\circ \cos \theta + \sin 90^\circ \sin \theta = \sin \theta$$

$$\sin(90^\circ - \theta) = \frac{m}{\sqrt{1+m^2}} = \sin 90^\circ \cos \theta + \cos 90^\circ \sin \theta = \cos \theta$$

所以  $\cos^2 \theta - \sin^2 \theta = \frac{m^2 - 1}{1 + m^2}$

替换变换矩阵中的和得：

$$T = \begin{bmatrix} \frac{1-m^2}{1+m^2} & \frac{2m}{1+m^2} & 0 \\ \frac{2m}{1+m^2} & \frac{m^2-1}{1+m^2} & 0 \\ \frac{-2bm}{1+m^2} & \frac{b(1-m^2)}{1+m^2} + b & 1 \end{bmatrix}$$

上述变换用代数方程表示为：

$$x' = \frac{1-m^2}{1+m^2} x + 2(y-b) \frac{m}{1+m^2}$$

## 6.1.6 错切变换

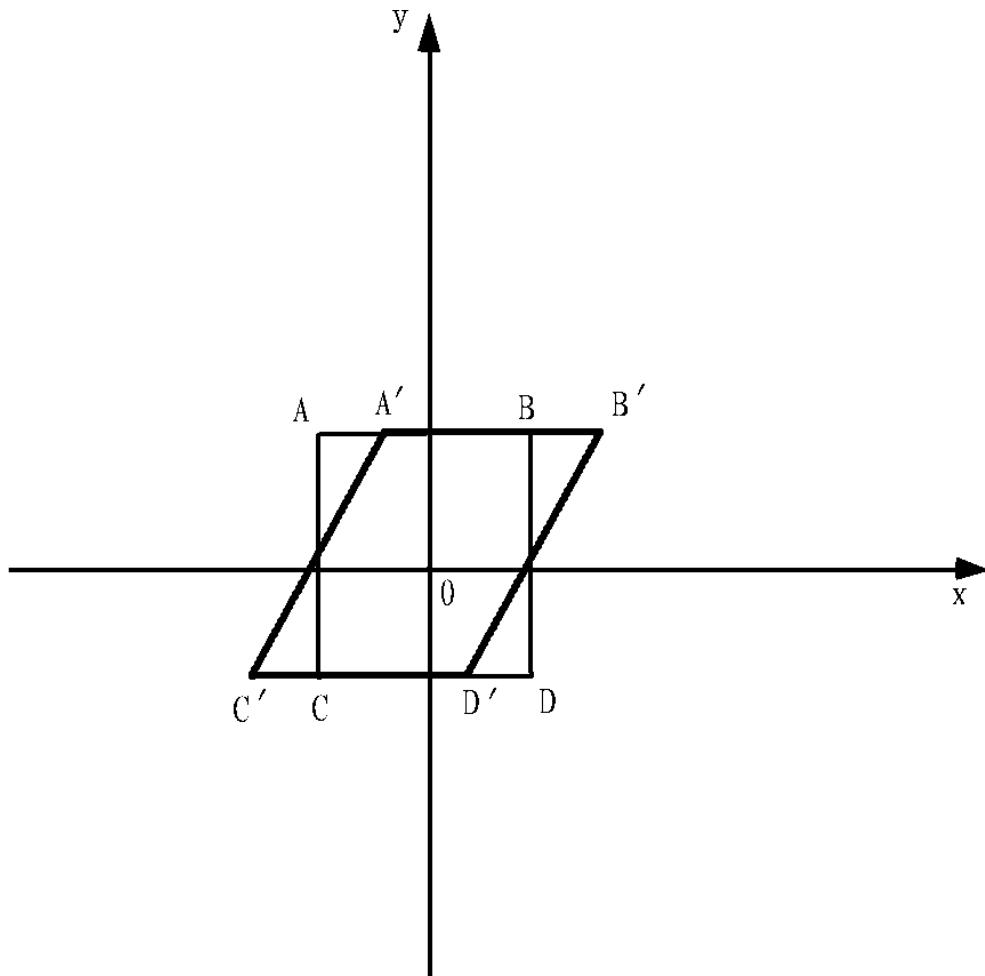
错切（shear）变换是轴上点不动，其它点沿平行于此轴方向移动变形的变换。错切变换也称为剪切、错位或错移变换。常用的错切变换有两种：改变x坐标值和改变y坐标值。

# 1. 沿X轴方向关于Y的错切

变换前和变换后y坐标不变，而x坐标根据y坐标值呈线性变化。变换前后点的坐标之间的关系为：

$$\begin{cases} x' = x + cy \\ y' = y \end{cases}$$

式中c为错切系数。若  $c > 0$ ，则沿+X方向错切，若  $c < 0$ ，则沿-X方向错切。右图说明了矩形ABCD经错切变换后变为A'B'C'D'的结果。



变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

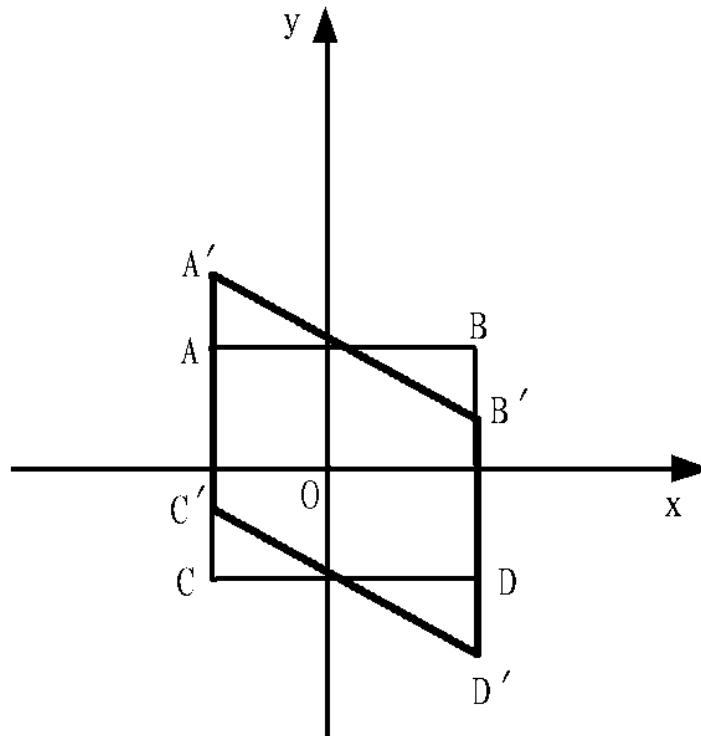
$$T = \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## 2. 沿Y轴方向关于X的错切

变换前和变换后x坐标不变，而y坐标根据x坐标值呈线性变化。变换前后点的坐标之间的关系为：

$$\begin{cases} x' = x \\ y' = dx + y \end{cases}$$

式中d为错切系数。若d  
 $>0$ ，则沿+Y方向错切，  
若d<0，则沿-Y方向错  
切。右图说明了矩形  
ABCD经错切变换后结果  
为A'B'C'D'。



变换方程写成齐次坐标矩阵形式为：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} 1 & d & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中变换矩阵：

$$T = \begin{bmatrix} 1 & d & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

除了沿X轴方向和沿Y轴方向的错切变换外，还可以使用沿平行于X轴方向的轴线或沿平行于Y轴方向的轴线以及任一轴线的错切变换。对于这些变换，可以通过先平移、旋转轴线，转化为沿X轴方向或沿Y轴方向的错切变换。

- 错切变换不仅改变图形的形状，而且改变图形的方位，还可能使图形发生畸变。

上面讨论的五种变换给出的都是点变换的公式，图形的变换实际上都可以通过点变换完成。例如直线段的变换可通过变换两个端点，并重画新端点间的线而得到。多边形的变换可通过变换每个顶点，并用新的顶点来生成多边形而实现。曲线的变换可通过变换控制点并重画线来完成。

符合下面形式：

$$\begin{cases} x' = a_{11}x + a_{12}y + a_{13} \\ y' = a_{21}x + a_{22}y + a_{23} \end{cases}$$

的坐标变换称为二维仿射变换（Affine Transformation）。变换的坐标 $x'$ 和 $y'$ 都是原始坐标 $x$ 和 $y$ 的线性函数。参数 $a_{ij}$ 是由变换类型确定的常数。仿射变换具有平行线转换成平行线和有限点映射到有限点的一般特性。

# 二维图形的几何变换

- 设二维图形变换前坐标为 $(x,y,1)$ , 变换后为 $(x^*,y^*)$ 二维变换矩阵

$$T_{2D} = \begin{pmatrix} a & d & g \\ b & e & h \\ c & f & i \end{pmatrix}$$

# 二维图形的几何变换

- 从变换功能上可把  $T2D$  分为四个子矩阵

$\begin{pmatrix} a & d \\ b & e \end{pmatrix}$ : 对图形进行缩放、旋转、对称、错切等变换。

$\begin{pmatrix} c & f \end{pmatrix}$ : 对图形进行平移变换。

$\begin{pmatrix} g \\ h \end{pmatrix}$ : 对图形做投影变换。

$g$ : 在  $x = \frac{1}{g}$  处产生一个灭点。

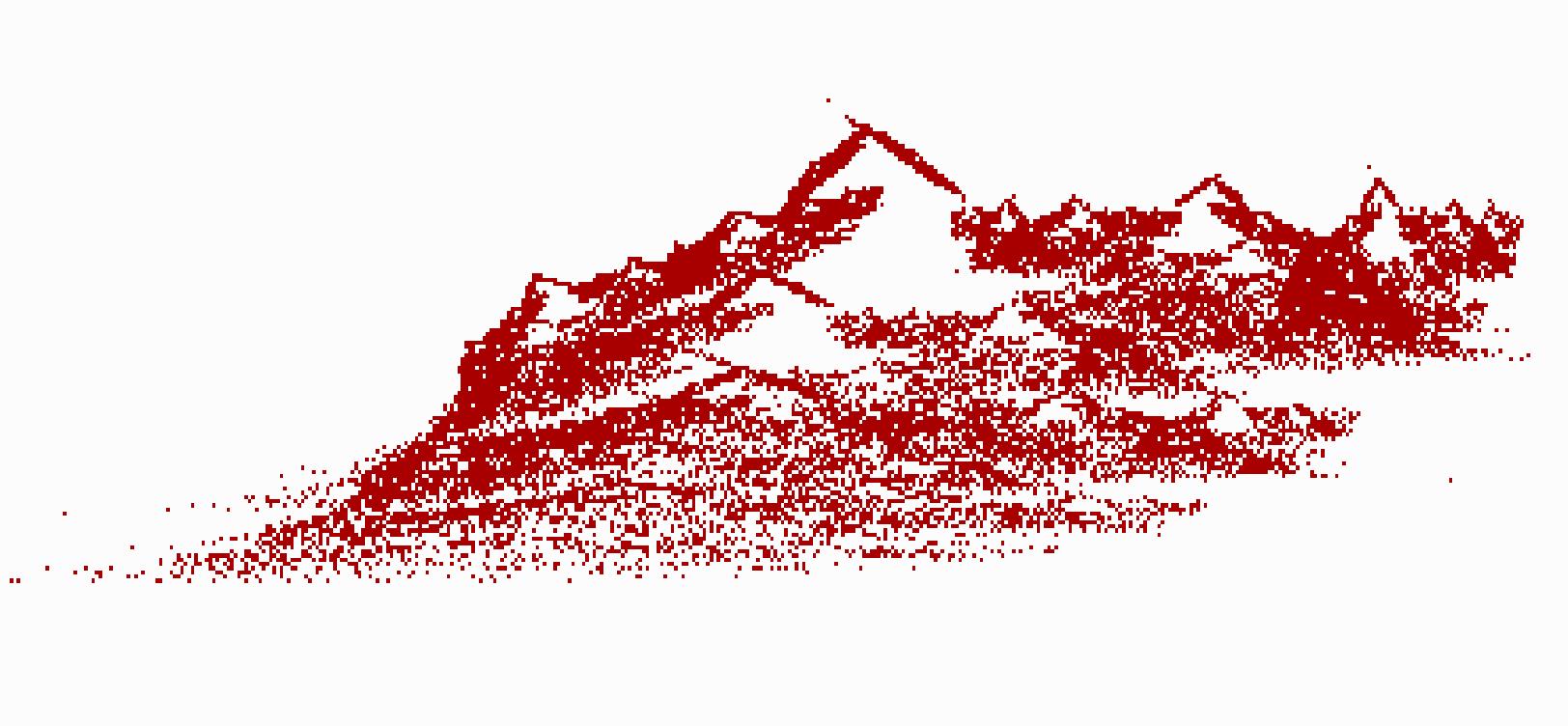
$h$ : 在  $x = \frac{1}{h}$  处产生一个灭点。

$(i)$ : 对整体图形进行伸缩变换。

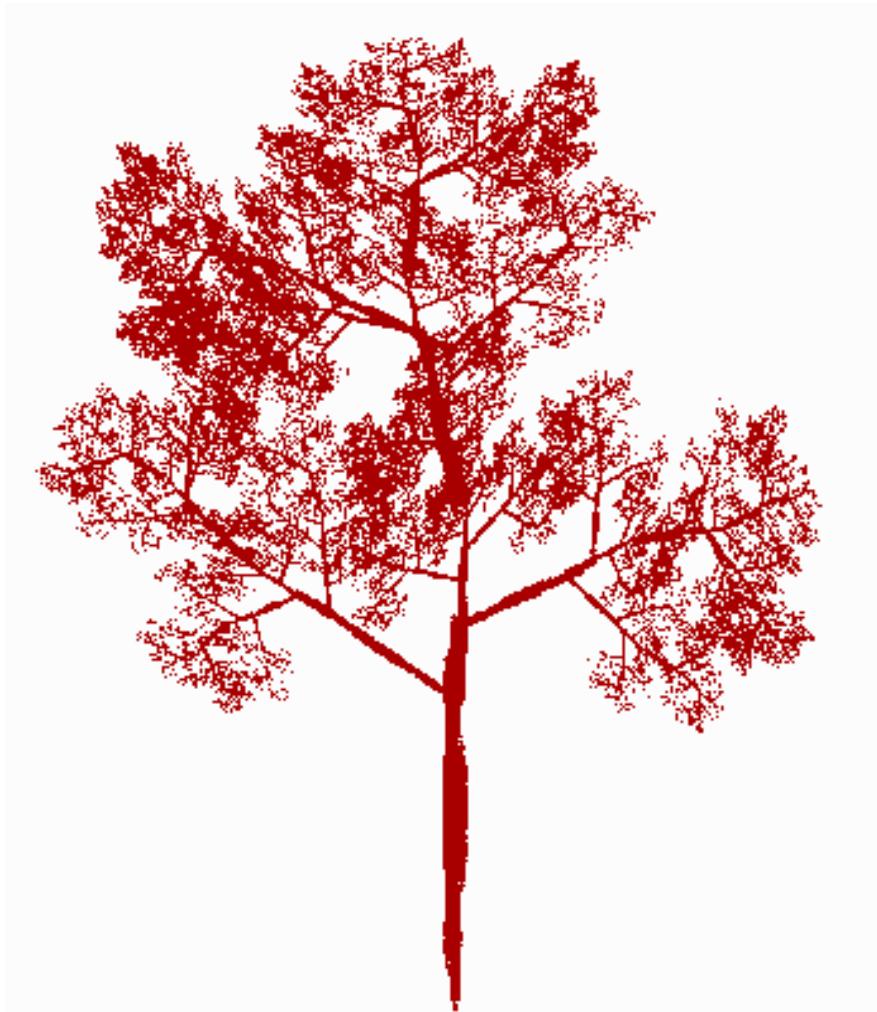
$$\therefore \begin{pmatrix} x^* & y^* & 1 \end{pmatrix} = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & i \end{pmatrix}$$

$\therefore$  若  $i > 1$ , 则总体缩小; 否则, 总体放大。

平移、比例、旋转、对称和错切变换都是二维仿射变换的特例，任何常用的二维仿射变换总可表示为这五种变换的组合。平移、比例、旋转、对称的仿射变换保持变换前后两直线间的角度、平行关系和长度之比不改变。







## 6.1.7 复合变换

所谓二维图形的复合变换，就是在XY平面内，对一个已定义的图形，按一定顺序进行多次变换而得到新的图形。一般把上面讨论的五种变换称为基本的图形变换，绝大部分复杂的图形变换都可以通过这些基本变换的适当组合来实现。利用前面所提供的矩阵表示，就可通过计算单个变换的矩阵乘积，将任意顺序变换的矩阵建立为复合变换矩阵。

•例：一个二维图形首先绕坐标原点旋转30度，然后沿x轴、y轴分别平移3个单位和2个单位，最后作对称于x轴的变换，求总的变换矩阵。要求写出每一步变换的变换矩阵。

•

$$T1 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \quad T3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T = T1 * T2 * T3$$

# 第六章

# 三维变换和投影



# 本章学习目标

---

- 三维图形基本几何变换矩阵
  - 平行投影
  - 透视投影
-

# 本章内容

---

- 三维基本几何
  - 三维基本几何变换矩阵
  - 三维复合变换
  - 投影变换
  - 透视变换
  - 本章小结
  - 习题
-

# 三维基本几何变换

---

□ 三维变换矩阵

□ 三维几何变换

---

# 二维变换

---

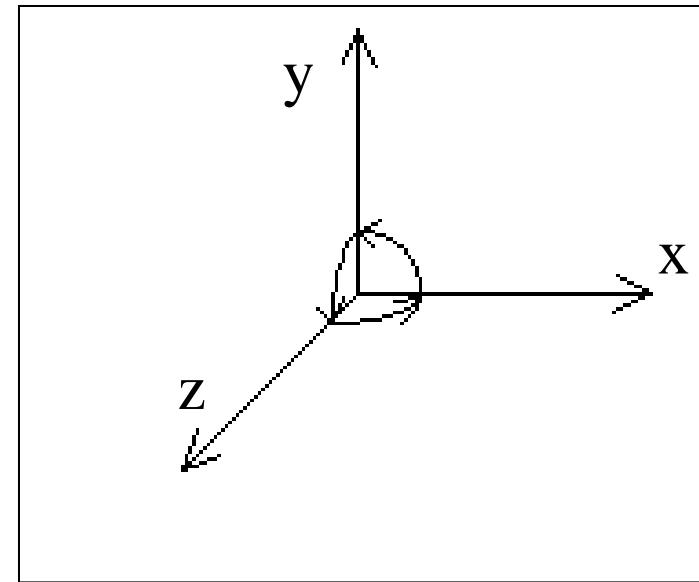
$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & b & q \\ c & d & p \\ l & m & s \end{bmatrix}$$

- **abcd**对图形作缩放、旋转、对称、错切变换
  - **lm**对图形作平移变换
  - **qp**对图形作投影变换
  - **s**对图形作整体变换
  - **acl**对x'起作用, **bdm**对y'起作用, **qps**对整体起作用。
-

# 三维齐次坐标

## □ 三维坐标，右手坐标系

- ◆ 旋转轴 正的旋转方向
- ◆ x      y→z
- ◆ y      z→x
- ◆ z      x→y



## □ 三维齐次坐标

- ◆  $(x, y, z)$  点对应的齐次坐标为  $(x_h, y_h, z_h, h)$

$$x_h = hx, y_h = hy, z_h = hz, h \neq 0$$

- ◆ 标准齐次坐标  $(x, y, z, 1)$

# 三维变换矩阵

---

## □ 三维几何变换

- ◆ 三维几何变换是二维几何变换的推广
- ◆ 三维几何变换在齐次坐标空间中可以用 $4\times 4$ 的变换矩阵表示
- ◆ 变换矩阵：

$$\left[ \begin{array}{ccc|c} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ \hline l & m & n & s \end{array} \right]$$

# 三维基本几何变换

- 三维基本几何变换都是相对于坐标原点和坐标轴进行的几何变换
- 假设三维形体变换前一点为 $p(x,y,z)$ , 变换后为 $p'(x',y',z')$ 。

$$p' = \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = p \cdot T_{3D} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{bmatrix}$$

---

□ 其中  $T_1 = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$  对图形进行比例、旋转、反射和错切变换。

□  $T_2 = [l \ m \ n]$  对图形进行平移变换。

□  $T_3 = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$  对图形进行投影变换。

□  $T_4 = [s]$  对图形进行整体比例变换。

---

# 三维几何变换

---

- 对于线框模型的变换，通常是以点变换为基础
  - 三维几何变换的基本方法是把变换矩阵作为一个算子，作用到变换前的图形顶点集合的坐标矩阵上，得到变换后新的图形顶点集合的坐标矩阵
  - 连接变换后的新的图形顶点，可以绘制出变换后的三维图形。
-

设图形变换前的顶点集合的规范化齐次坐标矩阵为：

$$P = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \dots & \dots & \dots & \dots \\ x_n & y_n & z_n & 1 \end{bmatrix}$$

变换后的顶点集合的规范化齐次坐标矩阵为：

$$P' = \begin{bmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ \dots & \dots & \dots & \dots \\ x'_n & y'_n & z'_n & 1 \end{bmatrix}$$

变换矩阵为：

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ l & m & n & s \end{bmatrix}$$

则三维图形基本几何变换有  $P' = P \cdot T$

即：

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \dots & \dots & \dots & \dots \\ x_n & y_n & z_n & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \dots & \dots & \dots & \dots \\ x_n & y_n & z_n & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ l & m & n & s \end{bmatrix}$$

# 三维基本几何变换矩阵

---

- 平移变换
  - 比例变换
  - 旋转变换
  - 反射变换
  - 错切变换
-

# 平移变换

平移变换的坐标表示为：

$$\begin{cases} x' = x + T_x \\ y' = y + T_y \\ z' = z + T_z \end{cases}$$

因此，三维平移变换矩阵为：

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

$T_x, T_y, T_z$ 是平移参数。

# 比例变换

比例变换的坐标表示为：

$$\begin{cases} x' = xS_x \\ y' = yS_y \\ z' = zS_z \end{cases}$$

因此，三维比例变换矩阵为：

$$T = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

这里  $S_x, S_y, S_z$  是比例系数

返 回

# 旋转变换

---

- 三维旋转一般看作是二维旋转变换的组合
  - 可以分为：绕x轴的旋转， 绕y轴的旋转，  
绕z轴的旋转。
  - 转角的正向满足右手定则：大拇指指向旋  
转轴，四指的转向为正向。
-

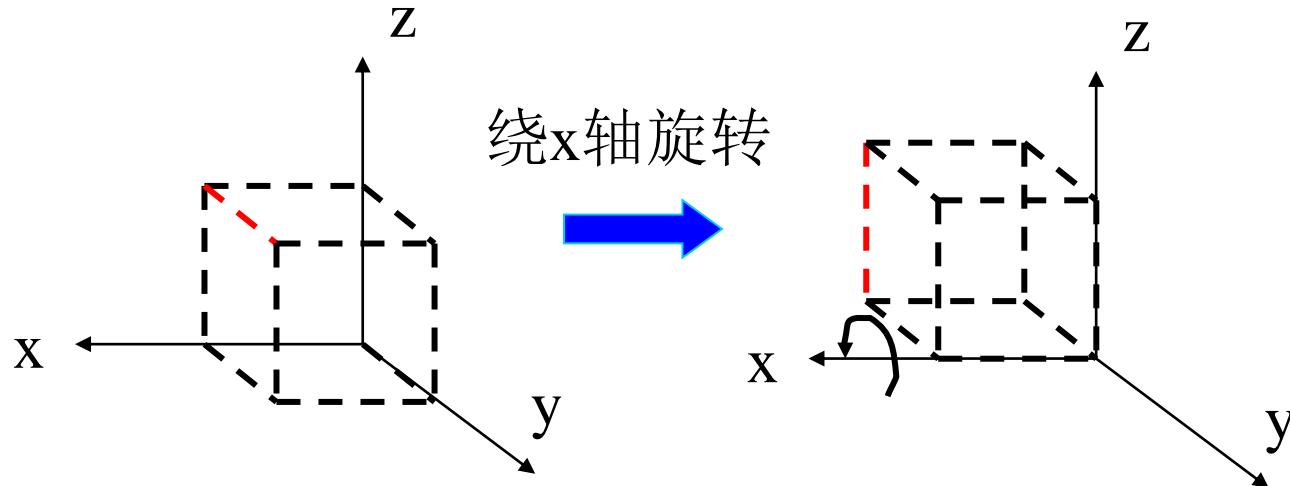
# 1. 绕x轴旋转

$$\begin{aligned}\cos(x+y) &= \cos x \cos y - \sin x \sin y \\ \sin(x+y) &= \sin x \cos y + \cos x \sin y\end{aligned}$$

绕x轴旋转变换的坐标表示为：

$$\begin{cases} x' = x \\ y' = y \cos \theta - z \sin \theta \\ z' = y \sin \theta + z \cos \theta \end{cases}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

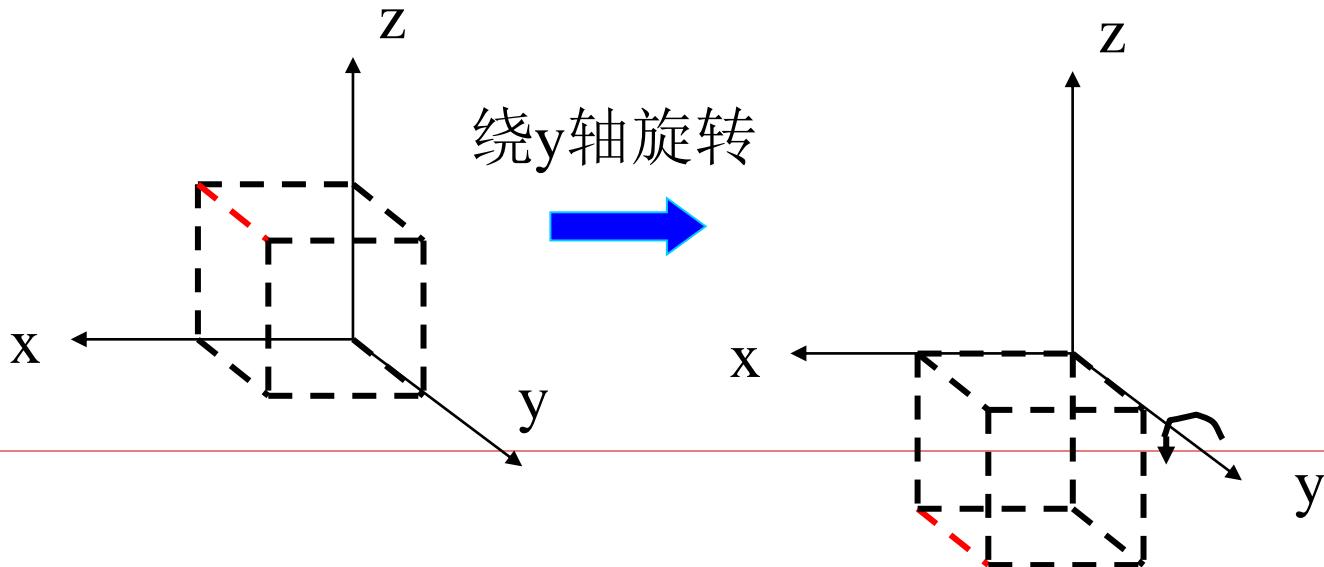


## 2. 绕y轴旋转

同理可得，绕y轴旋转变换：

$$\begin{cases} x' = z \sin \beta + x \cos \beta \\ y' = y \\ z' = z \cos \beta - x \sin \beta \end{cases}$$

$$T = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

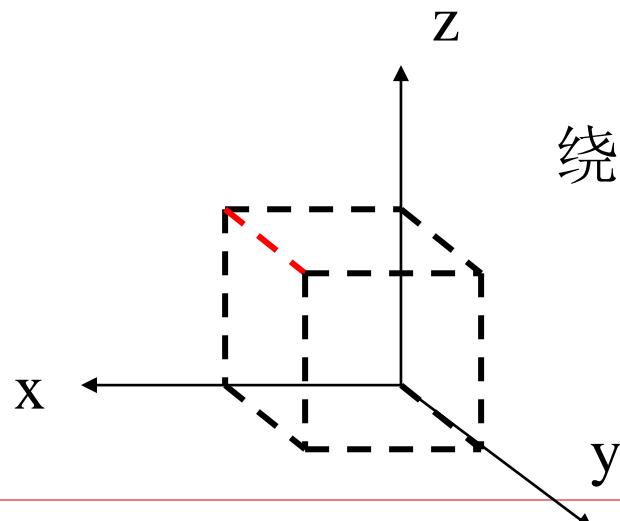


### 3. 绕z轴旋转

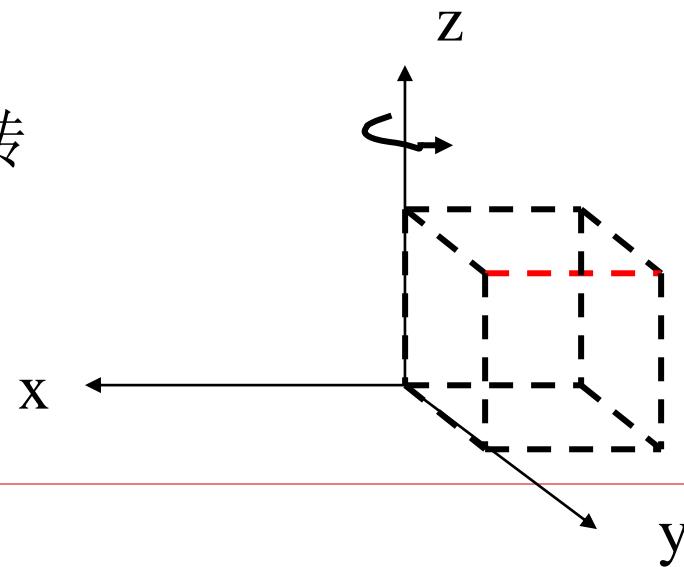
绕z轴旋转变换表示为：

$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \\ z' = z \end{cases}$$

$$T = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



绕z轴旋转



# 反射变换

---

三维反射分为两类：

- ◆ 关于坐标轴的反射
- ◆ 关于坐标平面的反射

## 1. 关于x轴的反射

坐标表示为：

$$\begin{cases} x' = x \\ y' = -y \\ z' = -z \end{cases}$$

变换矩阵为：

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 2. 关于y轴的反射

---

变换的坐标表示为：

$$\begin{cases} x' = -x \\ y' = y \\ z' = -z \end{cases}$$

变换矩阵为：

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 3.关于z轴的反射

---

变换的坐标表示为：

$$\begin{cases} x' = -x \\ y' = -y \\ z' = z \end{cases}$$

变换矩阵为：

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

## 4. 关于xoy面的反射

---

变换的坐标表示为：

$$\begin{cases} x' = x \\ y' = y \\ z' = -z \end{cases}$$

变换矩阵为：

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

## 5.关于yoz面的反射

---

坐标表示为：

$$\begin{cases} x' = -x \\ y' = y \\ z' = z \end{cases}$$

变换矩阵为：

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

## 6.关于zox面的反射

坐标表示为:

$$\begin{cases} x' = x \\ y' = -y \\ z' = z \end{cases}$$

变换矩阵为:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

返 回

# 错切变换

□ 三维错切变换的坐标表示为：

$$\begin{cases} x' = x + dy + gz \\ y' = y + bx + hz \\ z' = z + cx + fy \end{cases}$$

□ 因此，三维错切变换矩阵为：

$$T = \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ g & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# 错切变换

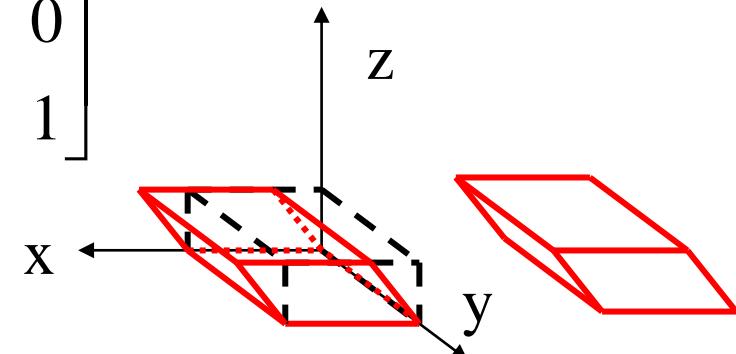
$$T = \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ g & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 三维错切变换中，一个坐标的变化受另外两个坐标变化的影响。
- 如果变换矩阵第一列中元素d和g不为0，产生沿x轴方向的错切；
- 第二列中元素b和h不为0，产生沿y轴方向的错切
- 第三列中元素c和f不为0，产生沿z轴方向的错切

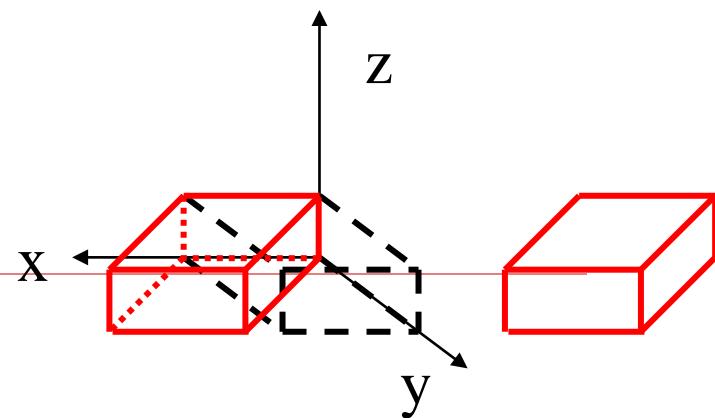
# 1. 沿x方向错切

$$b = 0, h = 0, c = 0, f = 0$$

$$\begin{cases} x' = x + dy + gz \\ y' = y \\ z' = z \end{cases} \quad T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d & 1 & 0 & 0 \\ g & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- ◆  $d = 0$  时，错切平面离开z轴，  
沿x方向移动gz距离；
- ◆  $g = 0$  时，错切平面离开y轴，  
沿x方向移动dy距离。



## 2. 沿y方向错切

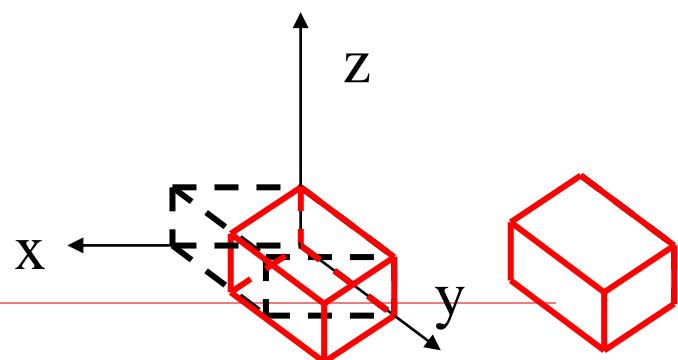
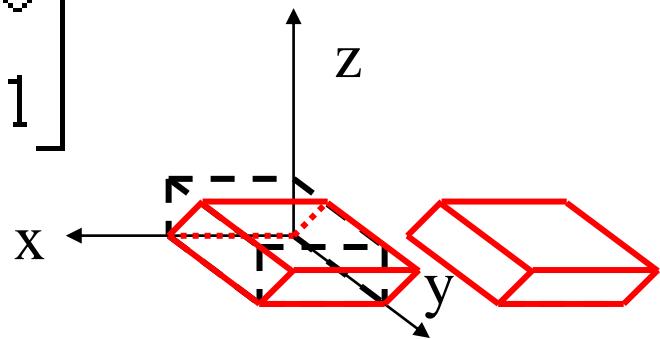
◆  $d = 0, g = 0, c = 0, f = 0$ 。

$$\begin{cases} x' = x \\ y' = y + bx + hz \\ z' = z \end{cases}$$

$$T = \begin{bmatrix} 1 & b & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

◆  $b = 0$ 时，错切平面离开z轴，  
沿y方向移动 $hz$ 距离；

◆  $h = 0$ 时，错切平面离开x轴，  
沿y方向移动 $bx$ 距离。



### 3. 沿z方向错切

◆  $d = 0, g = 0, b = 0, h = 0$

$$\begin{cases} x' = x \\ y' = y \\ z' = z + cx + fy \end{cases}$$

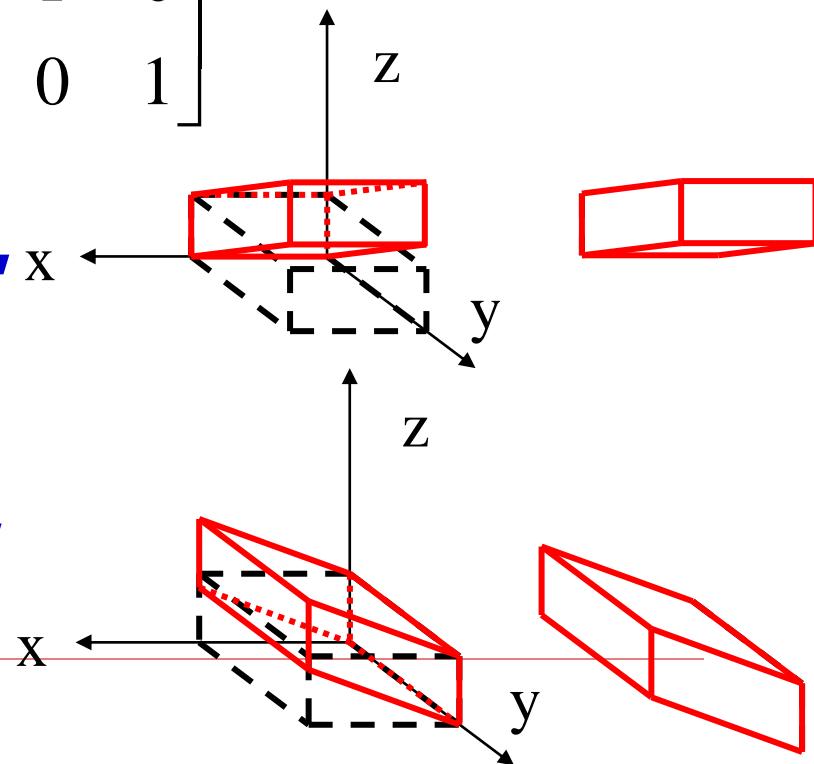
$$T = \begin{bmatrix} 1 & 0 & c & 0 \\ 0 & 1 & f & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

◆  $c = 0$  时，错切平面离开y轴，

沿z方向移动fy距离；

◆  $f = 0$  时，错切平面离开x轴，

沿z方向移动cx距离。



---

# 三维复合变换

---

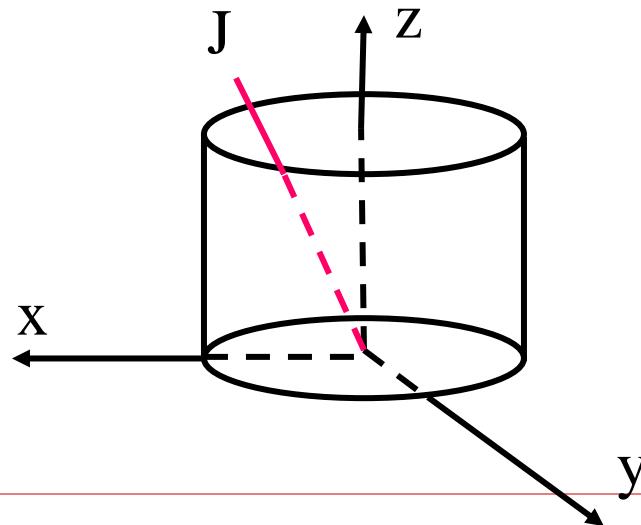
# 三维复合变换

---

- 基本几何变换是相对于坐标原点和坐标轴进行的几何变换
- 相对于任意点和任意方向的几何变换通过三维复合变换来实现
- 对三维图形按顺序进行多个基本变换，即可完成三维复合变换，**复合变换矩阵是每一步变换矩阵相乘的结果**

# 三维复合变换

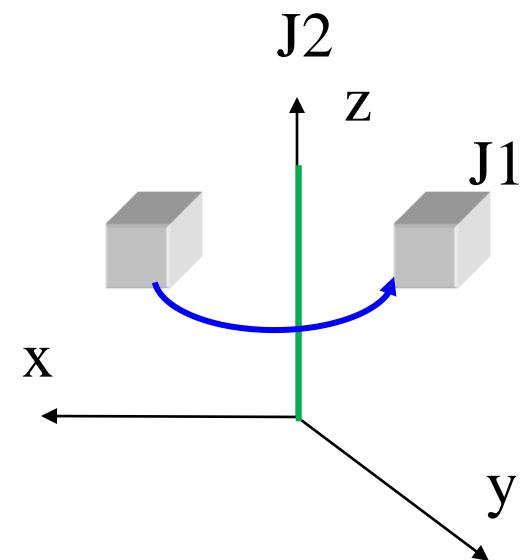
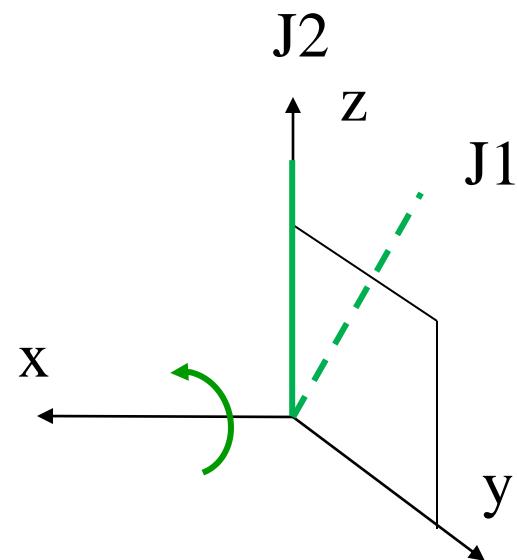
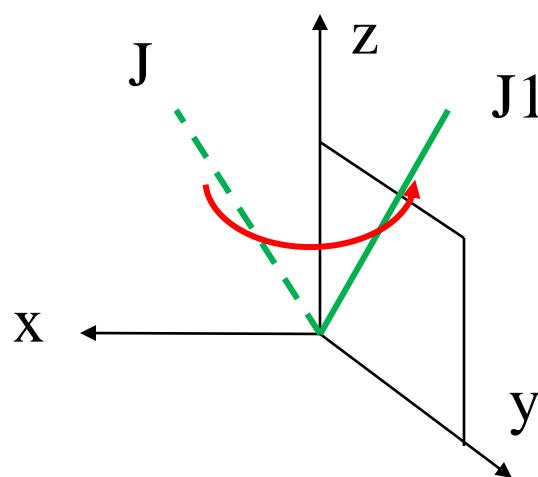
- 例子：使三维图形绕J轴旋转 $\theta$ 角
- 思路：将J轴重合Z轴之后，使立体**旋转 $\theta$ 角**，然后返回



# 三维复合变换

## □ 步骤：

1. J轴绕Z轴转 $\phi$ 角至yoz平面，成为J1。
2. J1轴绕X轴转 $\gamma$ 角后与z轴平行，成为J2。
3. 立体绕J2轴转 $\theta$ 角
4. 从J2返回J1。
5. 从J1返回J。

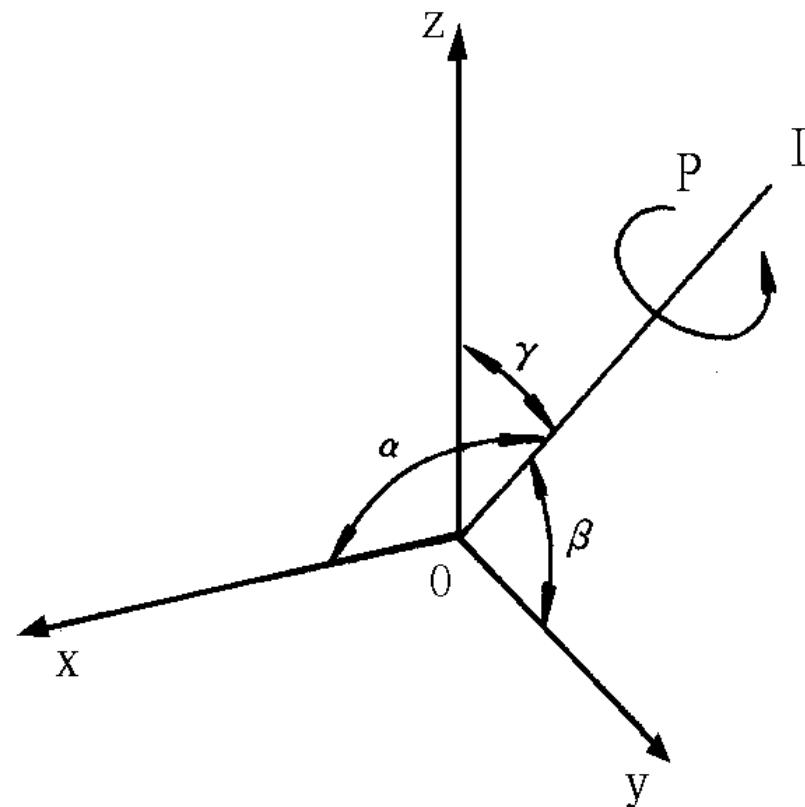


# 绕过原点的任意轴旋转变换

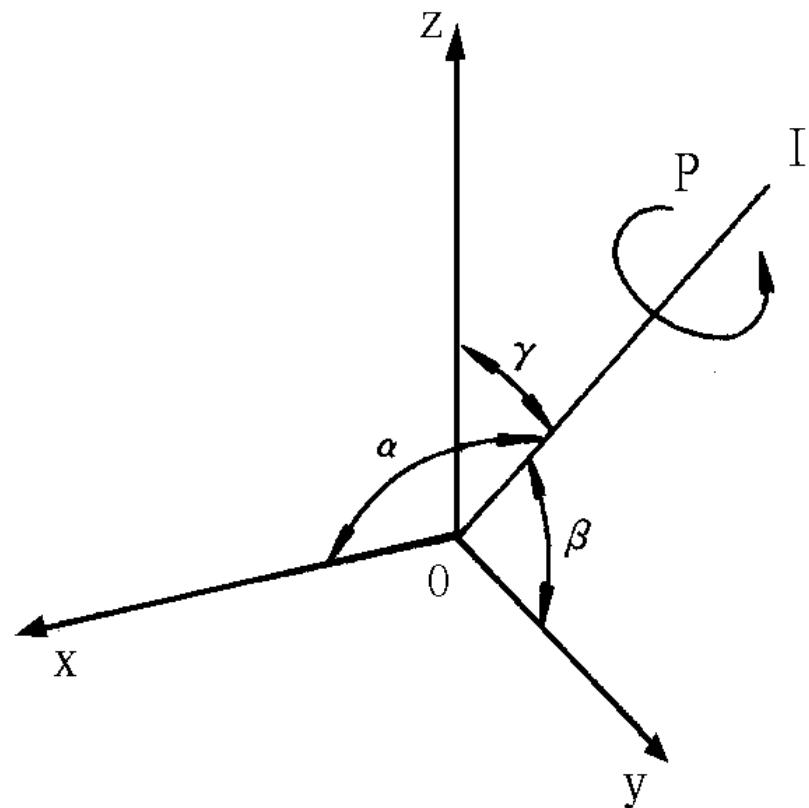
已知空间一点的坐标是  $P(x,y,z)$ , 设给定的旋转轴为  $I$ ,  
它对三个坐标轴的方向余弦分别为:

$$\begin{cases} n_1 = \cos\alpha \\ n_2 = \cos\beta \\ n_3 = \cos\gamma \end{cases}$$

如右图所示

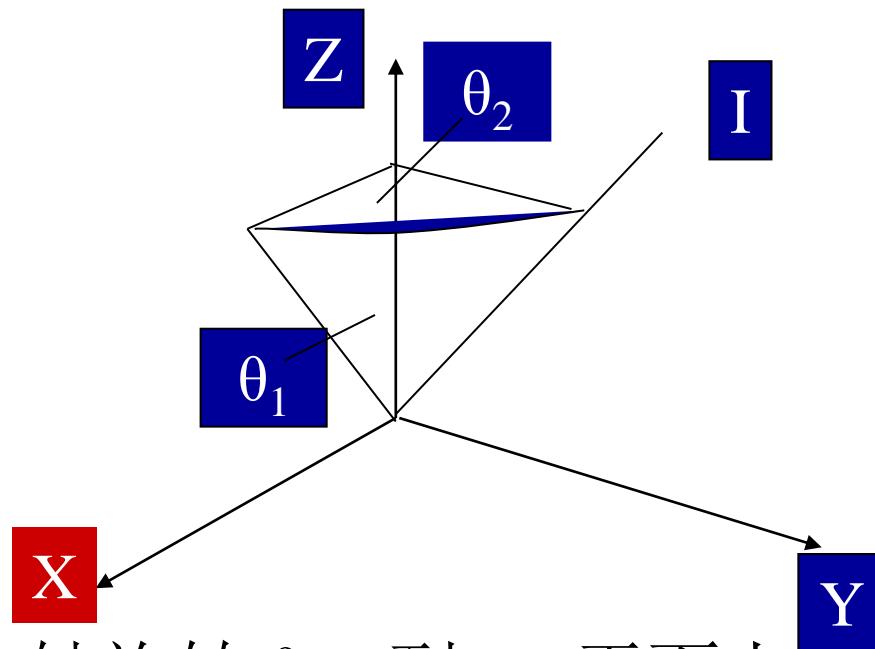


•基本思想：因I轴不是坐标轴，应设法旋转该轴，使之与某一坐标轴重合，然后进行旋转 $\theta$ 角的变换，最后按逆过程，恢复该轴的原始位置。



# 绕任意轴的旋转变换

能否转换成绕X、Y或Z轴旋转的变换？



I绕Z轴旋转  $\theta_2$  到XOZ平面上，然后再绕Y轴  
旋转  $\theta_1$ ，即可与Z轴重合。

这样，可得空间上任一点绕I轴旋转的变换过程如下：

- 1) 首先通过两次旋转，使I轴与Z轴重合；
- 2) 然后使点绕Z轴旋转  $\theta$  角；
- 3) 最后通过与1) 相反的旋转，使I轴回到原来的位置。

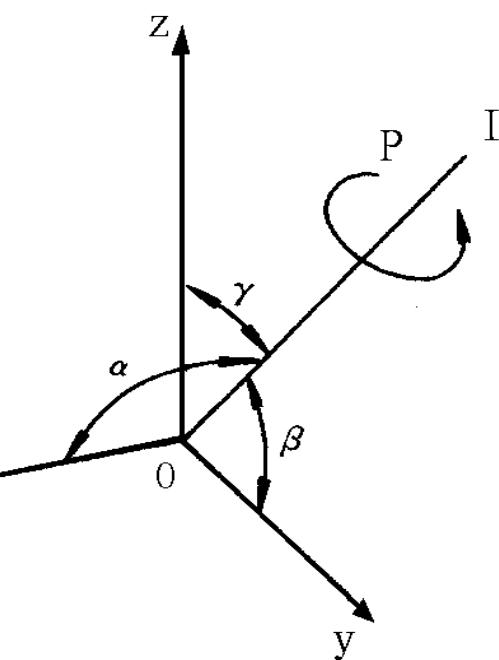
假设，绕Z轴的旋转-  $\theta_2$  矩阵为  $T_1$

绕Y轴的旋转-  $\theta_1$  矩阵为  $T_2$

绕Z轴的旋转  $\theta$  矩阵为  $T_3$

绕Y轴的旋转  $\theta_1$  矩阵为  $T_4$

绕Z轴的旋转  $\theta_2$  矩阵为  $T_5$



则总体变换矩阵为：

$$T = T_1 \ T_2 \ T_3 \ T_4 \ T_5$$

由上推导可看出，只要能求出  $\theta_1$ 、 $\theta_2$  的值，即可通过上式获得绕 I 轴的变换矩阵。

由于矢量  $(0 \ 0 \ 1)$  绕 Y 轴旋转  $\theta_1$ ，再绕 Z 轴旋转  $\theta_2$  即可与 I 轴重合。即：

$$[n_1 \ n_2 \ n_3 \ 1] = [0 \ 0 \ 1 \ 1] \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

$$[n_1 \ n_2 \ n_3 \ 1] = [\sin \theta_1 \cos \theta_2, \ \sin \theta_1 \sin \theta_2, \ \cos \theta_1, \ 1]$$

$$n_1 = \sin \theta_1 \cos \theta_2$$

$$n_2 = \sin \theta_1 \sin \theta_2$$

$$n_3 = \cos \theta_1$$

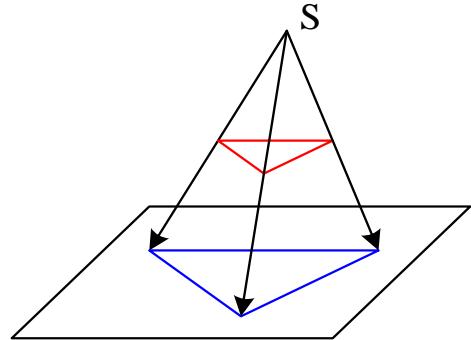
从而通过上式即可得到  $\theta_1$ 、 $\theta_2$  的值。

问题：当任一轴线的端点不在原点时，此时应如何计算变换矩阵？

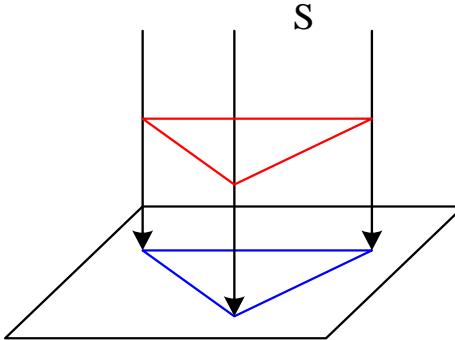
---

# 投影变换

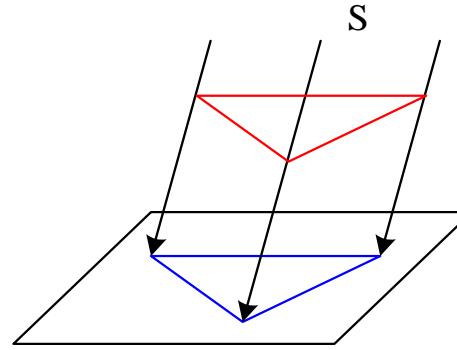
# 投影变换



(a) 透视投影



(b) 正投影

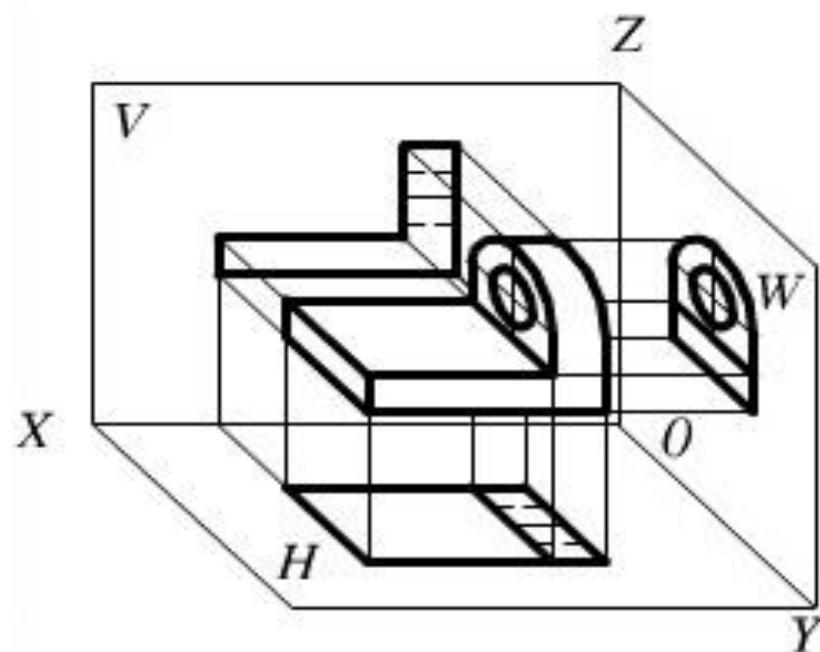


(c) 斜投影

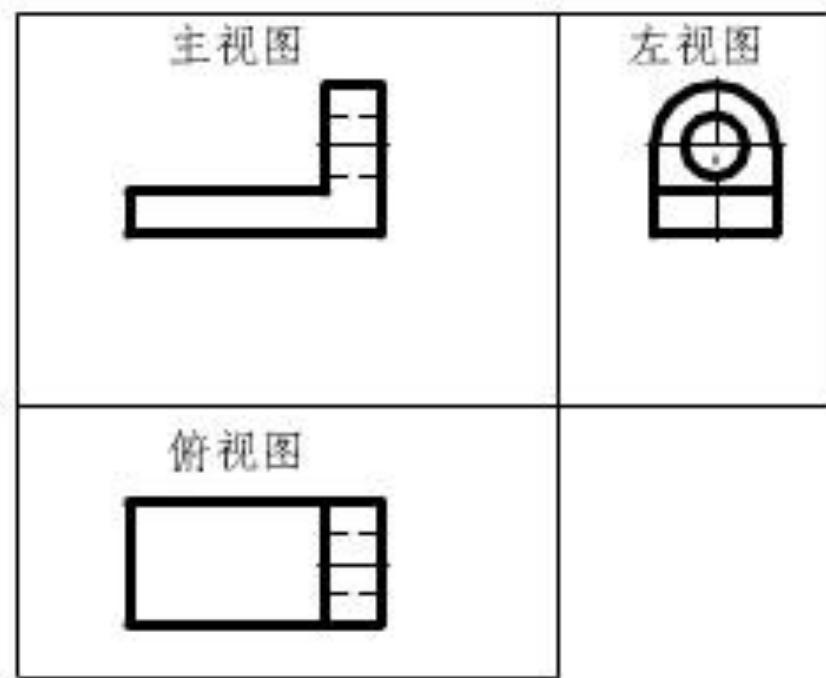
投影变换可分为两大类：

- 透视投影的投影中心到投影面之间的距离是有限的；
- 平行投影的投影中心到投影面之间的距离是无限的；
- 平行投影的最大特点是无论物体距离视点多远，投影后的物体尺寸保持不变。
- 平行投影可分成两类：正投影和斜投影。

# 三视图



(a) 空间投影图



(b) 三视图的展开

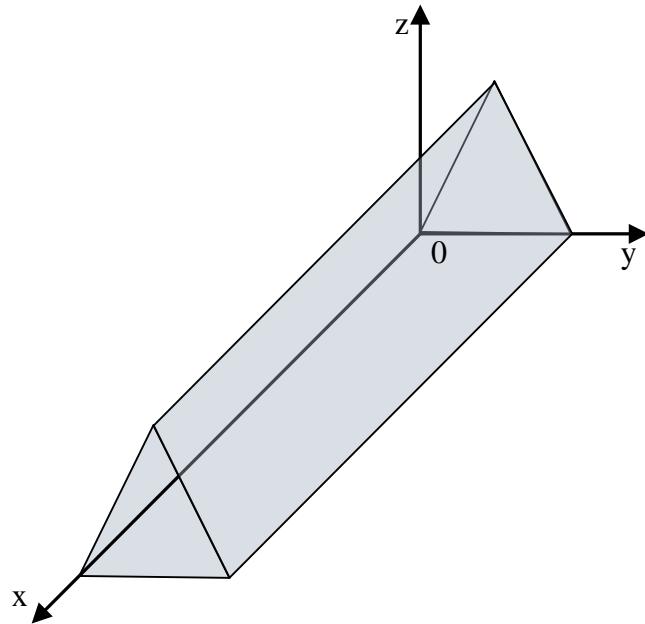
# 三视图

---

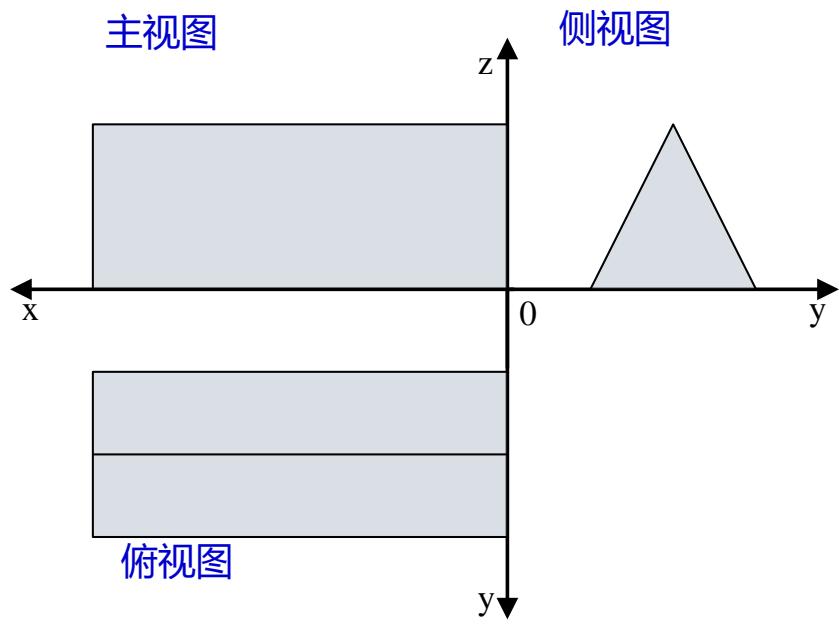
- 三视图是**正投影**视图
  - 包括主视图、俯视图和侧视图
  - 投影面分别与y轴、z轴和x轴垂直
  - 将三维物体分别对正面、水平面和侧平面做正投影得到三个基本视图
-

# 三视图

---



正三棱柱的立体图



正三棱柱的三视图

主视图的形成：

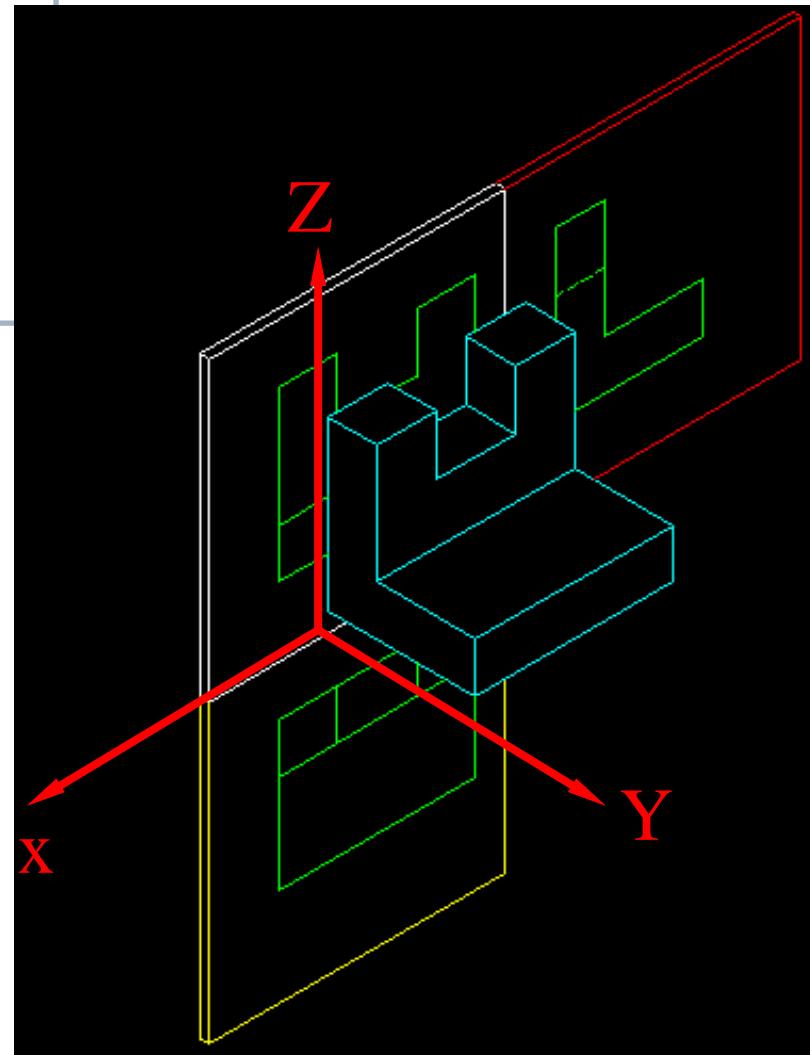
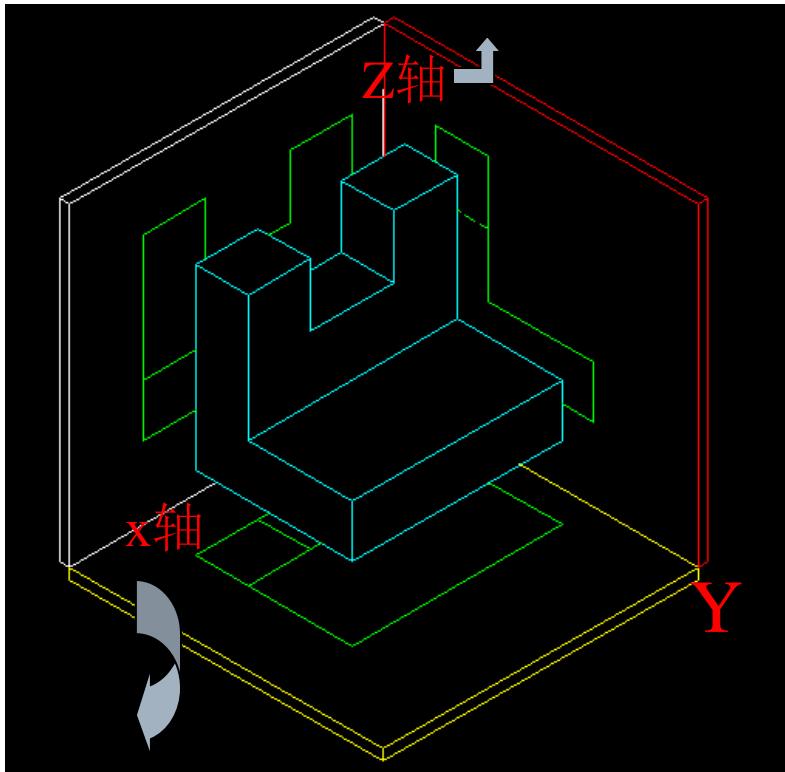
直接向V面（XOZ坐标面）投影；

俯视图的形成：

绕X轴向下旋转90度，平移N距离；

左视图的形成：

绕Z轴向后旋转90度，平移L距离。



# (1) 主视图

□ 将三棱柱向xoz面作平行投影，得到主视图。

- ◆ 设三棱柱上任一点坐标用P(x, y, z)表示
- ◆ 它在xoz面上投影后坐标为P'(x', y', z')
- ◆ 坐标关系  $x' = x$ ,  $y' = 0$ ,  $z' = z$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & 0 & z & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

□ 主视图投影变换矩阵为：

$$T_V = T_{xoz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## (2)俯视图

将三棱柱向xoy面作平行投影得到俯视图。

- ◆ 设三维物体上任一点坐标用P(x, y, z)表示
- ◆ 它在xoy面上投影后坐标为P' (x', y', z')
- ◆ 其中x' = x, y' = y, z' = 0。

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

xoy面投影变换矩阵为：

$$T_{xoy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## (2)俯视图

为了使俯视图和主视图在一个平面内，使xoy面绕x轴顺时针旋转90°，旋转变换矩阵为：

$$T_{Rx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-\frac{\pi}{2}) & \sin(-\frac{\pi}{2}) & 0 \\ 0 & -\sin(-\frac{\pi}{2}) & \cos(-\frac{\pi}{2}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

为了使俯视图和主视图有一定的间距，还要使xoy面沿z负方向平移一段距离 $z_0$

$$T_{Tz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -z_0 & 1 \end{bmatrix}$$

## (2)俯视图

俯视图的投影变换矩阵为上述三个变换矩阵的乘积：

$$T_H = T_{xoy} \cdot T_{Rx} \cdot T_{Tz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -z_0 & 1 \end{bmatrix}$$

俯视图总投影变换矩阵为：

$$T_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -z_0 & 1 \end{bmatrix}$$

### (3)侧视图

□将三维形体向yoz面作垂直投影得到侧视图。

- ◆设三维物体上任一点坐标用P(x, y, z)表示,
- ◆它在yoz面上投影后坐标为P'(x', y', z')。
- ◆其中x'=0, y'=y, z'=z。

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 0 & y & z & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

yoz面投影变换矩阵为：

$$T_{yoz} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### (3)侧视图

为了在xoz平面内表示侧视图，需要将yoz面绕z轴逆时针旋转90°，旋转变换矩阵为：

$$T_{Rz} = \begin{bmatrix} \cos \frac{\pi}{2} & \sin \frac{\pi}{2} & 0 & 0 \\ -\sin \frac{\pi}{2} & \cos \frac{\pi}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

为了使侧视图和主视图之间有一定的间距，还要将yoz面沿x轴负向平移一段距离 $x_0$

$$T_{Tx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & 0 & 0 & 1 \end{bmatrix}$$

### (3)侧视图

侧视图的投影变换矩阵为上面三个变换矩阵的乘积：

$$T_W = T_{yoz} \cdot T_{Rz} \cdot T_{Tx} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & 0 & 0 & 1 \end{bmatrix}$$

侧视图总投影变换矩阵为：

$$T_w = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & 0 & 0 & 1 \end{bmatrix}$$

# 三视图

---

- 从三视图的变换矩阵可以看出，三个视图中的y坐标始终为0，表明三个视图均落在xoz平面上，即三维物体用二维视图来表示。
  - 三视图是工程上常用的图样，由于三视图中物体的投影面平行于坐标平面，其投影能真实地反映物体的实际尺寸，三个视图具有长对正、高平齐、宽相等的特点，因此，机械工程中常用三视图来测量形体间的距离、角度等尺寸。
  - 但是三视图缺乏立体感，只有将主视图、俯视图和侧视图结合在一起加以抽象，才能获得物体的空间结构。
-

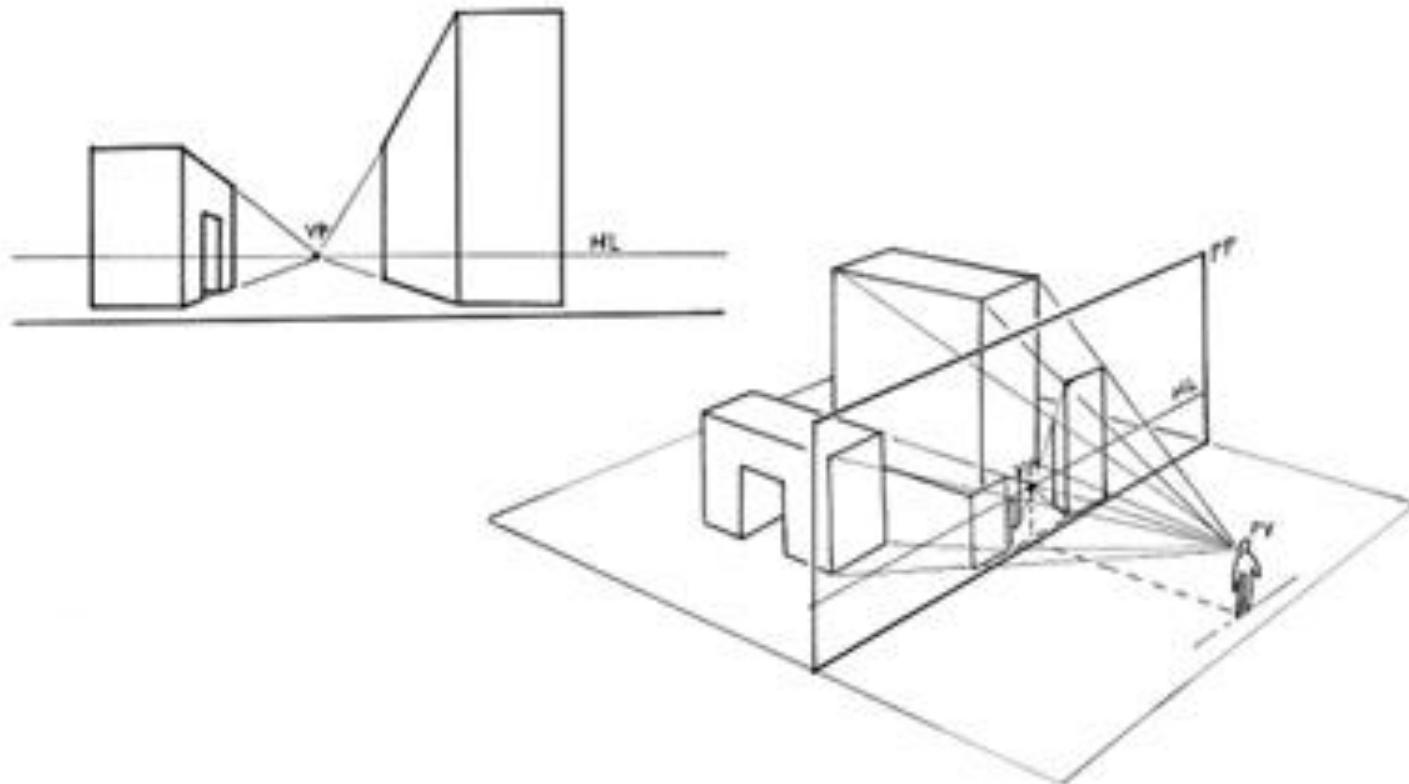
---

# 透视变换

---

# 透视变换

## □ 透过玻璃观察物体，玻璃上的投影



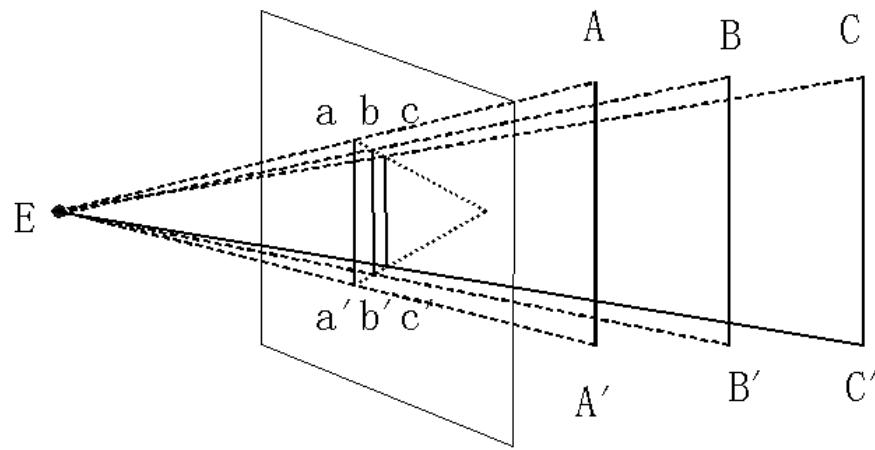
# 透视的基本知识

---

- 透视投影是一种中心投影法，在日常生活中，我们观察外界的景物时，常会看到一些明显的透视现象。
  - 如：我们站在笔直的大街上，向远处看去，会感到街上具有相同高度的路灯柱子，显得近处的高，远处的矮，越远越矮。这些路灯柱子，即使它们之间的距离相等，但是视觉产生的效果则是近处的间隔显得大，远处的间隔显得小，越远越密。观察道路的宽度，也会感到越远越窄，最后汇聚于一点。这些现象，称之为透视现象。
  - 产生透视的原因，可用下图来说明：
-

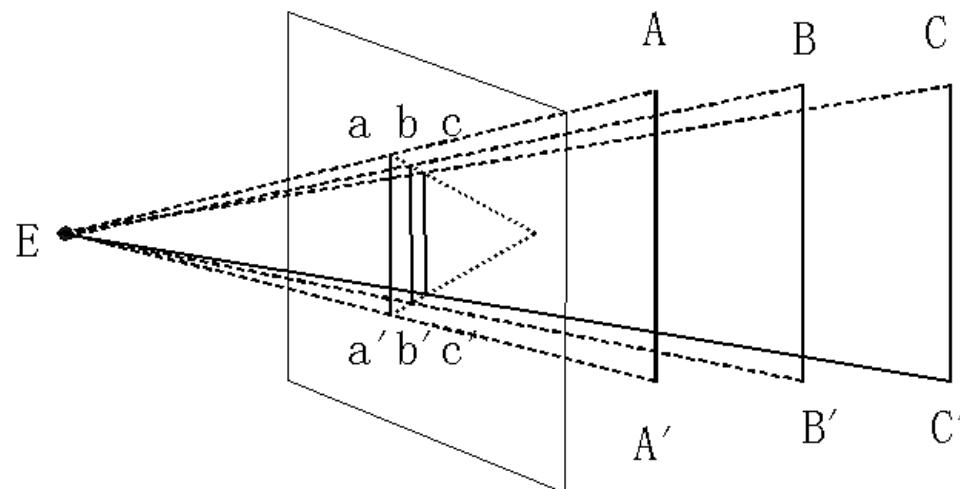
# 透视的基本知识

- 图中， $AA'$ , $BB'$ , $CC'$ 为一组高度和间隔都相等，排成一条直线的电线杆，从视点 $E$ 去看，发现
- $\angle AEA' > \angle BEB' > \angle CEC'$
- 若在视点 $E$ 与物体间设置一个透明的画面 $P$ ,让 $P$ 通过 $AA'$ ，则在画面上看到的各电线杆的投影 $aa' > bb' > cc'$
- $aa'$ 即 $EA, EA'$ 与画面 $P$ 的交点的连线；
- $bb'$ 即为 $EB, EB'$ 与画面 $P$ 的交点的连线。
- $cc'$ 即为 $EC, EC'$ 与画面 $P$ 的交点的连线。
- $\therefore$ 近大远小



# 透视的基本知识

- 若连 $a,b,c$ 及 $a',b',c'$ 各点，它们的连线汇聚于一点。
- 然而，实际上， $A,B,C$ 与 $A',B',C'$ 的连线是两条互相平行的直线，这说明**空间不平行于画面（投影面）的一切平行线的透视投影**，即 $a,b,c$ 与 $a',b',c'$ 的连线，**必交于一点**，这点我们称之为灭点



# 平面几何投影-透视投影

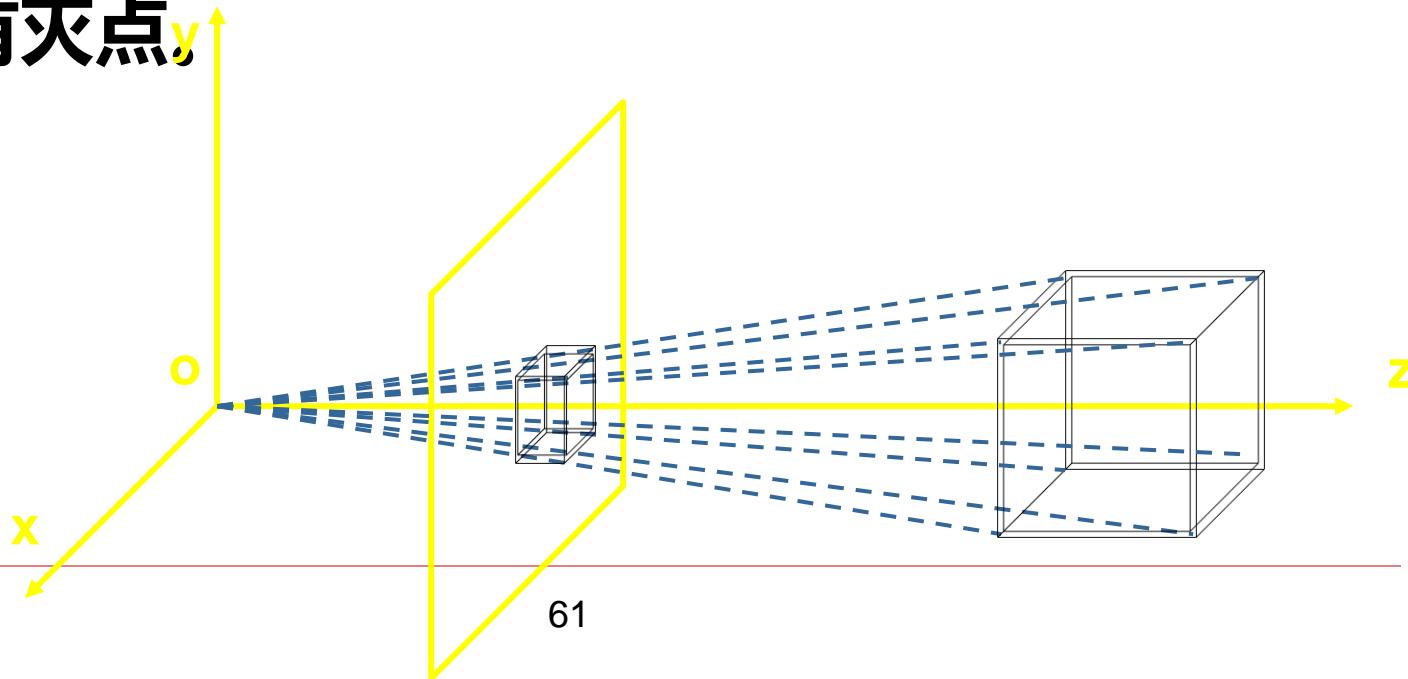
---

## - 透视投影

- 投影中心与投影平面之间的距离为有限
  - 灭点：不平行于投影平面的平行线，经过透视投影之后收敛于一点，称为灭点。
  - 主灭点：平行于坐标轴的平行线产生的灭点。
    - 一点透视
    - 两点透视
    - 三点透视
  - 特点：产生近大远小的视觉效果，由它产生的图形深度感强，看起来更加真实。
-

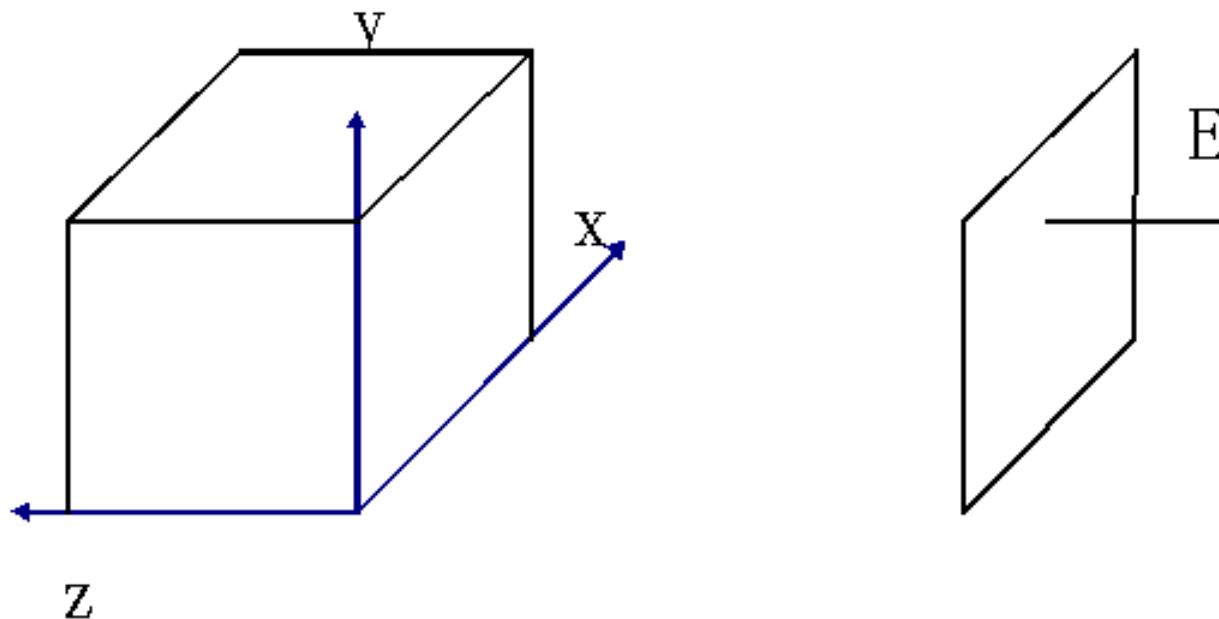
# 透视投影

□ 主灭点数是和投影平面切割坐标轴的数量相对应的, 即由坐标轴与投影平面交点的数量来决定的。如投影平面仅切割z轴, 则z轴是投影平面的法线, 因而只在z轴上有一个灭点, 平行于x轴或y轴的直线也平行于投影平面, 因而没有灭点



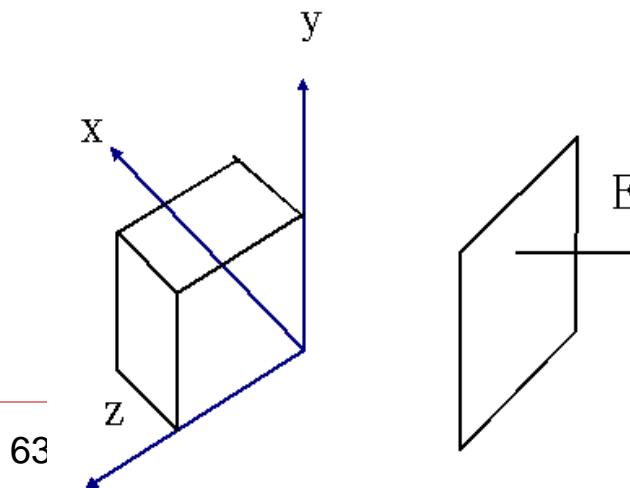
# 一点透视（平行透视）

□ 人眼从正面去观察一个立方体，当z轴与投影平面垂直时，另两根轴 $ox, oy$ 轴平行于投影平面。这时的立方体透视图只有一个灭点，即与画面垂直的那组平行线的透视投影交于一点。



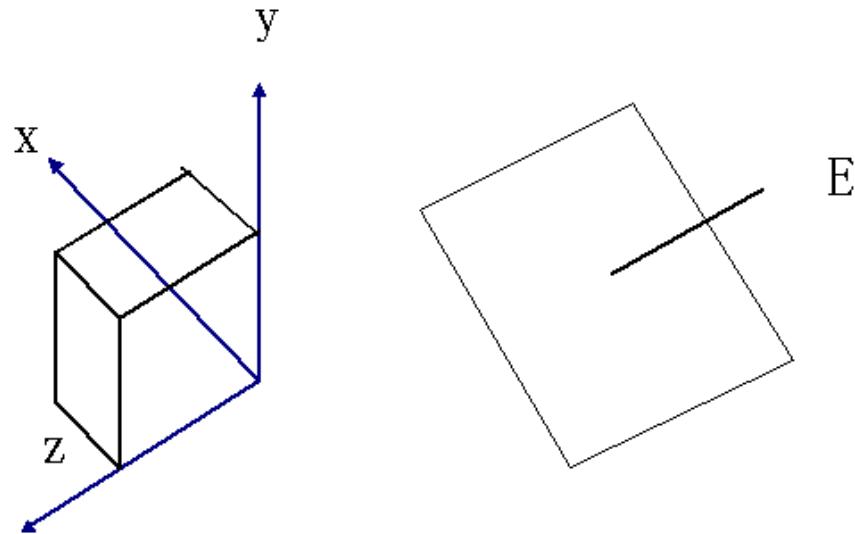
## 二点透视（成角透视）

- 人眼观看的立方体是绕 $y$ 轴旋转一个角度之后，再进行透视投影。三坐标轴中 $oy$ 轴与投影平面平行，而其它两轴与画面倾斜，这时除平行于 $oy$ 轴的那组平行线外，其它两组平行线的透视投影分别在投影平面的左右两侧，作出的立方体透视图产生两个灭点。

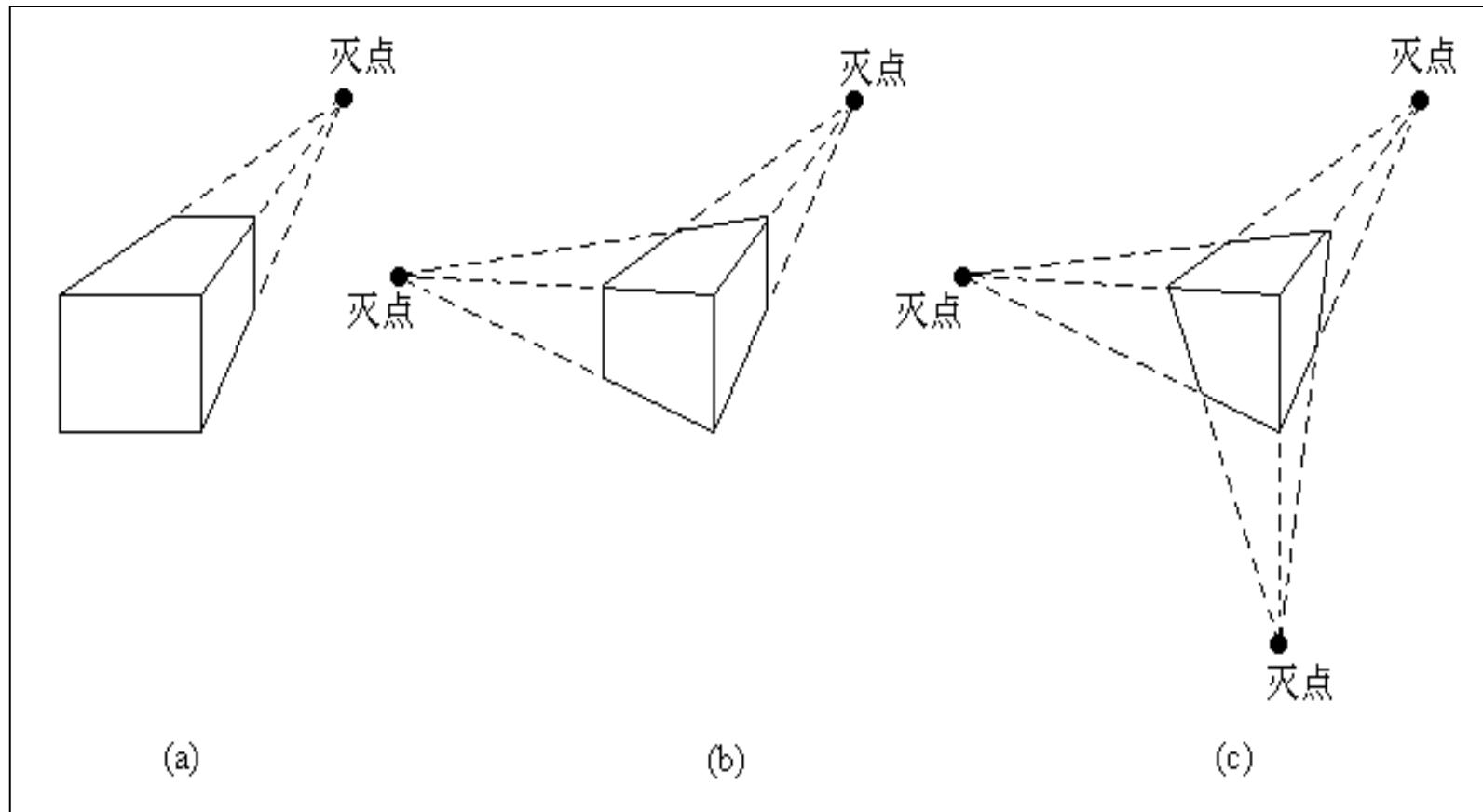


# 三点透视 (斜透视)

- 此时，投影平面与三坐标轴均不平行。
- 这时的三组平行线均产生灭点。



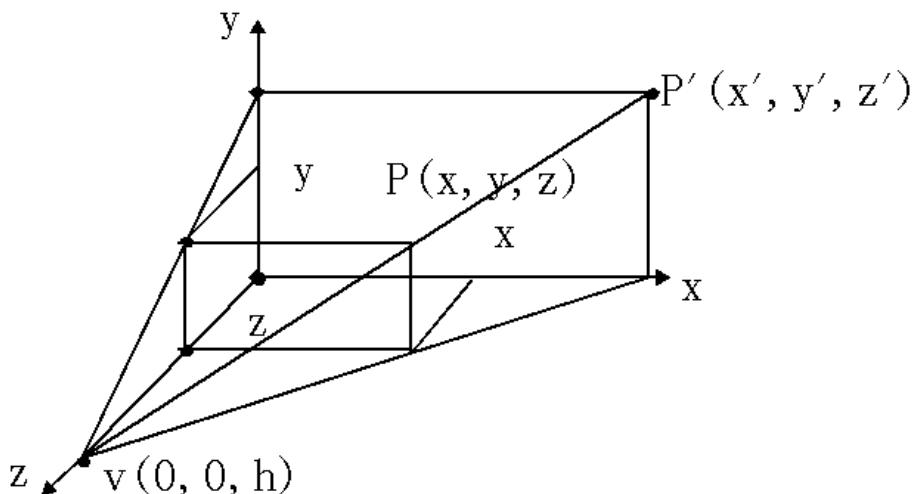
# 透视举例



# 一点透视投影的变换矩阵

- 1) 一点透视
- 设 $z$ 轴上有一观察点 (即视点)  $V(0,0,h)$
- 从 $V$ 点出发将物体上的点 $P(x,y,z)$ 投影到 $XOY$ 平面上得到 $P'$  ( $x',y',0$ )
- 由相似三角形可知:

$$\frac{x}{x'} = \frac{y}{y'} = \frac{h-z}{h}$$



# 一点透视投影的变换矩阵

□ 令：

$$\begin{aligned}x' &= \frac{x}{1 - \frac{z}{h}} \\y' &= \frac{y}{1 - \frac{z}{h}} \\z' &= 0\end{aligned}$$

$$H = 1 - \frac{z}{h} \quad X = x'H \quad Y = y'H \quad Z = z'H$$

# 一点透视投影的变换矩阵

□ 这是变换矩阵为

$$M_{rz} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{h} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

□ 的齐次坐标变换

$$\begin{pmatrix} X & Y & Z & 1 \end{pmatrix} = \begin{pmatrix} x & y & z & 1 \end{pmatrix} M_{rz}$$

□ 它可以看作是先作变换

$$M_r = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{h} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

→ 透视变换

# 一点透视投影的变换矩阵

## □ 再做变换

$$M_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow \text{向 } Z = 0 \text{ 平面的正投影变换}$$

## □ 的合成。

# 一点透视投影的变换矩阵

□ 在透视变换Mr下有：

$$\left\{ \begin{array}{l} x' = \frac{x}{1 - \frac{z}{h}} \\ y' = \frac{y}{1 - \frac{z}{h}} \\ z' = \frac{z}{1 - \frac{z}{h}} \end{array} \right.$$

# 一点透视投影的变换矩阵

- 当 $z \rightarrow \infty$ 时,  $x' \rightarrow 0, y' \rightarrow 0, z' \rightarrow -h$
- $\therefore (0,0,-h)$ 为该透视的一个灭点。
- 同样, 视点在 $(h,0,0)$ 的透视变换, 灭点在 $(-h,0,0)$
- 变换矩阵为

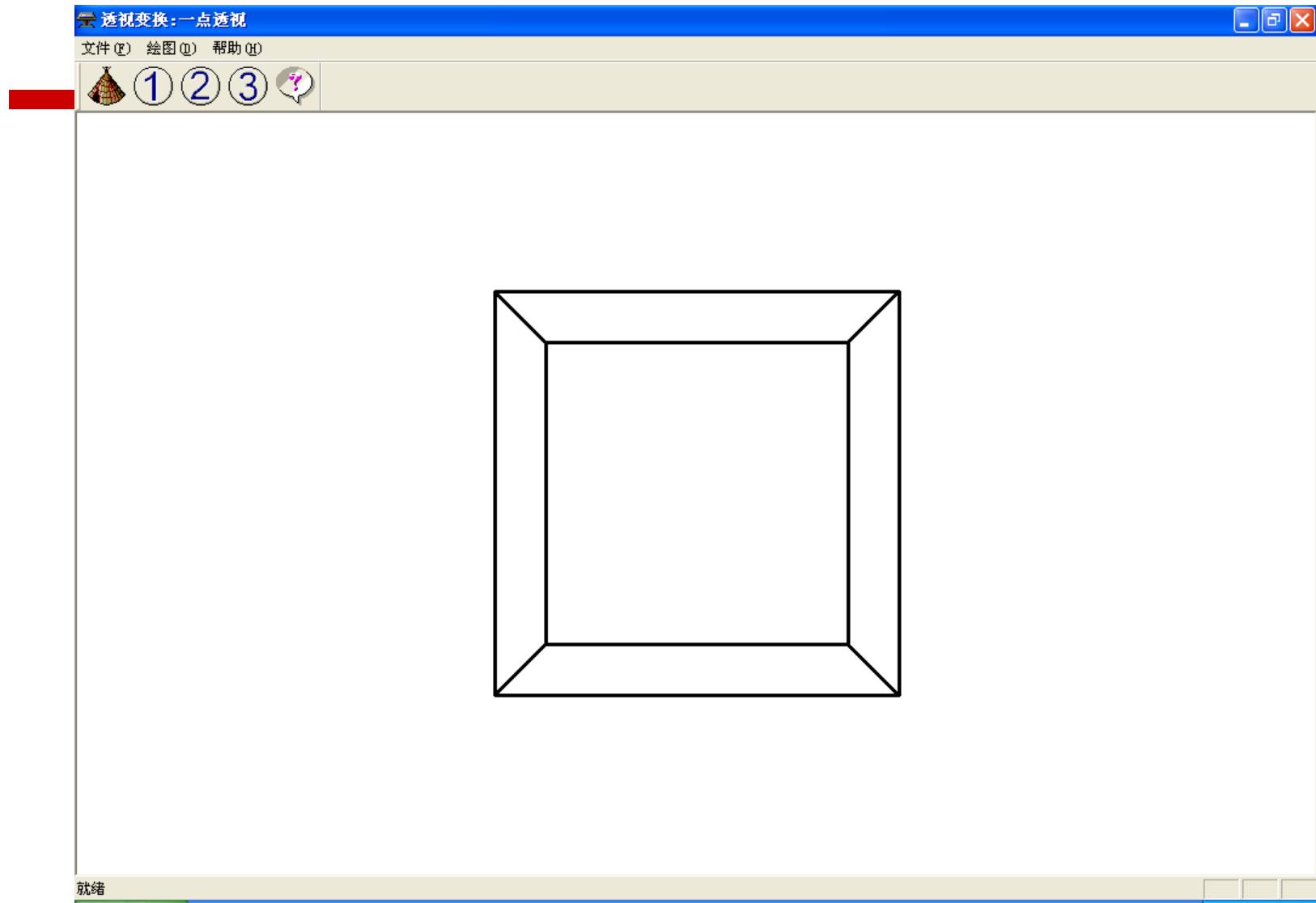
$$M_{rx} = \begin{pmatrix} 1 & 0 & 0 & -\frac{1}{h} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# 一点透视投影的变换矩阵

- 视点在(0,h,,0)的透视变换，灭点在(0,-h,0)
- 变换矩阵为

$$M_{ry} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\frac{1}{h} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$M_{rx}$ 、 $M_{ry}$ 、 $M_{rz}$ 均称为一点透视变换。



立方体的一点透视投影图

# 一点透视投影的变换矩阵

□ 在变换矩阵中，第四列的 $p, q, r$ 起透视变換作用

$$M = \begin{pmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# 一点透视投影的变换矩阵

当p、q、r中有一个不为0时的变换。假定  
 $q \neq 0, p=r=0.$

对空间上任一点 $(x, y, z)$ 进行透视变换结果如下。

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & qy + 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{x}{qy+1} & \frac{y}{qy+1} & \frac{z}{qy+1} & 1 \end{bmatrix}$$

---

◆ 当 $y=0$ 时：

$$x' = x$$

$$y' = 0$$

$$z' = z$$

即处于 $y=0$ 平面上的点， 经过透视变换后没有变化。

◆ 当 $y=\infty$ 时

$$x' = 0$$

$$y' = 1/q$$

$$z' = 0$$

即当 $y \rightarrow \infty$ 所有点的变换结果都集中到Y轴的 $1/q$ 处，  
也即所有平行于Y轴的直线， 变换后都将沿伸相交  
于该点。

---

# 二点透视投影的变换矩阵

- 2) 二点透视
- 在变换矩阵中，第四列的 $p, q, r$ 起透视线变换作用

$$M = \begin{pmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

当 $p, q, r$ 中有两个不为0时的透视线变换称为二点透视线变换。假定 $p \neq 0, r \neq 0, q=0$ ；

将空间上一点 $(x, y, z)$ 进行变换，可得如下结果：

# 二点透视投影的变换矩阵

$$[x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ z \ px + rz + 1]$$

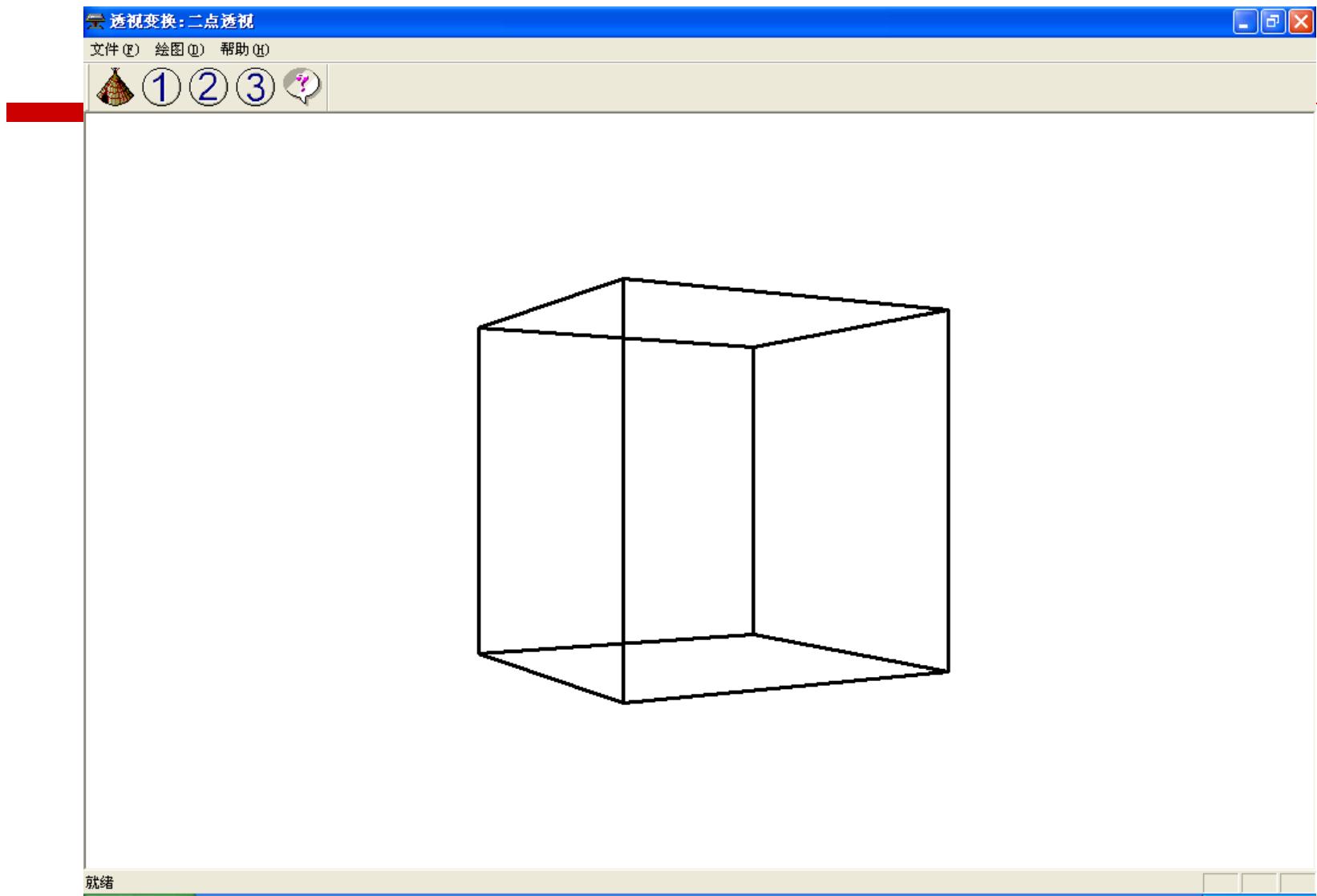
经齐次化处理后得：

$$\begin{cases} x' = x / (px + rz + 1) \\ y' = y / (px + rz + 1) \\ z' = z / (px + rz + 1) \end{cases}$$

由上式可看出：

当  $x \rightarrow \infty$  时，在 X 轴上  $1/p$  处有一个灭点；

当  $z \rightarrow \infty$  时，在 Z 轴上  $1/r$  处有一个灭点；



立方体的二点透视投影图

# 三点透视投影的变换矩阵

- 3) 三点透视
- 类似，若  $p, q, r$  都不为 0，则可得到有三个灭点的三点透视。

$$[\begin{matrix} x & y & z & 1 \end{matrix}] \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [\begin{matrix} x & y & z & px + qy + rz + 1 \end{matrix}]$$

经齐次化处理后得：

$$\begin{cases} x' = x / (px + qy + rz + 1) \\ y' = y / (px + qy + rz + 1) \\ z' = z / (px + qy + rz + 1) \end{cases}$$

# 三点透视投影的变换矩阵

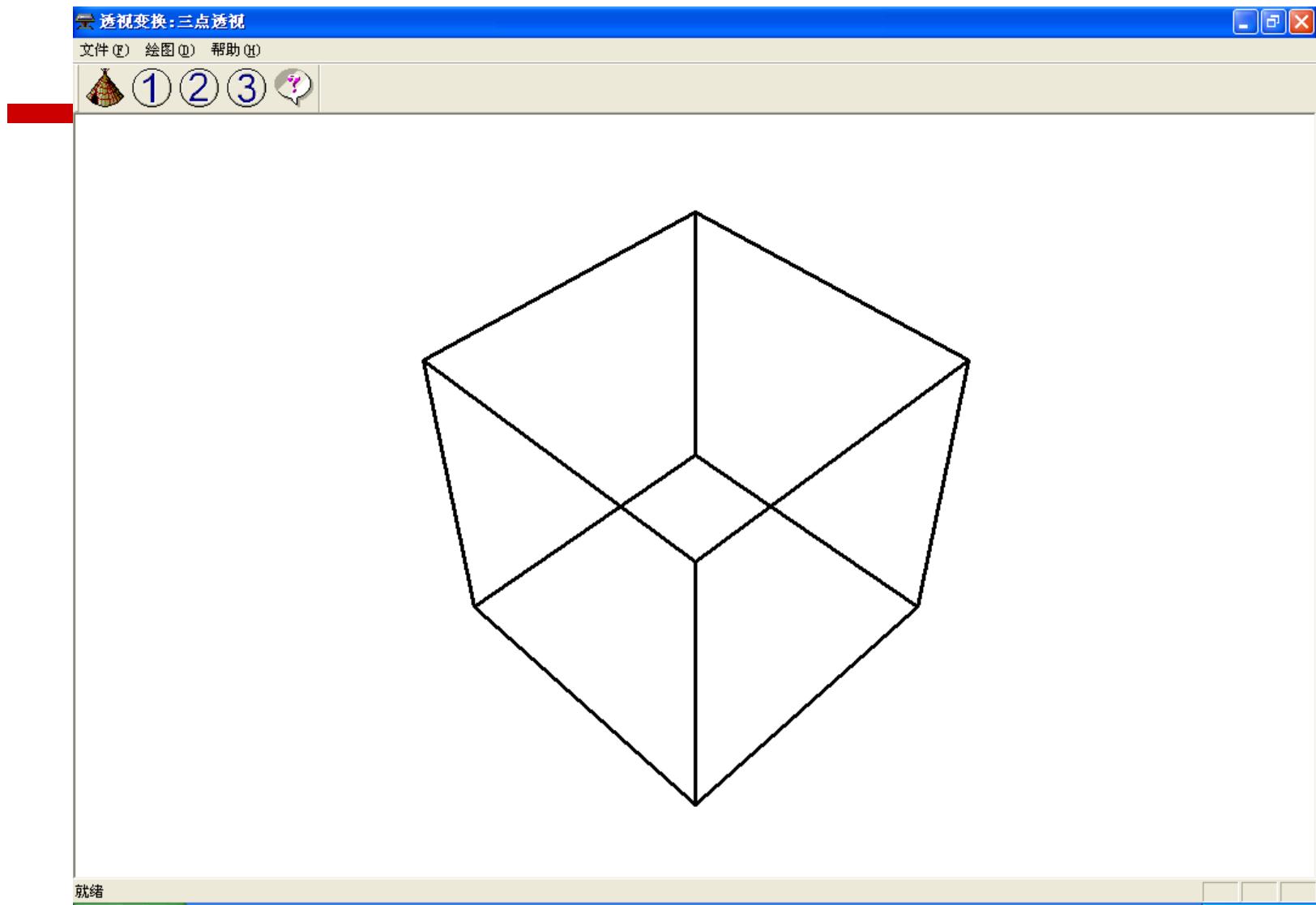
---

由上式可看出：

当 $x \rightarrow \infty$ 时，在X轴上 $1/p$ 处有一个灭点；

当 $y \rightarrow \infty$ 时，在Y轴上 $1/q$ 处有一个灭点；

当 $z \rightarrow \infty$ 时，在Z轴上 $1/r$ 处有一个灭点；



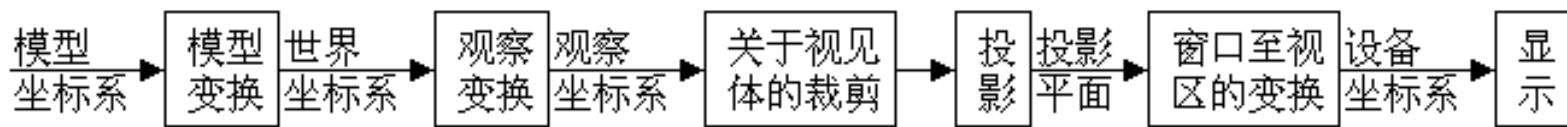
就绪

立方体的三点透视投影图

返回

# 三维图形的显示流程图

## □ 显示流程图



◆ 观察变换：从世界坐标系到观察坐标系的变换

# 三维图形的显示流程图

## □ 何时裁剪

### ◆ 投影之前裁剪----三维裁剪

➤ 优点

✓ 只对可见的物体进行投影变换

➤ 缺点

✓ 三维裁剪相对复杂

### ◆ 投影之后裁剪----二维裁剪

➤ 优点

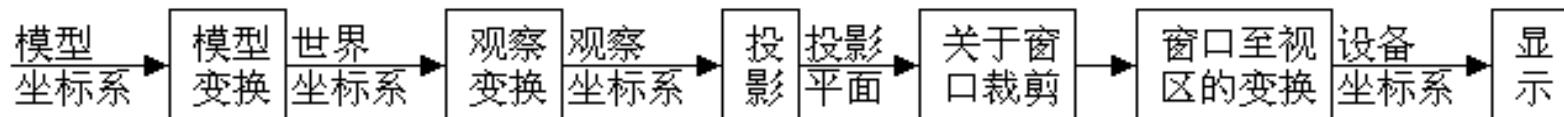
✓ 二维裁剪相对容易

➤ 缺点

✓ 需要对所有的物体进行投影变换

# 三维图形的显示流程图

## ◆ 采用二维裁剪的三维图形显示流程图

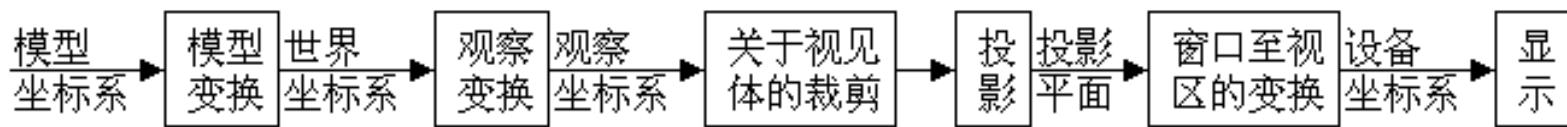


## ◆ 在投影之前裁剪的理由

- 三维物体的表面通常被离散表示成多边形或折线，而对这类简单图元，三维裁剪同样比较简单。
- 三维图形在显示过程中需要被消隐，做这个工作要有图形的深度信息，所以必须在投影之前完成。消隐很费时，如果在此之前裁剪（或部分裁剪）掉不可见的图形，可使需要消隐的图形减至最小。

# 三维图形的显示流程图

## □ 显示流程图



◆ 观察变换：从世界坐标系到观察坐标系的变换

# 三维图形的显示流程图

## □ 何时裁剪

### ◆ 投影之前裁剪----三维裁剪

➤ 优点

✓ 只对可见的物体进行投影变换

➤ 缺点

✓ 三维裁剪相对复杂

### ◆ 投影之后裁剪----二维裁剪

➤ 优点

✓ 二维裁剪相对容易

➤ 缺点

✓ 需要对所有的物体进行投影变换

# 三维图形的基本问题

---

## 1. 在二维屏幕上如何显示三维物体?

- ◆ 显示器屏幕、绘图纸等是二维的
- ◆ 显示对象是三维的
- ◆ 解决方法——投影
- ◆ 三维显示设备正在研制中

## 2. 如何表示三维物体?

- ◆ 二维形体的表示——直线段, 折线, 曲线段, 多边形区域
- ◆ 二维形体的输入——简单 (图形显示设备与形体的维数一致)

# 三维图形的基本问题

- ◆ 三维形体的表示——空间直线段、折线、曲线段、多边形、曲面片
- ◆ 三维形体的输入、运算、有效性保证——困难
- ◆ 解决方法——各种用于形体表示的理论、模型、方法

## 3. 如何反映遮挡关系？

- ◆ 物体之间或物体的不同部分之间存在相互遮挡关系
- ◆ 遮挡关系是空间位置关系的重要组成部分
- ◆ 解决方法——消除隐藏面与隐藏线

# 三维图形的基本问题

---

## □ 4. 如何产生真实感图形?

- ◆ 何谓真实感图形
  - 逼真的
  - 示意的
- ◆ 人们观察现实世界产生的真实感来源于
  - 空间位置关系——近大远小的透视关系和遮挡关系
  - 光线传播引起的物体表面颜色的自然分布
- ◆ 解决方法——建立光照明模型、开发真实感图形绘制方法

# 三维图形的基本问题

---

## 三维图形的基本研究内容

1. 投影
  2. 三维形体的表示
  3. 消除隐藏面与隐藏线
  4. 建立光照明模型、开发真实感图形绘制方法
-

# 本章小结

---

- 三维基本几何变换、变换矩阵
- 三维复合变换
- 投影变换
  - ◆ 平行投影:三视图及其变换矩阵
  - ◆ 透视投影变换矩阵

---

返 回

# 习题

1、长方体如图6-21所示，八个坐标分别为  $(0, 0, 0)$ ,  $(2, 0, 0)$ ,  $(2, 3, 0)$ ,  $(0, 3, 0)$ ,  $(0, 0, 2)$ ,  $(2, 0, 2)$ ,  $(2, 3, 2)$ ,  $(0, 3, 2)$ 。试对长方体进行  $S_x = 1/2$ ,  $S_y = 1/3$ ,  $S_z = 1/2$  的比例变换，求变换后的长方体各顶点坐标。

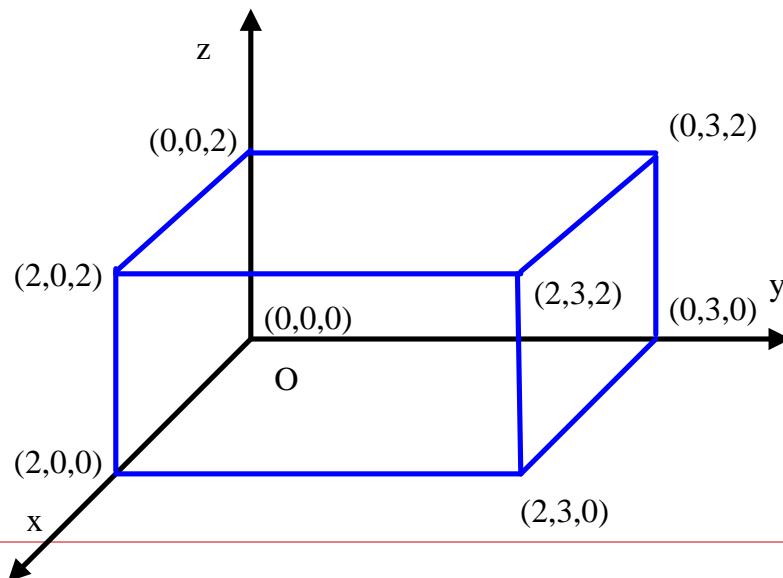


图6-21 长方体比例变换

2. 空间四面体的顶点坐标为  $A(2, 0, 0)$ ,  $B(2, 2, 0)$ ,  $C(0, 2, 0)$ ,  $D(2, 2, 2)$ , 如图6-22所示, 求解: (1) 关于点  $P(2, -2, 2)$  整体放大2倍的变换矩阵。 (2) 变换后的空间四面体顶点坐标。

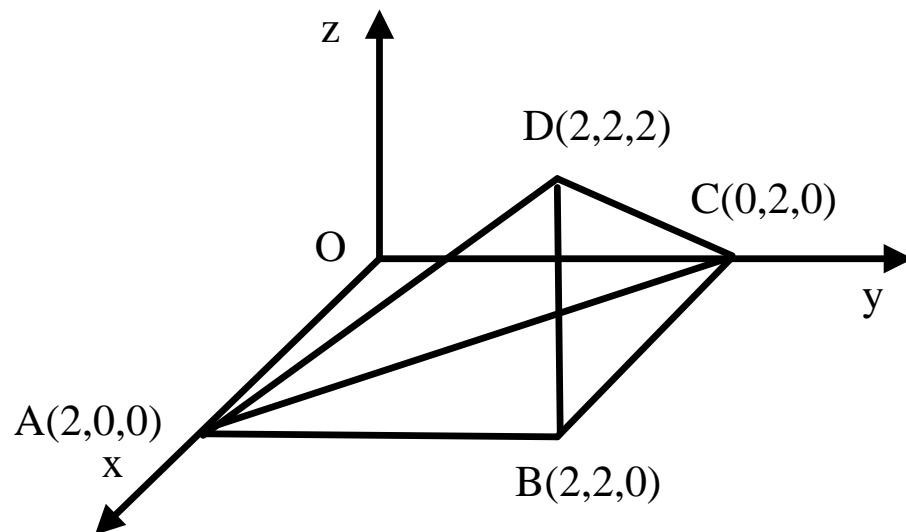


图6-22 四面体

## □ 实施以下坐标系之间变换：

(1) 将xOy坐标系原点O(0,0)平移到点O' (5,5)，变换矩阵为T<sub>1</sub>；

(2) 将坐标系逆时针旋转45°，变换矩阵为T<sub>2</sub>；

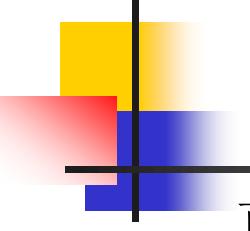
(3) 将x轴作反射变换，指向O点，变换矩阵为T<sub>3</sub>

□ 坐标系变换矩阵为  $T = T_1 \cdot T_2 \cdot T_3$ .

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -5 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

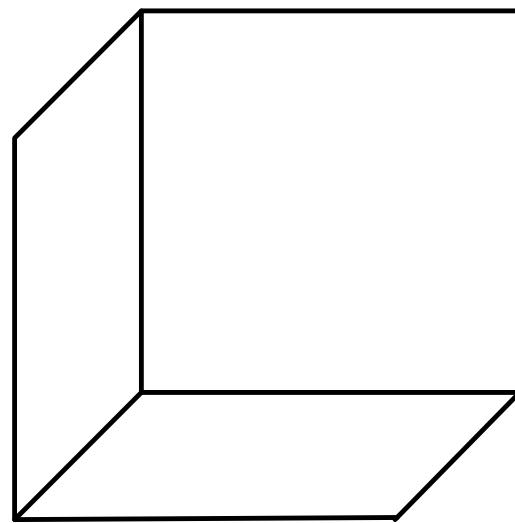
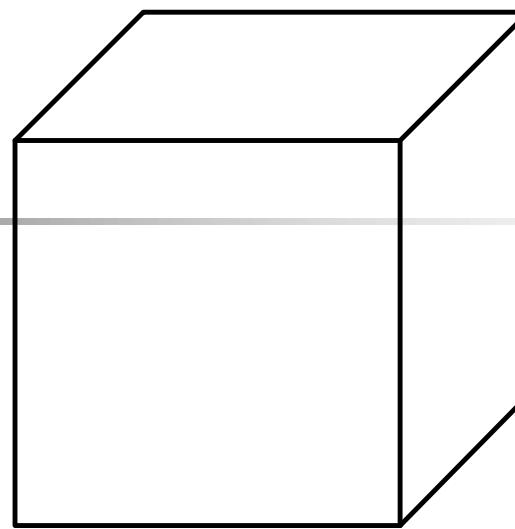
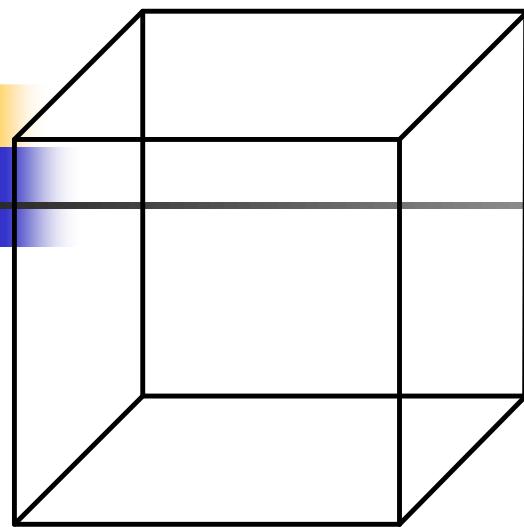
$$T_3 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



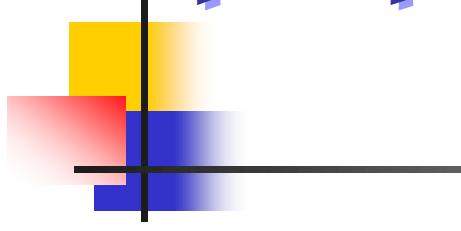
# 第七章 真实感图形生成技术

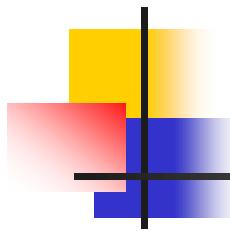
真实感图形生成技术涉及到的主要问题

- 隐藏线面的消除
- 明暗处理
- 阴影处理
- 纹理处理等



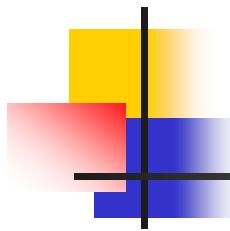
# 本章內容

- 
- 图形的数据结构
  - 消隐算法分类
  - 隐线算法
  - 隐面算法



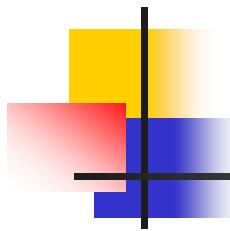
# 图形的数据结构

W.K.Giloi在其著作  
Interactive Computer Graphics中提出：  
Computer Graphics=Data Structure  
+Graphics  
Algorithms+Language



# 图形的数据结构

- 图形的几何信息和拓扑信息
- 基本图形的数据结构
- 立体表示模型



# 图形的几何信息和拓扑信息

- 几何信息：描述几何元素空间位置的信息。
- 拓扑信息：描述几何元素之间相互连接关系的信息。
- 描述一个物体不仅要有几何信息的描述而且还要有拓扑信息的描述。因为只有

五个顶点，几何信息已经确定，如果拓扑信息不同，则可产生不同图形

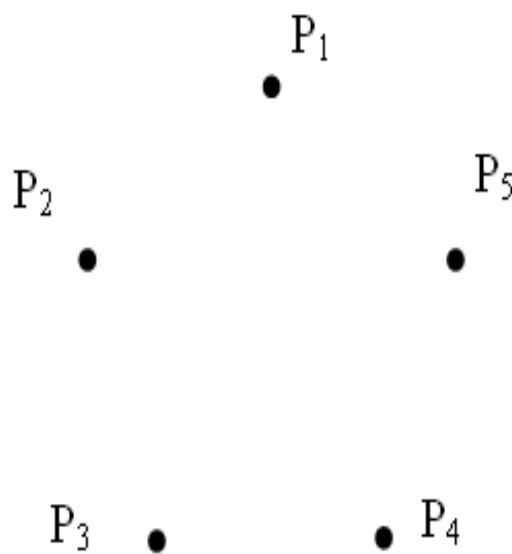
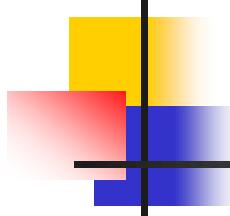
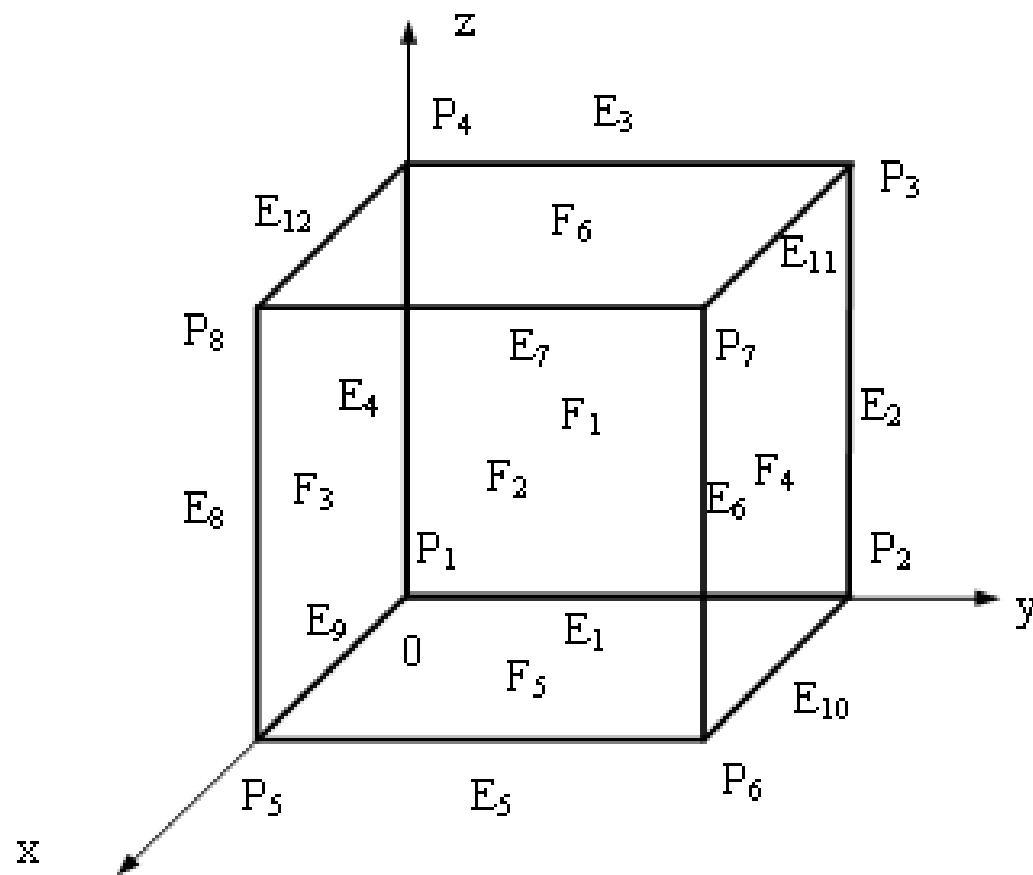


图 9-1 五个顶点



# 基本图形的数据结构

- 在三维坐标系下，描述一个物体不仅需要顶点表，而且还需要边表和面表，才能完全表达清楚



立体的数据结构

顶点	x坐标	y坐标	z坐标
P <sub>1</sub>	$x_1 = 0$	$y_1 = 0$	$z_1 = 0$
P <sub>2</sub>	$x_2 = 0$	$y_2 = a$	$z_2 = 0$
P <sub>3</sub>	$x_3 = 0$	$y_3 = a$	$z_3 = a$
P <sub>4</sub>	$x_4 = 0$	$y_4 = 0$	$z_4 = a$
P <sub>5</sub>	$x_5 = a$	$y_5 = 0$	$z_5 = 0$
P <sub>6</sub>	$x_6 = a$	$y_6 = a$	$z_6 = 0$
P <sub>7</sub>	$x_7 = a$	$y_7 = a$	$z_7 = a$
P <sub>8</sub>	$x_8 = a$	$y_8 = 0$	$z_8 = a$

表1 立方体顶点表

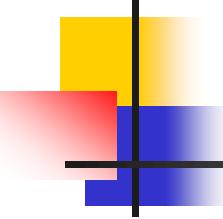
边	起点	终点
$E_1$	$P_1$	$P_2$
$E_2$	$P_2$	$P_3$
$E_3$	$P_3$	$P_4$
$E_4$	$P_4$	$P_1$
$E_5$	$P_5$	$P_6$
$E_6$	$P_6$	$P_7$
$E_7$	$P_7$	$P_8$
$E_8$	$P_8$	$P_5$
$E_9$	$P_1$	$P_5$
$E_{10}$	$P_2$	$P_6$
$E_{11}$	$P_3$	$P_7$
$E_{12}$	$P_4$	$P_8$

表2 立方体边表

面	边1	边2	边3	边4
$F_1$	$E_1$	$E_4$	$E_3$	$E_2$
$F_2$	$E_5$	$E_6$	$E_7$	$E_8$
$F_3$	$E_9$	$E_8$	$E_{12}$	$E_4$
$F_4$	$E_{10}$	$E_2$	$E_{11}$	$E_6$
$F_5$	$E_1$	$E_{10}$	$E_5$	$E_9$
$F_6$	$E_3$	$E_{12}$	$E_7$	$E_{11}$

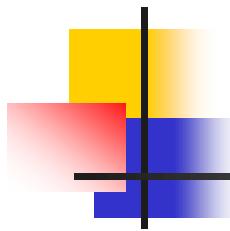
表3 立方体面表

返 回



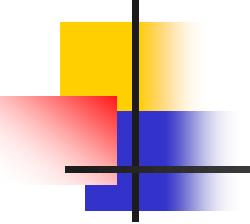
# 立体表示模型

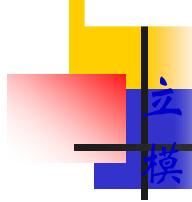
- 计算机三维模型的描述经历了
    - 线框模型
    - 表面模型
    - 实体模型
- 的发展，所表达的几何体信息越来越完整



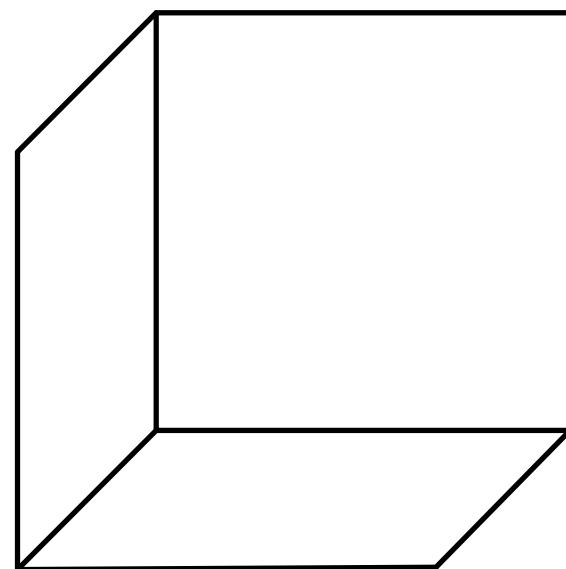
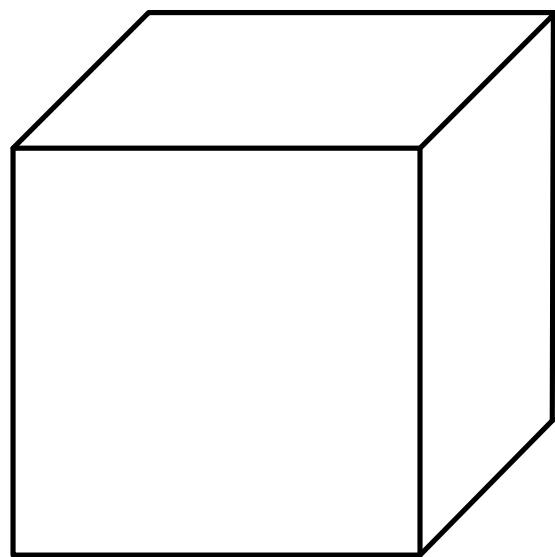
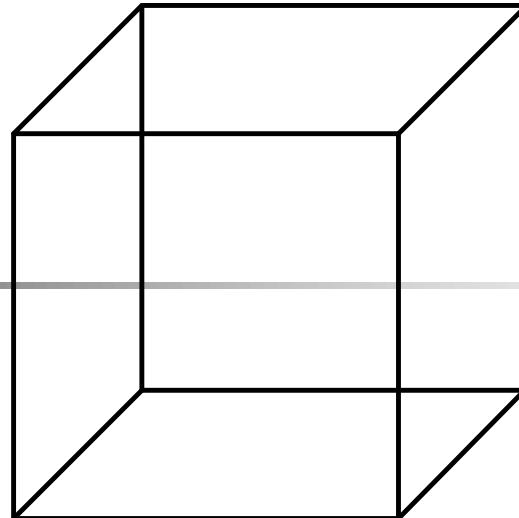
# 线框模型 (Wireframe Model)

- 线框模型只是用几何体的边线来表示立体的外形，就如同用边线搭出的框架一样，线框模型中没有表面、体积等信息。
- 线框模型只使用顶点表和边表两个数据结构描述面点之间的拓扑关系

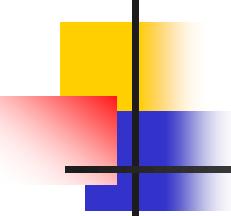
- 
- 线框模型的**优点**: 可以产生任意方向视图, 视图间能保持正确的投影关系
  - 线框模型的**缺点**:
    - 因为所有棱边全部绘制出来, 理解方面容易产生二义性
    - 线框模型不能进行两个面的求交运算



## 立体线框 模型表示



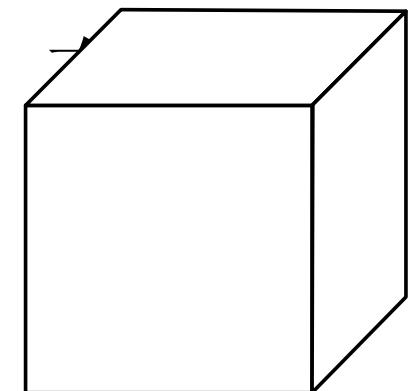
线框模型表示的二义性



# 表面模型 (Surface Model)

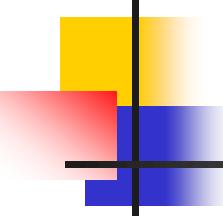
- 表面模型是利用立体的外表面来构造模型
- 就如同在线框模型上蒙上了一层外皮，使立体具有了一定的轮廓，可以进行消隐处理
- 与线框模型相比，表面模型增加了一个

- 表面模型的优点：可以进行面着色，隐藏面消隐，以及表面积计算，格划分等。



- 缺点：
  - 表面模型仍缺乏体积的概念，是一个立体的空壳

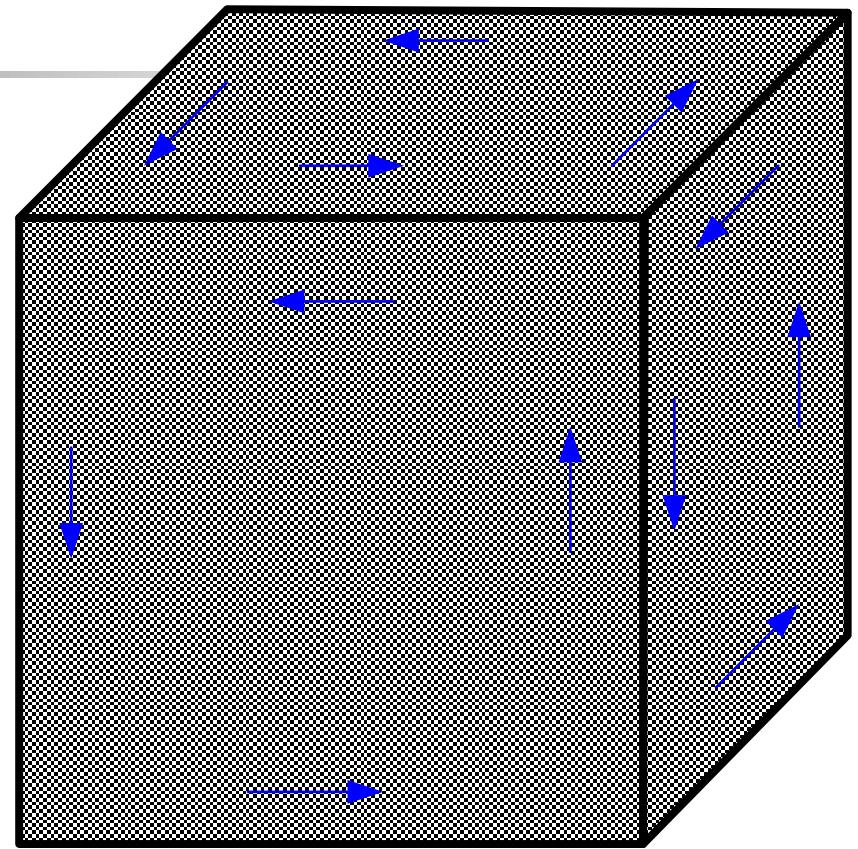
无法进行立体之间的相交运算



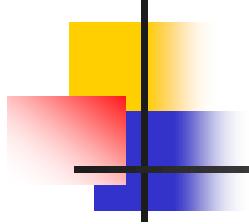
# 实体模型 (Solid Model)

- 实体模型，如同在立体表面模型的内部进行了填充，使之具有了如体积和重量等特性，更能反映立体的真实性，这时的立体才具有“体”的概念。
- 实体模型与表面模型的不同之处在于确定了表面的哪一侧存在实体
- 在表面模型的基础上采用有向棱边隐含地表示表面的外法矢方向，使用右手法则取向：四个手指沿闭合的棱边方向，

拓扑合法的立体  
在相邻两个面的  
公共边界上，棱边的  
方向正好相反



立方体实体模型表示

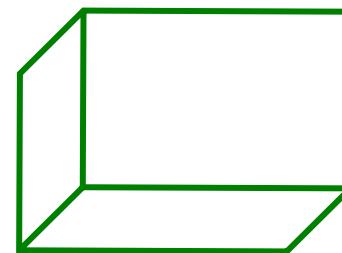
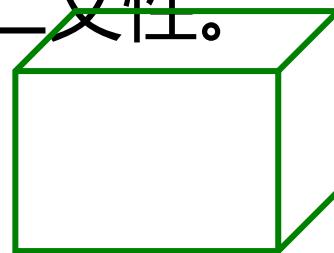
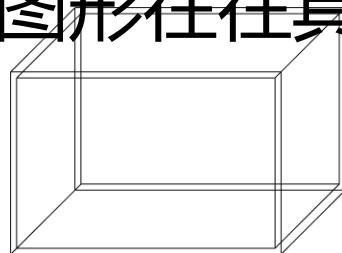
- 
- 实体模型只需将面表改成环表形式，就可确切地分清体内体外
  - 实体模型和线框模型、表面模型的根本区别在于其数据结构不仅记录了全部几何信息，而且记录了全部点、线、面、体的拓扑信息

# 消隐

## 消隐

- 什么叫做“消隐”？为什么要进行“消隐”？

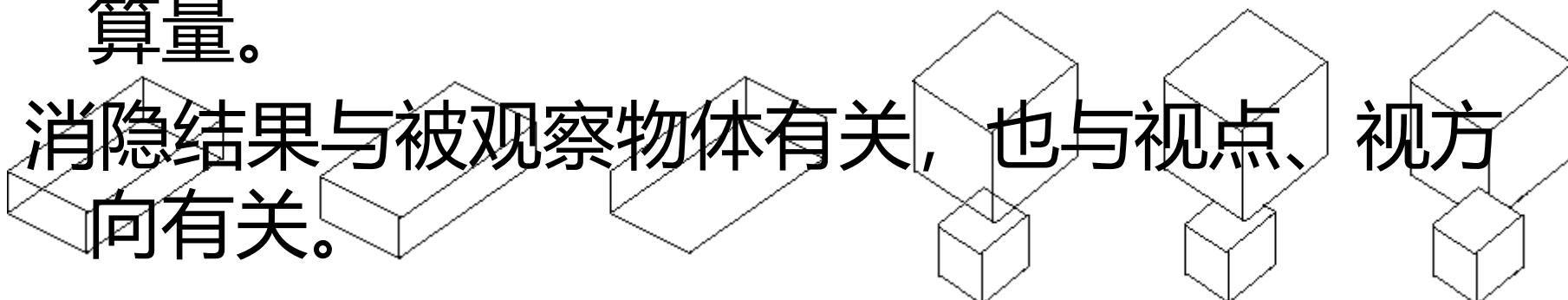
将三维场景绘制在计算机二维显示屏上必须经过投影变换，投影变换将三维信息变换到二维平面上，这个过程中**深度信息被丢失**，生成的图形往往具有二义性。



要消除二义性，就必须在绘制时消除被遮挡的不可见的线或面，习惯上称作消除隐藏线和隐藏面，简称为“消隐”。

- 经过消隐得到的投影图称为物体的真实图形。
- 通过消隐，也可降低光照、纹理等算法的计算量。

消隐结果与被观察物体有关，也与视点、视方  
向有关。



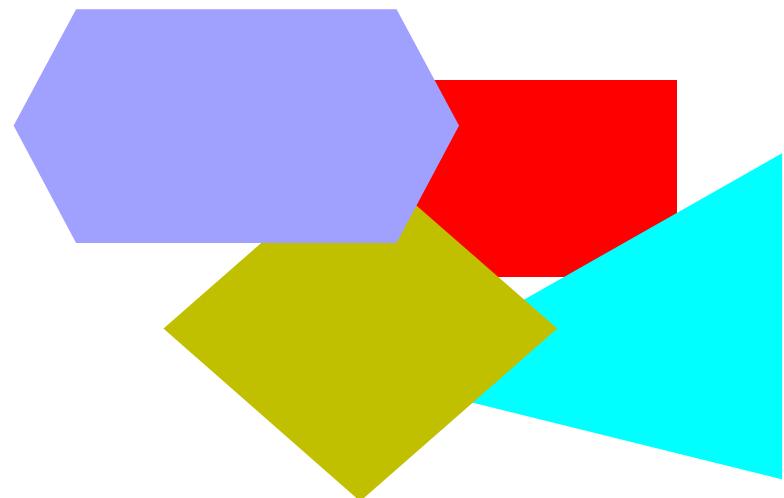
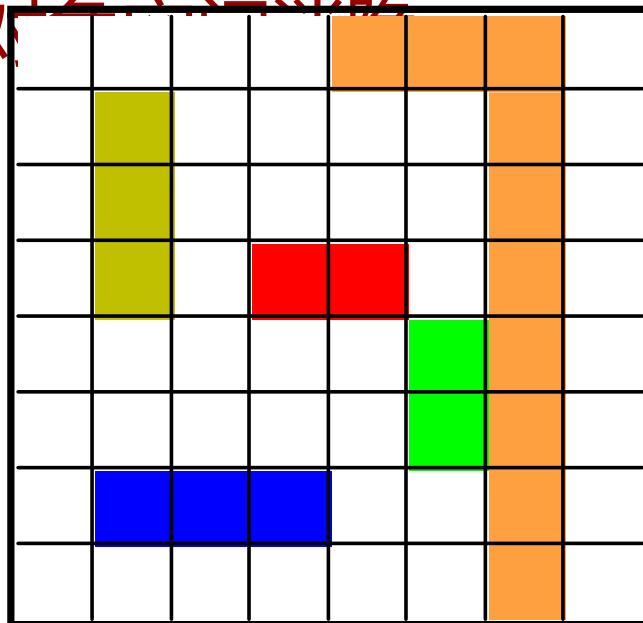
# 消隐

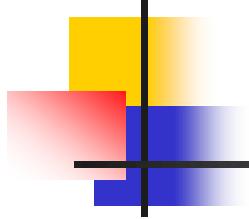
## 消隐的分类

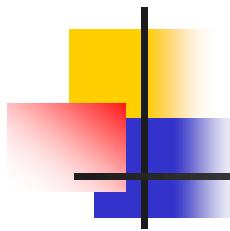
- 根据消隐算法实现时所在的坐标系(空间)进行分类：

- 图像空间消隐

- 对象空间消隐

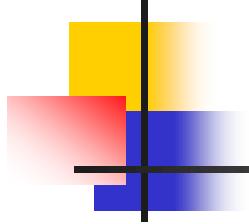


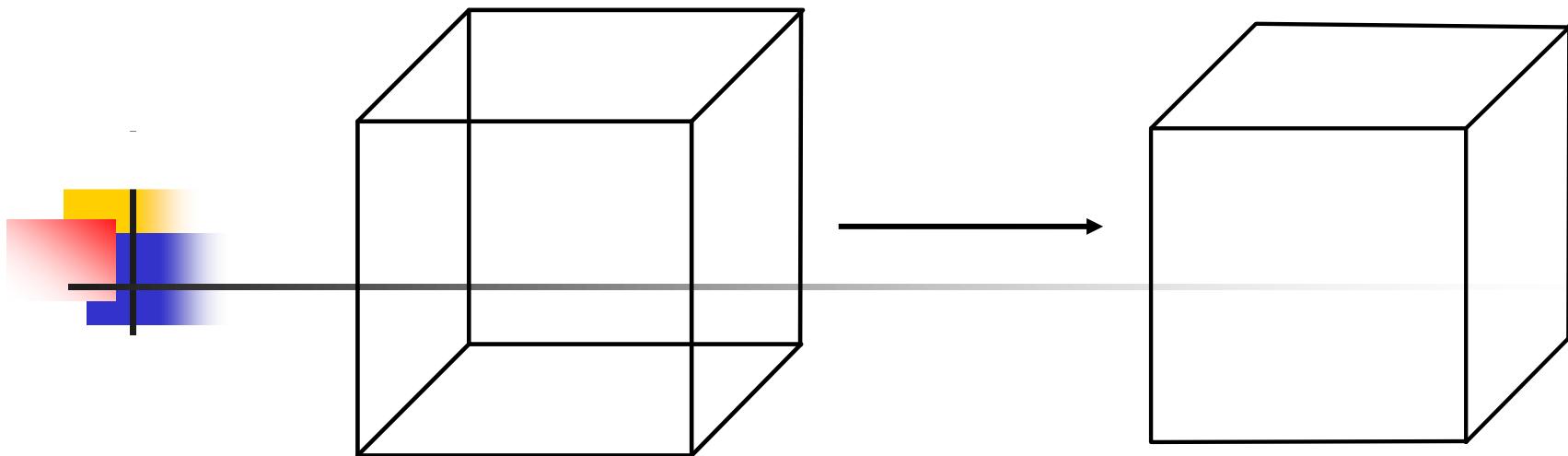
- 
- 消隐算法
  - 背面剔除算法
  - 画家算法
  - BSP树算法
  - 深度缓冲器算法
  - Warnock 算法



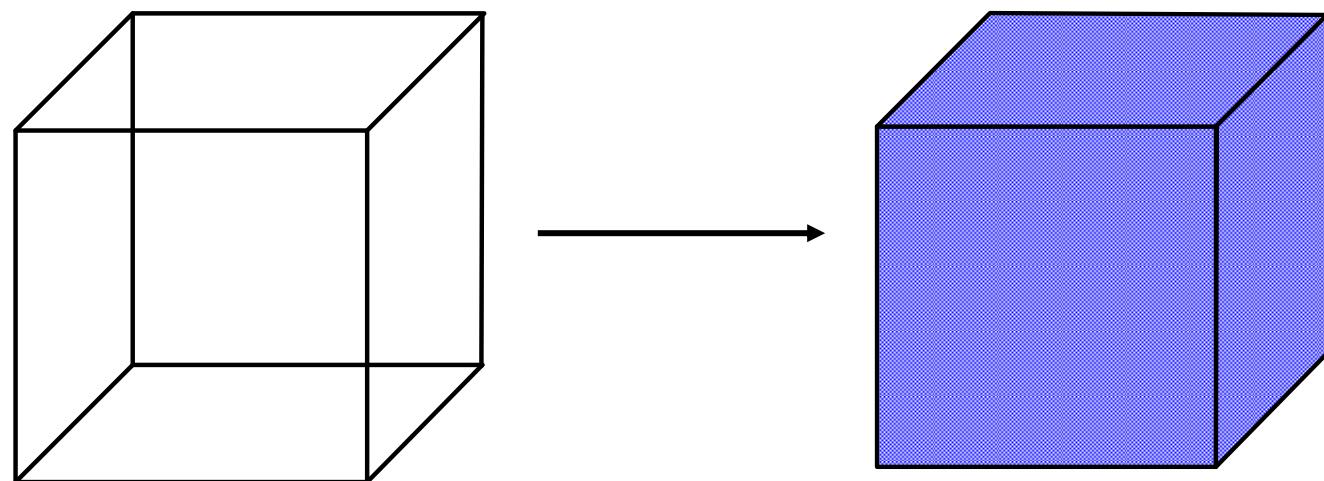
# 消隐算法分类

- 无论多么复杂的空间物体，沿视线方向上都只能看到一部分表面，其余的表面背向观察者不可见
- 计算机图形学的一个重要任务就是对空间物体的表面进行可见性检测，绘制出可见边线和表面

- 
- 根据消隐方法的不同，消隐算法可分为两类：
    - 隐线算法
      - 用于消除物体上不可见的边界线
      - 主要针对线框模型，要求画出物体的各可见棱边
    - 隐面算法
      - 用于消除物体上不可见的表面
      - 主要针对表面模型，不仅要求画出物体的各个可见棱边，而且还要求填充各个表面



隐线算法

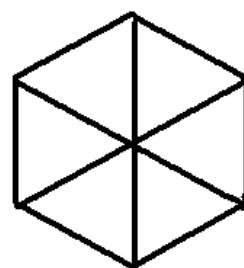


隐面算法

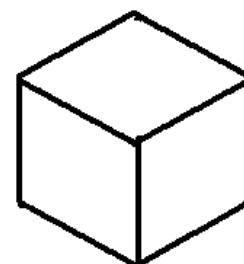
## 7.1 隐藏线面的消除

在用计算机生成三维图形时，形体的所有部分都将被表示，不管是可见的还是不可见的，这样的图形显示出来形状是不清楚的，甚至是不确定的。图1（a）所示是一个通过棱边表示的立方体的图形，如果不消隐不易辨别。通过适当删除不可见的隐藏部分，可以得到表示明确的图形。图1（b）是观察点在立方体的前上方的消隐图，图1（c）是观察点位于立方体的前下方消隐图，消隐图表示了明确的立体感形体。

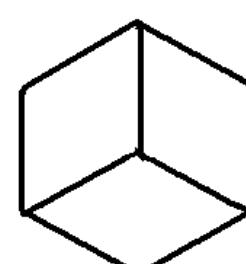
图1



(a)



(b)



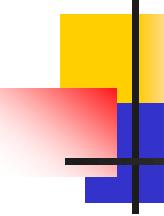
(c)



观察点确定后，找出并消除图形中不可见的部分，称为消隐。经过消隐得到的图形称为消隐图。

消除隐藏线和隐藏面是计算机图形学中一个较为困难的问题，消隐算法是决定相对于空间给定位置的观察者，哪些棱边、表面或物体是可见的，哪些是不可见的。消隐不仅与消隐对象有关，还与观察点、观察方向、投影面等的设置方位有关。改变这些设置，物体上某些可见的部分将会变成不可见，某些不可见的部分又会变成可见。

虽然各种消隐算法的基本思想有所不同，但它们大多采用了排序和相关性以提高效率。排序的主要目的是区分体、面、边、点与观察点间几何距离的远近。因为一个物体离观察点愈远，它愈有可能被另一距观察点较近的物体部分地或全部遮挡。消隐算法的效率在很大程度上取决于排序的效率。通常利用画面在局部区域内的相关性来提高排序过程的效率。



消隐算法一般可以分为两类。如果算法是在物体所定义的空间实现，那么这种算法称为**对象空间算法**；如果算法是在物体投影后的屏幕坐标空间实现，那么这种算法就称为**图象空间算法**。一般说来，对象空间算法有比较高的精度，而图象空间算法在精度上受屏幕分辨率的限制，但可以方便地利用图象空间中各种相关性获得较高的计算效率。

### 7.1.1 凸多面体的消隐算法

在消隐问题中，凸多面体是最简单情形。凸多面体是由多个凸多边形平面包围而成的立体，连接形体上不属于同一表面的任意两点的线段完全位于形体的内部。对于单个凸多面体，背向观察点的面是不可见面，如图2所示。因此，只要判断出这些“朝后面”，即可达到隐藏面消除的目的。

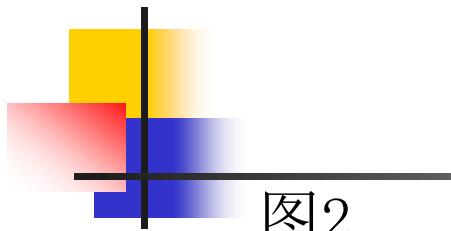
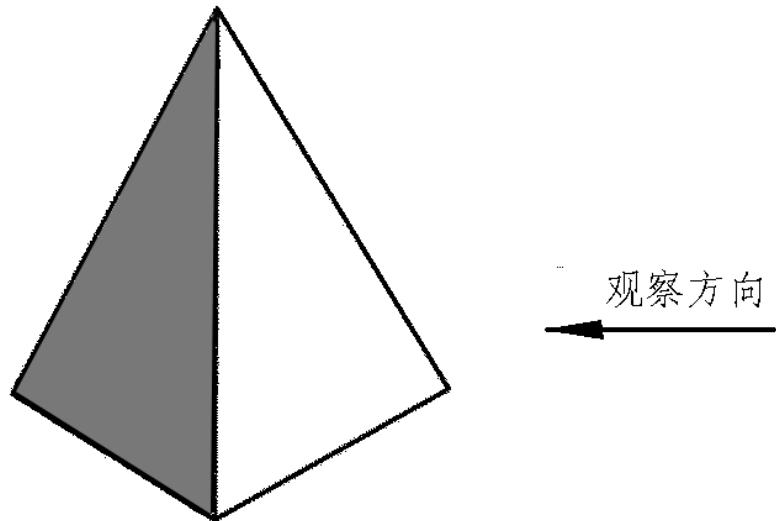


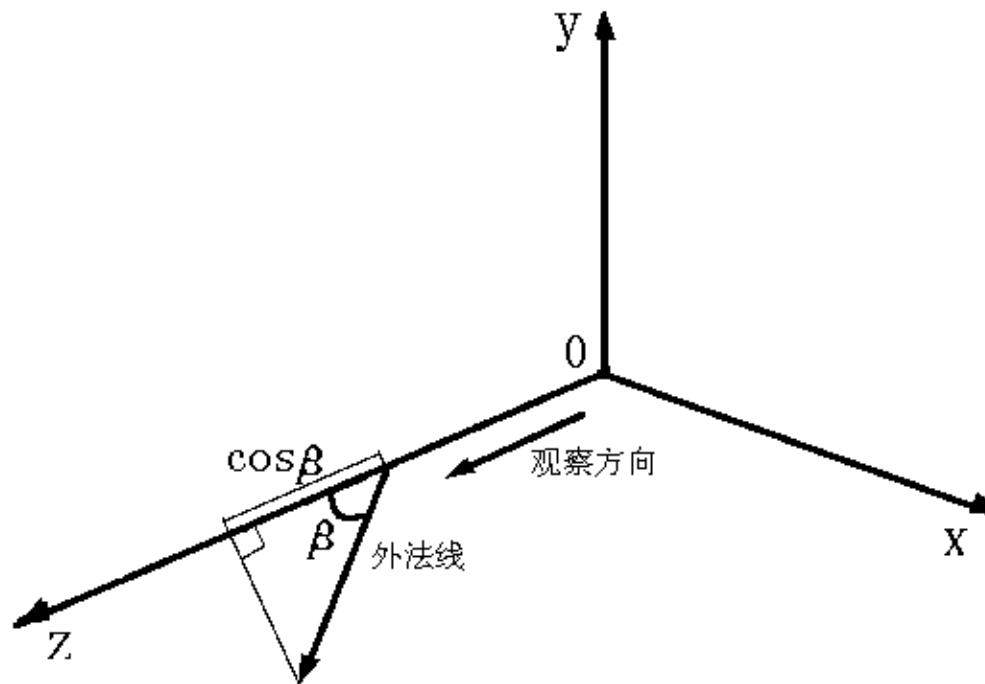
图2



构成多面体的每个平面都有其法线。通常规定法线的方向是由多面体的内部指向多面体的外部，称为“外法线”。

假定在右手坐标系中，观察点位于原点，投影面平行于XY坐标平面，以Z轴作为深度坐标轴，视线平行于Z轴，如图3所示，则平面外法线同Z轴方向的夹角，就是外法线同视线的夹角。很显然，对于单个凸多面体，当外法线同视线的夹角小于90°时，其平面背向观察点为不可见面。

图3



设平面外法线同Z轴方向的夹角为 $\beta$ ，则 $\cos \beta$  为单位平面外法线矢量在Z轴上的分量。 $\beta$  角同可见性的关系为：

(1) 当  $0^\circ \leqslant \beta \leqslant 90^\circ$  时， $\cos \beta > 0$ ，此面背向观察者为不可见面。

(2) 当  $= 90^\circ$  时， $\cos \beta = 0$ ，此面平行于Z轴，可以认为是不可见面。

(3) 当  $90^\circ \leqslant \beta \leqslant 180^\circ$  时， $\cos \beta < 0$ ，此面朝向观察者的，为可见面。

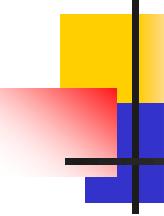
设平面方程为

$$Ax + By + Cz + D = 0$$

法向矢量为

$$\mathbf{N} = A\mathbf{i} + B\mathbf{j} + C\mathbf{k}$$

则 $\cos \beta = C/|\mathbf{N}|$ 。作为判断依据，只需要知道 $\cos \beta$  的正负号就够了。因为 $|\mathbf{N}|$  恒大于 0，所以 $\cos \beta$  的符号由C决定，因此，当 $C < 0$  时，为可见面。当 $C \geqslant 0$  时，为不可见面。



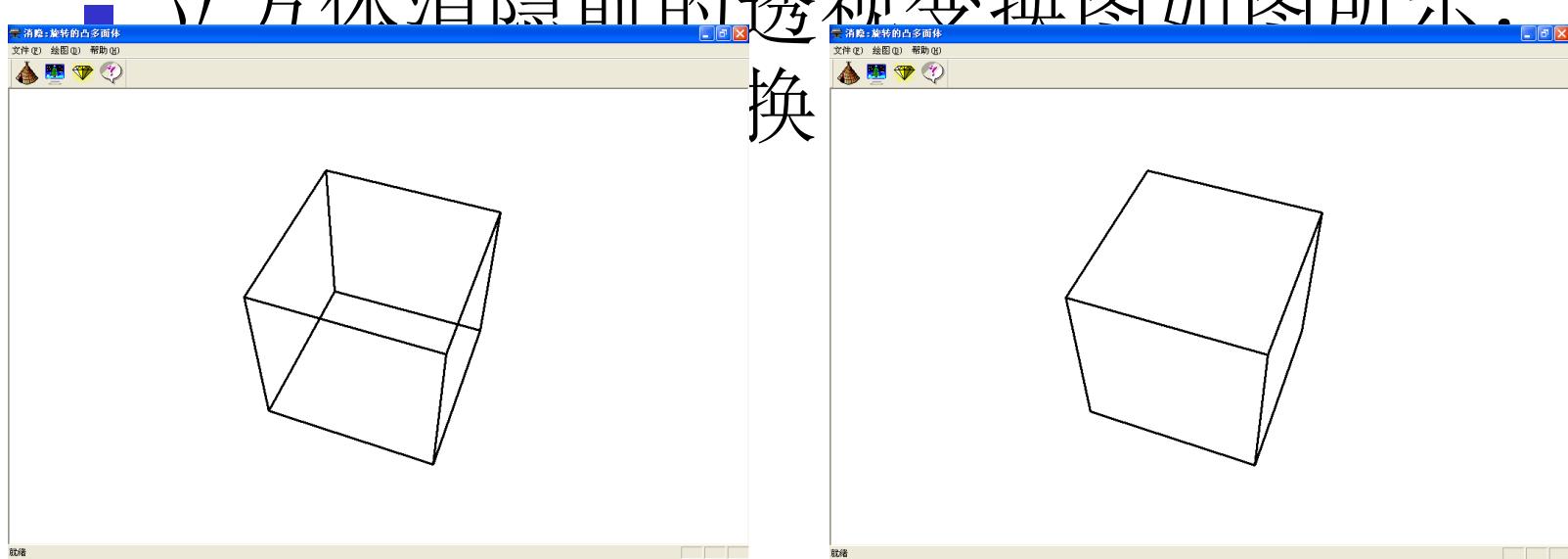
由于三点可以构成一个平面，和三点可以构成两个矢量，由两矢量的叉积可以求出平面的法线。对于凸多面体，任取构成平面多边形的三个相邻点  $P_0(x_0, y_0, z_0), P_1(x_1, y_1, z_1), P_2(x_2, y_2, z_2)$ ，按右手规则确定点的顺序，此时有：

$$C = \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_1 & y_2 - y_1 \end{vmatrix} = (x_1 - x_0)(y_2 - y_1) - (x_2 - x_1)(y_1 - y_0)$$

为了决定一个凸多面体的不可见面，对于每一个面按上述公式进行计算，当  $C \geq 0$  时为不可见面。

对于单个凸多面体，该方法可判别出所有隐藏面，因为每个面或是完全可见，或是完全不可见。对于其它形体，如凹多面体或由多个物体组成的复杂形体，则还需进行更多的测试来检查是否存在被其它面或其它物体完全或部分遮挡的表面。通常，凸多面体消隐处理可消除一半左右的隐藏面。

立方体消隐前的透視變換圖如圖所示。



消隱前的立方体

图 消隱后的立方体

返  
回

## 7.1.2 画家算法

- 由来：画家的作画顺序暗示出所画物体之间的相互遮挡关系
- 算法基本思想：
  - 1) 先把屏幕置成背景色
  - 2) 先将场景中的物体按其距观察点的远近进行排序，结果放在一张线性表中；（线性表构造：距观察点远的优先级低，放在表头；距观察点近的优先级高，放在表尾。该表称为深度优先级表）
  - 3) 然后按照从远到近（从表头到表尾）的顺序逐个绘制物体。
- 关键：如何对场景中的物体按深度（远近）排序，建立深度优先级表？

深度优先级表的建立是动态进行的。假定观察方向同Z轴同向，则最初可按各面的最小z值排序。但这一初步排序可能出现差错，如图4所示的情况。图中尽管面 $S_1$ 的最小z值小于面 $S_2$ 的最小z值，但正确的顺序是面 $S_2$ 位于面 $S_1$ 前。因此在实际将z值最大的面S写入帧缓冲器之前，需与其它面比较以确定是否在Z方向存在重叠。若无重叠，则对S进行写入，若存在重叠，则需作一些比较以决定是否有必要重新排序。如果存在两面相交和循环遮挡，如图5中所示的情况，这时简单的排序是无法解决问题的，必须求交分割后再进行排序。

图4

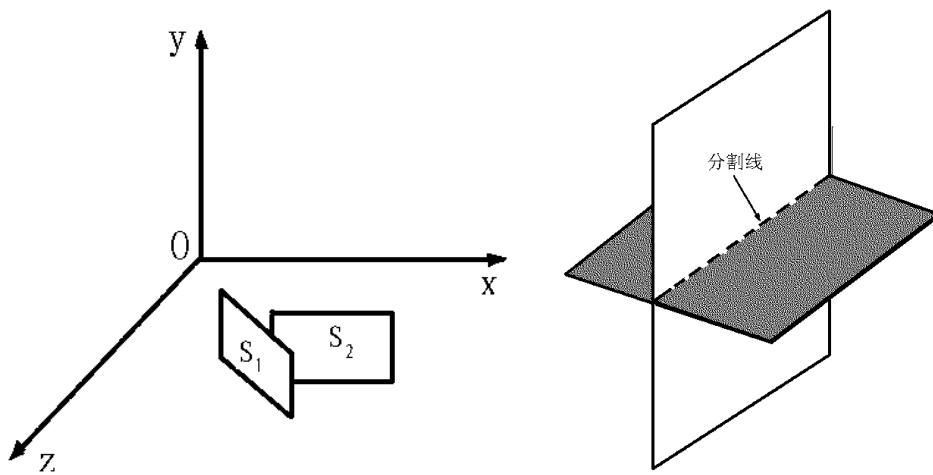
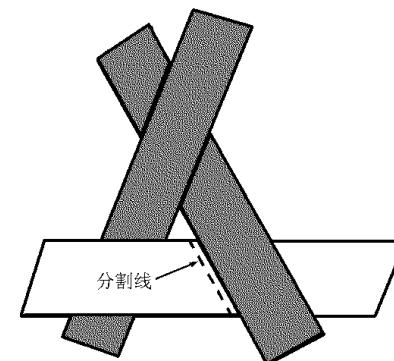
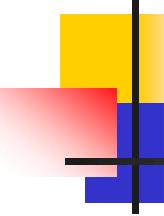


图5





下面给出这种算法过程的简单描述

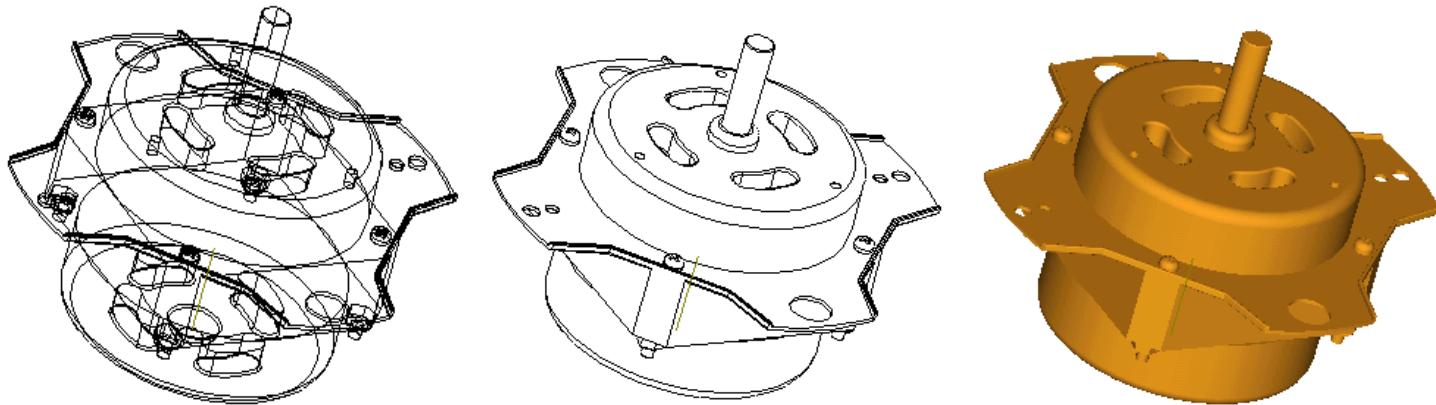
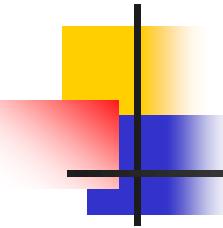
(1) 计算各面最小z值 $z_{min}$ , 并以此值的优先级进行排序, 建立初步的深度优先表;

(2) 表空结束。否则取表中深度最大的面 $S_n$ , 检查表中其它各面 $S_k$  ( $k=0, 1, \dots, n-1$ ) 与 $S_n$ 是否在Z方向存在重叠的关系。存在, 记重叠面为 $S_j$ 转(3)。否则, 将 $S_n$ 写入帧缓冲器,  $n=n-1$ 转(2);

(3) 检查 $S_n$ 是否遮挡 $S_j$ , 不遮挡则将 $S_n$ 写入帧缓存器,  $n=n-1$ 转(2)。否则, 交换 $S_n$ 和 $S_j$ 在表中位置, 转(2)。如果 $S_n$ 和 $S_j$ 已经交换过位置, 则两面交叉遮挡, 转(4);

(4) 用 $S_n$ 和 $S_j$ 的交线分割 $S_n$ 为两部分, 转(1)。

画家算法的优点是简单, 容易实现, 缺点是算法中的深度排序计算量大。



## 7.2 简单光照模型

### -计算某一点光强度的模型

当光照射到一个不透明的物体表面时，部分被反射，部分被吸收并转化为热。对一个透明的表面，部分入射光被反射，而另一部分被透射。其中，反射或透射部分的光使物体可见。如果入射光全部被吸收，物体将不可见，该物体称为黑体。一物体表面呈现的颜色是由物体表面向视线方向辐射的光能中各种波长的分布所决定的。光能中被吸收、反射或透射的数量决定于光的波长和物体的表面特性。由于光照射到物体表面产生的现象是很复杂的，它与光源的性质、形状、数量、位置有关，还与物体的几何形状、光学性质、表面纹理等许多因素有关，甚至与人眼对光的生理与心理视觉因素有关，我们不可能把这一切都准确计算出来。我们将讨论计算光强度的一些较为简单的方法，这些经验模型为计算物体表面某点处的光强度提供了简单有效的途径，并能在许多应用场合获得较好的效果。

# 简单光照明模型

模拟物体表面的光照明物理现象的数学模型—光照明模型

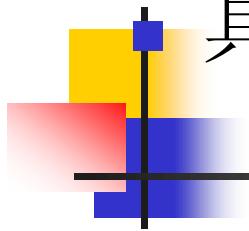
简单光照明模型亦称局部光照明模型，其假定物体是不透明的，只考虑光源的直接照射，而将光在物体之间的传播效果笼统地模拟为环境光。

可以处理物体之间光照的相互作用的模型称为整体光照明模型

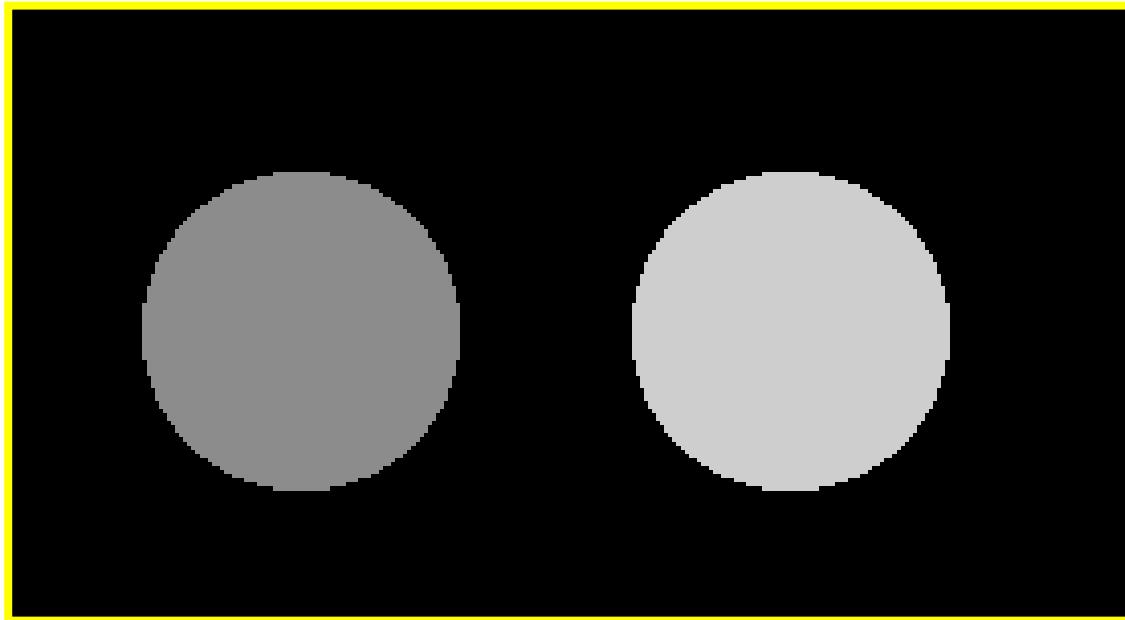
## 7.2.1 简单光照明模型-环境光

假定物体是不透明的（即无透射光）

- **环境光：** 在空间中近似均匀分布，即在任何位置、任何方向上强度一样,记为  $I_a$
- **环境光反射系数  $K_a$ ：** 在分布均匀的环境光照射下，不同物体表面所呈现的亮度未必相同，因为它们的环境光反射系数不同。
- 光照明方程（仅含环境光）：  $I_e = K_a I_a$   
 $I_e$ 为物体表面所呈现的亮度。



# 具有不同环境光反射系数的两个球



$$K_a = 0.4$$

$$K_a = 0.8$$

# 简单光照明模型-环境光

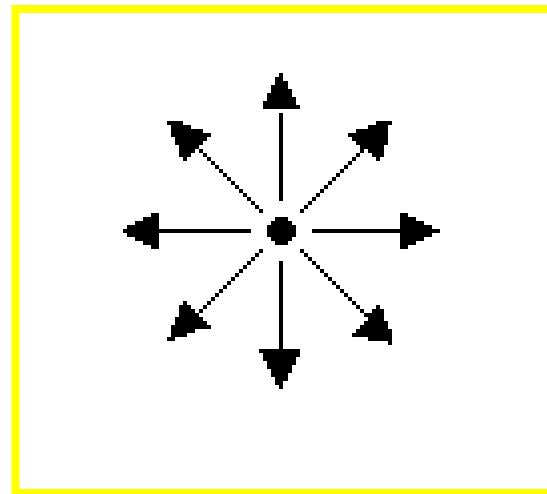


缺点：虽然不同的物体具有不同的亮度，但是同一物体的表面的亮度是一个恒定的值，没有明暗的自然过度。

# 简单光照明模型

考虑引入点光源。

点光源：几何形状为一个点，位于空间中的某个位置，向周围所有的方向上辐射等强度的光。记其亮度为 $I_p$



■ 点光源的照射：在物体的不同部分其亮度也不同，亮度的大小依赖于物体的朝向及它与点光源之间的距离。

## 7.2.2 简单光照明模型：-漫反射角度余弦的推导

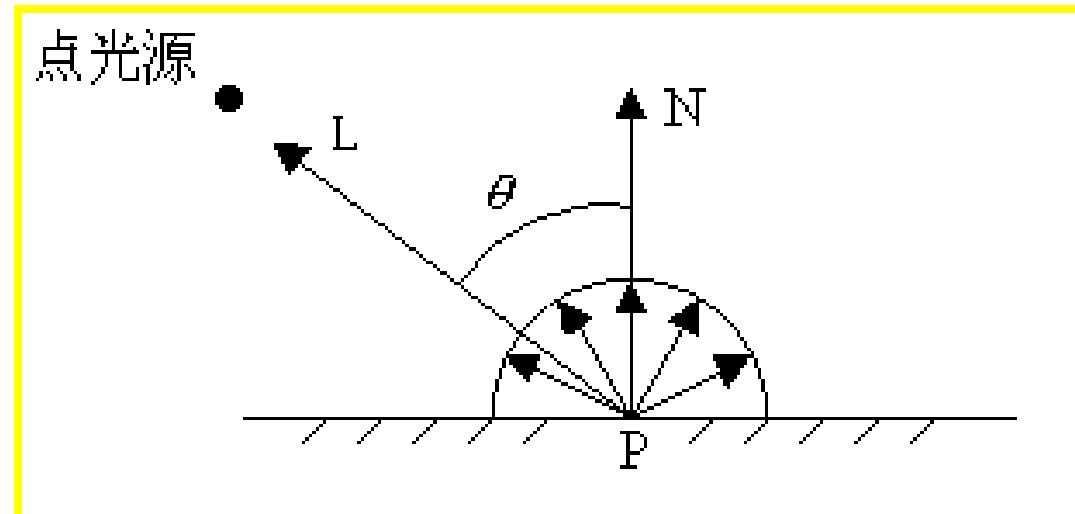
### 漫反射

- 粗糙、无光泽物体（如粉笔）表面对光的反射
- 光照明方程

$$I_d = I_p K_d \cos\theta \quad \theta \in [0, \frac{\pi}{2}]$$

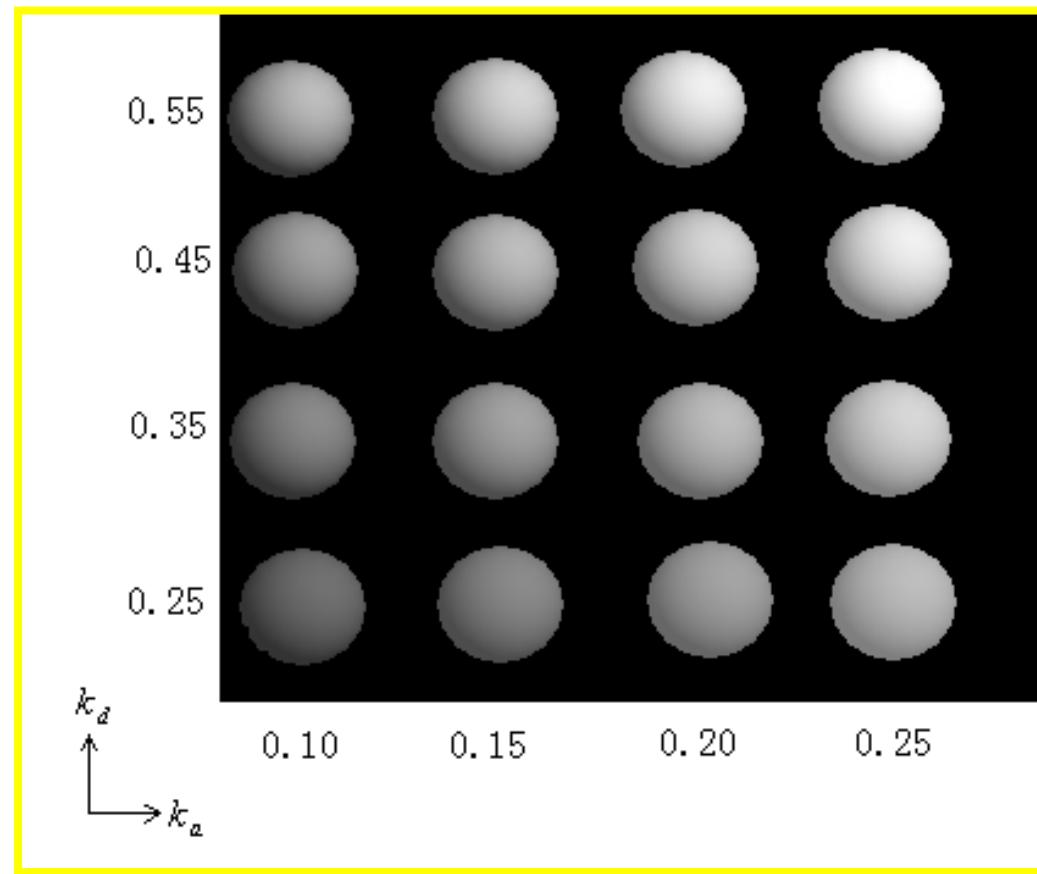
- $I_d$  漫反射的亮度
- $I_p$  点光源的亮度
- $K_d$  漫反射系数
- $\theta$  入射角

漫反射光的强度  
只与入射角有关



# ■ 将环境光与漫反射结合起来

## ■ 例子





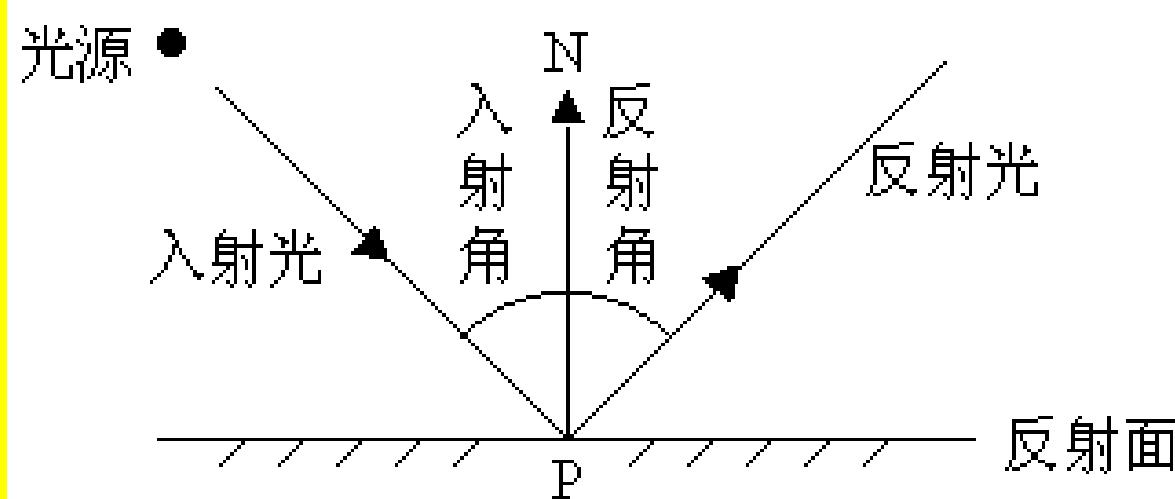
**缺点：**对于许多物体，使用上式计算其反射光是可行的，但对于大多数的物体，如擦亮的金属、光滑的塑料等是不适用的，原因是这些物体还会产生镜面发射。

# 简单光照明模型-镜面反射

## 镜面反射

- 光滑物体（如金属或塑料）表面对光的反射
- 高光
  - 入射光在光滑物体表面形成的特别亮的区域

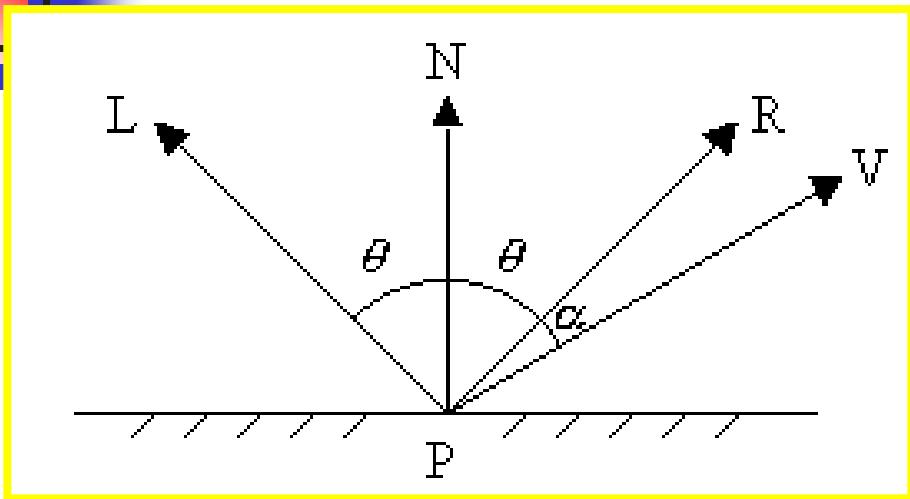
## 理想镜面反射



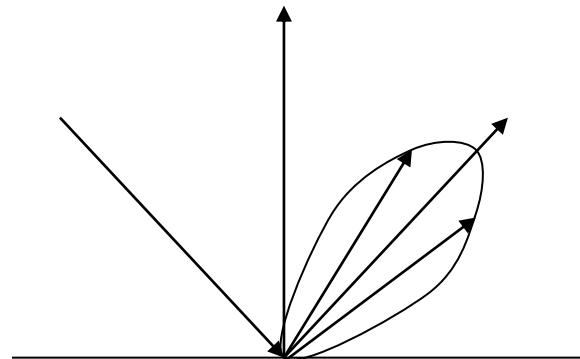
- 观察者只能在反射方向上才能看到反射光，偏离了该方向则看不到任何光。

## 非理想镜面反射

$$I = I_p K_s \cos^n \alpha$$



镜面



光滑平面

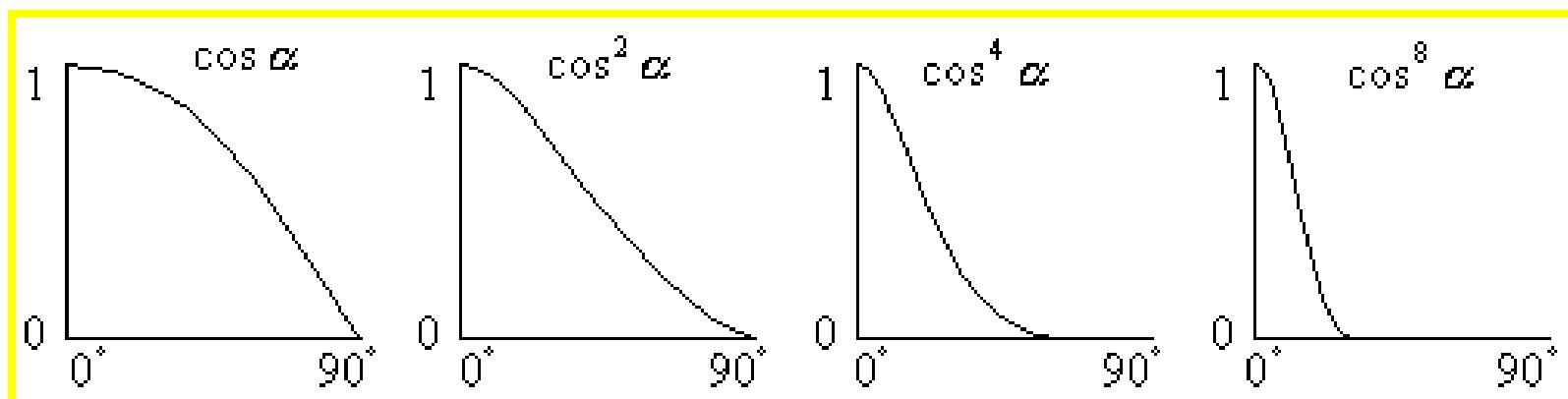
- P为物体表面上一点，L为从P指向光源的单位矢量，N为单位法矢量，R为反射单位矢量，V为从P指向视点的单位矢量

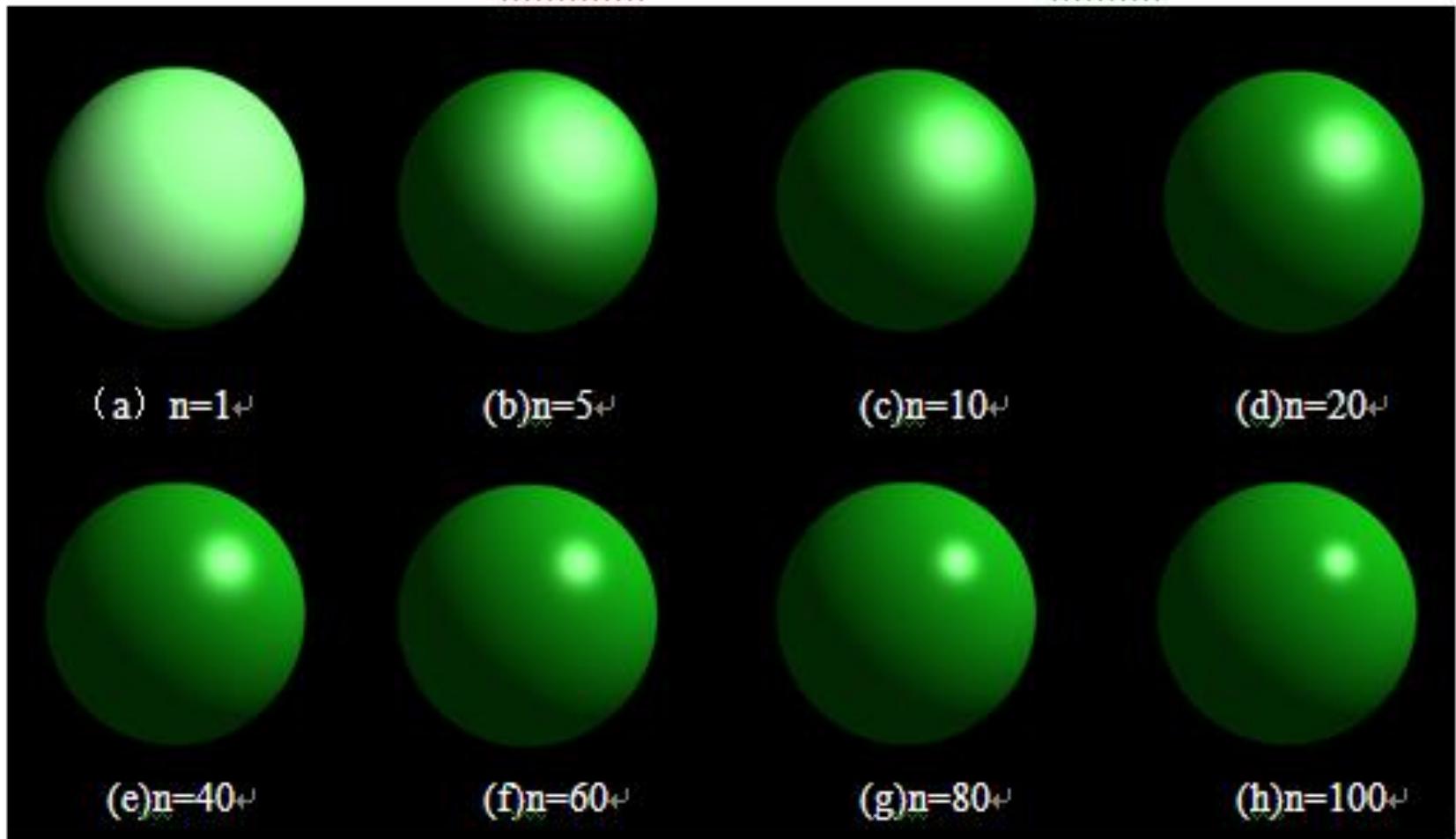
## ■ 镜面反射

$$I_s = I_p K_s \cos^n \alpha \text{ 或 } I_s = I_p K_s (V \cdot R)^n$$

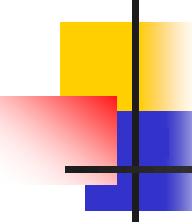
$I_s$ 为镜面反射光强。 $I_p$ 点光源的亮度

- $K_s$ 是与物体有关的镜面反射系数。n为镜面反射指数，n越大，则 $I_s$ 随 $\alpha$ 的增大衰减的越快。
- n的取值与表面粗糙程度有关。
  - n越大，表面越平滑（散射现象少，稍一偏离，明暗亮度急剧下降）
  - n越小，表面越毛糙（散射现象严重）





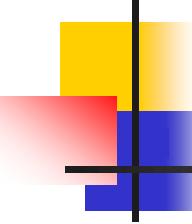
# 简单光照明模型-Phong光照明模型



简单光照明模型模拟物体表面对光的反射作用, 光源为点光源

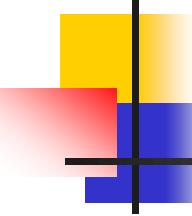
- 反射作用分为
  - 物体间作用环境光(Ambient Light)
  - 漫反射(Diffuse Reflection)
  - 镜面反射(Specular Reflection)

# 简单光照明模型-Phong光照明模型



Phong光照明模型的综合表述：由物体表面上一点 $P$ 反射到视点的光强 $I$ 为环境光的反射光强 $I_e$ 、理想漫反射光强 $I_d$ 、和镜面反射光 $I_s$ 的总和。

$$\begin{aligned} I &= I_e + I_d + I_s \\ &= I_a K_a + I_p [K_d (L \cdot N) + K_s (V \cdot R)^n] \end{aligned}$$



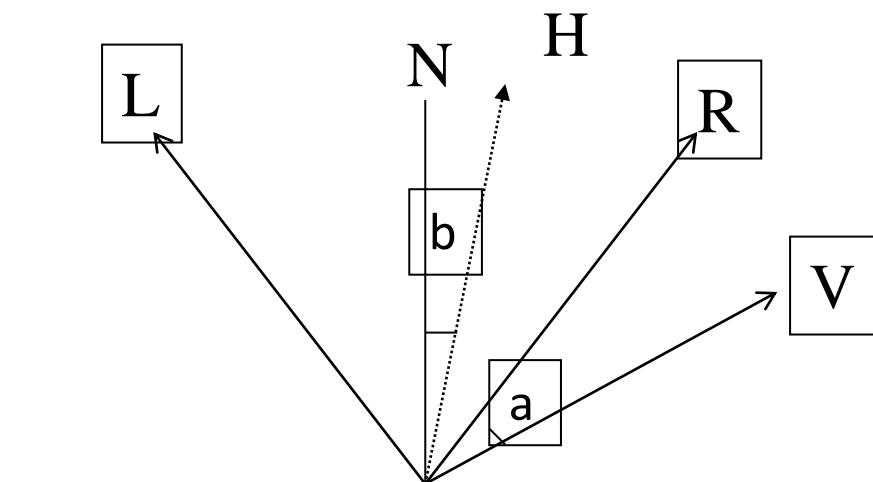
另外，反射光强度还和物体与点光源的距离d的平方成反比，即物体离光源愈远，显的愈暗。因此，若要得到真实感的光照效果，在光照明模型中必须考虑这一因素。然而，若采用因子 $1/d^2$ 来进行光强度衰减，简单的点光源照明并不总能产生真实感的图形。当d很小时， $1/d^2$ 会产生过大的强度变化，而d很大时反射光强度项将无意义。另外，人对物体的视觉也同视点与物体的距离有关，因此综合考虑，可以用以下修正公式，它是根据经验，取同距离的关系为线性衰减。

$$I = k_a I_a + I_p (k_d (N \cdot L) + k_s (R \cdot V)^n) / (d + K)$$

常数项K为一调整常数，它的存在可以防止当d很小时 $1/d$ 值太大。

■ 对物体表面上的每个点 $P$ , 均需计算光线的反射方向。为了减少计算量, 假设:

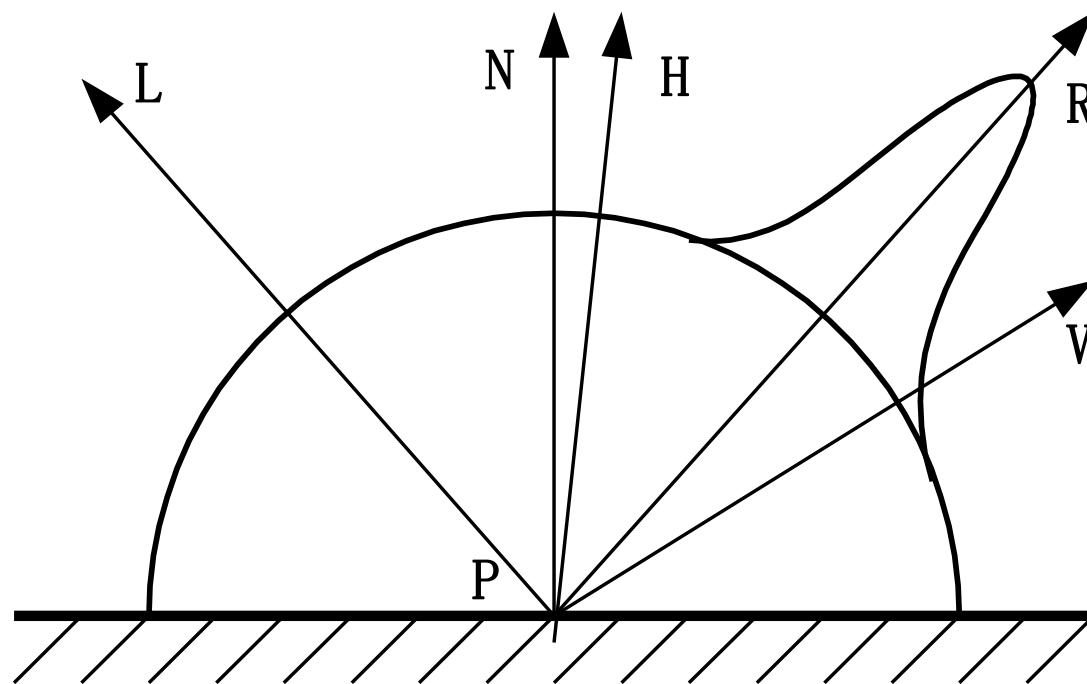
- 光源在无穷远处,  $L$ 为常向量
- 视点在无穷远处,  $V$ 为常向量
- $(H \cdot N)$  近似  $(R \cdot V)$ ,  $H$ 为 $L$ 与 $V$ 的平分向量



H----L和V的角平分线

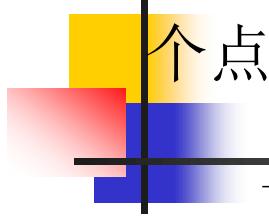
- 对所有的点总共只需计算一次 $H$ 的值, 节省了计算时间

## •Phong模型几何



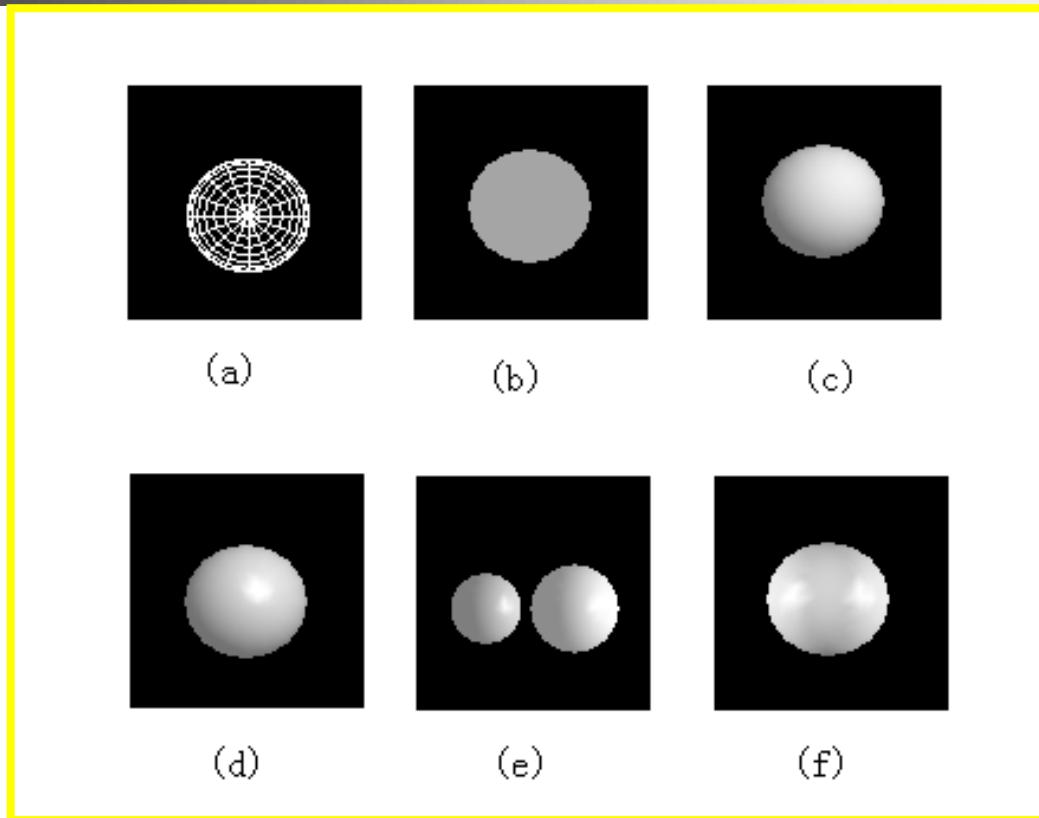
## 7.2.4 多点光源和颜色

若存在多个点光源，则物体表面上的反射光强度是各个点光源照射效果的线性相加，这时光照模型为：


$$I = k_a I_a + \sum_{i=1}^m \frac{I_{li}}{d_i + K_i} (k_d (N \cdot L_i) + k_s (N \cdot H_i)^n)$$

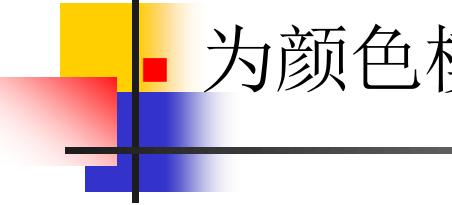
其中， $m$ 为点光源的数目。

■ 例子：其中**a**图：线框图   **b**图：环境光  
**c**图：增加漫反射   **d**图：增加镜面反射  
**e**图：增加光的衰减   **f**图：两个点光源

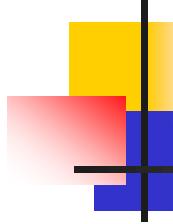


## ■ 产生彩色

- 选择合适的颜色模型----RGB模型
- 为颜色模型中的每一种基色建立光照明方程


$$\begin{cases} I_R = I_{aR}K_{aR} + I_{pR}[K_{dR}(L \cdot N) + K_s(N \cdot H)^n]/(d + k) \\ I_G = I_{aG}K_{aG} + I_{pG}[K_{dG}(L \cdot N) + K_s(N \cdot H)^n]/(d + k) \\ I_B = I_{aB}K_{aB} + I_{pB}[K_{dB}(L \cdot N) + K_s(N \cdot H)^n]/(d + k) \end{cases}$$

# 7.3 光滑着色



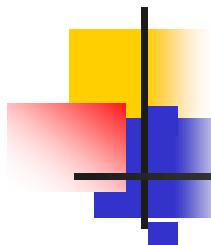
分类：均匀着色与光滑着色

## ■ 均匀着色

方法：任取多边形上一点，利用光照明方程计算出它的颜色，用这个颜色填充整个多边形

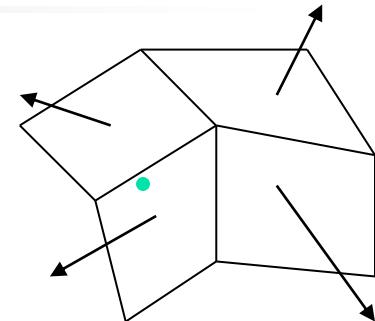
适用场合：

- 1) 光源在无穷远处；
- 2) 视点在无穷远处；
- 3) 多边形是物体表面的精确表示；



缺点：产生的图形效果不好。

如左图：相邻两个多边形的法向不同，计算出来的颜色也不同，因此造成整个物体表面的颜色过渡不光滑。

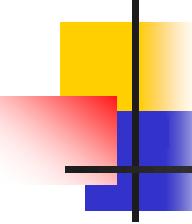


- 如何解决？
- 光滑着色，亦称插值着色

**Gouraud着色方法**

**Phong着色方法**

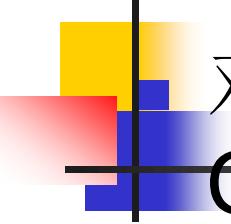
# Gouraud着色方法



Gouraud于1971年提出，又被称Gouraud  
明暗处理

- 基本思想：在每个多边形顶点处计算颜色，然后在各个多边形内部进行线性插值，得到多边形内部各点颜色。即它是一种颜色插值着色方法。
- 注意：Gouraud着色方法并不是孤立的处理单个多边形，而是将构成一个物体表面的所有多边形（多边形网格）作为一个整体来处理。

# Gourand 着色方法



对多边形网格中的每一个多边形，  
**Gourand** 着色处理分为如下四个步骤：

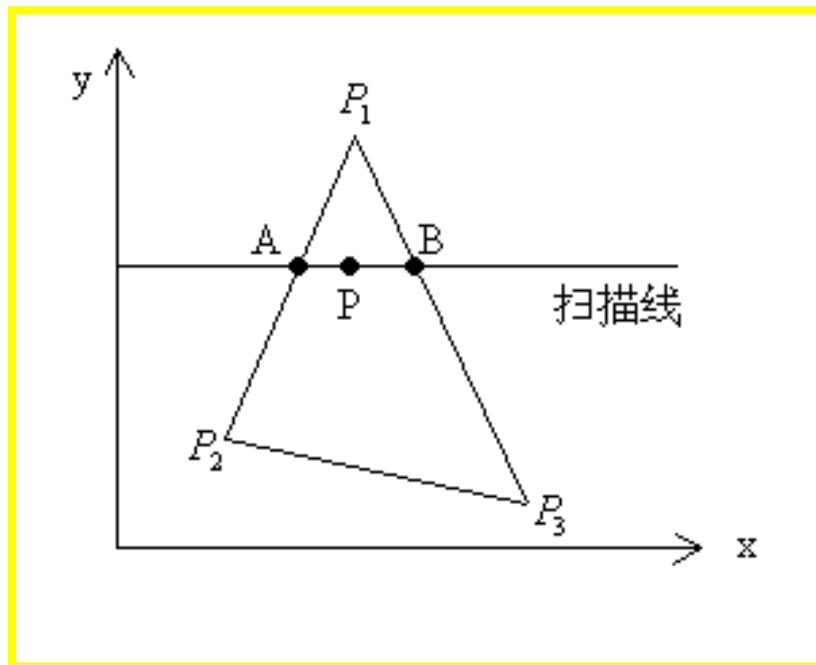
- 步骤

- 1、计算多边形的单位法矢量
- 2、计算多边形顶点的单位法矢量

# Gouraud 着色方法

3、利用光照明方程计算顶点光强（颜色）

4、对多边形顶点光强（颜色）进行双线性插值，获得多边形内部各点的光强（颜色）



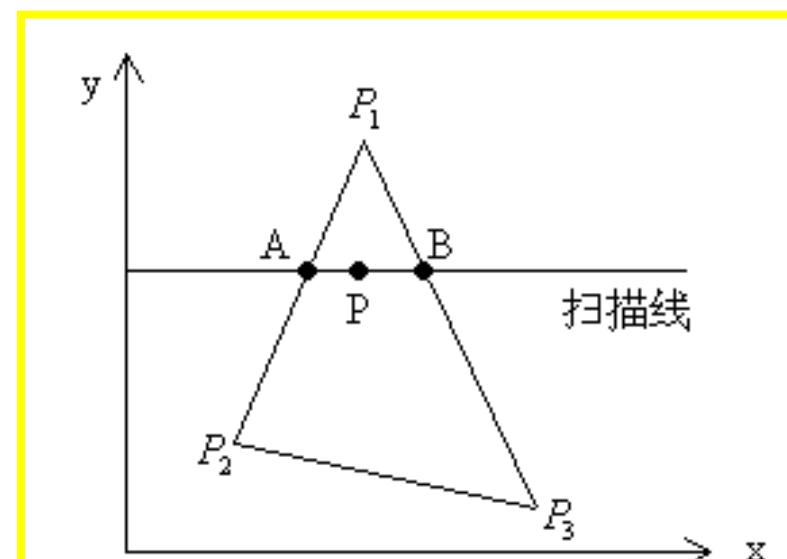
# Gouraud 着色方法-光强插值

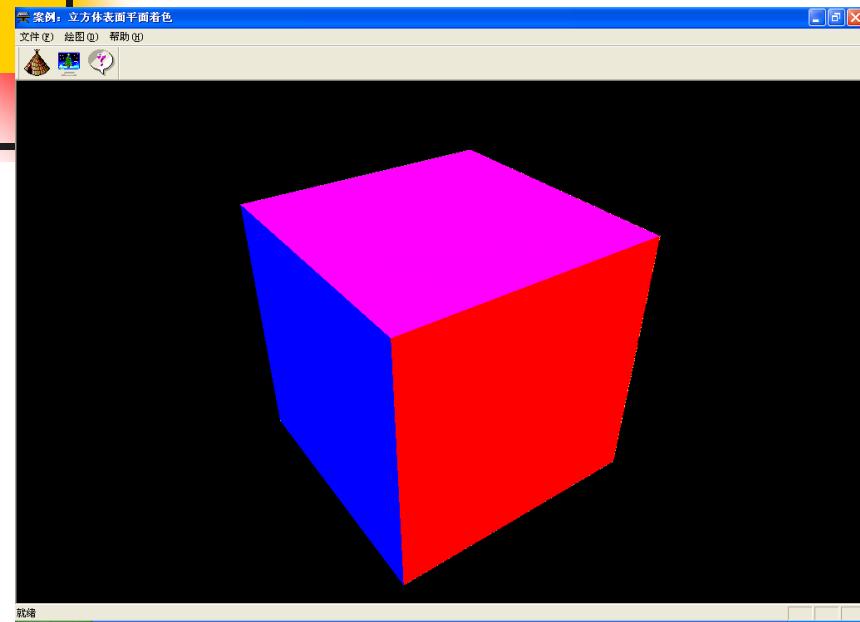
双线性光强插值：假设待绘制的三角形投影为  $P_1P_2P_3$ ,  $P_i$  的坐标为  $(x_i, y_i)$ ,  $i=1, 2, 3$ ; 一条扫描线与三角形的两条边分别交于  $A(x_A, y_A)$ ,  $B(x_B, y_B)$  两点。 $P(x, y)$  是  $AB$  上的一点。 $A$  点的颜色  $I_A$  由  $P_1$ 、 $P_2$  点的颜色  $I_1$ 、 $I_2$  线性插值得到

$$I_A = \frac{y_A - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_A}{y_1 - y_2} I_2$$

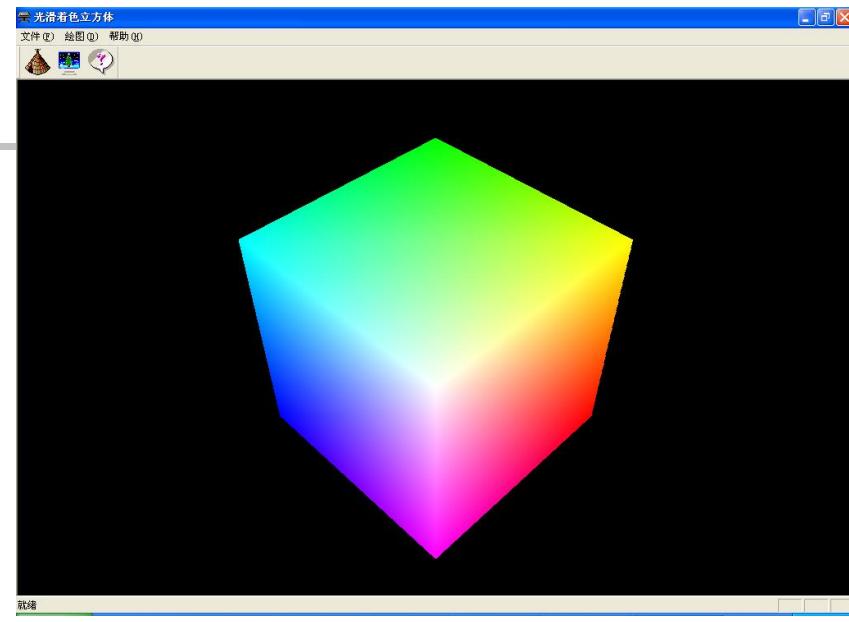
$$I_B = \frac{y_B - y_3}{y_1 - y_3} I_1 + \frac{y_1 - y_B}{y_1 - y_3} I_3$$

$$I_P = \frac{x_B - x}{x_B - x_A} I_A + \frac{x - x_A}{x_B - x_A} I_B$$





(a) 平面着色



(b) Gouraud光滑着色

立方体颜色填充



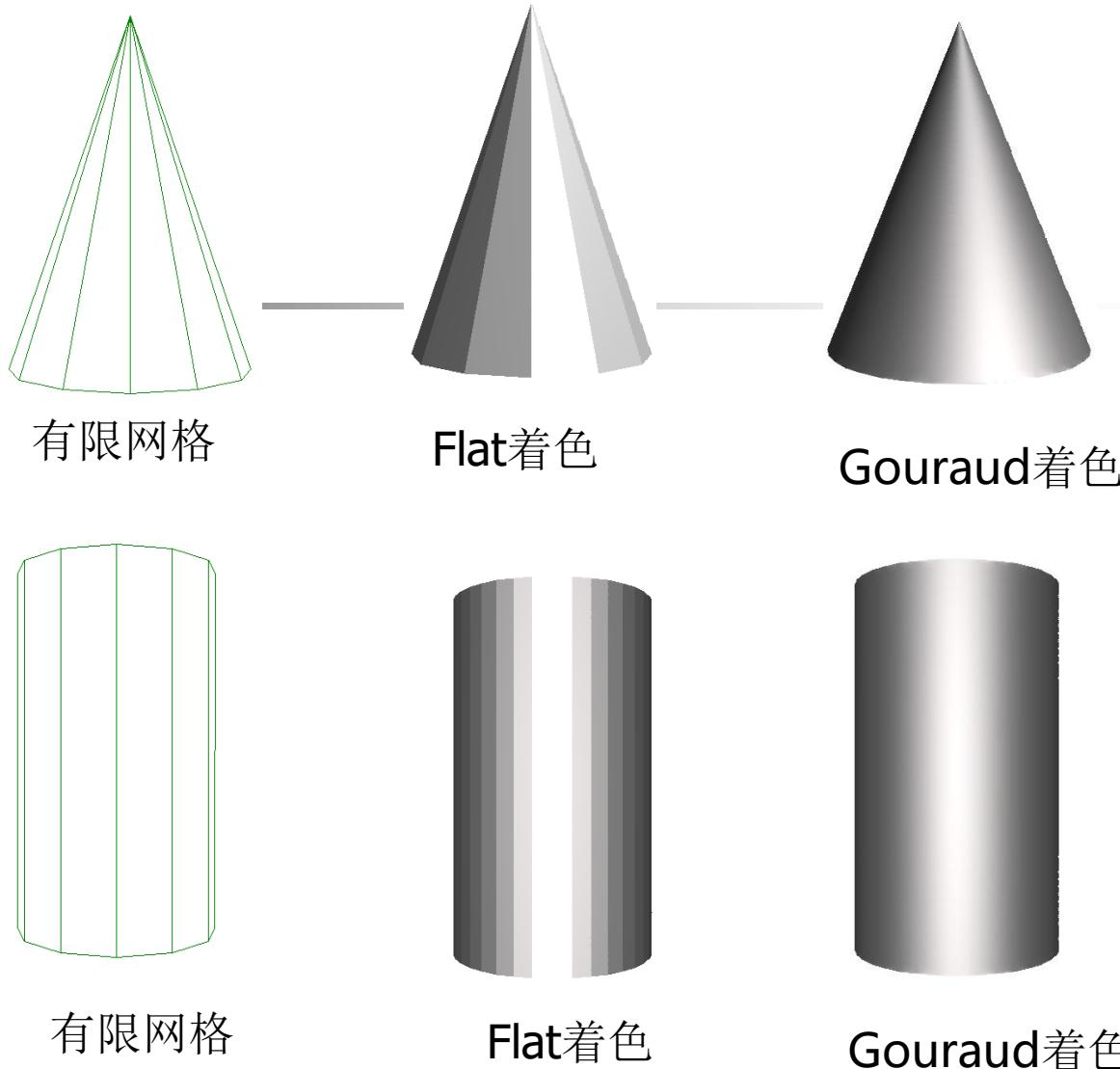
物体是使用平面多边形逼近的。上图光滑的效果是通过增加多边形数量来获得的么？

**设计目标：**

使用有限多边形来获得光滑表面，如何实现？



## 平均法矢量的作用



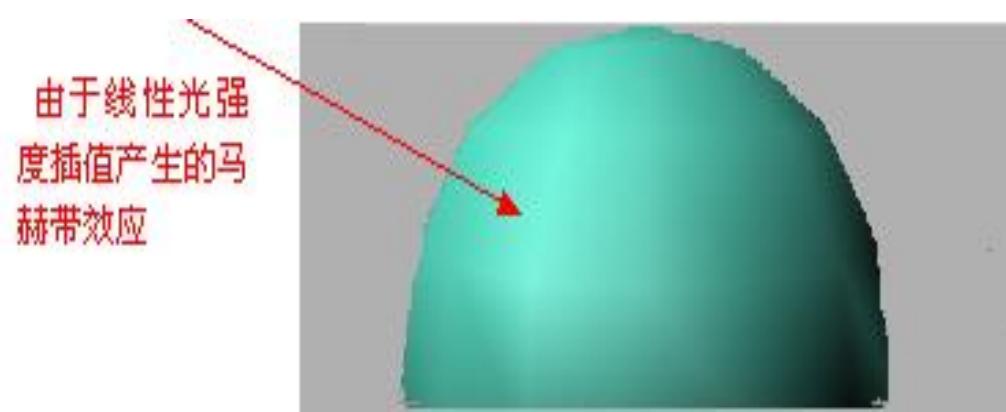
**Gouraud**插值算法使得图形变得光滑，主要是因为使用了相邻两个面的平均法矢量计算过渡光强，实现了“使用有限多边形构造光滑表面”的设计目标。

# Gouraud 着色方法

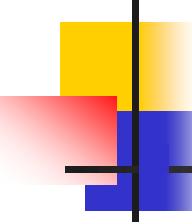
优点：能有效的显示漫反射曲面，计算量小

缺点：

- 1、高光有时会异常
- 2、当对曲面采用不同的多边形进行分割时会产生不同的效果。
- 3、Gouraud明暗处理会造成表面上出现过亮或过暗的条纹，称为马赫带（Mach\_band）效应
- 改进—Phong提出双线性法向插值，以时间为代价，解决高光问题



# Phong着色方法



基本思想：通过对多边形顶点的法矢量进行插值，获得其内部各点的法矢量，又称为法向插值着色方法。

## ■ 步骤

- 1、计算多边形单位法矢量
- 2、计算多边形顶点单位法矢量
- 3、对多边形顶点法矢量进行双线性插值，获得内部各点的法矢量
- 4、利用光照明方程计算多边形内部各点颜色

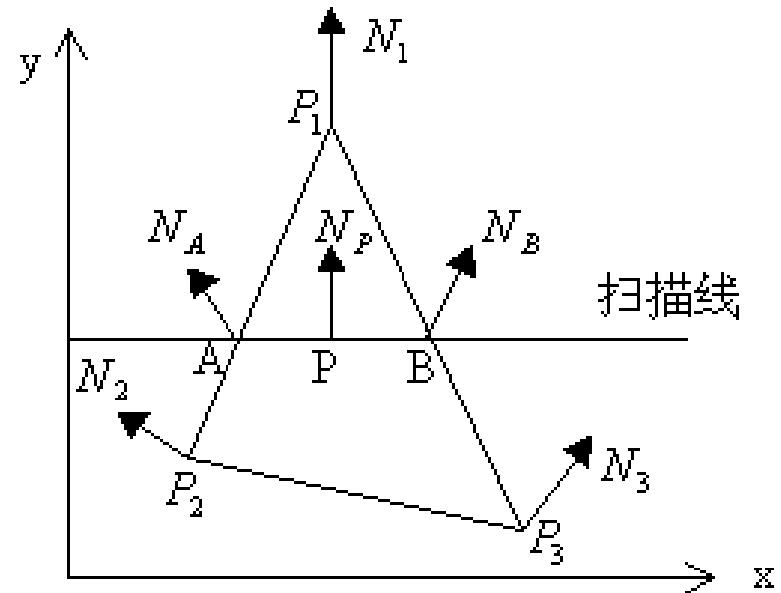
# Phong着色方法-法向插值

$N_A$ 由 $N_1$ 、 $N_2$ 线性插值得到：

$$N_A = \frac{y_A - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y_A}{y_1 - y_2} N_2$$

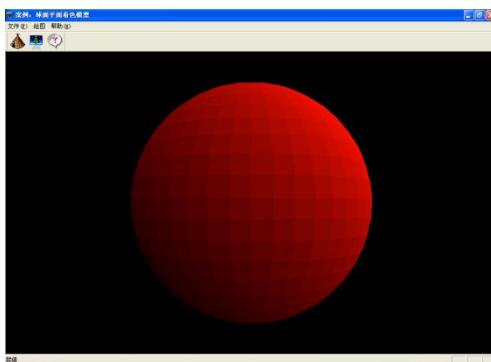
$$N_B = \frac{y_B - y_3}{y_1 - y_3} N_1 + \frac{y_1 - y_B}{y_1 - y_3} N_3$$

$$N_P = \frac{x_B - x}{x_B - x_A} N_A + \frac{x - x_A}{x_B - x_A} N_B$$

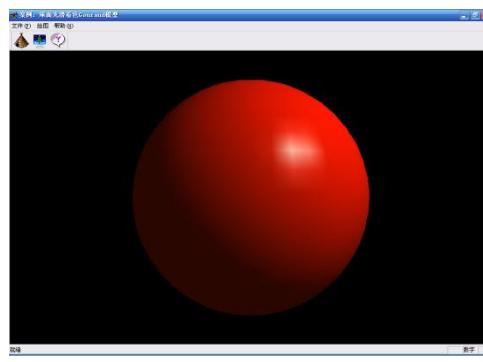


**Phong** 明暗处理优点：可以产生正确的高光区域。

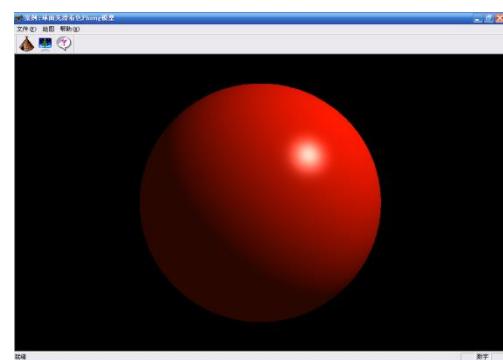
**Phong** 明暗处理缺点：既要通过三角形网格各顶点的法矢量来插值计算多边形内各点的法矢量，还要调用光照模型计算其光强，计算时间是**Gouraud**明暗处理的6~8倍。



(a) Flat

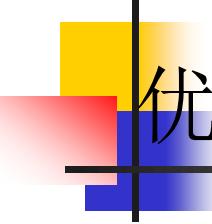


(b) Gouraud  
球面着色模式效果图



(c) Phong

# Phong着色方法



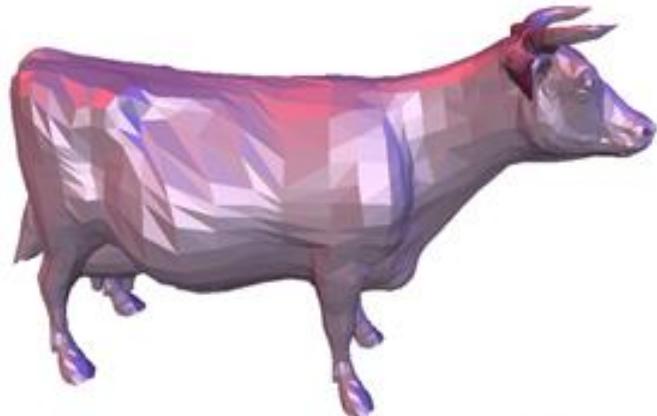
优点：

Phong着色方法绘制的图形比Gouraud方法更真实，体现在两个方面：高光区域的扩散，产生正确的高光区域

缺点：

- 1、Phong着色方法计算量远大于Gouraud着色方法
- 2、在处理某些多边形分割的曲面时，Phong算法还不如Gouraud算法好。

- 牛的三角网格模型
- 用简单光照明模型显示
- 用增量式光照明模型显示

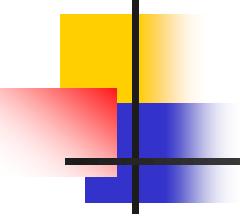


## 7.4 简单透明模型

简单透明模型是不考虑折射的影响的经验模型。该模型简单地将物体1上各像素处的光强与其后的另一个物体2上相应像素处的光强作线性插值以确定物体1上各像素最终显示的光强。

$$I = (1 - t) \cdot I_1 + t \cdot I_2 \quad , \quad t \in [0, 1]$$

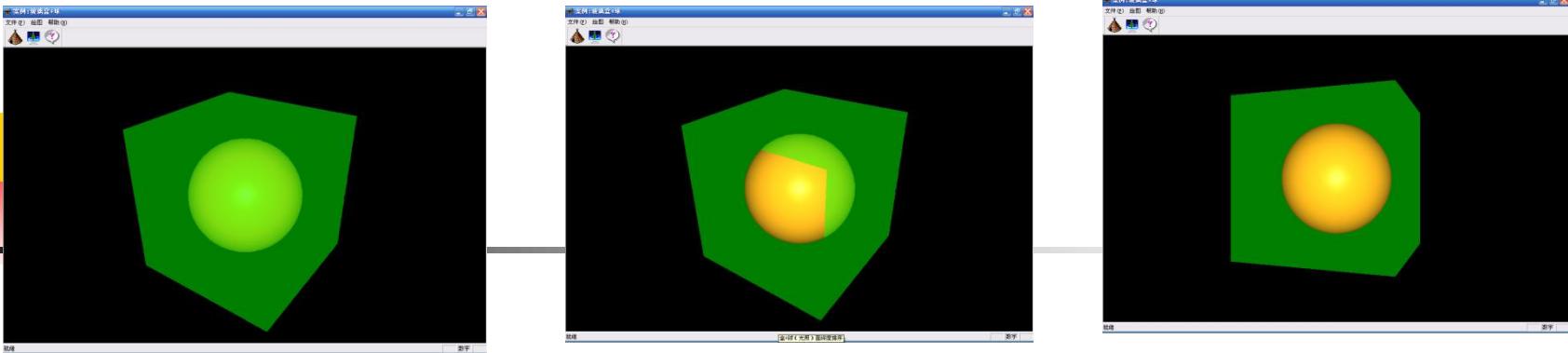
式中， $I_1$ 为物体1上某一像素的光强， $I_2$ 为物体2上相应像素的光强。 $t$ 为透明度，其值通常取自CRGB类的alpha分量。当 $t=1.0$ 时，物体1透明，可以完全看到物体2；当 $t=0.0$ 时，物体1完全不透明，物体2被物体1遮挡。当 $t$ 的取值位于[0.0, 1.0]内时，如 $t=0.5$ ，物体的最终颜色是物体1的颜色与物体2的颜色的线性融合，即物体1的颜色与物体2的颜色各占50%。假定物体2不透明，物体1的透明度的变化如图所示。


$$\text{颜色化: } \begin{cases} I_R = (1 - t) \cdot I_{1R} + t \cdot I_{2R} \\ I_G = (1 - t) \cdot I_{1G} + t \cdot I_{2G} \\ I_B = (1 - t) \cdot I_{1B} + t \cdot I_{2B} \end{cases} \quad (10-37)$$

在黑色场景中放置物体A和物体B，物体A为绿色立方体，物体B为黄色球体，球体内置于此立方体中心。为了演示透明效果，立方体只绘制5个表面，另一个表面作为“开窗”。

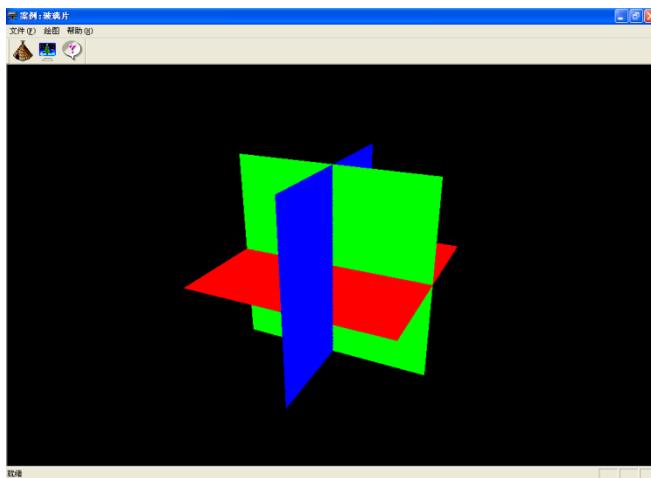
在黑色场景中绘制两两正交的红绿蓝玻璃片，效果如图所示。

简单透明模型的优点是计算简单，但只能模拟透明效果，不能模拟光的折射效果。计算机图形学中也常用简单透明模型模拟雾效果，使远处的物体看上去逐渐变得模糊，从而增加场景的真实感。



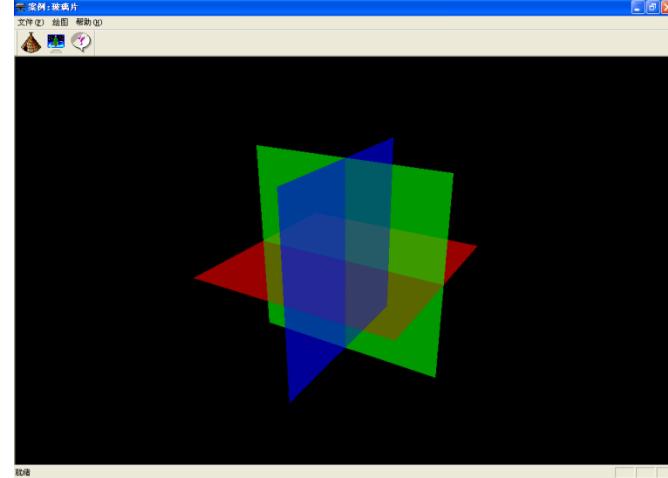
红色球+绿色盒子

图  $t=0.5$ 的透明立方体与内置球体



(a)  $t=0.0$

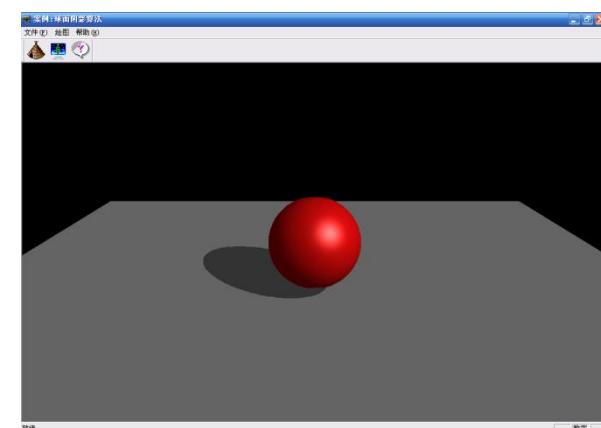
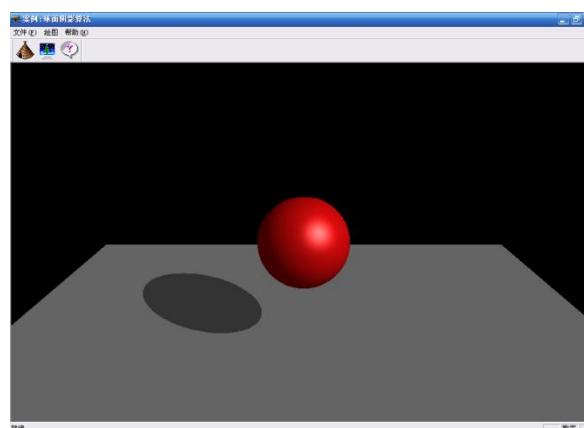
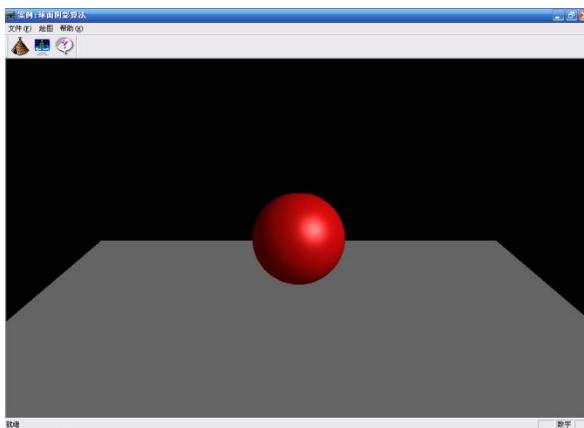
图 正交玻璃片



(b)  $t=0.6$

## 7.5 简单阴影模型

自然界中，物体只要受到光照，就会产生阴影。阴影效果对增强场景的真实感有着非常重要的作用。阴影可以反映物体之间的相对位置，增强场景的立体感和层次感。



阴影是由于物体截断了光线而产生的，如果光源位于物体的一侧，阴影总是位于物体的另一侧，也就是与光源相反的一侧。如果视点和光源在同一方向上，得不到光照的阴影面同时又是看不到的隐藏面，不会有阴影。

物体的多边形表面，如果在阴影区域内部，则该多边形的光强就只有环境光一项；否则就用正常的光照模型计算光强。采用这种简单的方法，就可以将阴影引入简单光照模型中，使产生的真实感图形更具有层次感。

对于单点光源，阴影算法与隐面算法相似。隐面算法确定哪些表面从视点看过去是不可见的，而阴影算法确定哪些表面从光源看过去是不可见的。从光源位置看过去不可见的区域就是阴影区域。计算阴影相当于两次消隐过程。

在单点光源的照射下，阴影分为自身阴影与投射阴影。假设单点光源位于立方体左面的无穷远处，视点位于立方体前方。这时产生的阴影包括两个部分：一部分是由于物体自身的遮挡而使光线照射不到它的某些表面产生自身阴影；另一部分是由于不透明的物体遮挡光线使得位于物体另一侧的区域受不到光照而形成投射阴影。

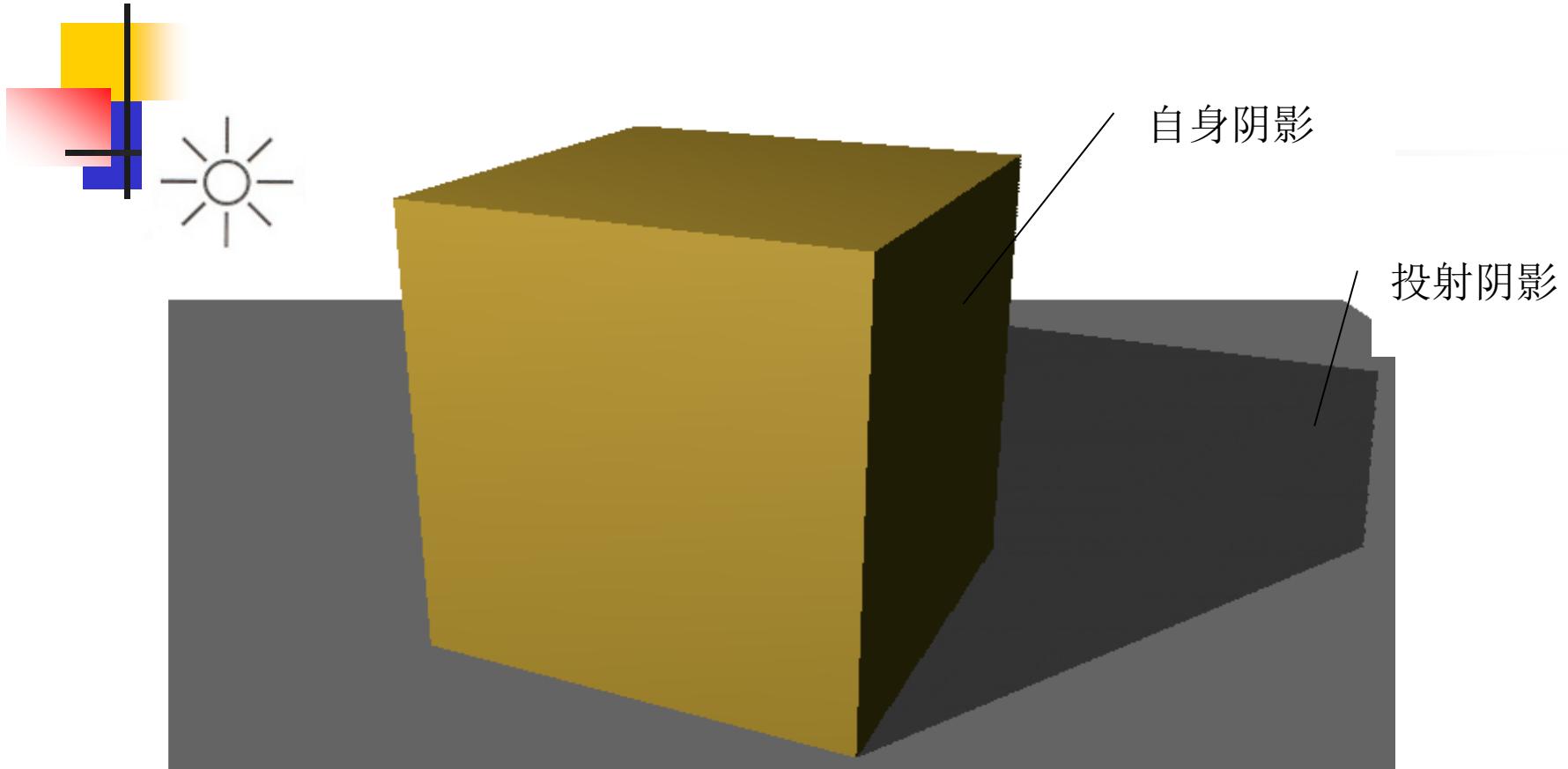
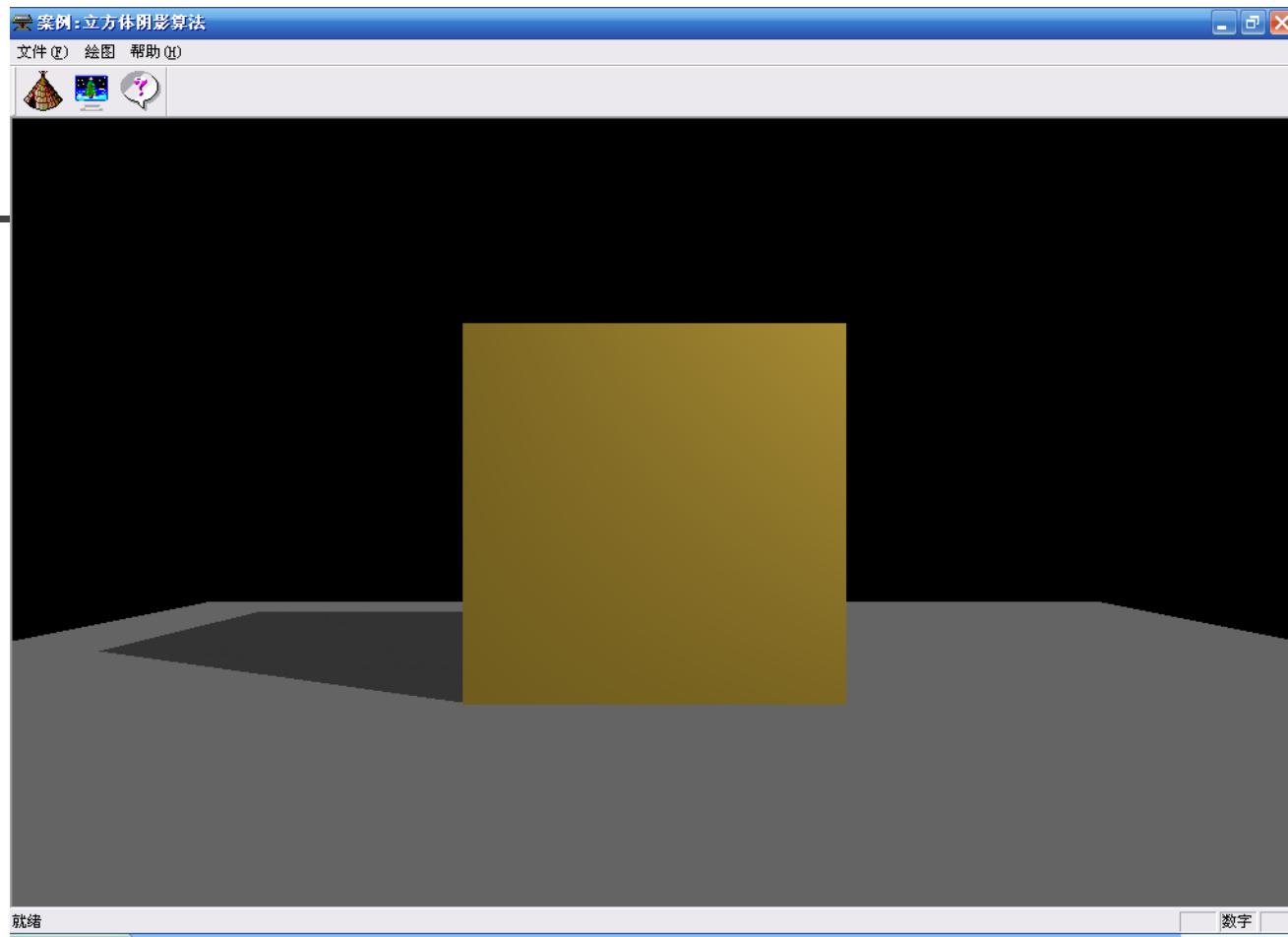


图10-34 阴影的分类

绘制自身阴影与投射阴影图形的算法如下：

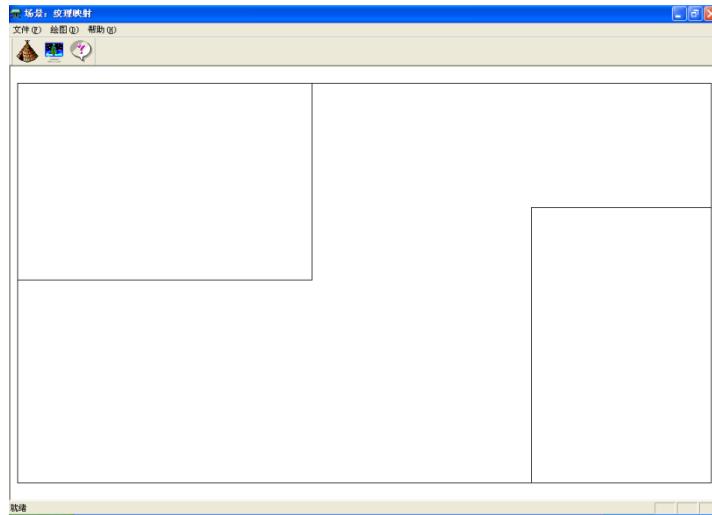
- (1) 根据视点原来的观察位置，对物体实施隐面算法，使用正常的光照模型计算光强来绘制可见表面。
- (2) 将视点移到光源的位置。从光源处向物体所有背光面投射光线，建立光线的参数方程，计算该光线与投影面（地面）的交点，使用深灰色填充交点所构成的阴影多边形，形成投射阴影。若选用简单光照模型，对于背光面，由于得不到光源的直接照射，只有环境光对其光强有贡献。



光照立方体产生的阴影

## 7.6 纹理映射

现实世界中的物体表面存在丰富的纹理细节，人们正是依据这些纹理细节来区分各种具有相同形状的物体。



(a) 空白场景

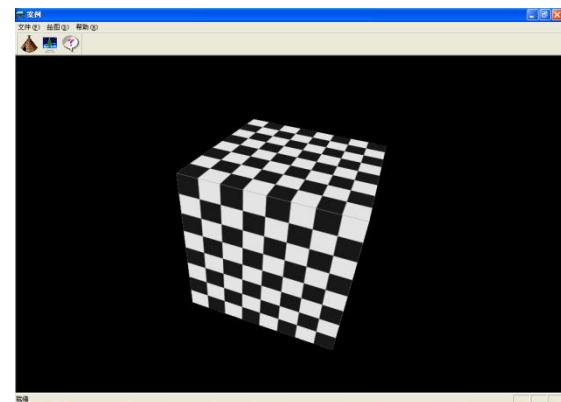
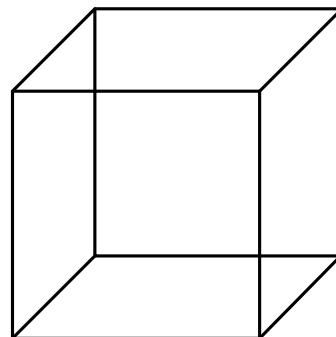
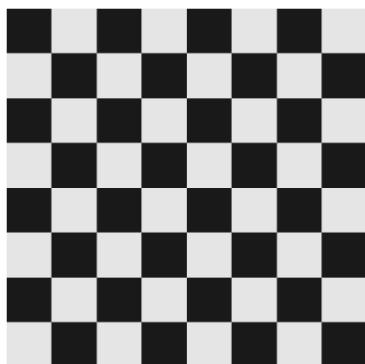


(b) 纹理映射场景

为场景添加纹理细节

为物体表面添加纹理的技术称为纹理映射（**texture mapping**）。

纹理映射是将纹理空间（**texture space**）坐标（ $u, v$ ）映射为物体空间（**object space**）坐标（ $x, y, z$ ），再进一步映射为屏幕图像空间（**screen space**）二维物体表面坐标（ $x, y$ ）的过程，如图所示。



(a) 纹理空间  
纹理映射

(b) 物体空间

(c) 图像空间

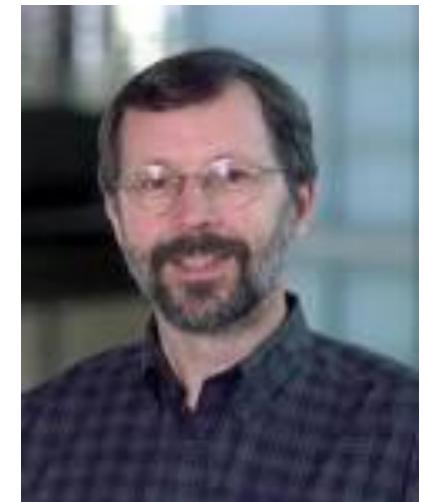
$$I = I_e + I_d + I_s = k_a I_a + k_d I_p \max(L \cdot N, 0) + k_s I_p \max(H \cdot N, 0)^n$$

当什么属性发生改变时，可以产生纹理效果呢？

漫反射光模型主要产生纹理效果。当光源的位置不变时，单位光矢量**L**是一个定值。影响光强的只有漫反射系数**k<sub>d</sub>**和单位法矢量**N**的方向。

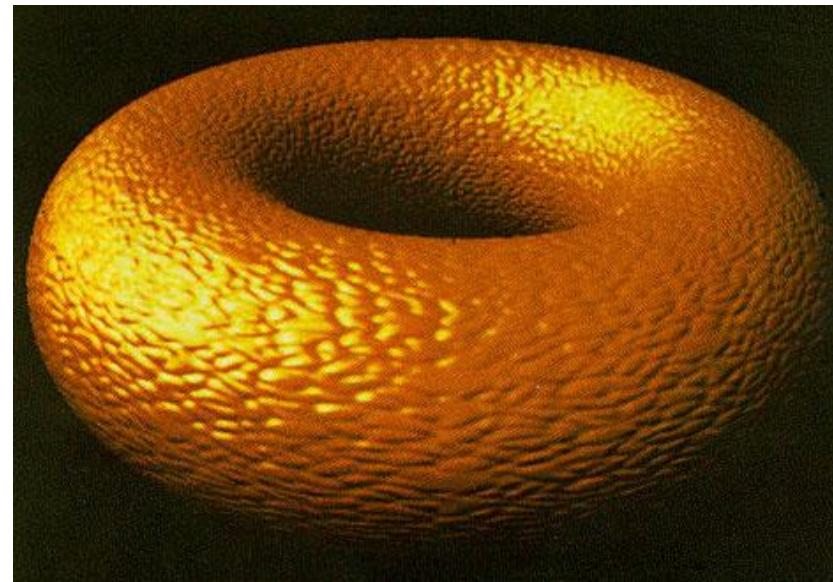
1974年，Catmull首先采用二维图像来定义物体表面材质的漫反射系数**k<sub>d</sub>**，这种纹理被称为**颜色纹理**。

1978年，Blinn提出了在光照模型中适当扰动物体表面的单位法矢量**N**的方向产生表面凹凸纹理的方法，被称为**几何纹理**。

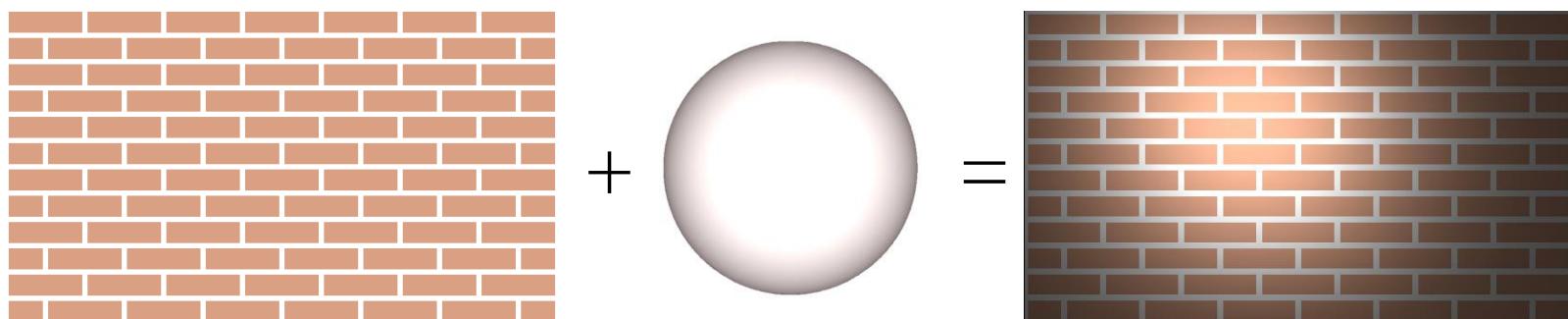


Edwin Earl Catmull  
88

Jim Blinn, 全名James F. Blinn, 世界图形学先驱。曾荣获Siggraph年度计算机图形学成就奖。系微软公司研究所图形学研究员。凸凹纹理映射: 由Jim Blinn发明, 用法向扰动技术模拟带褶皱的曲面。Bump mapping最初由Jim Blinn在1978年发明。



颜色纹理映射可以有两种实现方法，一种方法是直接用纹理的颜色替代物体表面的颜色。在这种情况下，不必进行光照计算。另一种方法是纹理数据参加光照计算。在这种情况下，物体表面的纹理会显示光照效果，但这会影响到镜面高光效果。因为镜面反射光的颜色是由光源颜色决定的，而与物体本身材质颜色无关。处理方法是将镜面反射光分离出来，通过设置材质漫反射系数 $k_d$ 完成纹理映射后，再将镜面反射光分量加上去。



## 7.6.1 颜色纹理

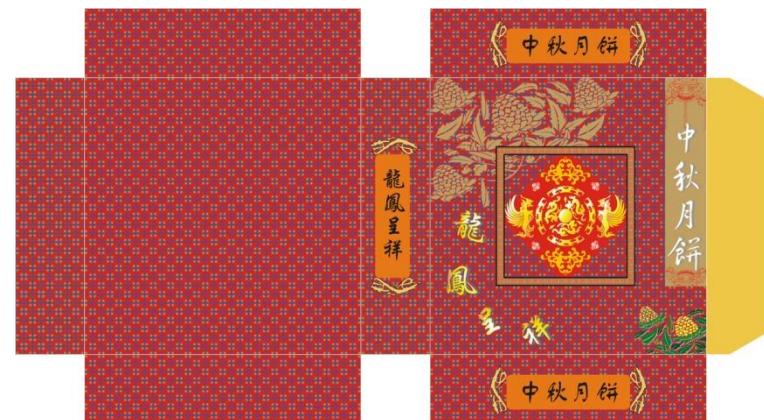
通过颜色变化表现出来的表面细节，称为**颜色纹理**。颜色纹理难以直接构造，常采用函数纹理或图像纹理来描述表面细节。为了使映射在物体表面的颜色纹理不因物体位置的改变而漂移，需要将颜色纹理绑定到物体的表面上。一般采用物体表面的参数化方法来确定表面的纹理坐标。



木纹



大理石



包装盒

## 1. 函数纹理

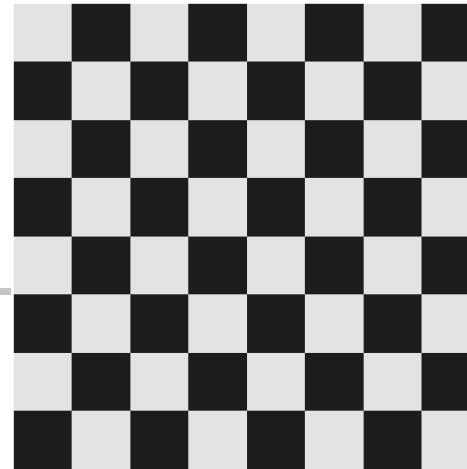
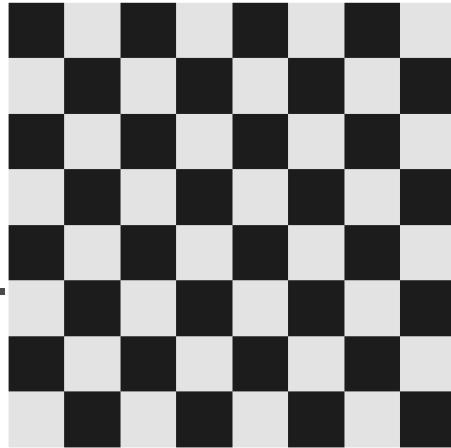
纹理一般定义在单位正方形区域 ( $0 \leq u \leq 1, 0 \leq v \leq 1$ )，称为纹理空间。理论上，任何定义在此空间内的函数都可以作为纹理函数，实际上，常采用一些特殊的函数来模拟现实世界中存在的纹理，如国际象棋棋盘纹理函数、粗布纹理函数等。

### (1) 国际象棋棋盘纹理函数

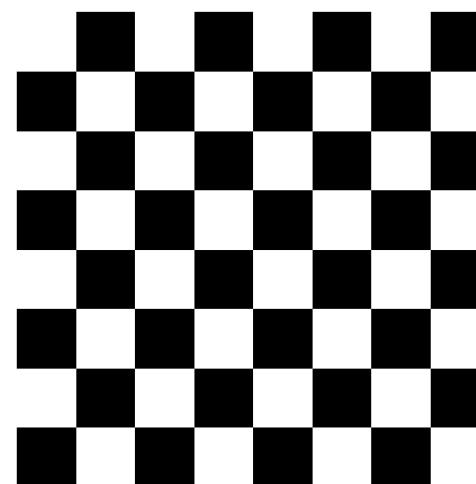
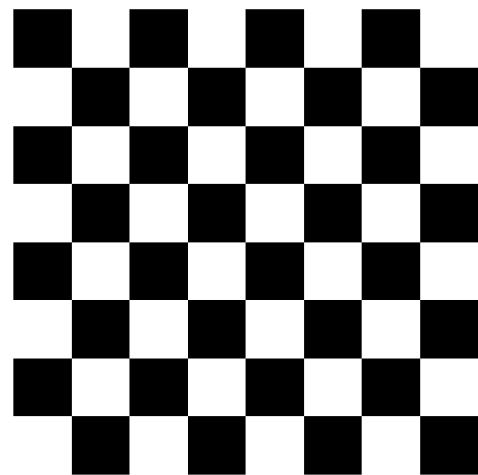
$$g(u, v) = \begin{cases} a & \lfloor u \times 8 \rfloor + \lfloor v \times 8 \rfloor \text{ 为偶数} \\ b & \lfloor u \times 8 \rfloor + \lfloor v \times 8 \rfloor \text{ 为奇数} \end{cases}$$

其中， $0 < a < b < 1$ ， $\lfloor x \rfloor$  表示不大于  $x$  的最大整数。

a 和 b 是 RGB 宏的颜色分量。



(a)  $a=0.9, b=0.1$     (b)  $a=0.1, b=0.9$



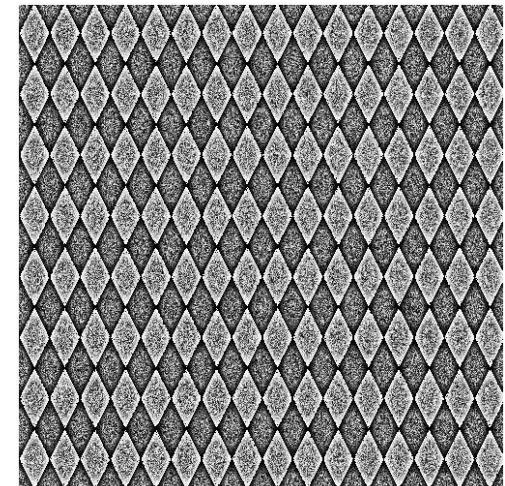
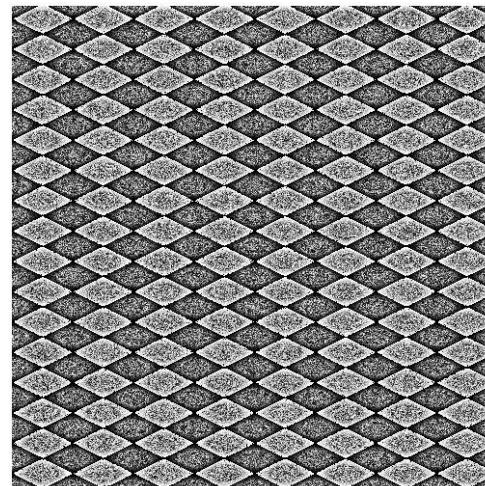
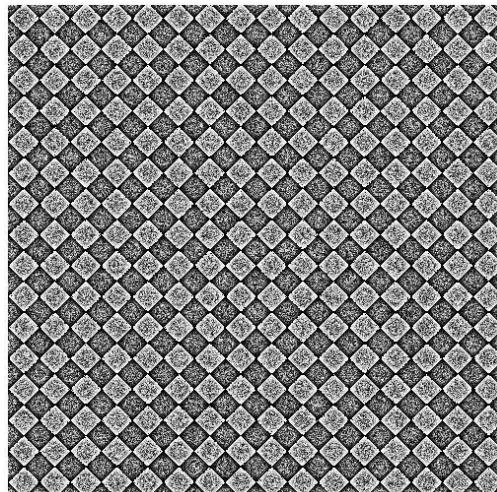
(c)  $a=1.0, b=0.0$     (d)  $a=0.0, b=1.0$

```
double a,b;
for(double u=0.0;u<=1.0;u+=0.001)
for(double v=0.0;v<=1.0;v+=0.001)
{
    if(0==(int(floor(u*8))+int(floor(v*8)))%2)//偶数
    {
        a=0.9;
        pDC->SetPixelV(Round(u*500-250),Round(v*500-250)
                        ,RGB(a*255,a*255,a*255));
    }
    else
    {
        b=0.1;
        pDC->SetPixelV(Round(u*500-250),Round(v*500-250)
                        ,RGB(b*255,b*255,b*255));
    }
}
```

## (2) 粗布纹理函数

$$f(u, v) = A(\cos(pu) + \cos(qv)) \quad (10-39)$$

式中，A为[0,1]上的随机变量，p, q为频率系数。



(a)  $p=100, q=100$

(b)  $p=50, q=100$

(c)  $p=100, q=50$

图10-40 粗布纹理

```
void CTestView::OnDraw(CDC* pDC)
{
    CTestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    CRect rect;
    GetClientRect(&rect);
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowExt(rect.Width(),rect.Height());
    pDC->SetViewportExt(rect.Width(),-rect.Height());
    pDC->SetViewportOrg(rect.Width()/2,rect.Height()/2);
    double p=100,q=100;
    for(double u=0.0;u<=1.0;u+=0.001)
        for(double v=0.0;v<=1.0;v+=0.001)
    {
        double A=double(rand())/RAND_MAX;
        double f=A*(cos(p*u)+cos(q*v));
        pDC->SetPixelV(Round(u*500-250),Round(v*500-250),RGB(f*255,f*255,f*255));
    }
}
```

### (3) 立方体表面纹理映射

立方体表面进行纹理映射的变换矩阵为

$$(x, y, z) = (u, v, 1) \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix}$$

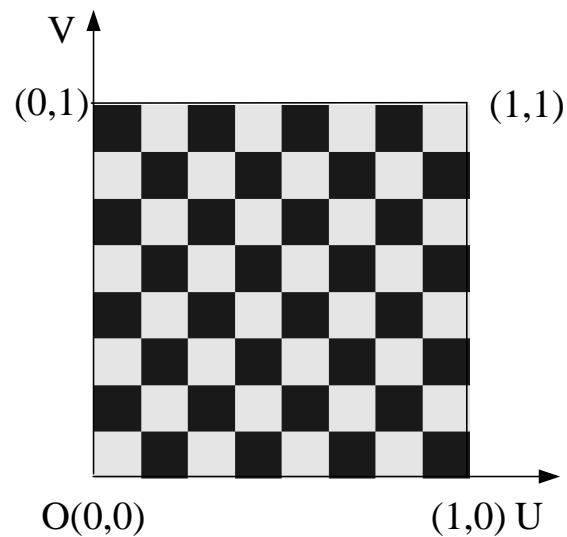
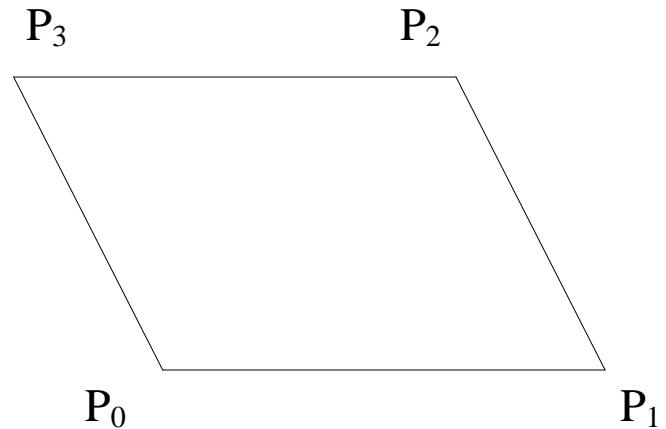
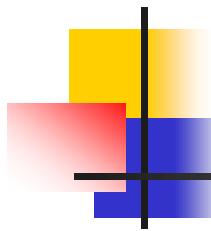


图10-41 二维函数纹理

图10-42 立方体的一个三维表面



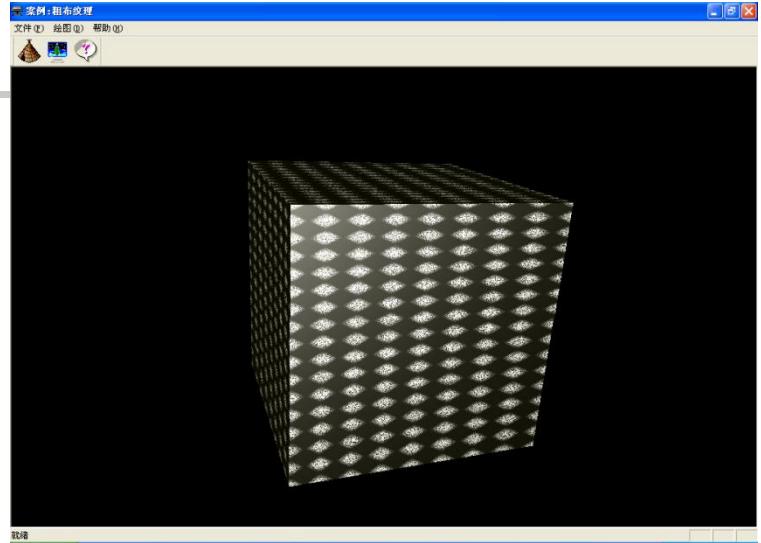
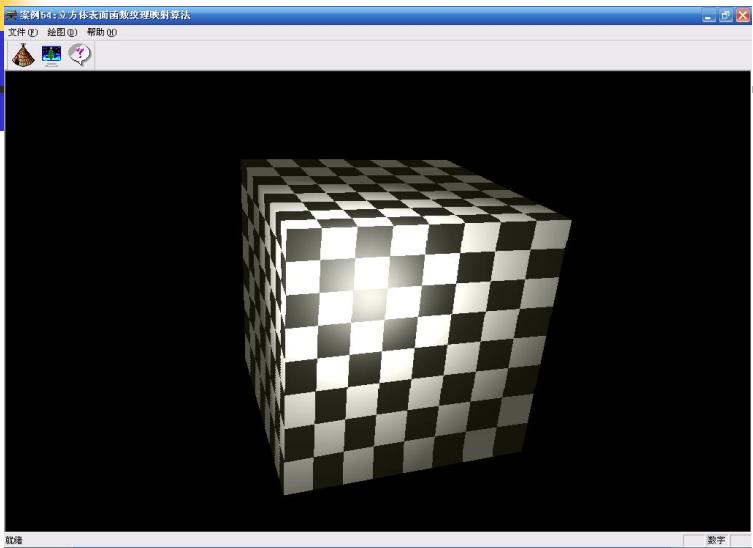


$$\begin{cases} P_0 \cdot x = C \\ P_0 \cdot y = F \\ P_0 \cdot z = I \end{cases} \quad \begin{cases} P_1 \cdot x = A + C \\ P_1 \cdot y = D + F \\ P_1 \cdot z = G + I \end{cases} \quad \begin{cases} P_2 \cdot x = A + B + C \\ P_2 \cdot y = D + E + F \\ P_2 \cdot z = G + H + I \end{cases}$$

$$\begin{cases} A = P_1 \cdot x - P_0 \cdot x \\ D = P_1 \cdot y - P_0 \cdot y \\ G = P_1 \cdot z - P_0 \cdot z \end{cases} \quad \begin{cases} B = P_2 \cdot x - P_1 \cdot x \\ E = P_2 \cdot y - P_1 \cdot y \\ H = P_2 \cdot z - P_1 \cdot z \end{cases} \quad \begin{cases} C = P_0 \cdot x \\ F = P_0 \cdot y \\ I = P_0 \cdot z \end{cases}$$

将立方体表面上的一点  $P(x, y, z)$  表达为  $(u, v)$  的参数形式，解得

$$P = (P_1 - P_0)u + (P_2 - P_1)v + P_0$$



(a) 国际象棋棋盘函数纹理

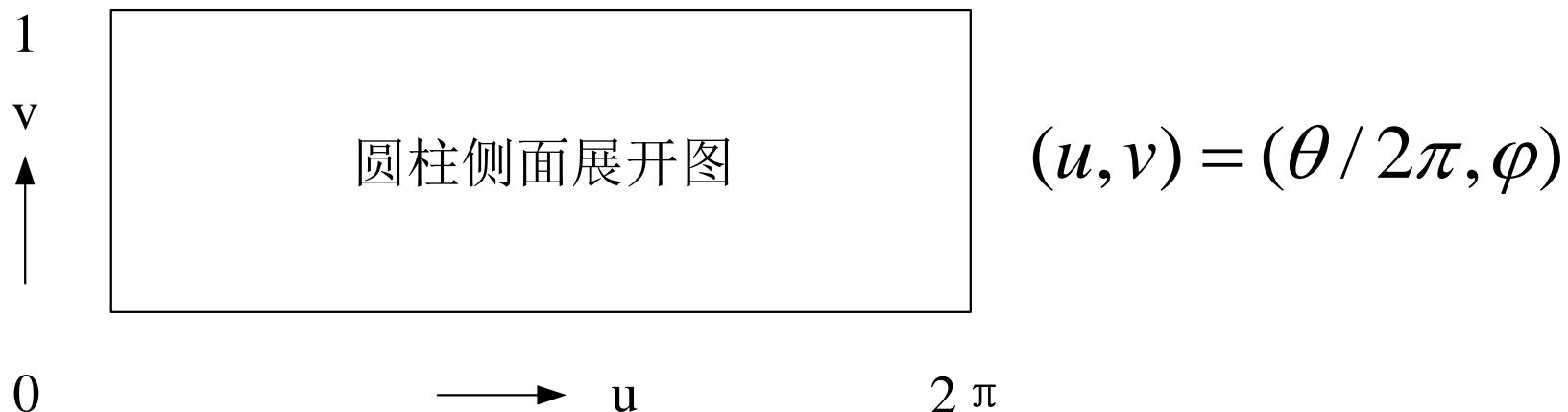
(b) 粗布函数纹理

立方体函数纹理映射

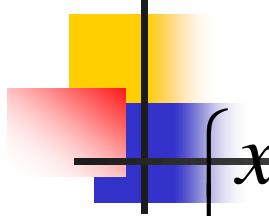
#### (4) 圆柱面函数纹理映射

高度为 $h$ 、截面半径为 $r$ 、三维坐标系原点位于底面中心。  
圆柱面的参数方程为

$$\begin{cases} x = r \cos \theta \\ y = h\varphi \\ z = r \sin \theta \end{cases} \quad 0 \leq \theta \leq 2\pi \quad 0 \leq \varphi \leq 1 \quad (10-43)$$



圆柱面的uv化表示为


$$\left\{ \begin{array}{l} x = r \cos(2\pi u) \\ y = hv \\ z = r \sin(2\pi u) \end{array} \right. , 0 \leq u \leq 1, 0 \leq v \leq 1 \quad (10-45)$$

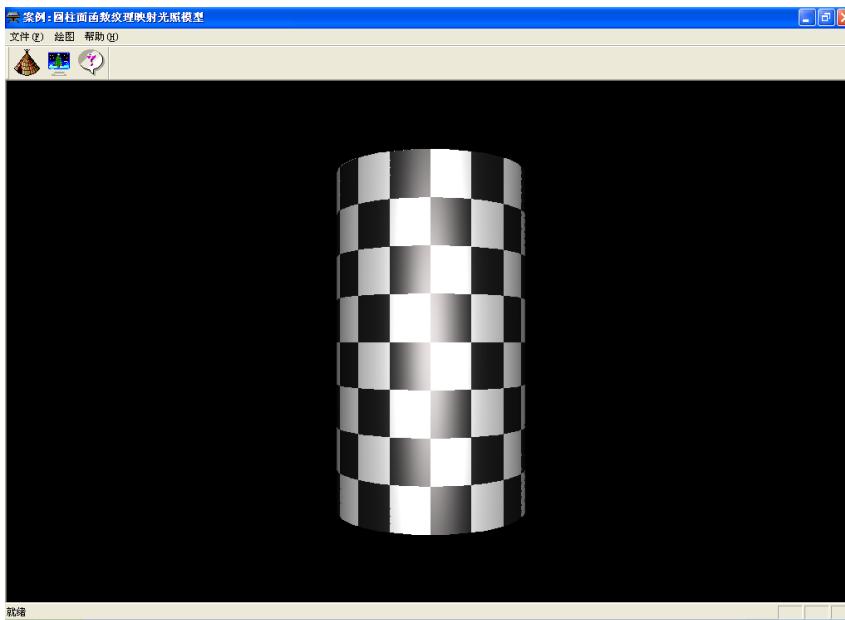


图10-45 圆柱面函数纹理映射

## (5) 圆锥面函数纹理映射

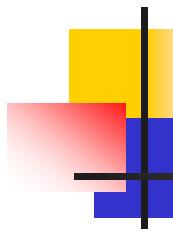
高度为 $h$ , 底面半径为 $r$ , 三维坐标系原点位于底面中心的圆锥面的参数方程为

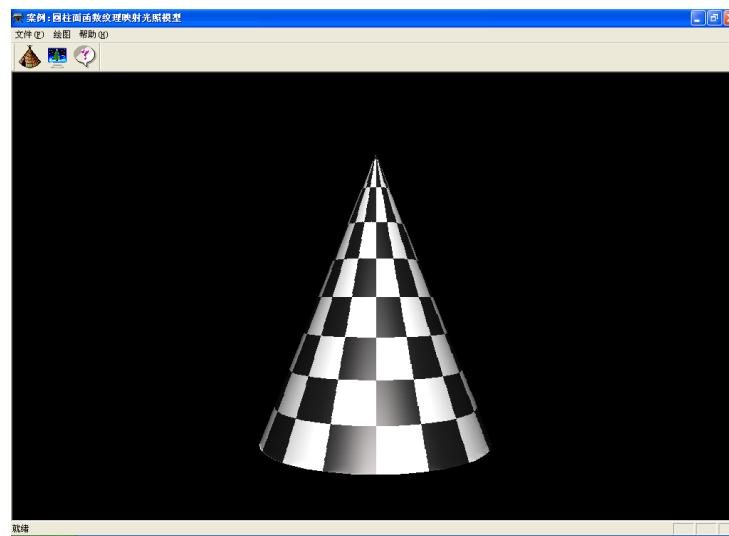
$$\begin{cases} x = \left(1 - \frac{y}{h}\right)r \cos \theta \\ y = h\varphi & 0 \leq \theta \leq 2\pi \quad 0 \leq \varphi \leq 1 \\ z = \left(1 - \frac{y}{h}\right)r \sin \theta \end{cases} \quad (10-46)$$

圆锥面侧面展开图是扇形, 通过下述线性变换将纹理空间 $[0,1] \times [0,1]$ 与物体空间 $[0,2\pi] \times [0,1]$ 等同起来。

$$(u, v) = (\theta / 2\pi, \varphi) \quad (10-47)$$

圆锥面的uv化表示为


$$\left\{ \begin{array}{l} x = (1 - \frac{y}{h})r \cos(2\pi u) \\ y = hv \\ z = (1 - \frac{y}{h})r \sin(2\pi v) \end{array} , 0 \leq u \leq 1, 0 \leq v \leq 1 \right. \quad (10-48)$$



圆锥面函数纹理映射

## (6) 球面函数纹理映射

球心位于三维坐标系原点，半径为r的球面参数方程为

$$\begin{cases} x = r \sin \alpha \sin \beta \\ y = r \cos \alpha \\ z = r \sin \alpha \cos \beta \end{cases} \quad (0 \leq \alpha \leq \pi, 0 \leq \beta \leq 2\pi) \quad (10-49)$$

球面是二次曲面，通过下述线性变换将纹理空间  $[0,1] \times [0,1]$  与物体空间  $[0,2\pi] \times [0,\pi]$  等同起来。

$$(u, v) = (\beta / 2\pi, \alpha / \pi) \quad (10-50)$$

球面的uv化表示为

$$\begin{cases} x = r \sin(\pi v) \sin(2\pi u) \\ y = r \cos(\pi v) \\ z = r \sin(\pi v) \cos(2\pi u) \end{cases}, 0 \leq u \leq 1, 0 \leq v \leq 1 \quad (10-51)$$

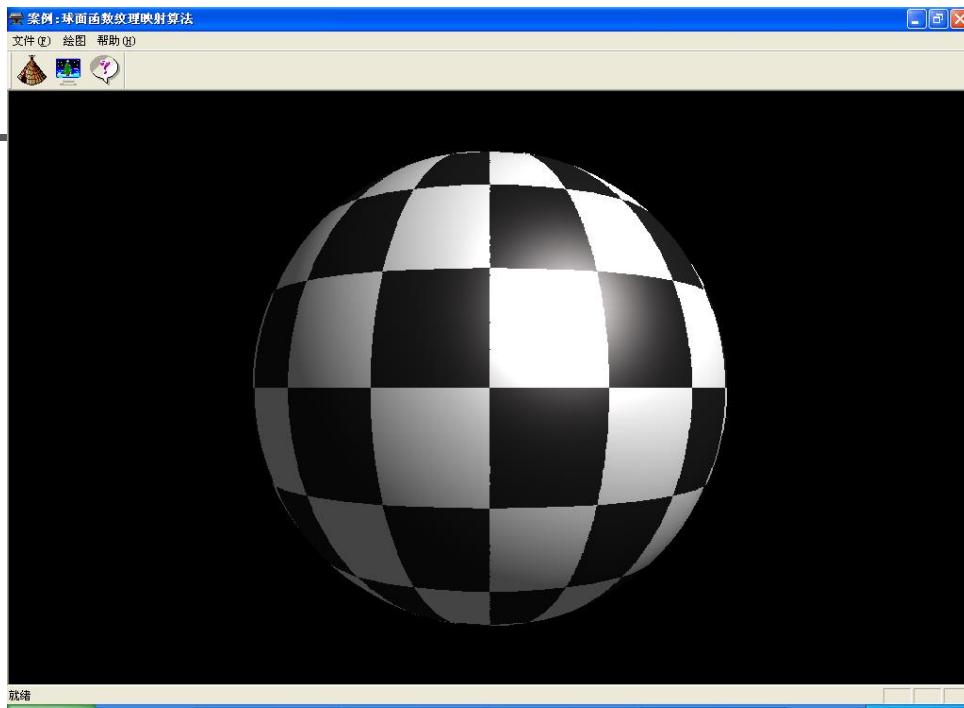


图10-47 球面函数纹理映射

## (7) 圆环面函数纹理映射

圆环中心位于三维坐标系原点，半径为 $r_1$ 和 $r_2$ 的圆环面的参数方程为

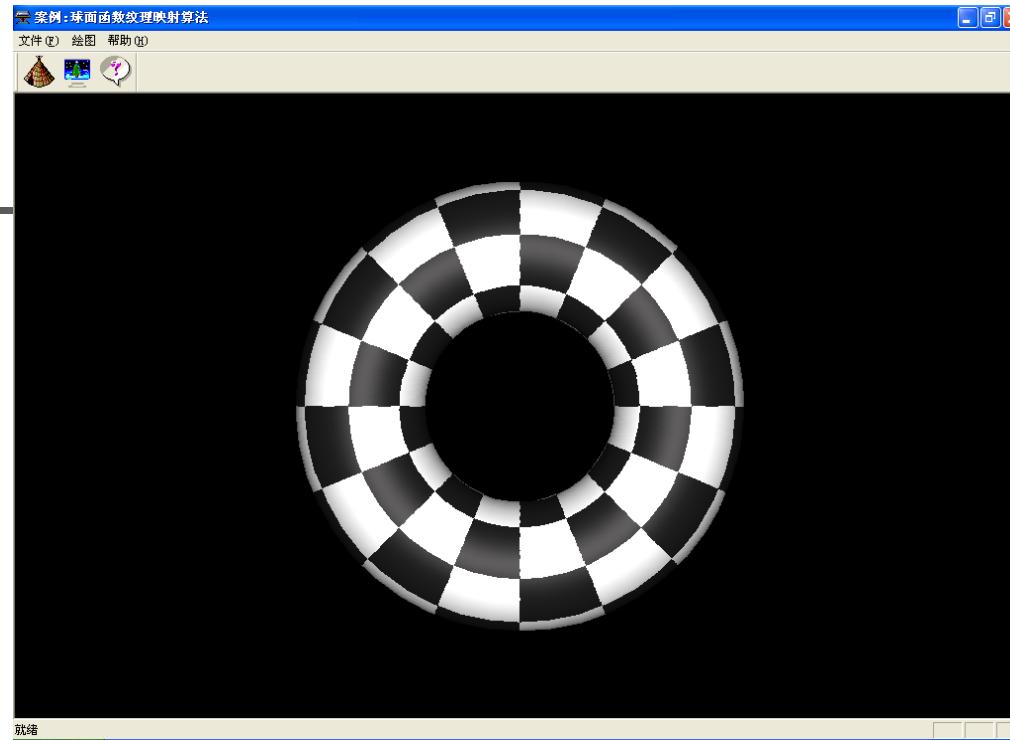
$$\begin{cases} x = (r_1 + r_2 \sin \beta) \sin \alpha \\ y = r_2 \cos \beta \\ z = (r_1 + r_2 \sin \beta) \cos \alpha \end{cases} \quad 0 \leq \alpha \leq 2\pi, 0 \leq \beta \leq 2\pi \quad (10-52)$$

圆环面是二次曲面，通过下述线性变换将纹理空间 $[0,1] \times [0,1]$ 与物体空间 $[0,2\pi] \times [0,2\pi]$ 等同起来。

$$(u, v) = (\beta / 2\pi, \alpha / 2\pi) \quad (10-53)$$

圆环面的uv化表示为

$$\begin{cases} x = (r_1 + r_2 \sin(2\pi u)) \sin(2\pi v) \\ y = r_2 \cos(2\pi u) \\ z = (r_1 + r_2 \sin(2\pi u)) \cos(2\pi v) \end{cases}, 0 \leq u \leq 1, 0 \leq v \leq 1 \quad (10-54)$$



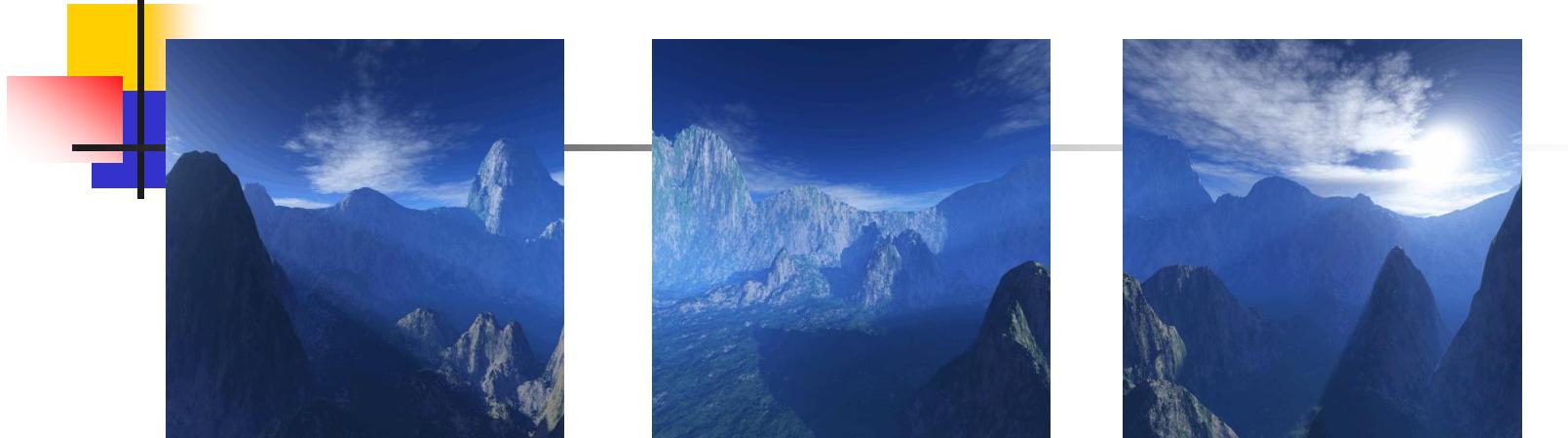
## 圆环面函数纹理映射

## 2. 图像纹理

函数纹理是使用数学方法定义的简单二维纹理图案，这种纹理规则而单调。为了增强纹理的表现力，一个自然的想法是将一幅来自数码相机的二维图像作为纹理映射到物体上。图像纹理映射需要建立物体表面上每一采样点与已知图像上各点（称为纹素，**texel**）的对应关系，取图像上点的颜色值作为物体表面上采样点的颜色值，然后采用光照模型来计算该点处的光强。

图像纹理映射既可以采用将图像纹理绑定到物体顶点上的方式实现也可以采用将图像纹理绑定到表面上的方式实现。前者一般用于映射单幅图像，仅需要正确处理图像接缝，后者一般用于映射多幅图像。

对于立方体，由于使用6幅图像纹理，一般采用绑定到表面上的方式实现。读入立方体6个表面所对应的位图的代码如下：



back

front

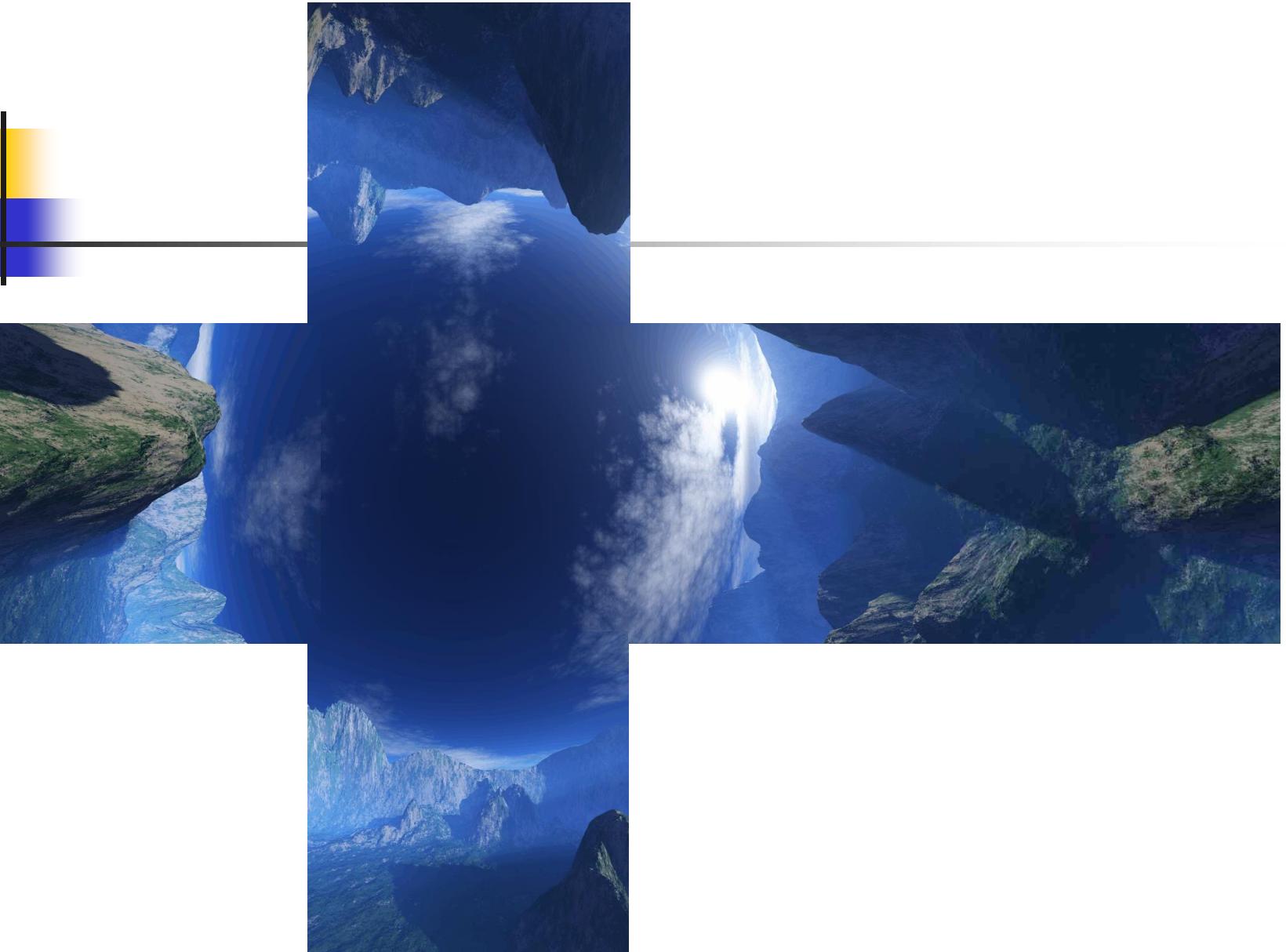
right

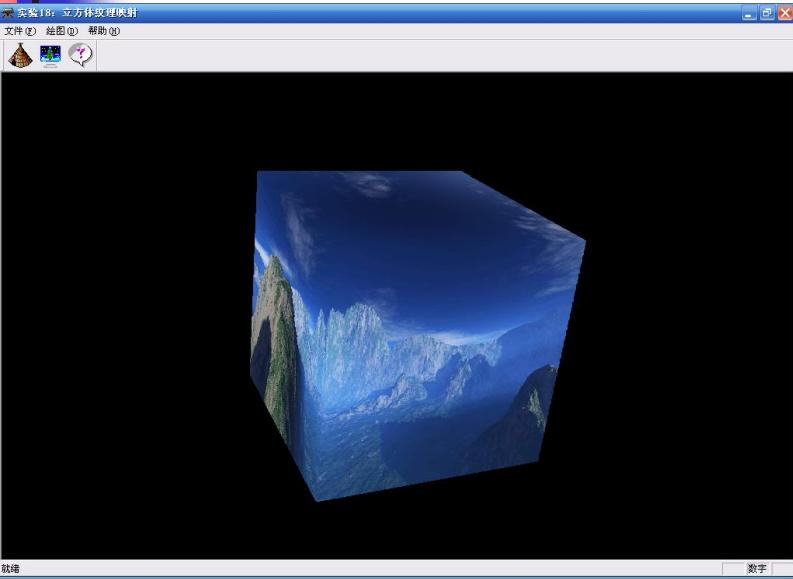


left

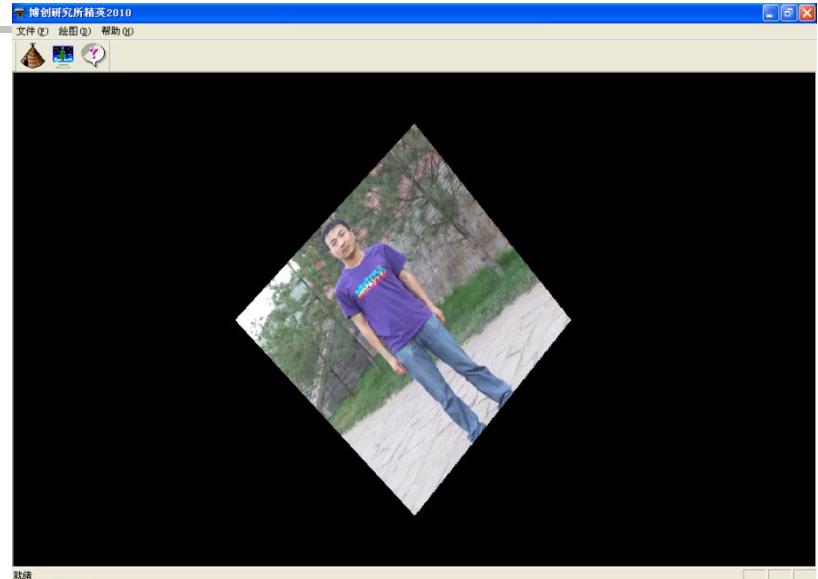
bottom

top





立方体图像纹理映射效果图



八面体贴图

圆柱面、圆锥面、球面和圆环面仅使用一幅位图就可以实现图像纹理映射，一般采用绑定到物体顶点上的方式实现。

圆柱面图像纹理映射效果如图所示。圆锥面图像纹理映射效果如图所示。球面图像纹理映射效果如图所示。圆环面图像纹理映射效果如图所示。

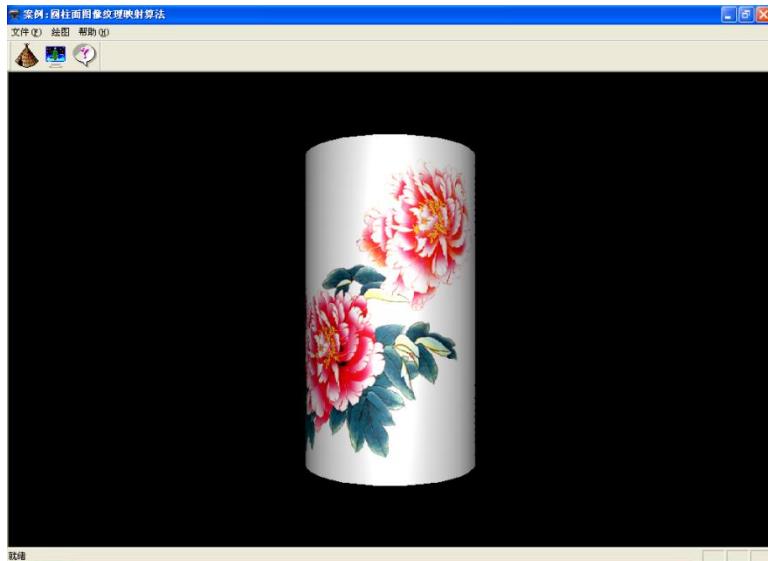


图 圆柱面图像纹理映射

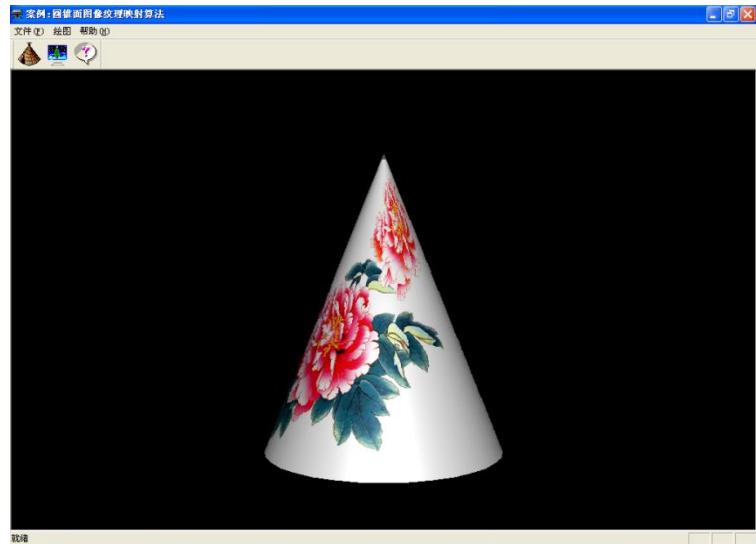


图 圆锥面图像纹理映射

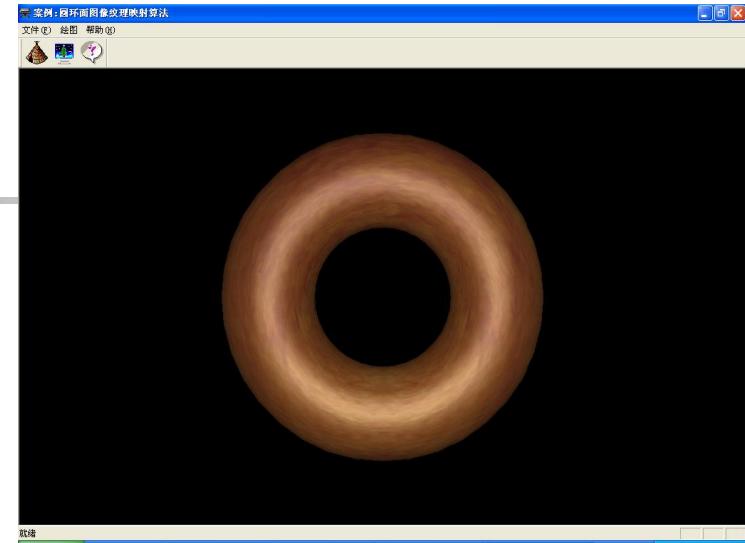
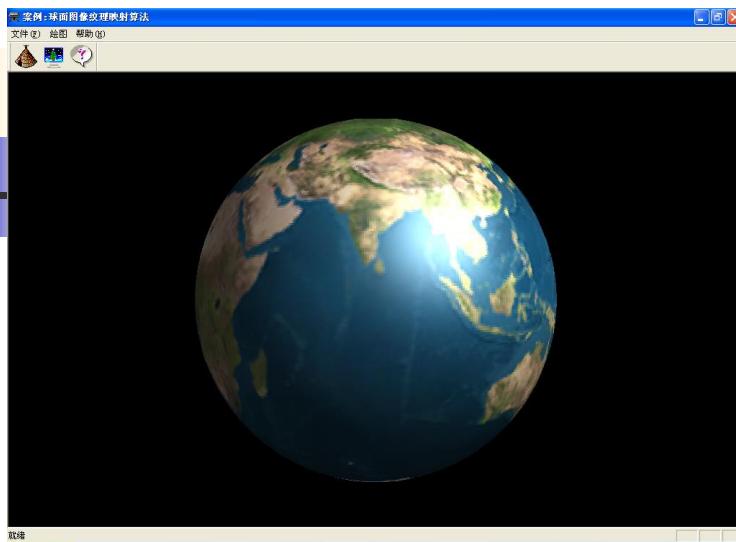


图 球面图像纹理映射  
(未进行纹理反走样处理)

图 圆环面图像纹理映射

## 7.6.2 三维纹理

由于纹理是二维的，而物体是三维的，很难在物体表面的连接处做到纹理自然过渡，极大地降低了图形的真实感。假如在三维物体空间中，物体上的每一个点 $P(x,y,z)$ 均有一个纹理值 $t(x,y,z)$ ，其值由纹理函数惟一确定。那么对于物体上的空间点，就可以映射到一个定义了纹理函数的三维空间上了。由于三维纹理空间与物体空间维数相同，在进行纹理映射时，只需把场景中的物体变换到纹理空间即可。

1985年，Peachey用一种简单的规则三维纹理函数首次成功地模拟了木制品的纹理效果。其基本思想是采用一组共轴圆柱面来定义三维纹理函数，即把位于相邻圆柱面之间的纹理函数值交替地取为明和暗。这样物体上任一点的纹理函数值可根据它到圆柱轴线所经过的圆柱面个数的奇偶性而取为明和暗。

上述定义的木纹函数过于规范，  
Peachey引入了三个简单的操作来克服  
这一缺陷：

- 扰动（perturbing）对共轴圆柱面的半径进行扰动。
- 扭曲（twisting）在圆柱轴向加一个扭曲量。
- 倾斜（tilting）将圆柱面圆心沿木块的截面倾斜。

取共轴圆柱面的轴向为y轴，横截面为x和z轴，如图10-56所示。则对于半径为 $r_1$ 的圆柱，参数方程为

$$r_1 = \sqrt{x^2 + z^2}$$

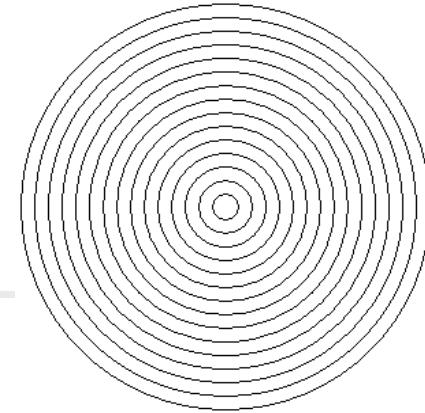


图10-55 共轴圆柱面的横截面

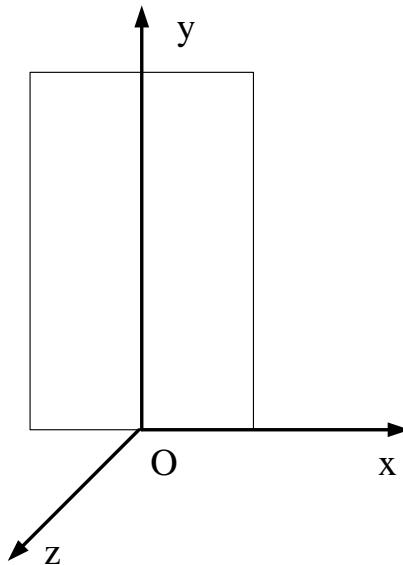


图10-56 共轴圆柱面坐标系  
115

若使用 $2\sin a\theta$ 作为木纹的不规则生长扰动函数，并在y轴方向附加的  $\frac{y}{b}$  扭曲量，得到

$$r_2 = r_1 + 2 \sin(a\theta + \frac{y}{b}) \quad (10-55)$$

式中：a, b为常数，  $\theta = arctg(\frac{x}{z})$

上式即为原半径 $r_1$ 的圆柱面经变形后为半径 $r_2$ 的表面方程，最后使用三维几何变换将纹理倾斜一个角度。取a=20, b=150时，绘制的光照长方体木纹纹理如图10-57所示。可以看出在公共边界处，木纹纹理具有连续性，这是使用二维纹理技术很难实现的。

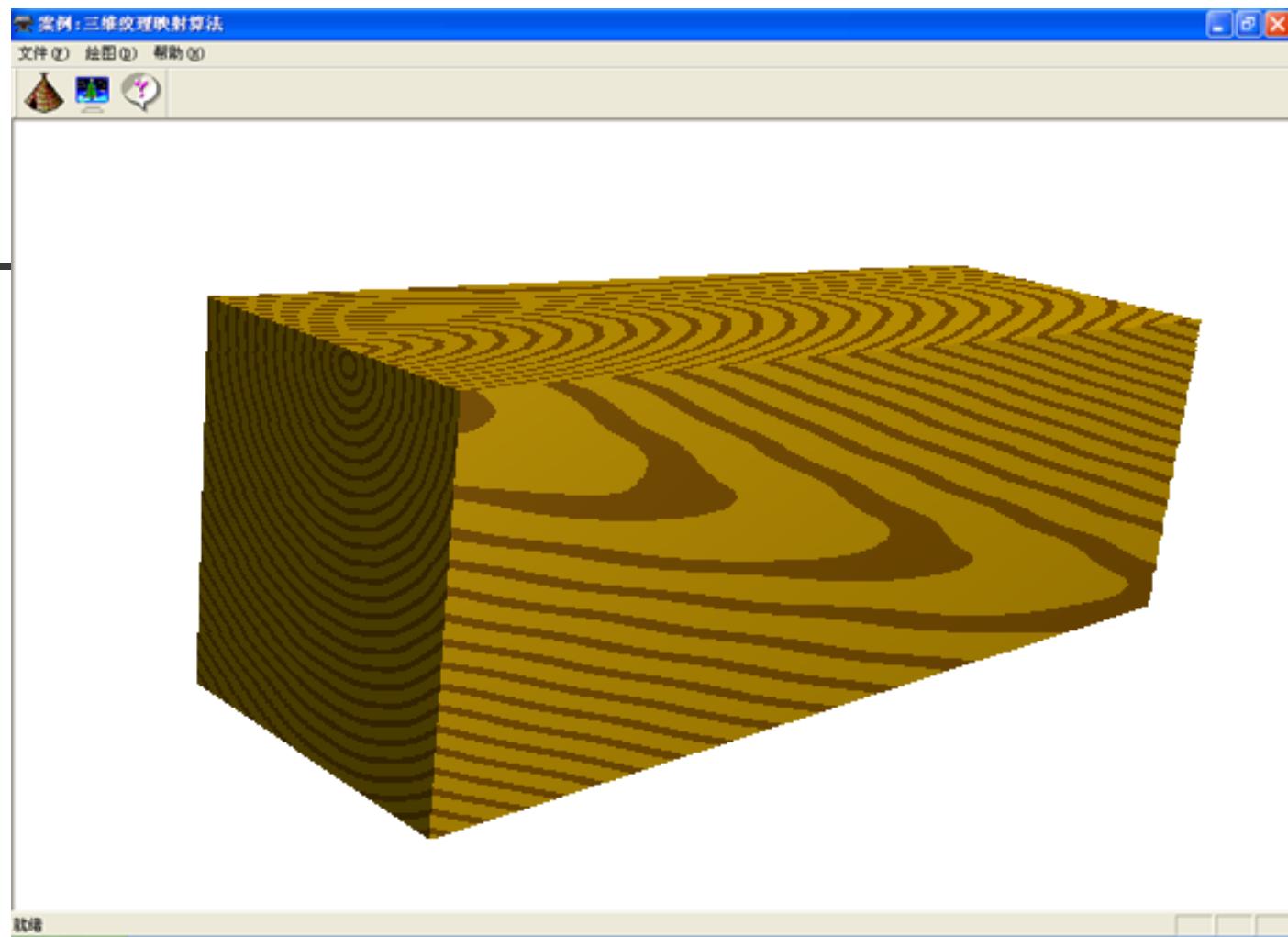


图 光照三维木纹纹理

## 7.6.3 几何纹理

现实世界中还存在另一类物体表面如橘子皮、树皮、混凝土墙面等凹凸不平的表面。虽然可以将上述位图作为颜色纹理映射到相应的几何表面以增加真实感，但却无法表达表面的凹凸不平。

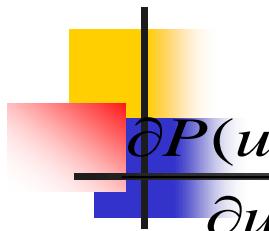
凹凸纹理（**bump map**）的基本思想是用简单光照模型计算物体表面的光强时，对物体表面的法向进行微小的扰动，导致表面光强的突变，产生凹凸不平的真实感效果。

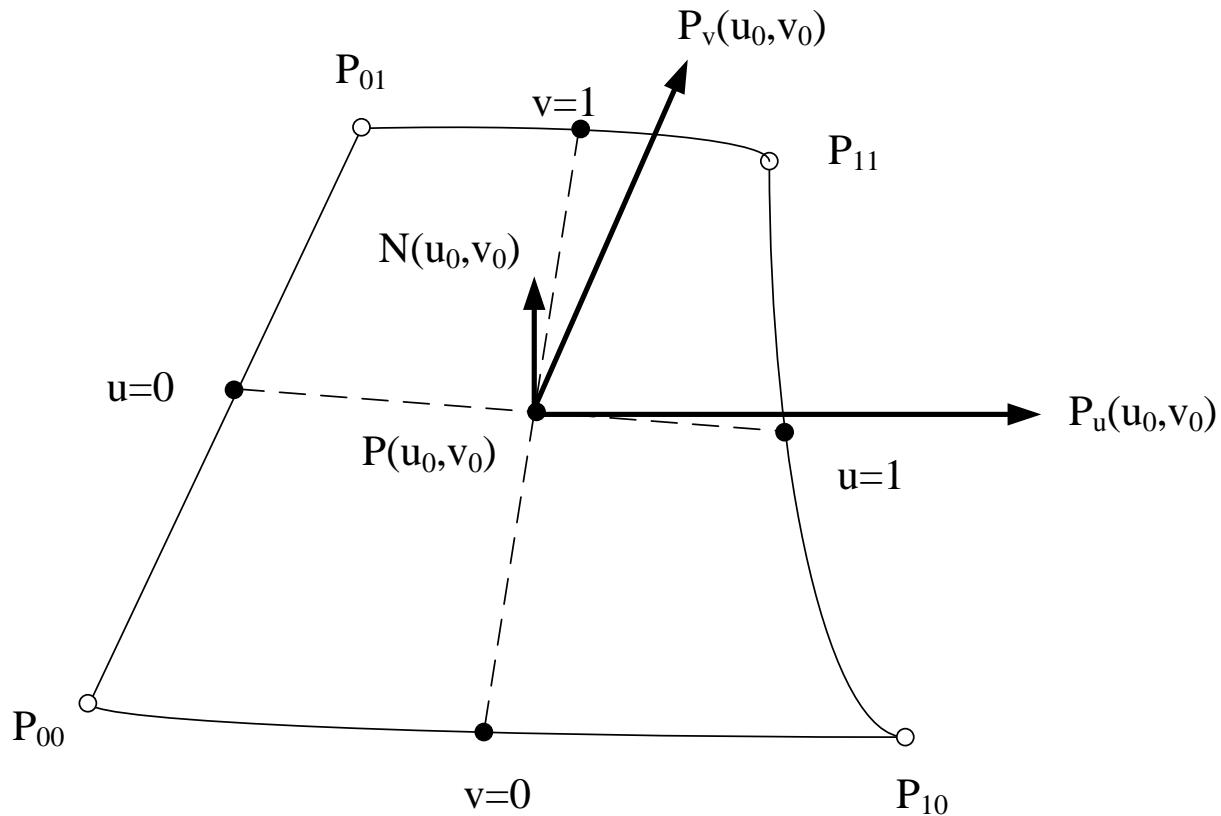
### 1. 参数曲面的定义

一张定义在矩形域上的参数曲面可以表示为

$$\begin{cases} x = x(u, v) \\ y = y(u, v) \\ z = z(u, v) \end{cases}, 0 \leq u \leq 1, 0 \leq v \leq 1$$

假定曲面上的点为  $P(u_0, v_0)$ , 该点的切矢量为

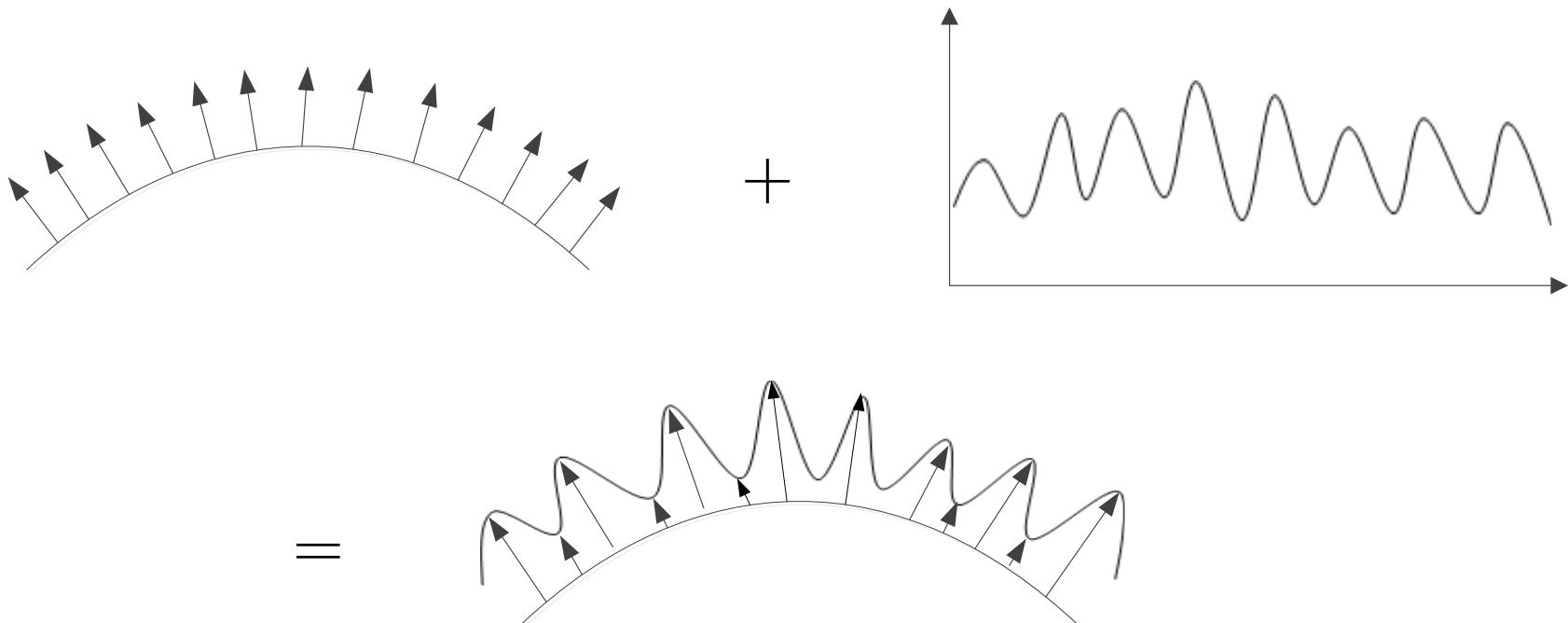

$$\left. \frac{\partial P(u, v)}{\partial u} \right|_{\substack{u=u_0 \\ v=v_0}} \text{ 和 } \left. \frac{\partial P(u, v)}{\partial v} \right|_{\substack{u=u_0 \\ v=v_0}}, \text{ 法矢量为 } \left. \frac{\partial P(u, v)}{\partial u} \right|_{\substack{u=u_0 \\ v=v_0}} \times \left. \frac{\partial P(u, v)}{\partial v} \right|_{\substack{u=u_0 \\ v=v_0}}$$



## 2. 映射原理

定义一个扰动函数  $B(u,v)$ ，对理想光滑表面作不规则的位移。  
物体表面上的每一个点  $P(u,v)$  都沿该点处的法矢量方向位移  $B(u,v)$  个单位长度，新的表面位置变为

$$P'(u,v) = P(u,v) + B(u,v) \frac{N}{|N|} \quad (10-55)$$



令

$$n = \frac{N}{|N|}$$

有

$$P' = P + Bn$$

新表面的法矢量

$$N' = P'_u \times P'_v$$

$$P'_u = \frac{\partial(P + Bn)}{\partial u} = P_u + B_u n + B n_u$$

$$P'_v = \frac{\partial(P + Bn)}{\partial v} = P_v + B_v n + B n_v$$

由于粗糙表面的凹凸高度相对于表面尺寸一般要小的多，因而B很小，可以忽略不计。

$$N' \approx (P_u + B_u n) \times (P_v + B_v n)$$

$$N' \approx P_u \times P_v + B_u (n \times P_v) + B_v (P_u \times n) + B_u B_v (n \times n)$$

由于  $n \times n = 0$ , 且  $N = P_u \times P_v$

$$N' \approx N + B_u (n \times P_v) + B_v (P_u \times n) \quad (10-60)$$

令

$$A = n \times P_v \quad B = n \times P_u$$

$$D = B_u (n \times P_v) - B_v (n \times P_u) = B_u A - B_v B$$

则扰动后的法矢量为

(10-61)

$$N' = N + D$$

第一项为原光滑表面上任意一点的法矢量, 而第二项为扰动矢量。这意味着, 光滑表面的法矢量**N**在u和v方向上被扰动函数**B**的偏导数所修改, 得到**N'**。将法矢量**N'**规范化为单位矢量, 可以用于计算物体表面的光强, 以产生**貌似**凹凸不平的效果。

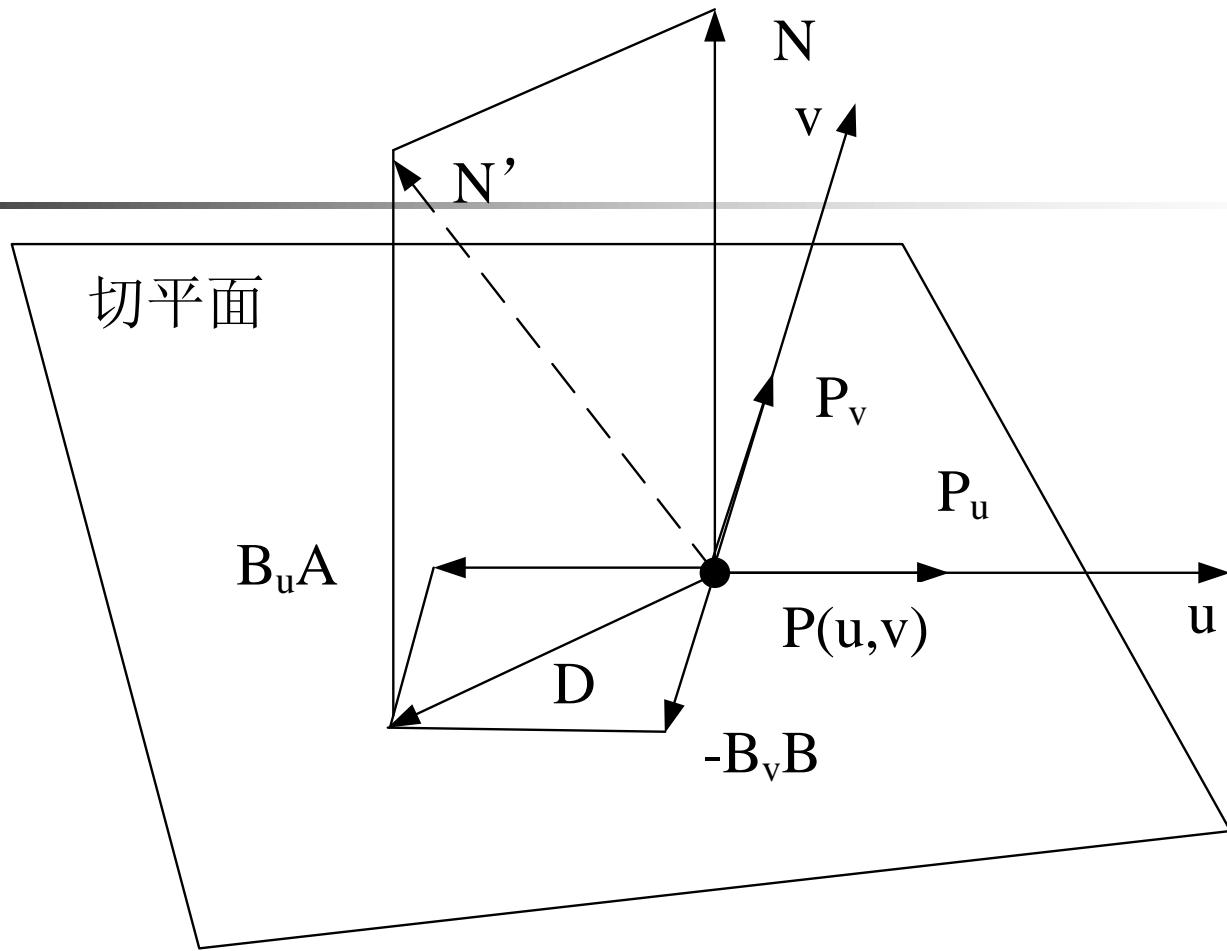
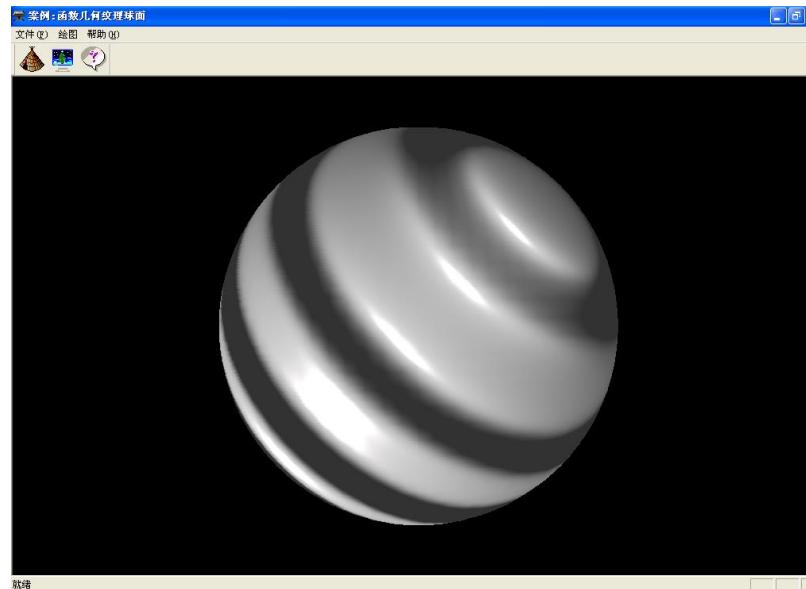


图10-59 法矢量扰动的几何关系

### 3. 几何纹理的分类

#### (1) 偏移矢量凹凸纹理

偏移矢量凹凸纹理 (offset vector bump map) 技术是使用函数，如正弦函数，来定义  $B_u$  和  $B_v$ 。使用  $B_u$  和  $B_v$  值对小面顶点的法向进行扰动，小面内的法矢量使用双线性插值计算。



图函数几何纹理球面

## (2) 高度场凹凸纹理

高度场凹凸纹理 (height field bump map) 的  $B_u$  和  $B_v$  是使用灰度图像定义的。灰度图像中白色纹理表示高的区域，黑色纹理表示低的区域。高度场中的  $B_u$  和  $B_v$  需要使用中心差分计算，相邻列的差得到  $B_u$ ，相邻行的差得到  $B_v$ 。

$$\begin{cases} B_u = P(x_i + 1, y_i) - P(x_i - 1, y_i) \\ B_v = P(x_i, y_i + 1) - P(x_i, y_i - 1) \end{cases} \quad (10-62)$$

由于 Blinn 方法不计算扰动后物体表面各顶点的新位置，而是直接计算扰动后表面新的法矢量，该方法难以在物体的轮廓线上表现出凹凸不平的效果。为此 Cook 采用位移映射 (displacement mapping) 技术来克服上述缺陷。位移映射方法通过沿表面法向扰动物体表面上个顶点的位置来模拟表面的粗糙不平的效果。

# 博创研究所

11 精心 精业 精品 20

高度场位图

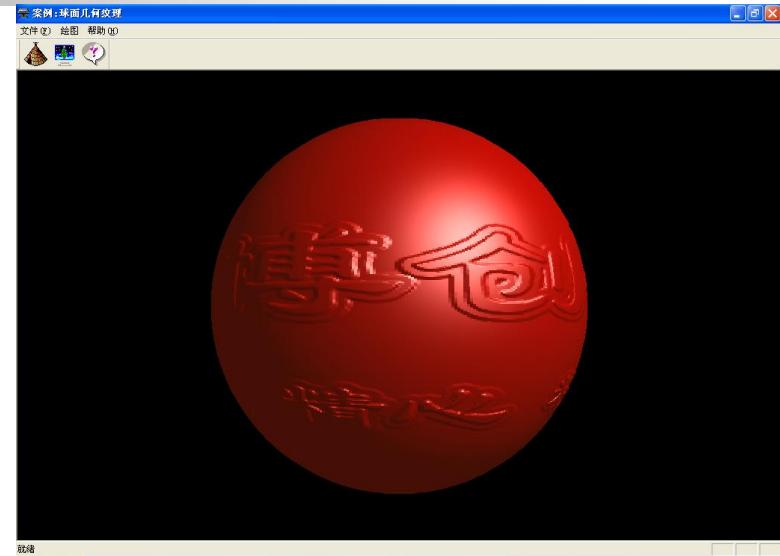
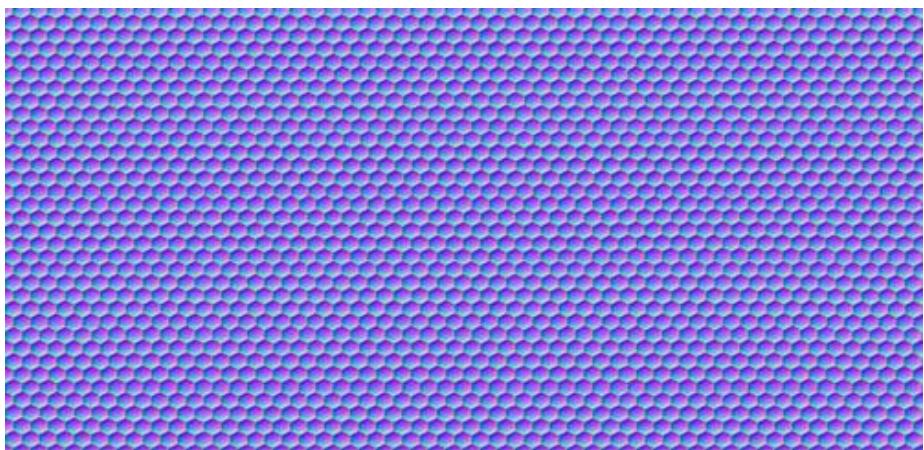
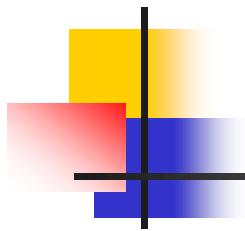
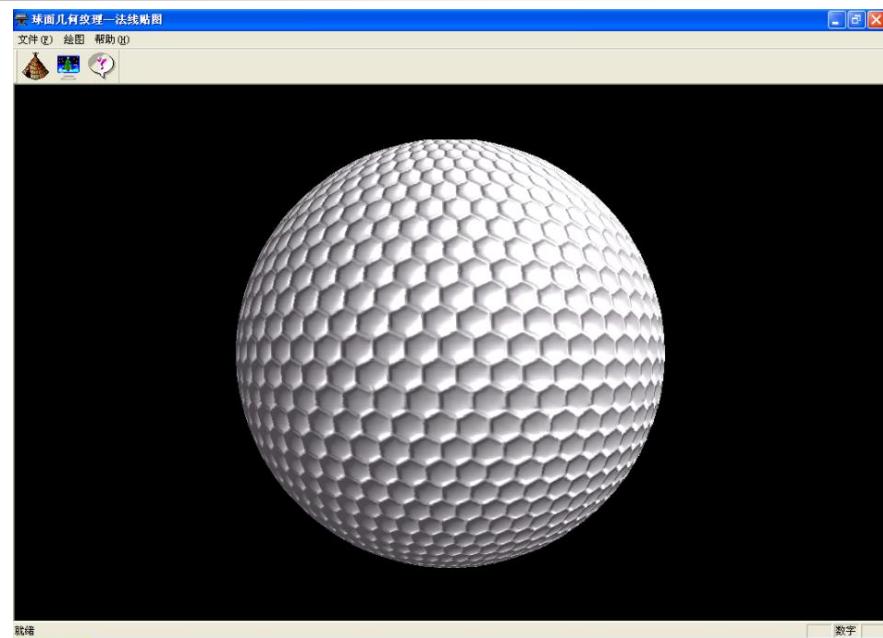


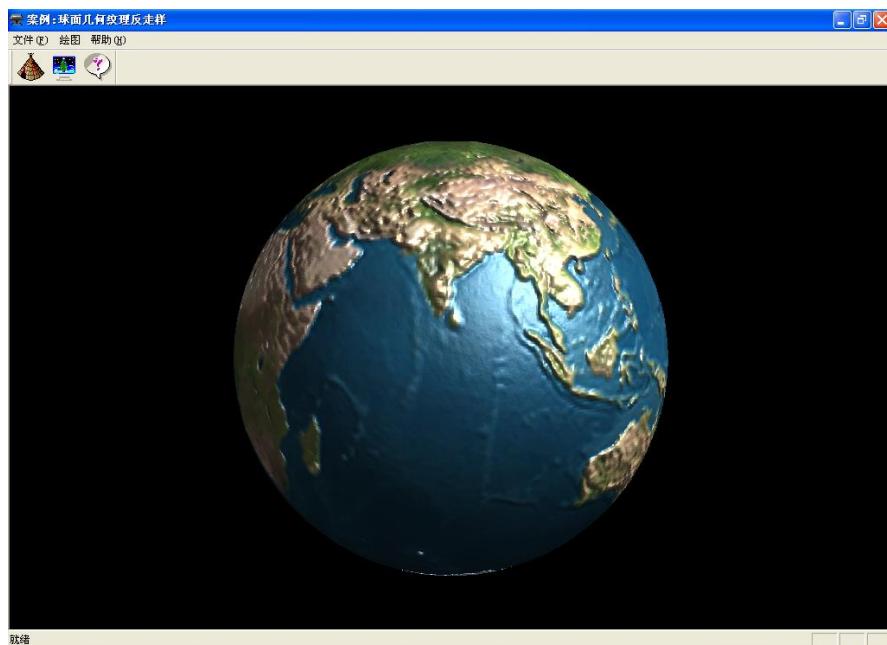
图10-61 高度场凹凸纹理



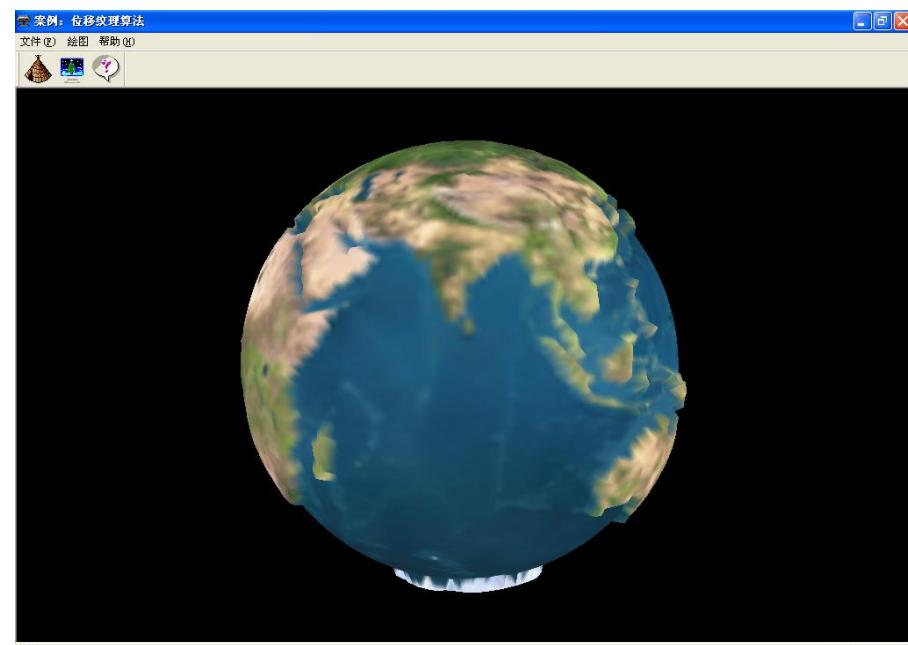
法线纹理



高尔夫球凹凸纹理

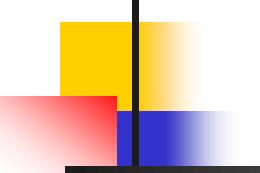


(a) 凹凸纹理映射

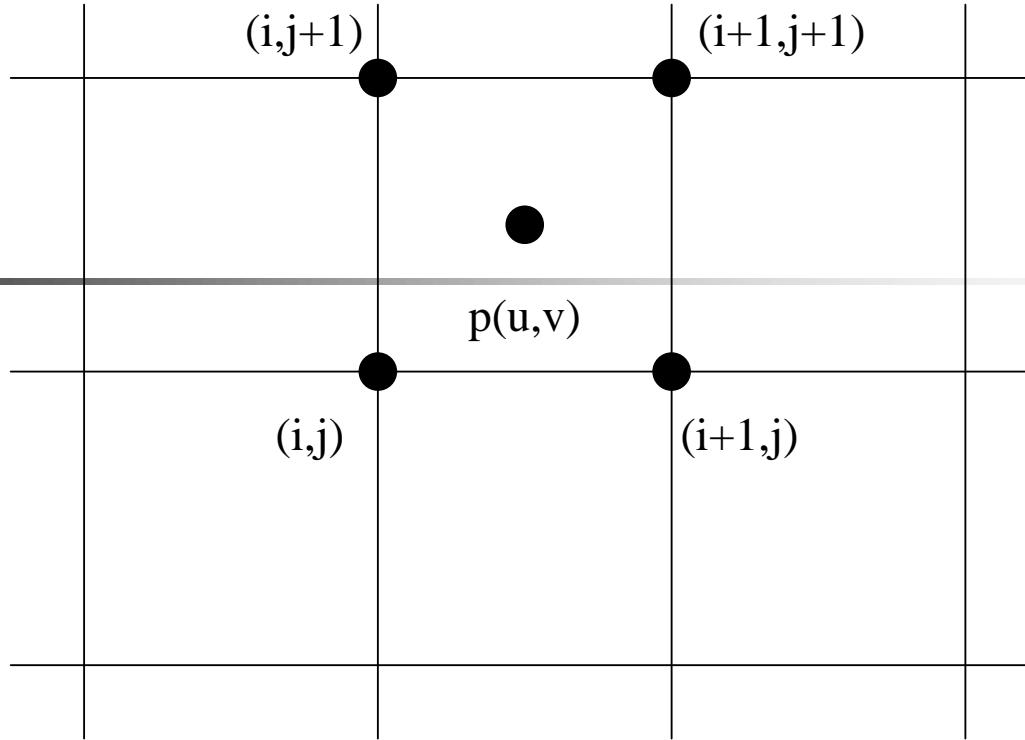


(b) 位移纹理映射

## 7.6.4 纹理反走样



纹理映射是将纹理图案映射到不同大小的物体表面上。若投影得到的象素数目比原始纹理大，则需要把纹理图像放大；若投影得到的象素数目比原始纹理小，则需要把纹理图像缩小。对于立方体、圆柱等物体，当纹理图案与屏幕像素之间相匹配时，可以实现一对一的映射。但是将二维纹理映射到曲面物体上时，如球面、圆环面等，会产生严重的走样。可以借助于不同的反走样技术予以改善，其中最简单的纹理反走样技术是双线性内插法。



$$f(u, v) = (1 - u)(1 - v)f(i, j) + (1 - u)v f(i, j + 1)$$

$$+ u(1 - v)f(i + 1, j) + uv f(i + 1, j + 1)$$

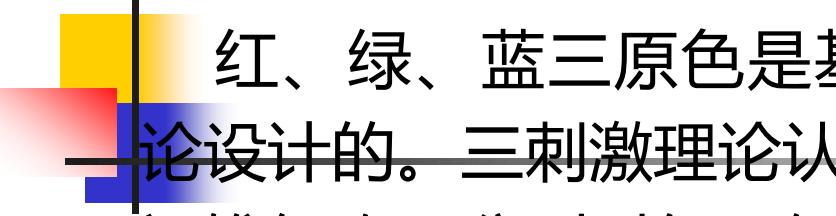


反走样前



反走样后

## 7.7 颜色模型



红、绿、蓝三原色是基于人眼视觉颜色感知的三刺激理论设计的。三刺激理论认为，人眼的视网膜中有三种类型的视锥细胞，分别对红、绿、蓝三种色光最敏感。人眼光谱灵敏度实验曲线证明，这些光在波长为700nm（红色）、546 nm（绿色）和435.8 nm（蓝色）时的刺激点达到高峰。三原色有这样的两个性质：(1) 三原色中的任意两种原色的组合都得不到第三种原色；(2) 通过三原色的混合可以得到可见光谱中的任何一种颜色。

计算机图形学中常用的颜色模型有RGB颜色模型、HSV颜色模型和CMYK颜色模型等。其中颜色模型RGB和CMYK是最基础的模型，其余的颜色模型在显示时都需要转换为RGB模型，在打印或印刷时都需要转换为CMYK模型。

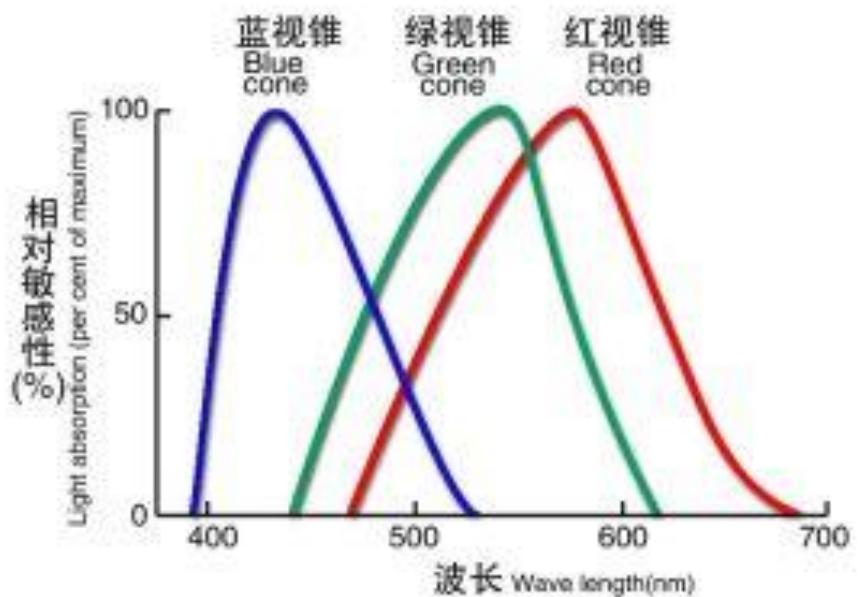
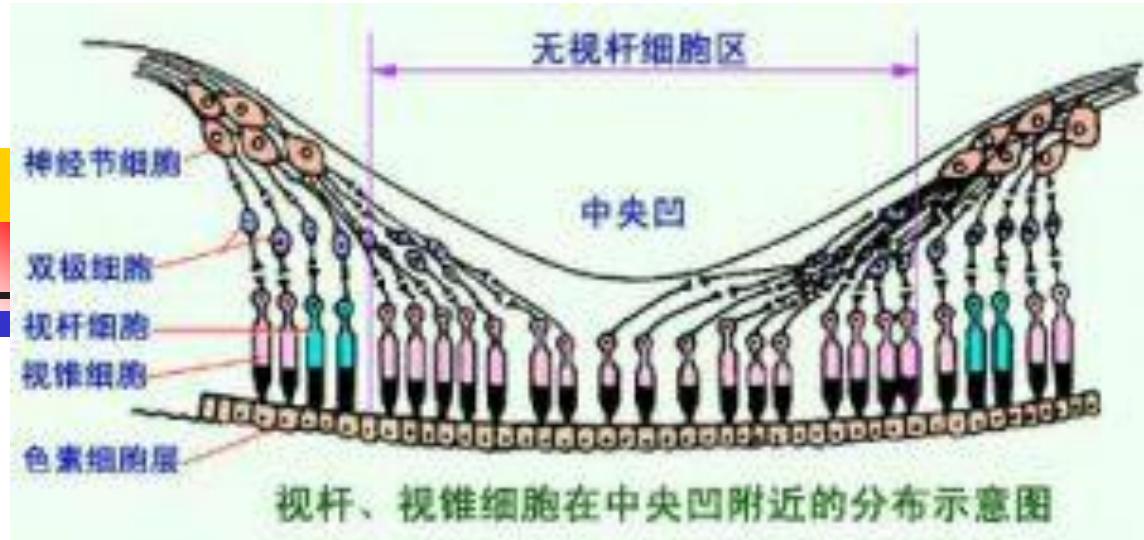
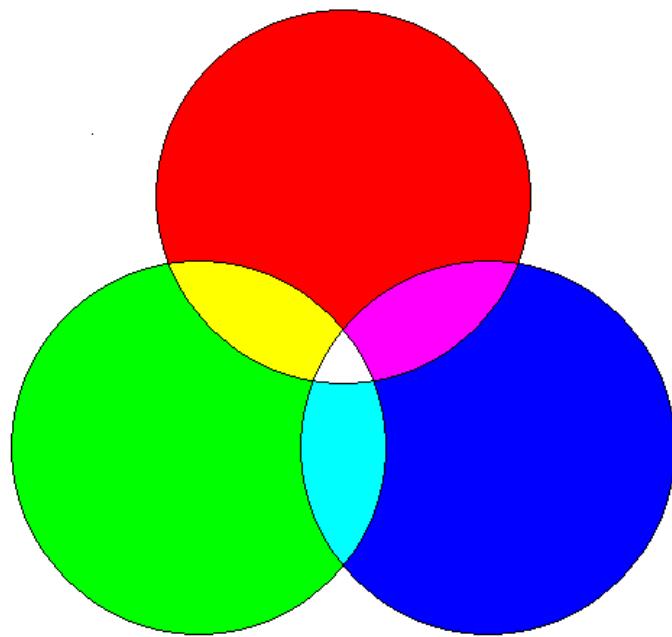
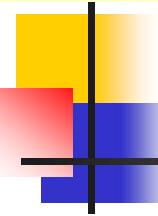


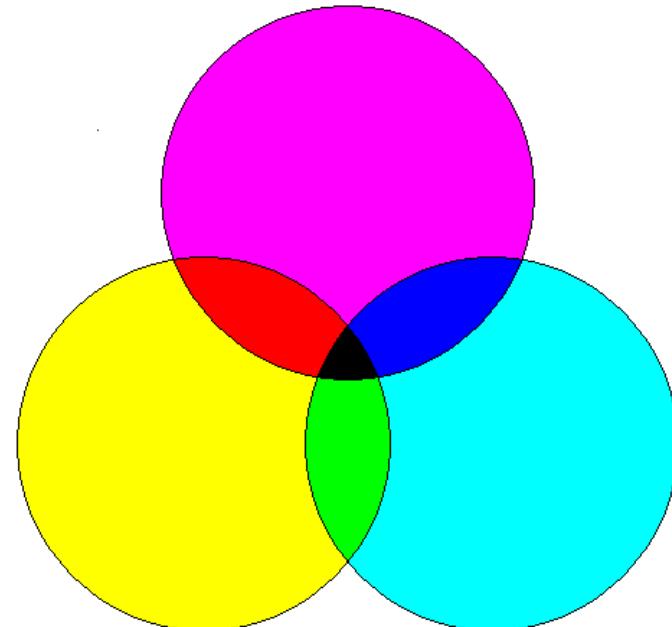
图 - 人视网膜中三种不同视锥细胞的光谱相对敏感性

视网膜存在两种感光细胞：视锥细胞与视杆细胞。视锥细胞在中央凹分布密集，而在视网膜周边区相对较少。视锥细胞对强光敏感。视杆细胞在中央凹处无分布，主要分布在视网膜的周边部，视杆细胞对暗光敏感。

## 7.7.1 原色系统

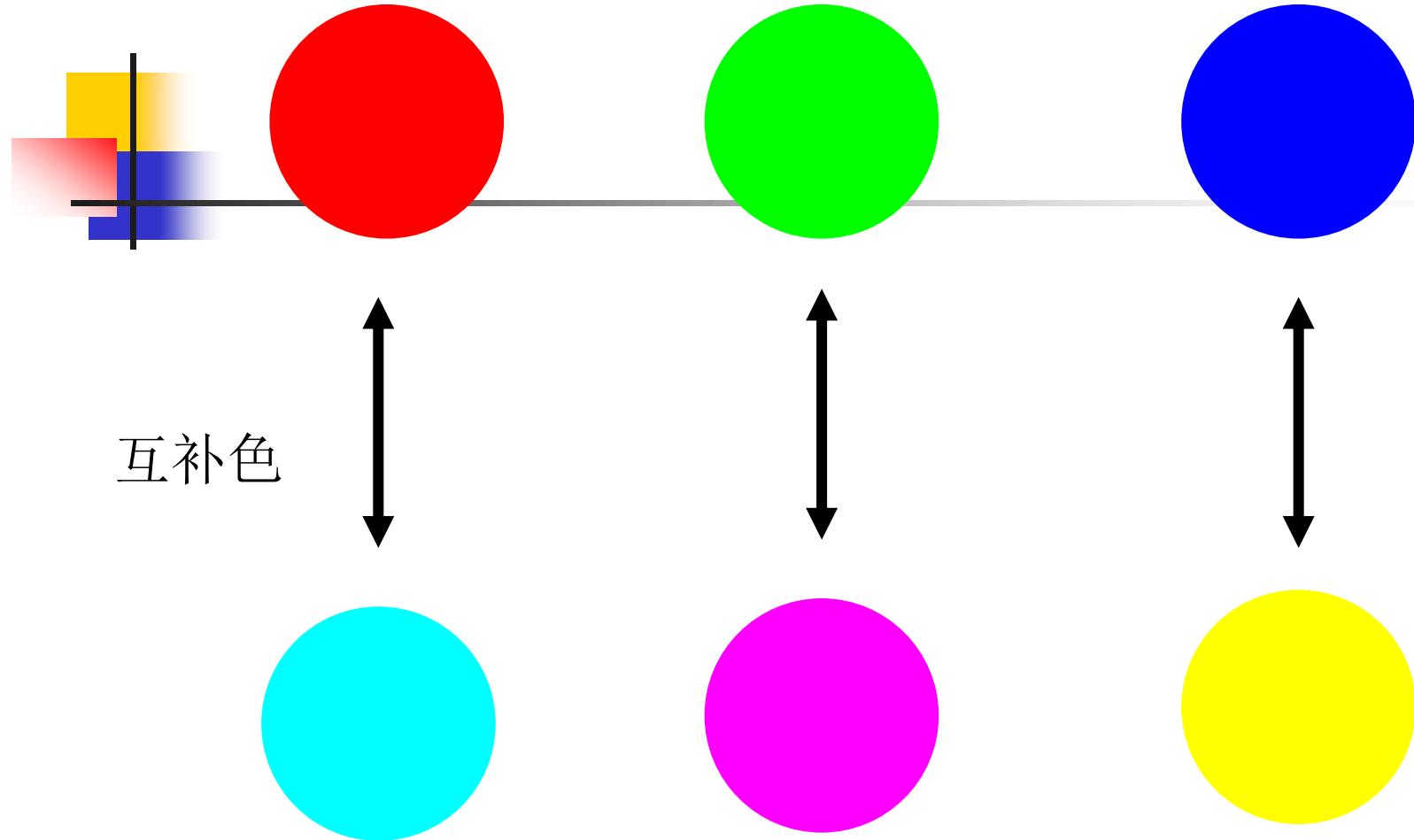


(a) RGB加色系统



(b) CMY减色系统

图 原色系统



互补色指完全不包含另一种颜色

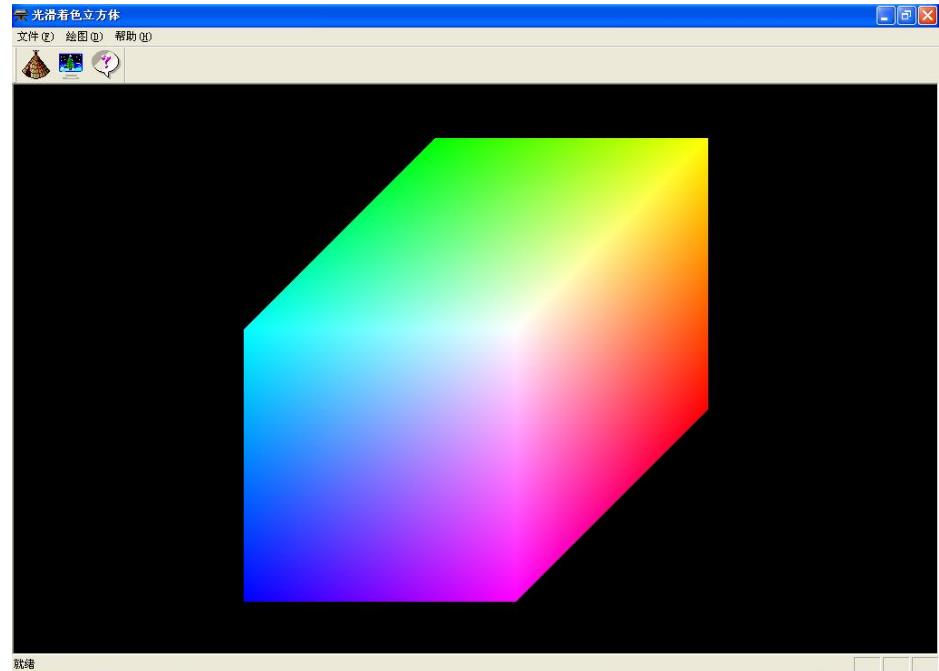
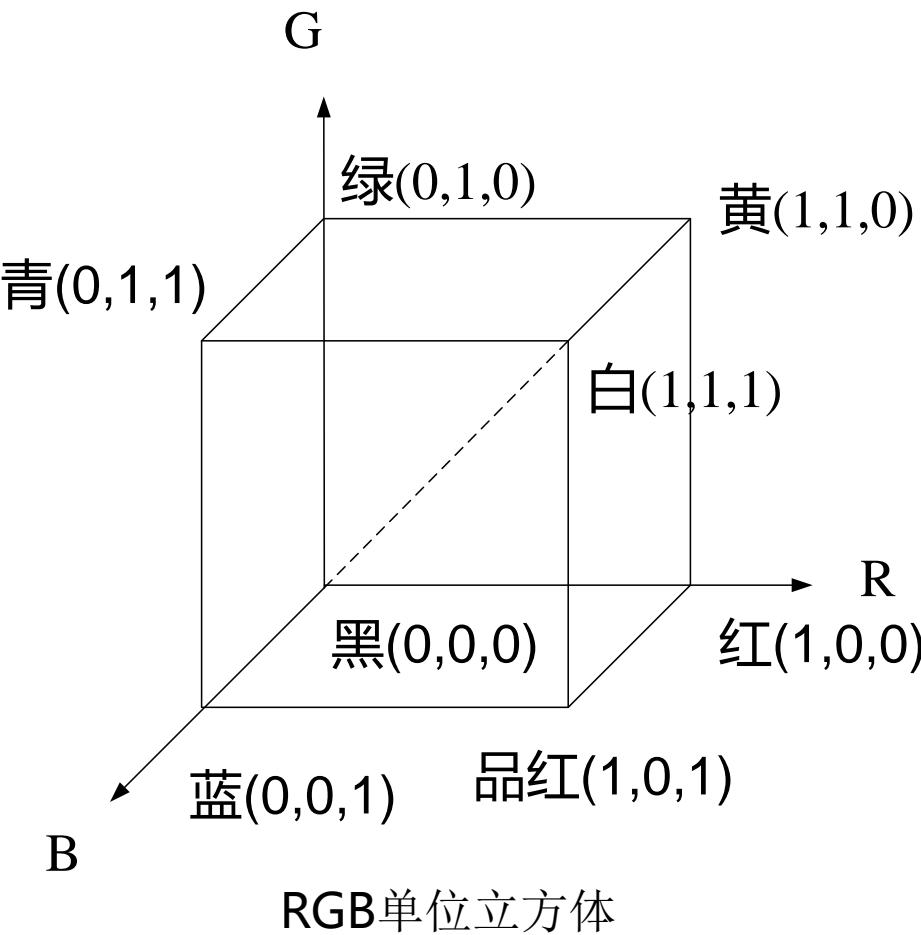
对于发光体使用的是RGB加色系统，对于反射体使用的是CMY减色系统。

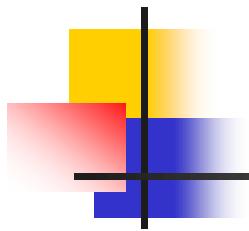
**加色系统**中，通过对颜色分量的叠加产生新颜色。红色和绿色等量叠加成为黄色，红色和蓝色等量叠加成为品红；绿色和蓝色等量叠加成为青色；如果红色、绿色和蓝色等量叠加，则成为白色。

**减色系统**中，通过消除颜色分量来产生新颜色。当在纸面上涂上品红油墨时，该纸面就不反射绿光；当在纸面上涂上黄色油墨时，该纸面就不反射蓝光；当在纸面上涂上青色油墨时，该纸面就不反射红光；如果在纸面上涂上了品红油墨、黄色油墨和青色油墨，那么所有的红光、绿光和蓝光都被吸收，纸面呈现黑色。

## 7.7.2 RGB颜色模型

RGB颜色模型是显示器的物理模型，无论软件开发中使用何种颜色模型，只要是绘制到显示器上，图像最终是以RGB颜色模型表示的。





```
class CRGBA
{
public:
    CRGBA ();
    virtual ~CRGBA ();
public:
    double red;           //红色分量
    double green;          //绿色分量
    double blue;           //蓝色分量
    double alpha;          //alpha分量
};
}
```

分量取值范围 [0,1]

## 7.7.3 HSV颜色模型



HSV颜色模型是一种直观的颜色模型，包含三个要素：色调（hue）、饱和度（saturation）和明度（value）。色调H是一种颜色区别于其它颜色的基本要素，如红、橙、黄、绿、青、蓝、紫等，当人们谈论颜色时，实际上是指它的色调，特别地，黑色和白色无色调。饱和度S是指颜色的纯度。没有与任何颜色相混合的颜色，其纯度为全饱和。要想降低饱和度可以在当前颜色中加入白色，鲜红色饱和度高，粉红色饱和度低。明度V是颜色的相对明暗程度。要想降低明度则可以在当前颜色中加入黑色，明度最高得到纯白，最低得到纯黑。

HSV模型是从RGB立方体演化而来。沿RGB立方体的主对角线由白色向黑色看去，在平面上的投影构成一个正六边形，RGB三原色和相应的补色分别位于正六边形的各个顶点上，其中红色、绿色、蓝色分别相隔 $120^\circ$ ，互补色相隔 $180^\circ$ （红色与青色、黄色与蓝色、绿色与品红分别相隔 $180^\circ$ ）

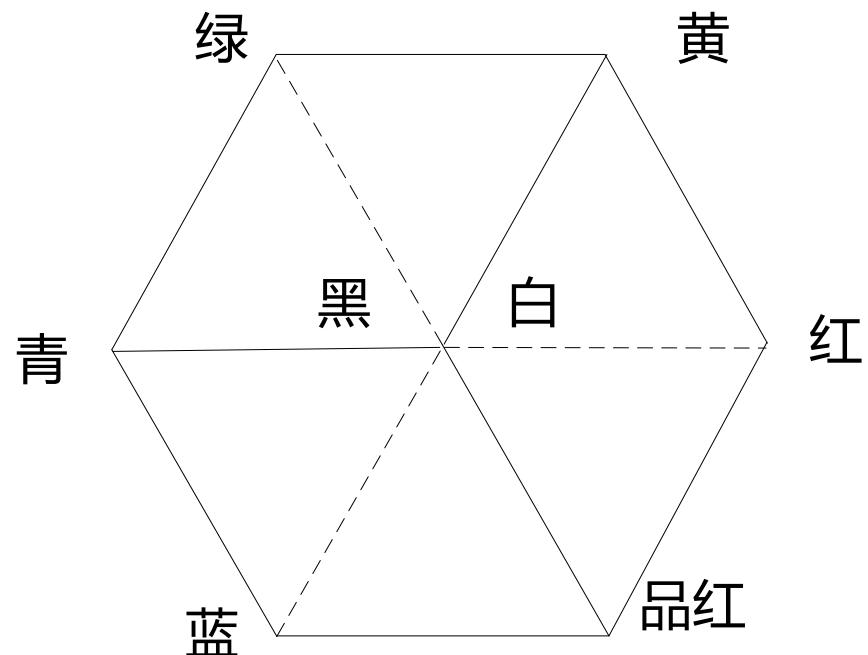
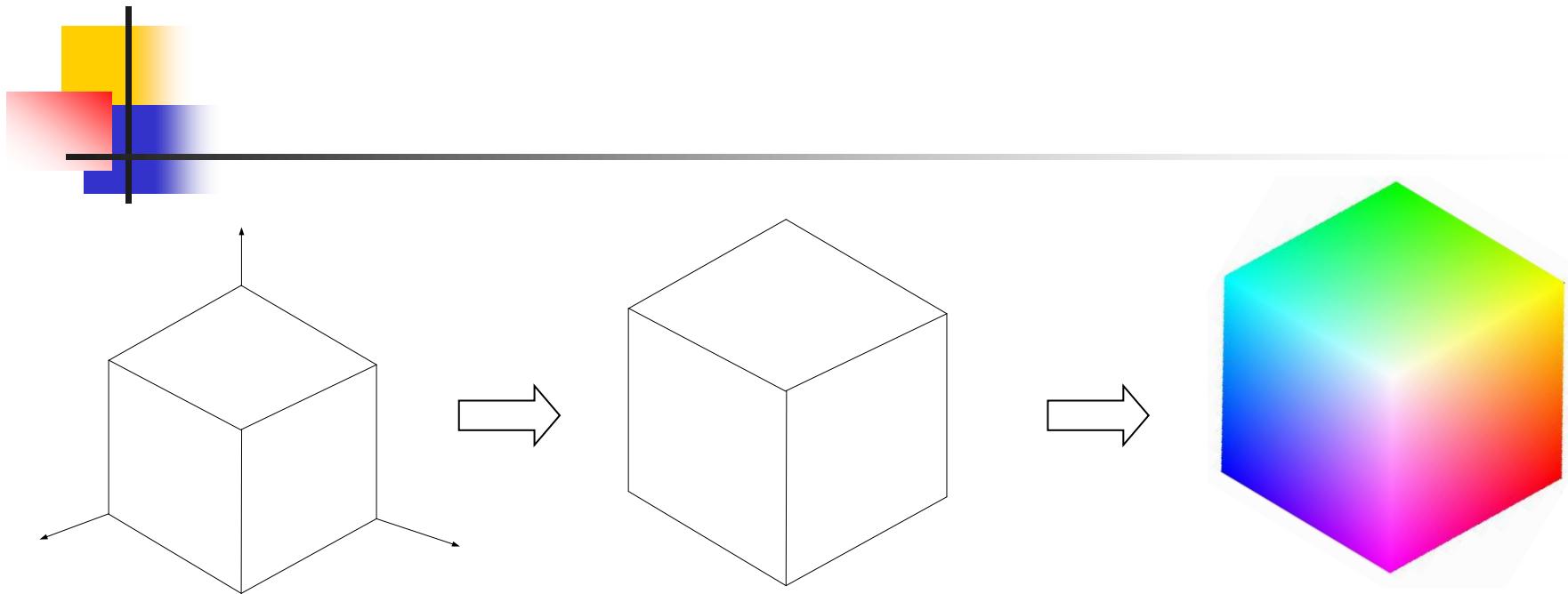


图 HSV正六边形



看做二维图，是  
个正六边形

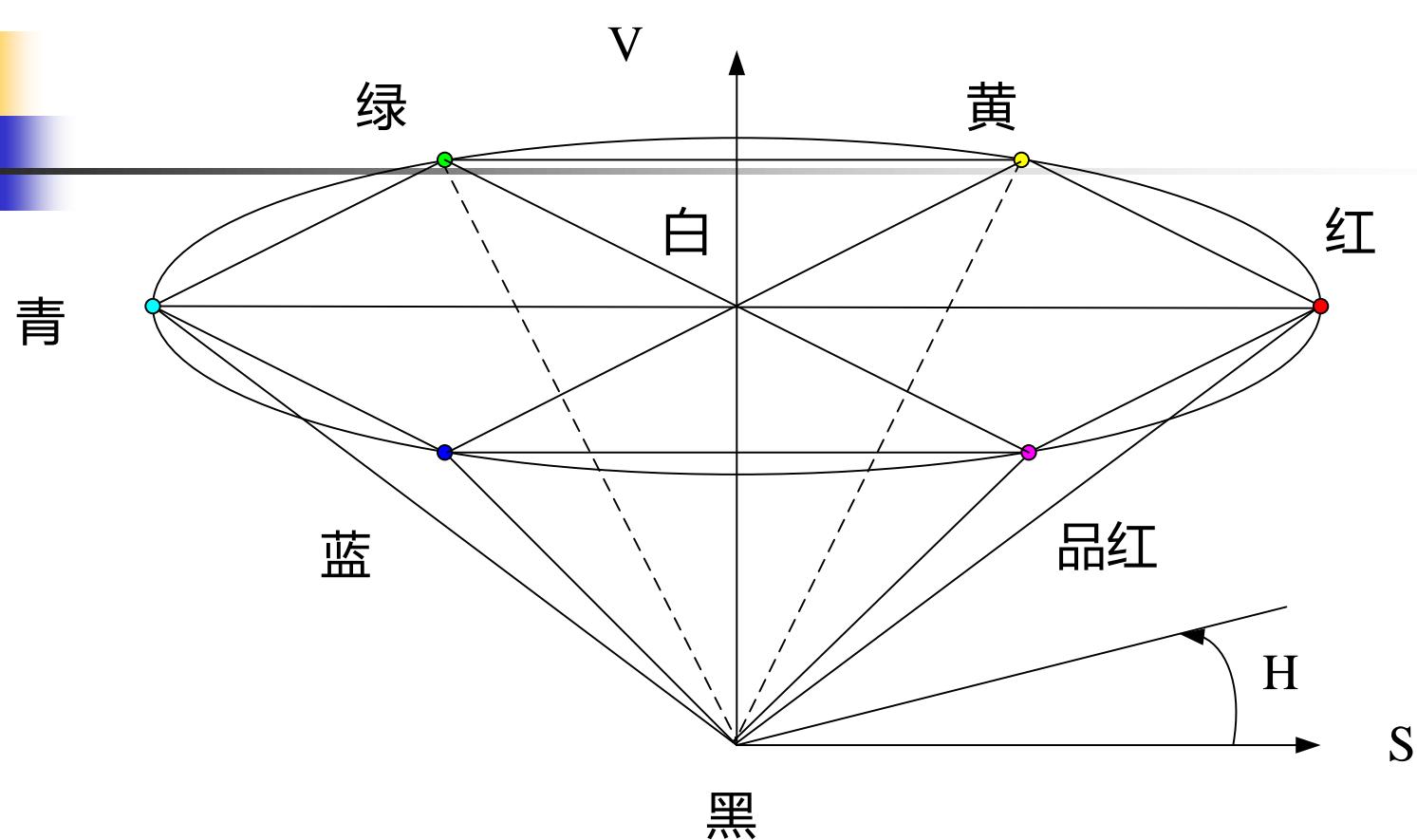
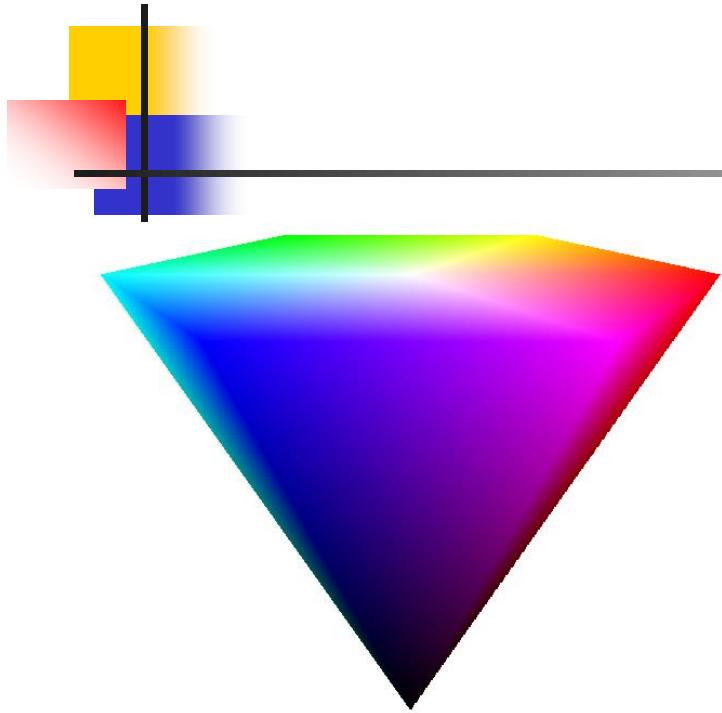
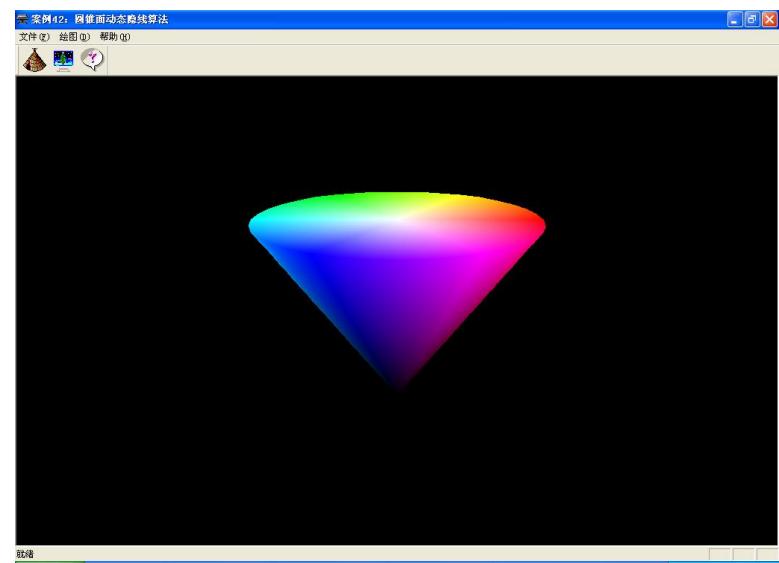


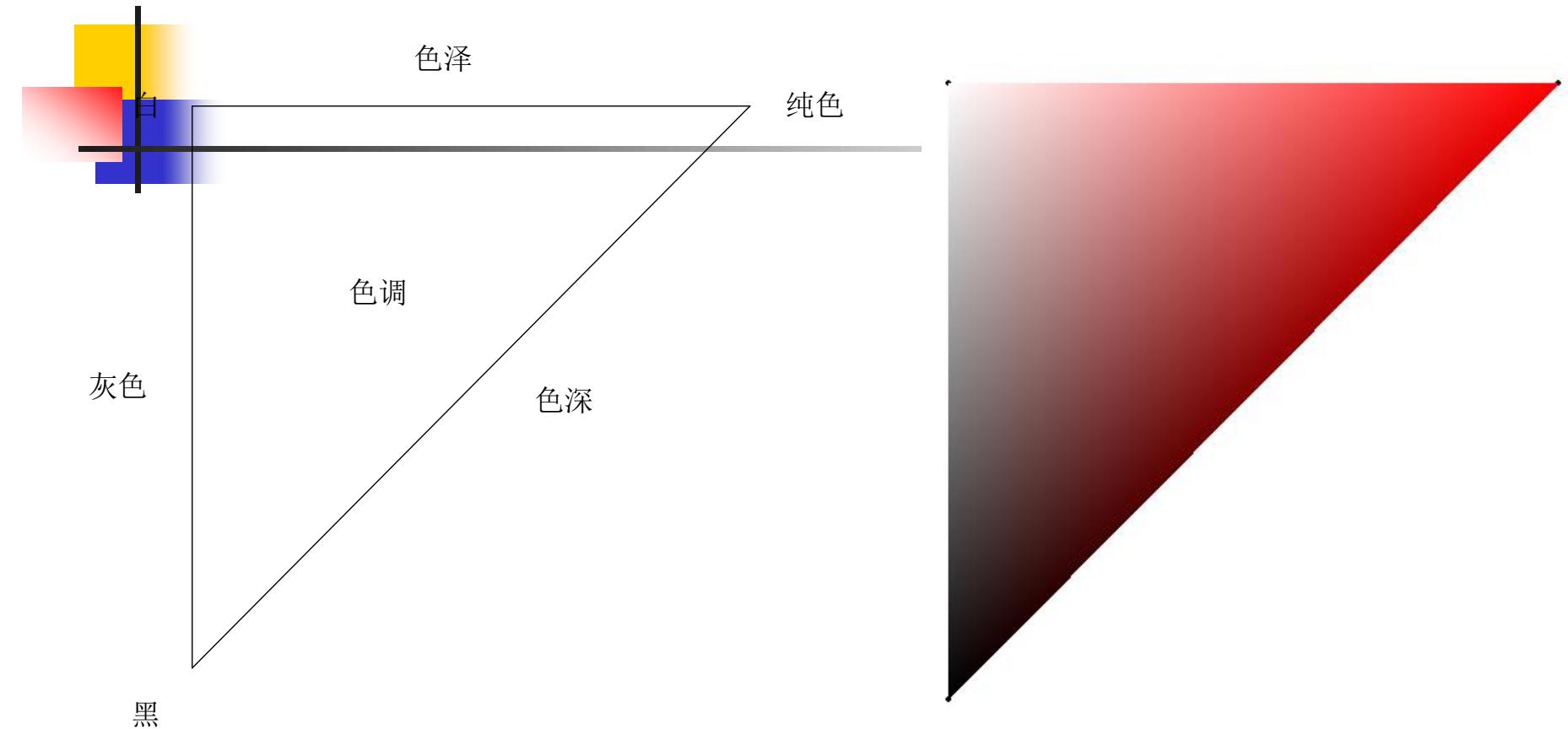
图 HSV 颜色模型



HSV六棱锥

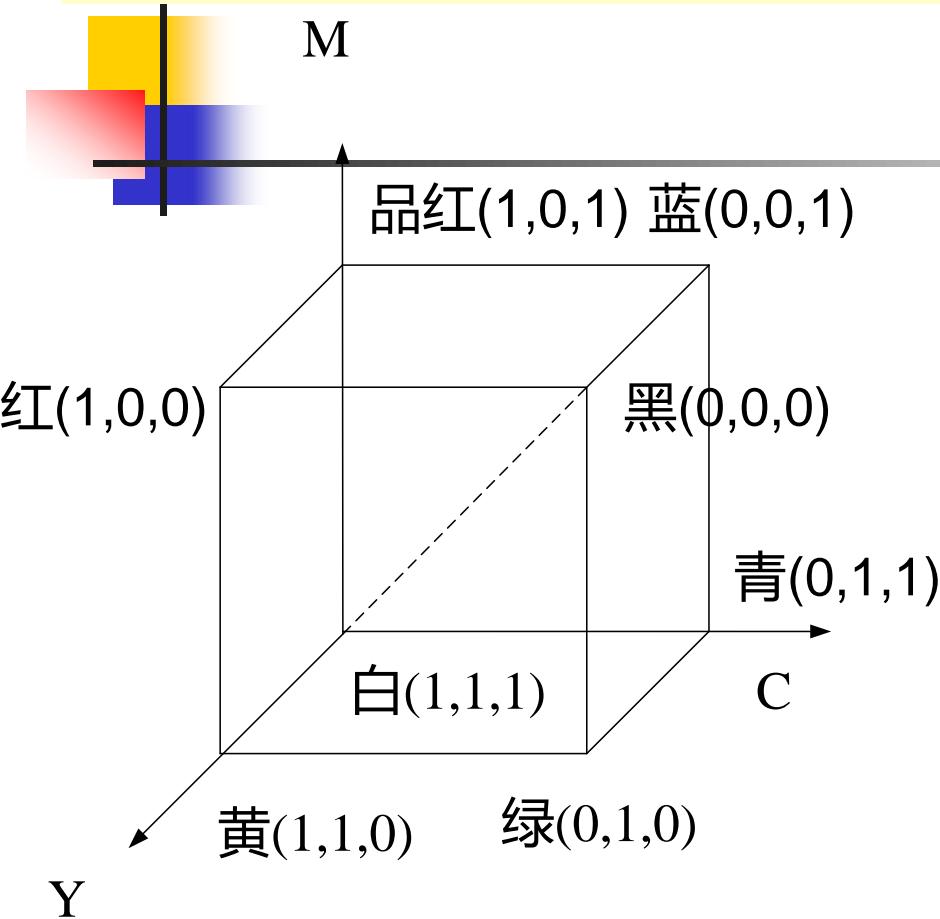


HSV圆锥

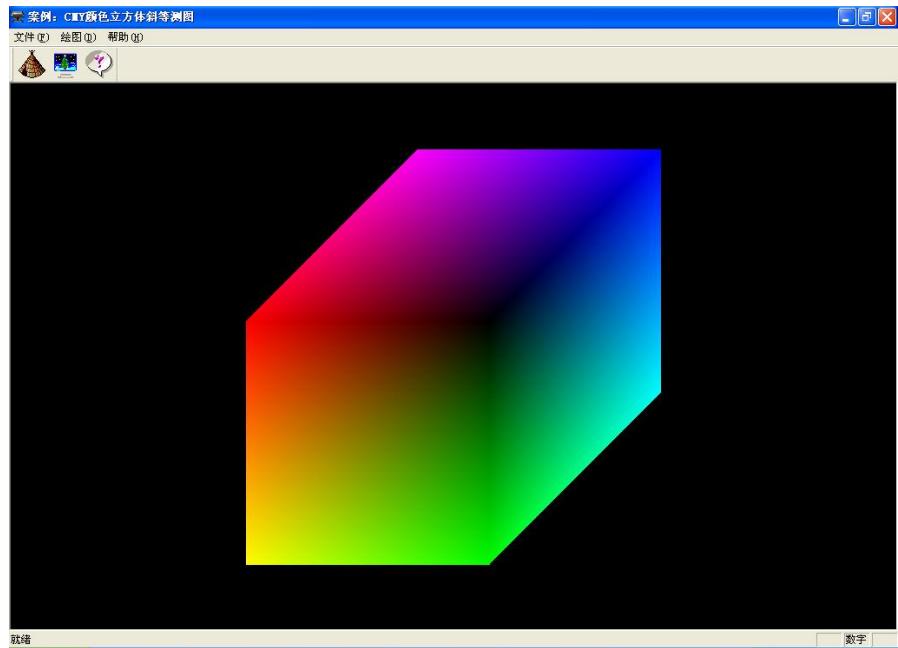


图色泽、色深和色调的关系图

## 7.7.4 CMYK颜色模型



CMY单位立方体

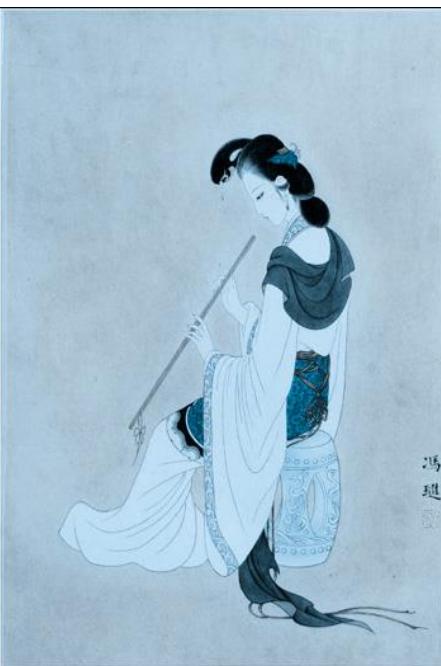


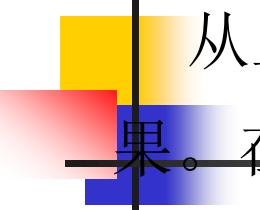
## 10.1.4 CMYK颜色模型



CMYK也称作印刷颜色模型，顾名思义就是用来印刷的。在印刷品上看到的图像，就使用了CMYK模型。其中K表示黑色（black），之所以不使用黑色的首字母B，是为了避免与蓝色（Blue）混淆。从理论上讲，只需要CMY三种油墨就足够了，浓度为100%的三种油墨加在一起就可以得到黑色。但是由于目前工艺还不能造出高纯度的油墨，CMY相加的结果实际是一种“灰”黑色。同时，由于使用一种黑色油墨要比使用青色、品红和黄色三种油墨便宜，所以黑色油墨被用于代替等量的青色、品红和黄色油墨。这就是四色套印工艺采用的CMYK模型的理由。

在图像交付印刷的时候，一般需要把这四个通道的灰度图制成胶片(称为出片)，然后制成硫酸纸等，再上印刷机进行印刷。传统的印刷机有4个印刷滚筒(形象比喻，实际情况有所区别)，分别负责印制青色、品红色、黄色和黑色。一张白纸进入印刷机后要被印4次，先被印上图像中青色的部分，再被印上洋红色、黄色和黑色部分，顺序如下图：





从上面的顺序中，可以很明显地感到各种油墨添加后的效果。在印刷过程中，纸张在各个滚筒间传送，可能因为热胀冷缩或者其他的一些原因产生了位移，这可能使得原本该印上颜色的地方没有印上。为了检验印刷品的质量，在印刷各个颜色的时候，都会在纸张空白的地方印一个+符号。如果每个颜色都套印正确，那么在最终的成品上只会看到一个+符号。如果有两个或三个，就说明产生了套印错误，将会造成废品。

喷墨打印机不会产生套印错误，这是为什么呢？印刷机的纸张要进出4个滚筒，套印错误就是在这进出之间产生的。而喷墨打印机是一次性打印，所以不存在套印错误。喷墨打印机如何实现一次性打印呢？喷墨打印机的将多个喷嘴前后依次排列。这样在打印的时候，纸张第一行先被喷上C，然后纸张向前移动一行，原先的第一行停在了M喷嘴下被喷上M色，同时新的空白的第二行被喷上C色。接着纸张再前移，已喷完C、M的那一行现在停在了Y色喷嘴下，被喷上Y色。而第二行被喷上M。新的空白第三行被喷上C。以此类推。如果在打喷墨打印机打印到一半的时取消打印，就会看到在图像的边缘分布着未完成的部分。

