

计算机图形学

马华东 教授

北京邮电大学计算机学院计算机应用中心

课程简介

- 计算机专业专业课: 课时34-40H

- 教材

- (1) 计算机图形学, 唐泽圣等, 清华大学出版社

- (2) 计算机图形学基础简明教程,

- 徐塞虹、杨峻, 北邮讲义

- (3) 计算机图形学, 杜世培, 重庆大学出版社

- (4) 计算机图形学教程,

- 唐荣锡, 汪嘉业, 彭群生等, 科学出版社

- (5) 计算机图形学, 孙家广等, 清华大学出版社

- 内容: 简洁→全面

- 价格: 便宜→较贵

- 选择: 考虑我们的学时数, 内容价格比适中

内 容

- (1) 计算机图形学概论
- (2) 计算机图形系统概述
- (3) 基本图形生成算法
- (4) 图形变换与裁剪
- (5) 人机交互技术
- (6) 曲线曲面的表示
- (7) 三维实体的造型
- (8) 消隐技术
- (9) 真实感图形技术
- (10) OPEN GL简介

第一章 计算机图形学概论

内容：计算机图形学的概念、发展和应用

目的：对计算机图形学有一个较全面了解

本章内容

- 1.1 计算机图形学的研究内容
- 1.2 计算机图形学的发展简史
- 1.3 计算机图形学的应用领域

1.1 计算机图形学的研究内容

- 什么是计算机图形学
- 图形的概念
- 计算机图形学研究的内容
- 计算机图形学的相关学科

1.1.1 什么是计算机图形学

- 国际标准化组织ISO在其数据处理字典中对计算机图形学有这样的定义：

计算机图形学(Computer Graphics)是研究通过计算机将数据转换为图形,并且在专用的显示设备上显示的原理、方法和技术的学科。

- 简言之：

计算机图形学是研究如何用计算机表示、生成、处理和显示图形的一门学科。

1.1.2 图形的概念

- 1) 什么是图形
- 2) 图形与图象
- 3) 图形表示方法
- 4) 矢量表示与位图表示比较

1) 什么是图形

图形是从客观世界的物体中抽象出来的带有灰度或者色彩以及形状的图或形。

2) 图形与图象

图象指计算机内以位图(bitmap)形式存在的灰度信息, 强调视觉表现; 而图形含有几何属性, 更强调场景的几何表示, 由场景的几何模型和景物的物理属性组成. (见图例)



图1.1 图形举例



图1.2 图象举例

3) 图形表示方法

方法1：矢量(参数法)

记录图形的形状参数和属性参数。适合于简单的线框图。

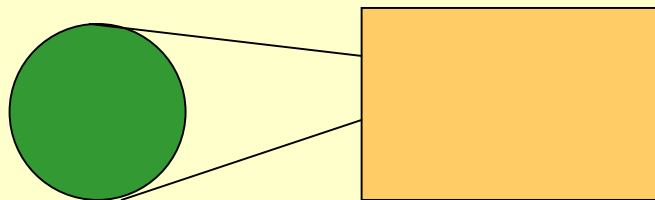


图1.3 一个适合于矢量表示的简单线框图

方法2：位图(像素图)

将一幅图形分割成栅格，栅格的每一像素的灰度或者色彩都单独记录。适合于真实感图形。

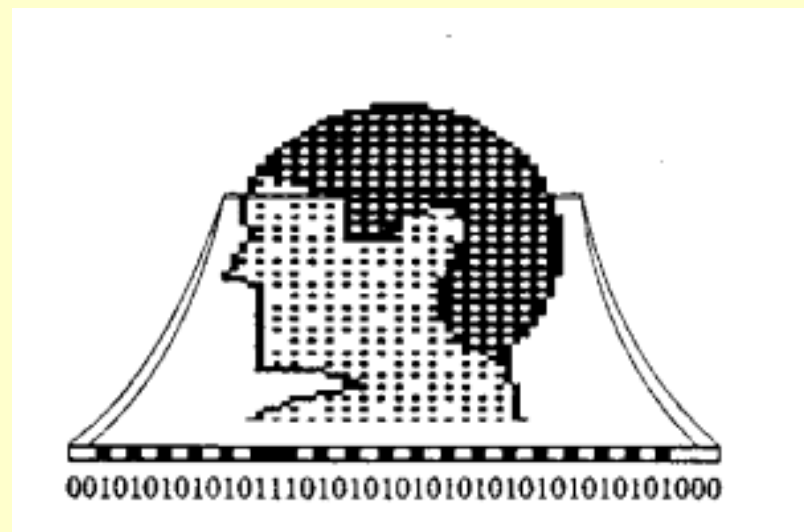


图1.4 位图图像
数据的一行

4) 矢量表示与位图表示比较

位图

- 可以表示任何图形
真实感效果好
- 应用方面的制约：
较强的处理和存储能力；
像素之间无内在联系；
分辨率固定；
- 实例：
三维场景表示是位图图形

矢量

- 画面简洁
速度快
- 对许多应用而言：
比位图更高效、更灵活；
表示方法有局限；
- 实例：
字符表示是矢量图形

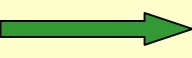
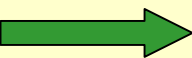
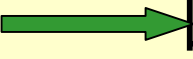
1.1.3 计算机图形学研究的内容

可以概括为以下3个方面：

- 图形系统的硬件配置，图形输入、输出设备的结构、原理、性能
- 图形系统支撑软件、图形应用软件和交互技术
- 图形处理中的各种算法（如图形元素的生成、几何变换、真实感处理、几何造型等）

1.1.4 计算机图形学的相关学科

与计算机图形学关系较为密切的学科有计算机图像处理、模式识别和多媒体技术等。彼此相互独立又相互渗透和关联。

- 计算机图像处理 图像  图像
- 模式识别 图像  数据(模型)
- 计算机图形学 数据(模型)  图形
- 多媒体技术 综合处理声、文、图形/图像等技术, 集成性、实时性、交互性

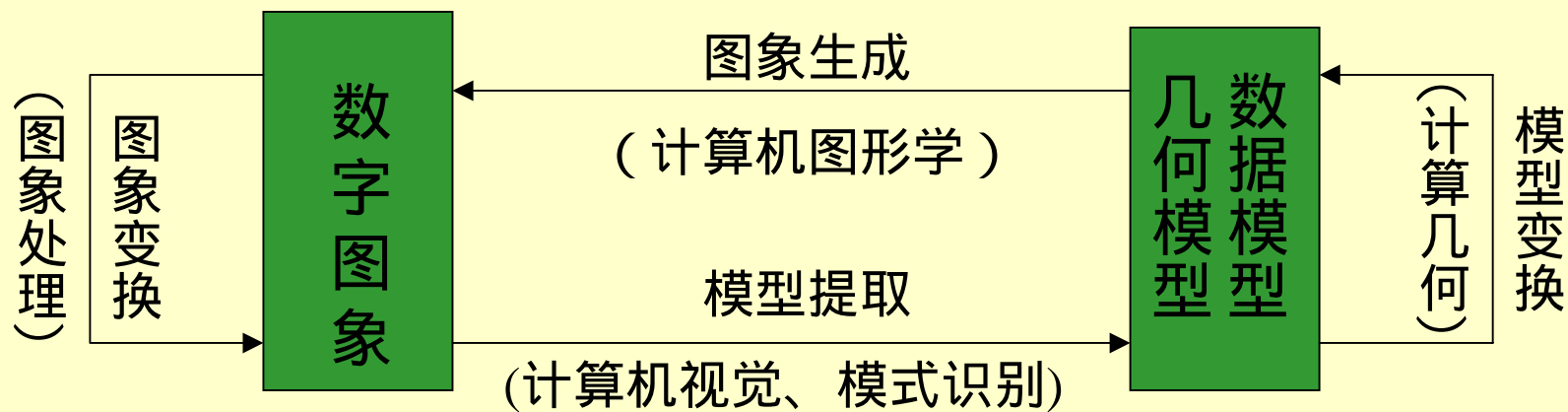


图1.5 计算机图形学的相关学科

1.2 计算机图形学的发展简史

- 1950年, 第一台图形显示器(Whirlwind I)在MIT诞生
- 20世纪50年代末, MIT开发SAGE空中防御系统, 交互式图形生成技术出现
- 1962年, MIT的Ivan E. Sutherland在它的博士论文首先使用Computer Graphics术语
- 20世纪60年代, 美、英等国开始大规模研究, 大型计算机连接图形终端成为图形生成技术的运行环境
- 1964年, MIT的Prof. Steven Coons提出插值4条任意的边界曲线来构造曲面-Coons曲面
- 20世纪60年代早期, 法国雷诺公司的Pierre Bezier提出Bezier曲线曲面理论用于几何外形设计
- 20世纪60年代中期, 矢量显示器问世
- 20世纪60年代后期, 存储管式显示器得到推广应用

- 20世纪70年代初, 基于电视技术的光栅扫描显示器出现
- 20世纪70年代, 真实感与实体造型技术产生
- 20世纪70年代, 图形系统标准化取得进展: GKS, PHIGS. 以小型机为基础的图形生成系统开始进入市场并形成主流
- 20世纪80年代, 全局光照模型和真实感绘制算法: Whitted模型, 光线跟踪(透射), 辐射度方法(多重漫反射)
- 20世纪80年代, 计算机工作站取代小型计算机成为图形生成的主要环境
- 20世纪90年代以来, 微型机性能迅速提高, 已成为计算机图形生成技术的重要环境
- ACM SIGGRAPH为代表的学术会议, 是展现国际图形学最新研究成果的论坛
- 国际内学术刊物、会议。国内图形学研究与应用。

1.3 计算机图形学的应用领域

- 计算机图形学的应用
- 计算机图形学的发展动向

1.3.1 计算机图形学的应用

计算机图形的应用无所不在！

- CAx领域

计算机辅助设计CAD；

计算机辅助制造CAM；

计算机辅助教学CAI；等

- 系统模拟、虚拟现实

航空、航天、建筑、体育等模拟与训练

- OA与电子出版系统

计算机图形的应用无所不在！

- 过程控制

飞船、卫星、导弹、工业生产过程等

- 绘制勘探、测量图

地形、地貌、矿藏、气象、GIS

- 艺术、娱乐和商业

平面设计、动画、影视制作

- 医学诊断技术

CT、核磁共振等数据分析诊断病因，手术模拟

- 科学计算可视化

复杂数据的直观表示，方便观察结果

（见计算机动画片）

1.3.2 计算机图形学的发展动向

- 造型技术

规则形体：可用欧氏几何描述的形体

几何造型技术（几何描述）

特征造型技术（特征作为形状描述的单元）

基于物理的造型技术（动画）

不规则形体：

过程式模拟, 如分形、粒子系统、基于文法生成

- 真实图形生成技术

简单局部光照模型、全局光照模型,

基于图象绘制技术

- 人-机交互技术

三维人-机交互技术, 虚拟环境, 多通道技术,
非精确交互

- 基于网络的图形技术

网络和多媒体技术, 分布式图形,
虚拟现实建模语言VRML

第2章 计算机图形系统概述

内容：计算机图形系统的构成以及图形软件国际标准

目的：对计算机图形系统构成和软件结构有较全面了解

本章内容

2.1 计算机图形系统的构成

2.2 图形输入输出设备

2.3 计算机图形系统标准化介绍

2.1 计算机图形系统的构成

- 图形系统与一般计算机系统最主要的差别：
具有图形的输入、输出设备以及必要的交互手段。而且作为系统核心的主计算机, 在运算速度和存储容量上均有较高的要求。
- 一个计算机图形系统是有关的硬件和软件的集合。
- 硬件设备包含:
交互设备; 输入输出设备; 存储设备
对系统要求: 运算速度(GHz), 内存(GB), 外存(TB)

2.1.1 计算机图形系统的结构

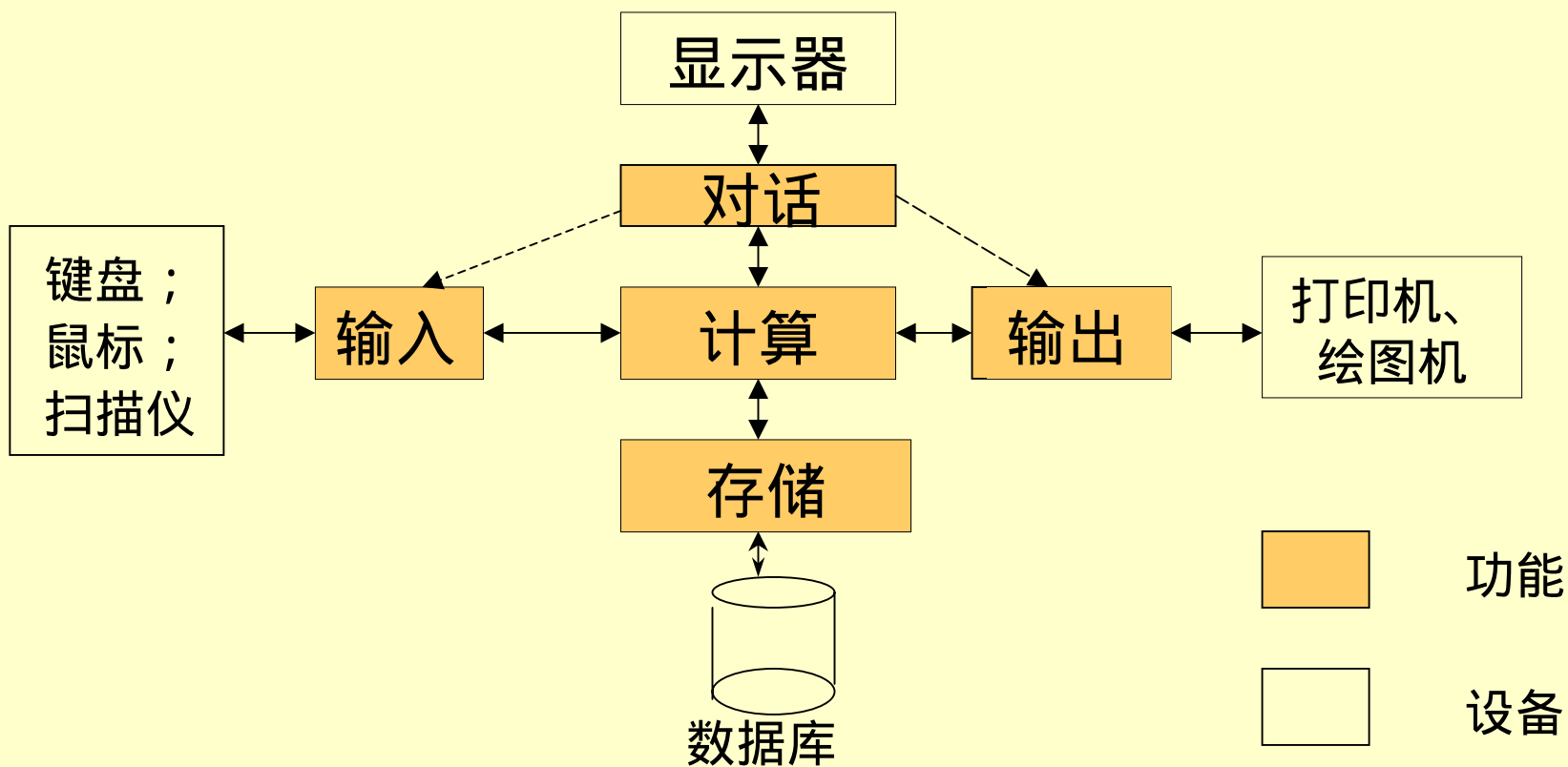


图2.1 图形系统的结构示意图

2.1.2 计算机图形系统的软件配置

通常包含4个层次如图所示：

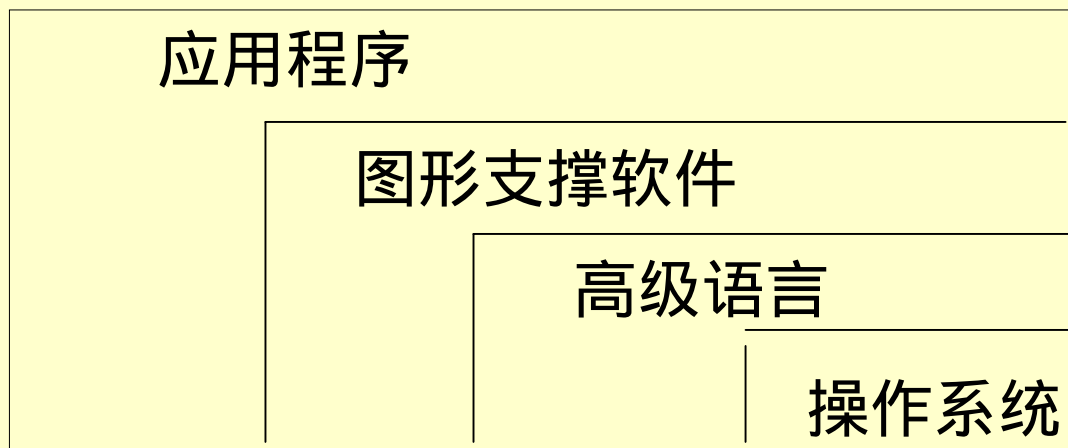


图2.2 计算机图形系统软件层次示意图

2.1.3 图形软件的开发途径

- 以某种高级语言为基础, 加上扩充处理图形功能的子程序包, C, GKS, GL
- 专用图形开发工具, AutoCAD

2.2 计算机图形输入、输出设备

2.2.1 计算机图形输入设备

2.2.2 计算机图形输出设备

2.2.3 光栅扫描显示技术介绍

2.2.1 计算机图形输入设备

- 输入设备的逻辑分类
- 常用的图形输入设备介绍

2.2.1.1 输入设备的逻辑分类

图形输入设备是将用户的图形数据、命令等转换成电信号传送给计算机。

可以分成以下6类逻辑输入设备：

- 定位(locator): 指定一个坐标点。对应的物理设备有鼠标器、键盘、数字化仪、触摸屏等。
- 笔划(stroke): 指示一个坐标点系列, 如指定一条曲线的控制点等。主要物理设备有数字化仪。
- 送值(valuator): 输入一个数值。最常用的物理设备是键盘的数字键。

- 字符串(string)：输入一个字符串。最基本的物理设备是键盘的数字字母键。
- 拾取(pick)：选择一个显示对象, 为应用程序处理确定目标。常用的物理设备是各种定位设备。
- 选择(choice)：在一个可选动作或选项中进行选择, 如选择菜单项。典型的物理设备是鼠标器、数字化仪、键盘的功能键等。

2.2.1.2 常用的图形输入设备介绍

- 1) 键盘
- 2) 鼠标器
- 3) 扫描仪
- 4) 数字化仪
- 5) 触摸屏
- 6) 数据手套

1) 键盘(Keyboard)

- 键盘是最常用的输入设备。键盘上有数字键、字母键、控制键、光标移动键、功能键等。键盘可用于输入数据值、字符串以及图形命令。
- 键盘的工作原理：

按下某一键, 通过开关元件(目前大多是电容开关), 产生电平信号, 控制编码器进行编码, 得到该键的ASCII码, 此码送入编码寄存器;

同时按键标志触发器置位向计算机发出键盘中断请求, 计算机响应中断后, 完成相应的中断服务程序, 如显示字符。

2) 鼠标器(Mouse)

- 鼠标是一种较小的手握设备。移动鼠标器可以驱动屏幕光标, 配合鼠标器上的几个按钮完成定位、拾取、选择等功能。
- 鼠标器的工作原理：
鼠标底部的转球可用来记录移动方向和偏移量。机械鼠标是将转球的运动转换成数字信号；光电鼠标通常需要一个特殊的垫板，其上有水平线和垂直线的网格, 光学感应器检测跨越网格线的移动量。

3) 图像扫描仪(Image scanner)

- 利用光学原理扫描图表、黑白和彩色照片、文本等, 经AD转换, 其灰度或彩色等级被记录下来, 并且按阵列形式存储, 由此计算机可以进行进一步的处理。
- 分类
 - 平板式
 - 手持式
 - 滚动式

4) 数字化仪(Digitizer)

- 是一种电子的图形数据转换设备。用户可以使用一种与图形输入板相接的特殊记录笔画图和写字, 以达到将图形输入计算机的目的。数字化仪可以向系统提供定位坐标以及按键状态。
- 按工作原理分为机械式、电位梯度式、电磁感应式、超声波式等, 电磁感应式使用较多。数字化仪的表面嵌有导线组成的网格, 构成感应阵列。手动定位头上常有若干按键, 可用来表示不同命令的含义。

5) 触摸屏(Touch panel)

- 触摸屏允许用户直接用手指或其它物体在屏幕上定位和选择。
- 触摸输入基于光学、电子或者声学的方法。
- 电子触摸屏有电阻式和电容式。
- 电阻式触摸屏感应器是一块覆盖电阻性栅格的玻璃,再在上面蒙上一层涂有导电涂层并有特殊模压凸缘的聚脂薄膜。物体的触摸导致导电层接触,这引起沿电阻板的电压降,该压降转换为所选屏幕位置的坐标值。
- 电容式触摸屏感应器是块透明的玻璃,表面有导电涂层,其上覆盖一层保护性玻璃外层。它工作时在感应器边缘的电极产生分布的电压场,

用手指或其它导电体触摸导电涂层时, 电容改变, 电压场变化, 控制器检测这些变化, 从而确定触摸的位置。

- 在光学触摸屏中, 沿一块玻璃板的水平方向和垂直方向, 产生红外线。触摸屏幕引起射线被手指遮挡反射到发射器。接触点的屏幕位置由射线从发送到反射回发射器的时间间隔计算。
- 在声波触摸屏中, 沿屏幕水平方向和垂直方向, 产生高频声波。触摸屏幕引起声波有一部分被手指反射到发射器。接触点的屏幕位置由每个波从发送到反射回发射器的时间间隔计算。

6) 数据手套(Data glove)

- 数据手套可以用来给虚拟场景中的对象定位或操纵该场景的二维投影以便在视频监视器上观察。三维投影可用头盔观察。
- 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。

2.2.2 计算机图形输出设备

常用的计算机图形输出设备可分为两大类：

- (1) 软拷贝设备——显示器
- (2) 硬拷贝设备

2.2.2.1 软拷贝设备——显示器

	出现时间	目前使用状况
向量显示器 (随机扫描显示器)	60 年代中期	已淘汰
存储管式显示器	60 年代末期	已淘汰
光栅扫描显示器	70 年代中期	主流产品
平板显示器 (液晶、等离子体)	80 年代末期 至目前	已进入实用阶段

2.2.2.2 硬拷贝设备

- 1) 点阵式打印机
- 2) 喷墨打印机
- 3) 激光打印机
- 4) 笔式绘图仪
- 5) 静电绘图仪
- 6) 摄像机

1) 点阵式打印机

- 工作原理：利用一个点阵打印头将成型字符或者图像通过色带印在纸上。打印头通常有7~24针, 每个针分别与相应的驱动电磁铁相连, 可以单独激活。打印头每次相对纸横向从一端走向另一端, 纸向前纵向推进一行。
- 这类打印机是光栅输出设备, 需要有扫描转换, 事先把矢量图转换成打印机用信号。利用彩色色带可以产生彩色硬拷贝。

2) 喷墨打印机

- 工作原理: 使墨水通过极细的喷嘴射出, 用电场控制喷出墨滴喷射到纸上的位置, 从而在绕在滚筒上快速旋转的纸上绘制图像。
- 若将分别喷射青绿、深红、黄和黑色墨水的功能相同的4支喷嘴并行排列, 运用彩色的空间混色效果, 就可以进行彩色图像打印。

3) 激光打印机

- 工作原理：利用一个涂有硒的、能旋转的鼓，其上带有正电荷，由信号控制的激光束被扫描到鼓上，激光束射到的部分失去电荷，而其余部分仍保留正电荷。带有负电荷的调色剂则粘附到鼓的正电荷区，从而形成黑色空拷贝。上述过程重复三次，每次一种基色(R,G,B)，将会产生彩色拷贝。
- 激光打印机可以有一微处理机，进行扫描转换及控制打印机。

4) 笔式绘图仪

- 工作原理：笔式绘图仪是矢量型设备, 绘图笔相对纸作随机移动。一个电脉冲通过驱动电机与传动机构使画笔移动的距离称为步距, 步距越小, 画出的图就越精细。
- 常用的笔式绘图仪可分为平板式和滚筒式两种。平板式的笔有两个坐标方向即X和Y方向, 它既可以横向也可以纵向运动, 而纸通常是静止不动的；
滚筒式的笔只有一个坐标方向即X方向, 它只能左右横向运动, 而纸代表另一个坐标方向即Y方向, 它随滚筒的旋转, 可以前后纵向运动。

5) 静电绘图仪

工作原理：事先使白纸或黑纸上带有负电荷, 而吸有调色剂的针尖带有正电荷, 当由程序控制的电压按阵列式输出并选中某针尖时, 就将调色剂附着到纸上形成图形。

6) 摄像机

- 摄像机可以拍下显示在CRT屏幕上的图像记录在胶片或者磁带上。输入到摄像机中的信号可以是光栅视频信号, 或是一张位图, 或是矢量类型结构的信号。
- 得到彩色胶片有两种基本技术：
 - 一种是摄像机直接从彩色CRT上得到彩色图像, 其图像质量受到影孔板及光栅扫描的限制；
 - 另一种是通过彩色滤波器拍摄黑白CRT上的图像, 按序显示图像的不同颜色分量, 这种技术可以得到很高质量的位图或者矢量图像。

2.2.3 光栅扫描显示技术介绍

- 像素、分辨率、屏幕坐标的定义
- 光栅扫描显示器的组成
- 显示存储器
- 彩色表
- 图形生成原理
- 光栅扫描显示器的主要性能参数

2.2.3.1 像素、分辨率、屏幕坐标的定义

- 1) 像素
- 2) 分辨率
- 3) 屏幕坐标

1) 像素

- 光栅扫描显示器的显示屏上有许多水平光栅扫描线, 每条光栅线由许多点组成, 这些点就是像素, 即图像的基本元素。
- 每一个像素是显示屏上可以编址控制的最小单元。

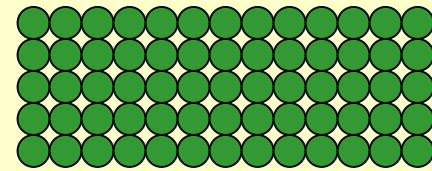


图2.3 显示屏上的像素

2) 分辨率

- 显示器的分辨率表示为：
水平分辨率×垂直分辨率
- 水平分辨率即每条光栅线上像素的总数目；垂直分辨率即屏上的水平光栅线数目。显示器的分辨率决定了显示屏上所拥有的像素总数。
- 例如, IBM PC彩色图形显示器(CGA)的中分辨率模式为 320×200 , 表明屏上共有 $320 \times 200 = 64000$ 个像素。又如, 目前常用 1024×768 。
- 对于同样尺寸的显示器, 分辨率越高, 记录图形的像素密度越大, 显示出的图形质量越好。

3) 屏幕坐标

- 显示屏上的每一像素被赋予一对整数值作为该像素的坐标。
- 屏上像素的坐标值一般是按照水平方向(X)从左到右、垂直方向(Y)从上向下两个方向从0开始整值排列。
- 显示屏上的坐标和像素位置一一对应。

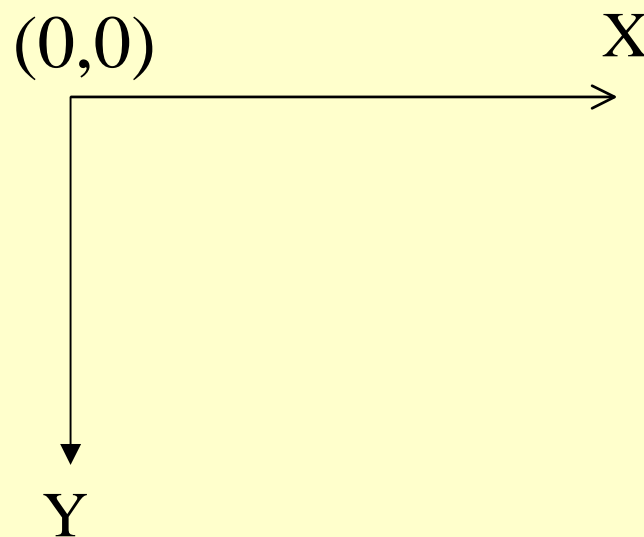


图2.4 显示屏上的坐标定义

2.2.3.2 光栅扫描显示器的组成

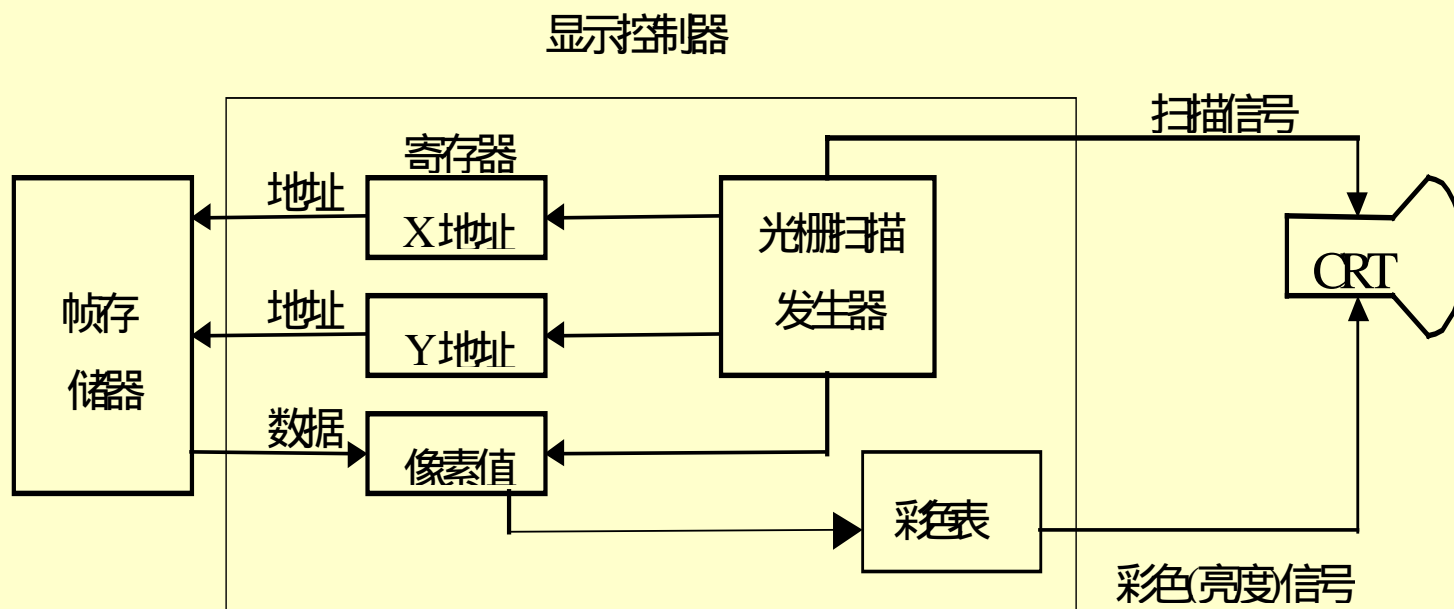


图2.5 光栅扫描显示器的硬件组成

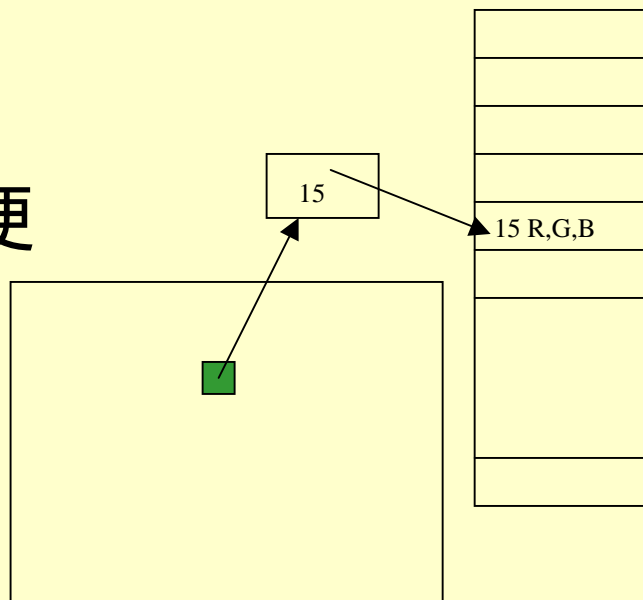
2.2.3.3 显示存储器

- 显示存储器(也称为位存储器、帧缓冲存储器)存储着屏幕画面上图形的映像。该图形映像对应一个二维矩阵,矩阵的每个元素就是屏幕对应点的图像信息,称为像素值。显示存储器的组织结构有多种形式。
- 显示存储器的分页处理是目前的常用技术,存储区划分成若干页,每一页存放一幅屏幕画面。正在显示的页称为可见页。被进行读写操作的页称为活动页。此技术可以使用户通过上下左右移屏功能看到存储器中的整幅画面,还用来支持动画效果。

2.2.3.4 彩色表

彩色表又称为调色板, 用来定义图像的不同颜色。

- 引入彩色表的目的
颜色编号,快速,节省存贮,变化方便
- 彩色表的工作原理
代号-颜色RGB值对照表



2.2.3.5 图形生成原理

(1)图形信息的产生有两种主要途径：

其一, 由计算机执行相应的图形应用程序, 图像生成系统接受指令将图形的矢量表示转换成像素表示, 再将像素值存入显示存储器;

其二, 图像生成系统直接把图形输入设备(摄像机、扫描仪等)输入的图形图像直接或经过主存储器间接地存放显示存储器中。

(2)显示控制器生成水平和垂直同步扫描信号送到监视器, 使CRT电子束进行水平扫描和垂

直扫描形成光栅; 另一方面又根据电子束在屏幕上的行、列位置, 不断地读出显示存储器中对应位置的像素值。

(3)利用彩色表将读出的像素值转换成R、G、B三原色的亮度值, 来控制CRT的R、G、B电子束, 在屏幕对应点生成需要的像素颜色。

为了使屏幕上显示的画面不闪烁, (2)~(3)应反复进行, 一般要求50~60帧/秒。

2.2.3.6 光栅扫描显示器的主要性能参数

- 显示分辨率 (SVGA: 1024×768)
- 颜色或者亮度等级数目 (16M色)
- 画图速度 (动画)
- 屏幕尺寸 (15英寸,17英寸)
- 刷新频率 (水平扫描30-82kHz, 垂直扫描50-120Hz)
- 屏幕纵横比 (4:3)

2.3 计算机图形软件国际标准简介

2.3.1 制定图形软件标准的目的

2.3.2 图形软件标准分类

2.3.3 目前常用的图形系统软件标准

2.3.1 制定图形软件标准的目的

制定图形软件标准的目的在于使图形软件能够在不同的计算机和图形设备之间进行移植, 以便提高图形软件的利用率, 降低开发成本, 缩短研制周期, 使图形软件向着通用、高级与设备无关的方向发展。

2.3.2 图形软件标准分类

目前已经制定或正在制定的一些图形标准都是接口标准。这些标准的功能旨在使图形系统中两部分之间的接口标准化, 可以分为两类：

- 数据接口标准
- 子程序接口标准

2.3.3 常见的图形软件系统标准

- 1) 图形核心系统GKS
(Graphics Kernel System)
- 2) 计算机图形元文件CGM
(Computer Graphics Metafile)
- 3) 计算机图形设备接口CGI
(Computer Graphics Interface)
- 4) 程序员层次结构图形系统PHIGS
(Programmer's Hierarchical Interactive Graphics System)
- 5) OPEN GL

1) GKS

- 1985年, 德国提出的GKS成为第一个计算机图形ISO国际标准(二维)
- 提供了在应用程序和图形输入输出设备之间的功能接口, 是一个子程序接口标准
- 将图形输入/输出设备定义为逻辑的工作站, 共有6种: 输入、输出、输入输出、独立图段存储、元文件输入、元文件输出
- GKS最基本的图形信息是输出图元, 包括折线、多重记号、正文、填充区、像素组5种基本图形元素和一种供扩充用的广义图元

- 一组图元的集合可构成段, 以段标识符命名, GKS提供的段操作有拷贝、转换、重命名、删除插入、设置可检取性、优先性修改、可见性修改等
- GKS定义了3种坐标系: 世界坐标系WC(用户)、标准化设备坐标系NDC(虚拟)、设备坐标系DC(物理)

标准化转换: $WC \rightarrow NDC$ (存储段和元文件)

工作站转换: $NDC \rightarrow DC$ (生成准确的图形)

- GKS定义了6类逻辑输入设备：
定位设备、笔画设备、拾取设备、选择设备、
值设备、字符串设备
- 每种设备可以用3种操作方式：
请求方式、采样方式、事件方式
- GKS采用元文件在图形系统之间传送图形信息
- GKS-3D是由GKS发展而来的完全三维图形软件标准, 功能包括三维输入、三维图素、三维几何属性、具有视图操作的三维变换以及隐藏线、面的消除等

2) CGM

- CGM是美国国家标准化协会ANSI 1986年公布的标准, 1987年成为ISO的第二个国际图形标准
- 它是一套与设备无关的语义、语法定义的图形文件格式, 提供了随机存取、传送、定义图像的手段, 是数据接口标准
- CGM是一个静态的图形生成元文件, 即它不能产生所定义图形的动态效果, 例如不能实现动态的几何变换

- CGM定义的元文件语义、语法主要包括8类元素：分隔元素、元文件描述元素、图像描述元素、控制元素、图形图像元素、属性元素、Escape元素、外部元素
- 一个完整的元文件的文件格式如图

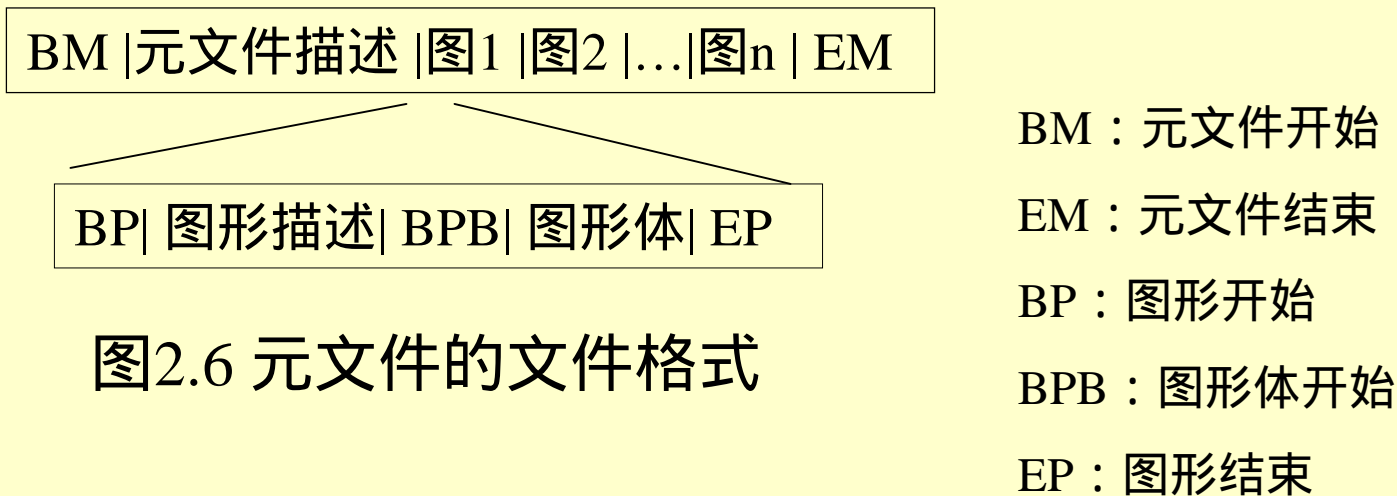


图2.6 元文件的文件格式

3) CGI

- CGI是ISO TC 97提出的图形设备接口标准草案，与ANSI 1985年公布的VDI (Virtual Device Interface) 标准一致
- CGI的目的是提供一种与设备无关的方法, 属于子程序接口标准
- 它的所有的功能大约有170个, 可以分成5组：
控制、查询及出错处理； 输出图素及其属性；
图段定义及处理； 输入及响应处理；
光栅图形处理

- GKS在图形描述和处理功能上优于CGI, 但实现这些功能需要可观的时间和空间开销, 因此CGI较之GKS更适合于软、硬件资源都比较有限的微机中

4) PHIGS

- 它是ANSI在1986年公布的图形系统软件标准, 是为应用程序员提供的控制图形设备的子程序接口标准
- 与GKS-3D功能类似, 但PHIGS的图形数据按照层次结构组织, 而且还提供了动态修改和绘制显示图形数据的手段, 是一个高交互、高动态的应用程序图形标准
- PHIGS的图形数据都被组织在称之为结构的单元中, 结构之间可以通过层次调用发生联系。每个结构由若干元素组成

- PHIGS的标准功能可划分为9个程序模块来分别实现, 各模块调用的公共子程序集中在一个公共子程序模块中, 系统逻辑结构清晰, 便于测试开发。9个程序模块是：

结构管理模块

系统查询管理模块

输入管理模块

设备驱动模块

输出管理模块

语言接口连编模块

系统和工作站管理模块

公共子程序模块

档案文件管理模块

- PHIGS定义了5种坐标系, 其输出流水线为 :



5) OPEN GL

- 上述几种标准并未得到真正推广。
- 目前, 在微机和工作站上运行的三维图形软件库OPEN GL成为事实上的图形开发标准。
- OPEN GL最初是SGI工作站采用的图形库, 广泛应用于真实感图形制作
- 其主要特点是三维造型和真实感处理能力强大, 支持动画功能。

第3章 基本图形的生成算法

知识点：直线段、圆弧、规则函数曲线以及字符等基本图形的生成算法；区域填充的算法

目的：了解二维平面上一些基本图元的生成方法和封闭区域的各种填充方法

本章内容

- 3.1 直线段的生成算法
- 3.2 圆弧的生成算法
- 3.3 规则曲线的生成方法
- 3.4 字符的生成方法
- 3.5 区域填充

3.1 直线段的生成算法

3.1.1 DDA算法

3.1.2 Bresenham算法

3.1.3 直线段属性

3.1.1 DDA算法

即数字微分分析法(Digital Differential Analyzer, DDA), 亦称为增量算法。

- 1) 算法思想
- 2) 算法原理
- 3) 算法描述
- 4) 算法特点

1) 算法思想

从直线段的起点像素出发，依据直线的微分方程依次确定描述直线的各个像素点。

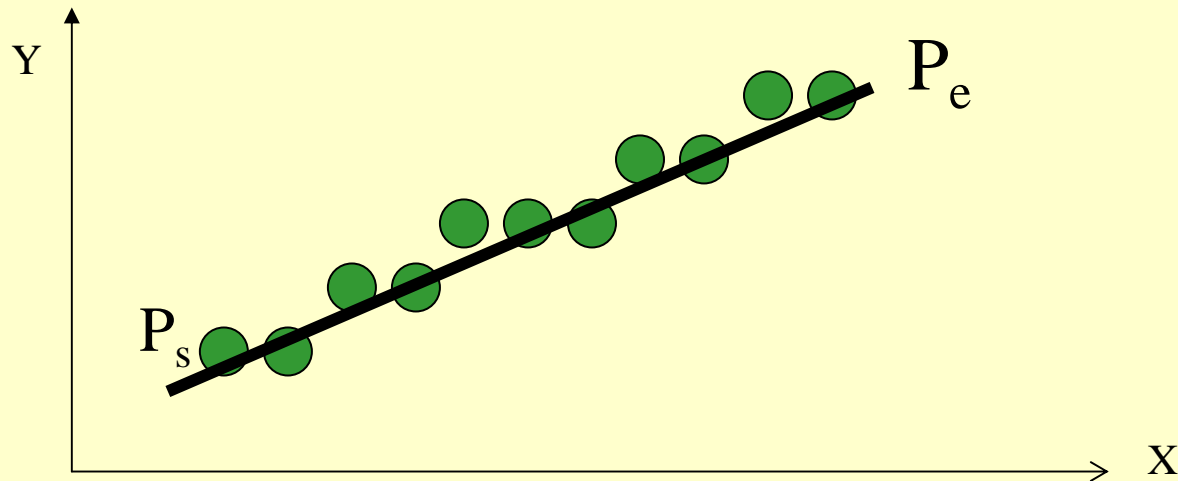


图3.1 理想直线段与像素点表示直线段的比较

微分方程: $dy/dx=m$, $y[i+1]=y[i]+m(x[i+1]-x[i])$

$$y[i+1]=y[i]+m \quad (0 < m < 1)$$

2) 算法原理

设一直线段的起点和终点坐标分别为 (x_s, y_s) 和 (x_e, y_e) 。则直线段在X和Y方向的增量分别为：

$$\Delta x = x_e - x_s, \quad \Delta y = y_e - y_s \quad (3-1)$$

$$\text{设 } \Delta t = \max(|\Delta x|, |\Delta y|) \quad (3-2)$$

取时间步长为 $1/\Delta t$,若当前像素点坐标为 (x_i, y_i) ,则下一个像素点的坐标可由以下两式确定：

$$\left\{ \begin{array}{l} x_{i+1} = x_i + dx = x_i + \Delta x / \Delta t \end{array} \right. \quad (3-3)$$

$$\left\{ \begin{array}{l} y_{i+1} = y_i + dy = y_i + \Delta y / \Delta t \end{array} \right. \quad (3-4)$$

3) 算法描述

Procedure DDA(x_1, x_2, y_1, y_2)

var k, i : integer; x, y, dx, dy : real;

begin

$k := \text{abs}(x_2 - x_1)$;

 if $\text{abs}(y_2 - y_1) > k$ then $k := \text{abs}(y_2 - y_1)$; //求最大增量

$dx := (x_2 - x_1) / k$; $dy := (y_2 - y_1) / k$;

$x := x_1$; $y := y_1$;

 for $i := 1$ to k do

 begin

 plot(round(x), round(y)); //下一个像素点坐标

$x := x + dx$; $y := y + dy$;

 end

end

注意: (1) 像素坐标是整数, 所以递推公式得到的坐标要用舍入的方法取整;

(2) 为保证计算精度, 公式中使用的 x_i 和 y_i 应保持取整前的数值。

4) 算法特点

- 原理简单.
- 但浮点增量的连续迭加中, 取整误差的积累会使长线段所计算的像素位置偏离实际线段, 而且取整操作和浮点运算较为耗时。

3.1.2 Bresenham算法

在直线生成的算法中, Bresenham算法是最有效的算法之一。

- 1) 算法思想
- 2) 算法原理
- 3) 算法描述
- 4) 例子
- 5) 算法特点

1) 算法思想

根据直线的斜率确定选择X或者Y方向作为计长方向, 在此方向上每次递增一个单位步长(或者一个像素单位), 另一个方向上是否同时产生一个单位增量由一个计算量很小的判别式来判断。

2) 算法原理

(以斜率在0~1之间的直线段为例)

- 这种情况下, 选择X方向为计长方向, 即增量 $dx=1$ 。如图3.2所示, 设 $P_i(x_i, y_i)$ 是已选定的离直线最近的像素, 现在要决定 P_{i+1} 是T还是S。
- 显然, 若 $d < 0.5$, 则S比较靠近直线, 应选S;
若 $d \geq 0.5$, 则应选T。 ($m = \Delta y / \Delta x$)
- 令 $e = d - 0.5$ (初值为 -0.5), 即有:
$$\begin{cases} e < 0 \text{ 时, 选 } P_{i+1}(x_i+1, y_i), \text{ 更新 } e = e + m; \\ e \geq 0 \text{ 时, 选 } P_{i+1}(x_i+1, y_i+1), \text{ 更新 } e = e + m - 1; \end{cases}$$

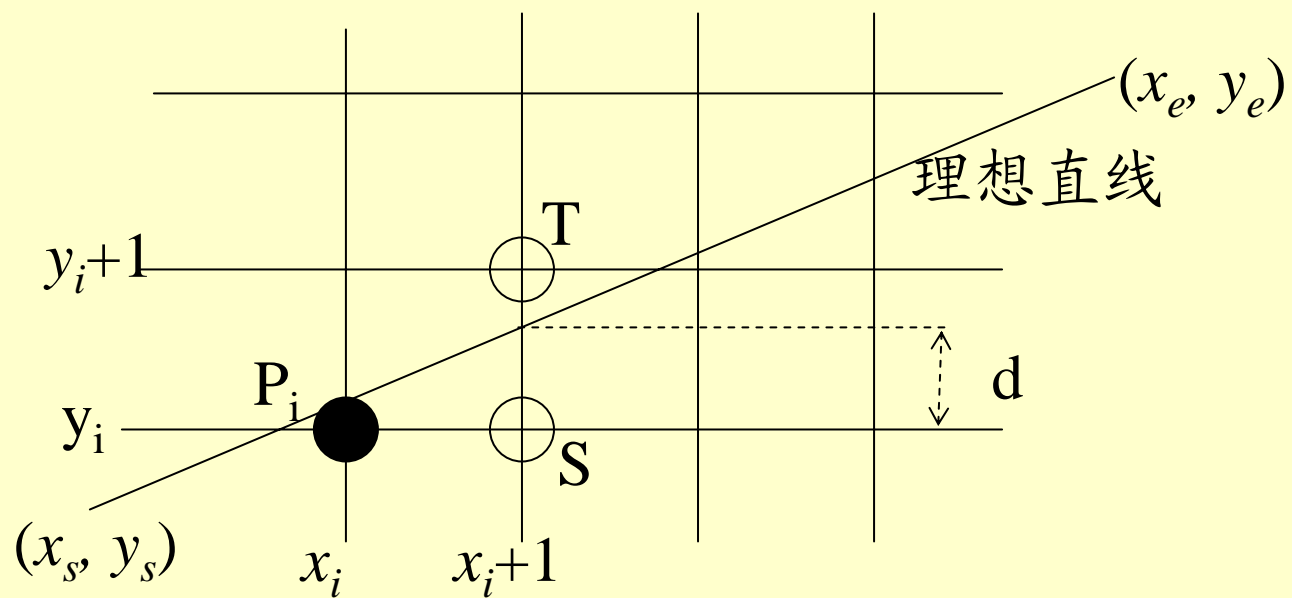


图3.2 Bresenham算法的几何示意图

- 为计算简便(整数计算), 将判别式 e 替换为

$$e = 2e\Delta x;$$

初值 $e = -\Delta x$

- 结论: 得到递推公式

$$\left\{ \begin{array}{l} e < 0 \text{ 时, 选 } P_{i+1}(x_i+1, y_i); \\ \quad e = e + 2\Delta y \\ e \geq 0 \text{ 时, 选 } P_{i+1}(x_i+1, y_i+1); \\ \quad e = e + 2\Delta y - 2\Delta x \end{array} \right.$$

其中, $\Delta x = x_e - x_s$; $\Delta y = y_e - y_s$.

3) 算法描述

Procedure Bresenhamline(xs,ys,xs,ys)

int x, y, deltax, deltay, e;

begin

deltax:=xs-xs; deltay:=ys-ys;

e:=-deltax; x:=xs; y:=ys;

for(i:=0 to deltax) do

begin

plot(x, y);

if (e>=0) then

begin

y:=y+1; e:=e-2*deltax;

end

x:=x+1; e:=e+2*deltay;

end

end

4) 例子

生成端点为(20,10)和(30,18)的直线段。

[点的确定过程]

$\text{deltax}=10, \text{deltay}=8, e=-10$

$x=20, y=10, e=6;$

$x=21, y=10, e=2;$

$x=22, y=11, e=-2;$

$x=23, y=12, \dots\dots$

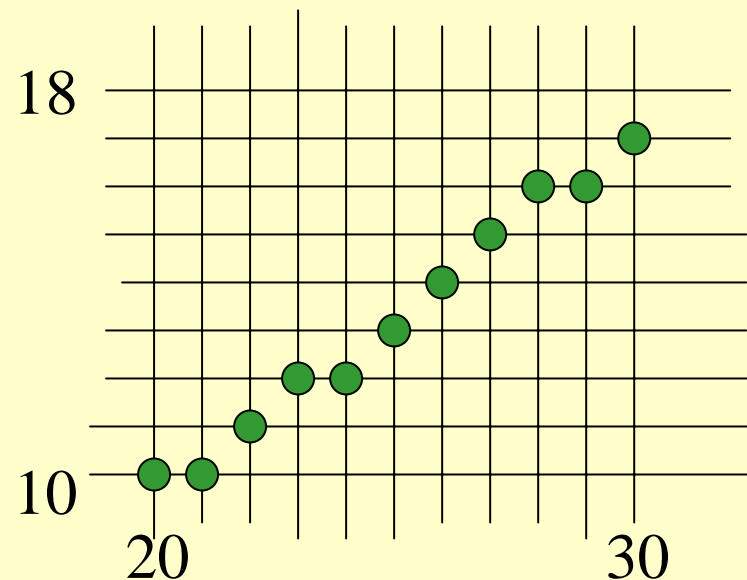


图3.3 用Bresenham算法生成的线段

5) 算法特点

- 算法中可以全部采用整数运算, 且判别式的计算中只含+、- 以及*2(计算机内以移位方法实现)运算, 计算效率高。而且便于硬件或者固件实现, 算法构思巧妙。
- 扩展。当斜率大于1时, 可以将x和y交换, 即以Y方向作为计长方向, 每次变化1个单位步长或者1个像素单位。对于任意走向的直线段, 则可以根据其端点的位置来控制其增量的正负。

3.1.3 直线段属性

- 直线段的基本属性包括线型、线宽和颜色。
- 在直线段的生成程序中，通常应允许用户选择直线段的属性参数。在实用系统中，可把常用属性值设置为默认值，如黑色的细实线。

1) 线型

- 线型属性可以有实线、虚线和点划线等。
- 要实现不同的线型, 需要修改画线算法:
使线段生成过程中, 可以按实线小段和空白小段相间的方式显示, 允许用户指定每个小段的长度参数(一般是定义在用户坐标下的)。
- 线型的默认值通常是实线。
- 例如, 在PHIGS应用程序中设置线型属性, 用户在画线之前需调用函数

`setlinetype(lt)`

*lt*为1,2,3,4分别生成实线、虚线、点线和点划线。

2) 线宽

- 线宽选择的实现取决于输出设备的能力。
- 通常把输出设备能产生的最小直线宽度(一个像素的宽度或者绘图笔的画线宽度)作为基准,再定义2倍宽或4倍宽的直线。
- 实现线宽的功能可根据宽度调整线段端点参数,重复调用线段生成程序画几次即可。此时,还应考虑直线的斜率,以便决定在端点处是沿X方向还是沿Y方向增减坐标。另外,对于粗线在连接处,往往还应作特殊处理。
- 可以用命令如 `setlinewidth(lw)` 来设置线宽属性,参数 `lw` 指定显示线段的相对线宽。

3) 线色

- 当系统提供颜色(或亮度)选择时,当前的可用颜色存放在系统的彩色查找表中。
- 可选颜色的数目取决于显示存储器中每个像素单元的大小(如8bit, 256种), 像素单元的内容是彩色查找表的索引(地址)值。
- 在PHIGS中用下列函数设置线的颜色值:

`setPolylineColourIndex(lc)`

参数 lc 是一非负整数,即彩色表的索引(地址)值

- 以背景色画的线是不可见的。用户可以利用指定背景颜色画线来消除以前显示的线。

3.2 圆弧的生成算法

3.2.1 Bresenham算法

3.2.2 正负法(逐点比较法)

3.2.3 多边形逼近法

3.2.1 Bresenham算法

- 1) 算法思想
- 2) 算法原理
- 3) 算法描述
- 4) 算法特点

1) 算法思想

与Bresenham直线生成算法一样, 其基本方法是从一个起点出发, 利用判别式选择下一个显示点。判别式的值通过简单计算获得, 其符号用作判断。

2) 算法原理

- 考虑圆心坐标在坐标系原点,位于第一象限的八分之一圆弧 P_1P_2 , 如图3.4所示。其他部分可用7个对称点绘出。
- 方程
$$x^2+y^2=R^2$$

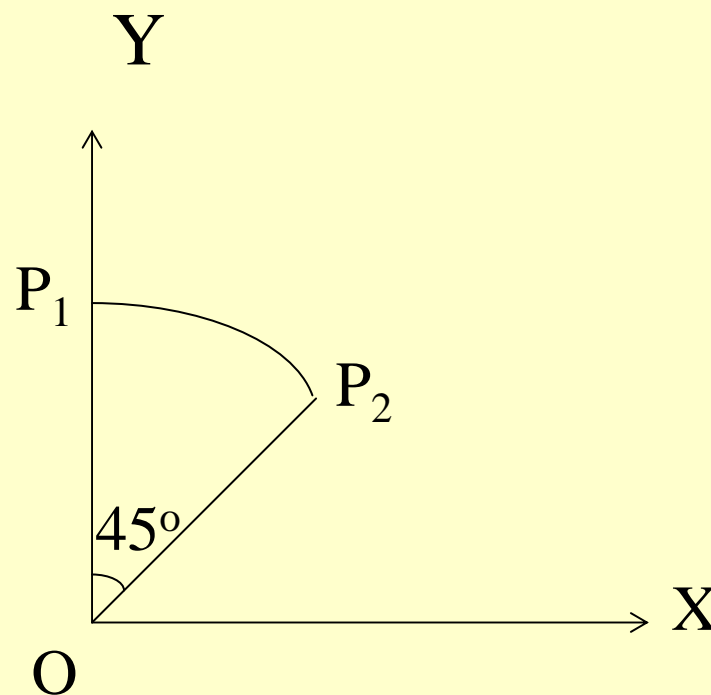


图3.4 第1象限的八分之一圆弧

- 此算法在每一步都选择一个离开理想圆周最近的点 $P_i(x_i, y_i)$, 使其误差项

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

在每一步都是极小值。

- 设 $P_{i-1}(x_{i-1}, y_{i-1})$ 已被选定为最靠近圆弧的点, 下一步 $x = x_{i-1} + 1$ 时, 参见图3.5, 要决定T还是S更接近理想的圆弧, 令

$$D(S) = (x_{i-1} + 1)^2 + y_{i-1}^2 - R^2$$

$$D(T) = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - R^2$$

- 显然, $|D(S)| \geq |D(T)|$ 时, 应该取T点; 反之则取S点。即若令

$$d_i = |D(S)| - |D(T)|$$

则 $d_i \geq 0$, 选择T点, 否则选取S点。

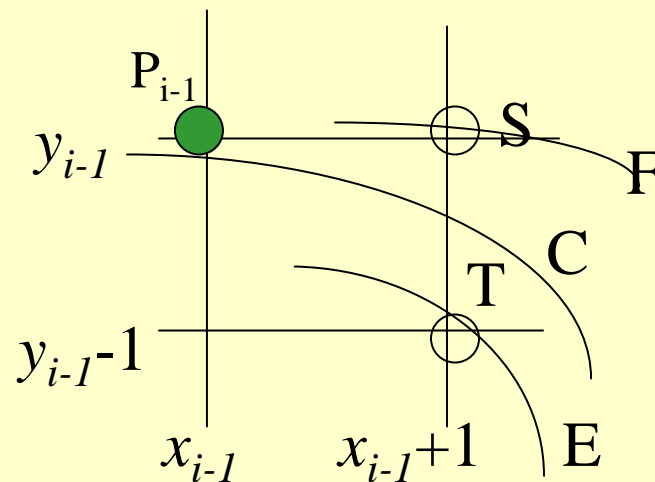


图3.5 Bresenham画圆算法的取点示意图

- 改用 $d_i = D(S) + D(T)$ 作为判别条件

$$d_i = D(S) + D(T) = 2(x_{i-1} + 1)^2 + y_{i-1}^2 + (y_{i-1} - 1)^2 - 2R^2$$

$d_i(R)$ 是递减函数。(d_i=0是到S和T近似相等的园C)

$d_i \geq 0$, 弧AB在园C的下方, 选择T点; 否则选取S点。

- 推导一个递推公式, 简化 d_i 的计算

$$x_0 = 0; y_0 = R; x_1 = x_0 + 1;$$

$$\begin{aligned} d_1 &= D(S) + D(T) = (1^2 + y_0^2 - R^2) + (1^2 + (y_0 - 1)^2 - R^2) \\ &= 3 - 2y_0 = 3 - 2R; \end{aligned}$$

$$\begin{aligned} d_i &= (x_i^2 + y_{i-1}^2 - R^2) + (x_i^2 + (y_{i-1} - 1)^2 - R^2) \\ &= 2x_i^2 + 2y_{i-1}^2 - 2y_{i-1} - 2R^2 + 1 \end{aligned}$$

$$\begin{aligned} d_{i+1} &= (x_i + 1)^2 + y_i^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 \\ &= 2x_i^2 + 4x_i + 2y_i^2 - 2y_i - 2R^2 + 3 \end{aligned}$$

- 结论:

$$d_1 = 3 - 2R;$$

如果 $d_i < 0$, 选 $P_i(x_i, y_{i-1})$, 将 $y_i = y_{i-1}$ 代入求 d_{i+1} :

$$d_{i+1} = d_i + 4x_i + 2 = d_i + 4x_{i-1} + 6$$

如果 $d_i \geq 0$, 选 $P_i(x_i, y_{i-1} - 1)$, 将 $y_i = y_{i-1} - 1$ 代入求 d_{i+1} :

$$d_{i+1} = d_i + 4x_i - 4y_{i-1} + 6 = d_i + 4(x_{i-1} - y_{i-1}) + 10$$

- 推广。利用圆周对坐标原点(圆心)及坐标轴的对称性, 就可以将 45° 圆弧扩展到整个圆周。如图3.6, 如果点 (x, y) 在圆周上, 则与之对称的另外7个点也在圆周上, 它们的坐标如图所示。

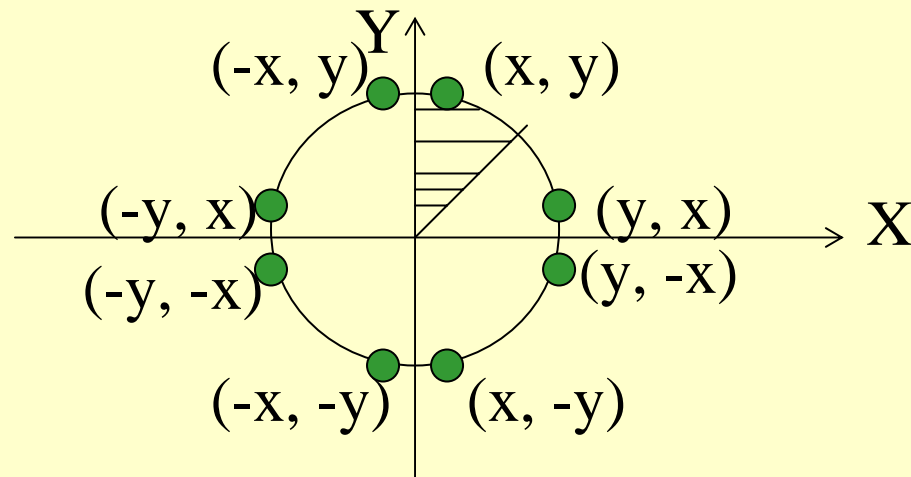


图3.6 圆周上的8个对称点

3) 算法描述

```
Procedure arc(r: integer);  
var x,y,d: integer;  
begin  
    x:=0; y:=r; d:=3-2*r;  
    while x<y do {  
        plot(x,y);  
        if d<0 then d:=d+4*x+6           //y不变  
            else {d:=d+4*(x-y)+10;       //y减1  
                y:=y-1};  
        x:=x+1;  
    }  
    if(x=y) then plot(x,y);  
end
```

4) 算法特点

Bresenham圆弧算法是最有效的算法之一,生成圆时利用第一象限的上八分之一圆弧对称扩展。选择点时所用判别式是递推表达式,仅用加、减和移位即可计算,算法效率高。

3.2.2 正负法（逐点比较法）

- 1) 算法思想
- 2) 算法原理
- 3) 算法特点

1) 算法思想

- 首先,区分不同象限的圆弧;
- 然后,选定圆弧起点后,在输出圆弧的过程中,根据当前点的位置与理想圆弧的关系和所在象限,决定下一步的走向,每次只在一个方向(X或 Y)走步取点;
- 这样一步步地逼近产生应显示的圆弧。

2) 算法原理

- 设要显示的圆的圆心在 $C(x_c, y_c)$, 半径为 R 。
- 以图3.7中第一象限画圆弧AB为例。

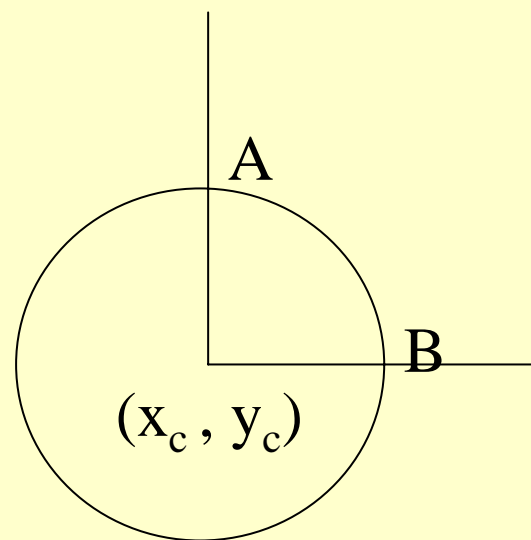


图3.7 正负法原理示意图

- 令 $F(x,y) = (x-x_c)^2 + (y-y_c)^2 - R^2$
 则圆的方程为: $F(x,y) = 0$
 当点 (x, y) 在圆内时, 有 $F(x,y) < 0$; 当点 (x, y) 在圆外时, 有 $F(x,y) > 0$ 。
- 取 $P_0(x_0, y_0)$ 为A点, 即 $x_0 = x_c, y_0 = y_c + R$ 。求得 $P_i(x_i, y_i)$ 后找下一点 P_{i+1} 的原则为:
 当 $F_i(x_i, y_i) < 0$ 时, 向+X方向(圆外方向)走步, 取

$$x_{i+1} = x_i + 1, y_{i+1} = y_i;$$
 当 $F_i(x_i, y_i) \geq 0$ 时, 向-Y方向(圆内方向)走步, 取

$$x_{i+1} = x_i, y_{i+1} = y_i - 1.$$
- $F(x,y)$ 时正时负, 这是算法名称来源

- 推导递推公式, 简化判别式

注意 $F(x_0, y_0)=0$;

当 $x_{i+1}=x_i+1, y_{i+1}=y_i$ (圆外方向走)时,

$$\begin{aligned} F(x_{i+1}, y_{i+1}) &= (x_i+1-x_c)^2 + (y_i-y_c)^2 - R^2 \\ &= (x_i-x_c)^2 + (y_i-y_c)^2 - R^2 + 2(x_i-x_c) + 1 \\ &= F(x_i, y_i) + 2(x_i-x_c) + 1 \end{aligned}$$

当 $x_{i+1}=x_i, y_{i+1}=y_i-1$ (圆内方向走)时,

$$\begin{aligned} F(x_{i+1}, y_{i+1}) &= (x_i-x_c)^2 + (y_i-1-y_c)^2 - R^2 \\ &= F(x_i, y_i) - 2(y_i-y_c) + 1 \end{aligned}$$

- 结论:

判别式初值 $F_0=0$;

当 $F_i < 0$ 时(在圆内), 选 $P_{i+1}(x_i + 1, y_i)$, 且

$$F_{i+1} = F_i + 2(x_i - x_c) + 1;$$

当 $F_i \geq 0$ 时(在圆外), 选 $P_{i+1}(x_i, y_i - 1)$, 且

$$F_{i+1} = F_i - 2(y_i - y_c) + 1.$$

- 算法描述

```
Procedure pnarc(r,xc,yc: integer);  
  var x,y,f: integer;  
  begin  
    x:=xc; y:=yc+r; f:=0;  
    while y>yc do {          //画 1/4 圆弧  
      plot(x,y);  
      if f>0 then { f:=f-2*(y-yc)+1;  
                    y:=y-1}    //向圆内走  
      else { f:=f+2*(x-xc)+1;  
             x:=x+1}    //向圆外走  
    }  
    if (y=yc) plot(x,y)  
  end
```

- 其他象限和不同走向的圆弧可类似画出
- 一般运算规律

对于逆圆的一、三象限及顺圆的二、四象限,沿运动走向的X绝对值是不断减小的,而Y绝对值是不断增加的,因而其运算规律相同;

对于顺圆的一、三象限,其X绝对值不断增加,Y绝对值不断减小,运算规律与逆圆的二、四象限相同。

3) 算法特点

- 计算机用正负法生成圆弧时运算只有加、减和移位(乘2)运算, 无乘除, 因此运算效率高, 这很适用于用硬件实现。
- 较之Bresenham算法, 正负法的运算更为简单, 但对于同一段圆弧而言, 由于正负法每次只是单向走步, 因而生成的点数比Bresenham算法生成的点数要多。

3.2.3 多边形逼近法

1) 原理

2) 求圆的内接正多边形的方法

1) 原理

- 当圆的内接多边形边数足够多时, 该多边形可以和圆接近到任意程度, 因此在允许的误差范围内(例如圆周和多边形之间的最大距离小于半个像素的宽度), 可用显示多边形代替显示圆。
- 显示多边形的边可以用Bresenham直线生成算法来实现。

2) 求圆的内接正多边形的方法

设要显示的圆的圆心为 $C(x_c, y_c)$, 半径为 r 。
如图3.8所示。

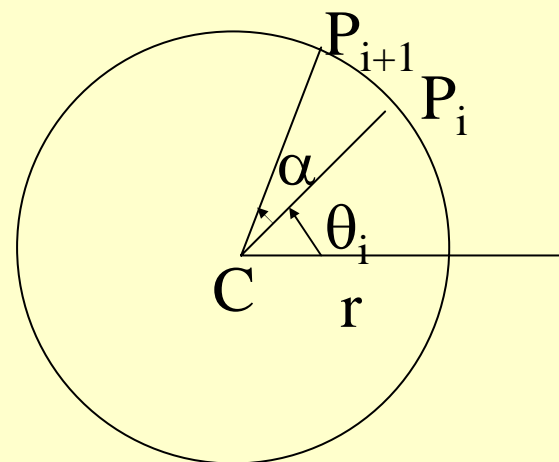


图3.8 用正多边形逼近圆弧

[方法1] 直接计算每个顶点坐标

- 设内接正多边形的一个顶点为 $P_i(x_i, y_i)$, CP_i 的幅角为 θ_i , 则

$$\begin{cases} x_i = x_c + r \cos \theta_i \\ y_i = y_c + r \sin \theta_i \end{cases}$$

- 分析：此方法的计算每次都包含三角函数以及乘法的运算，因而计算量较大。

[方法2] 利用递推公式计算顶点坐标

- 设多边形相邻顶点所对的圆心角为 α , 推导顶点 P_{i+1} 的坐标如下:

$$\begin{cases} y_{i+1} = y_c + r \sin(\theta + \alpha) = y_c + (x_i - x_c) \sin \alpha + (y_i - y_c) \cos \alpha \\ x_{i+1} = x_c + r \cos(\theta + \alpha) = x_c + (x_i - x_c) \cos \alpha - (y_i - y_c) \sin \alpha \end{cases}$$

- 分析: 在上述递推公式中, 因为 α 是常数, $\cos \alpha$ 和 $\sin \alpha$ 只要在开始时算一次, 这样求 P_{i+1} 的坐标需要作4次乘法。
- 另外, 凡是利用递推公式计算时要注意误差是否稳定, 以免误差多次积累后严重偏离真解。可以证明, 上述递推公式对初始值的误差是稳定的。
- 可用2个处理器流水线处理: 一个求点, 一个画图。

3.3 规则曲线的生成方法

3.3.1 规则曲线的一般生成方法

3.3.2 椭圆生成算法

3.3.3 抛物线生成算法

3.3.4 生成规则曲线时需注意的几点

3.3.1 规则曲线的一般生成方法

- 平面曲线可分为两大类：规则曲线(函数曲线)和不规则曲线(或称为自由曲线)。
- 规则曲线通常是指可以用函数直接描述的曲线，如圆、椭圆、双曲线、渐开线等；不规则曲线通常由一系列离散点(样本点)定义，不规则曲线的生成方法第6章讨论。
- 规则曲线生成的基本方法是将曲线离散化成小直线段，通过连接各直线段来逼近所要的曲线。

3.3.2 椭圆生成算法

椭圆是一种常见的二次曲线,下面介绍两种方法,说明选择合理的几何描述的重要性。

- 1) 标准方程法
- 2) 参数方程法

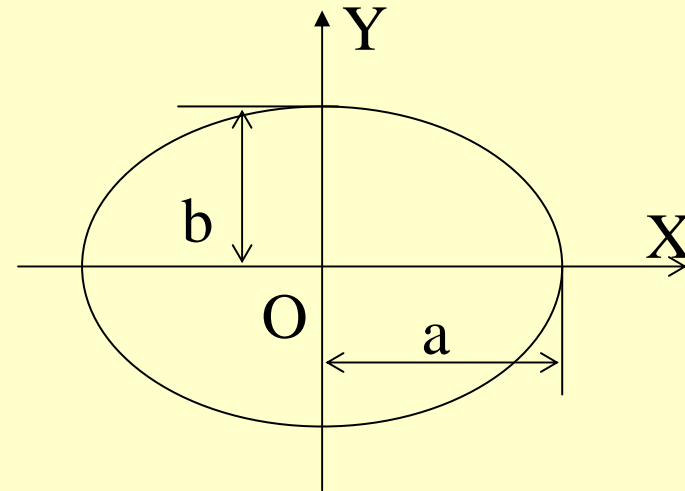


图3.9 椭圆

1) 标准方程法

- 椭圆的标准方程为：
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

可以解得：

$$y = \pm \frac{b}{a} \sqrt{a^2 - x^2} \quad (-a \leq x \leq a)$$

该式分别表示图3.9中上下两个半椭圆。

- 对每一个半椭圆, 可以通过以下步骤绘制:
 - (1) 选择x的步进增量为dx, 从x=-a,y=0出发;
 - (2) 计算x=x+dx时的相应y值, 并且将前一个点和(x, y)连接为一个直线段, 反复执行(2), 直至x=a为止.

- 分析

该方法明显的缺点是： x 坐标的步进增量 dx 被选定后,在整个过程中就是常数,但是曲率大的两端与曲率小的两侧的点的分布情况是不一样的。一个夸大的图形见图3.10,椭圆两端出现棱角。若 dx 取值减小,两端情况会好转,但是对于相对平滑的两侧而言,计算更多的点则有所浪费。

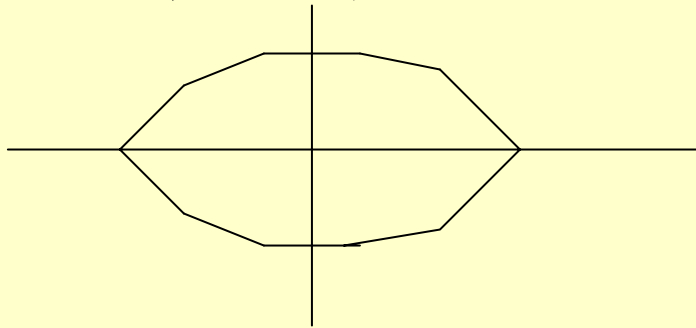


图3.10 步进增量 dx 拉大时生成的椭圆

2) 参数方程法

- 椭圆的参数方程为：
$$\begin{cases} x=a \cos t \\ y=b \sin t \end{cases} \quad (0 \leq t \leq 2\pi)$$
- 按以下步骤生成椭圆：
 - (1)选择幅角参数 t 的步进增量 dt ,计算 $t=0$ 时的点 (x, y)
 - (2)计算下一点坐标,即 $t=t+dt$ 时点 (x, y) ,并且与前一点连接为一个直线段,反复执行(2),直至 $t=2\pi$ 为止。
- 分析：

由椭圆的参数方程可以得到参数 t 有增量 dt 时,引起X和Y方向的增量 dx 和 dy 分别为

$$\begin{cases} dx = -a \sin t \cdot dt \\ dy = b \cos t \cdot dt \end{cases}$$

当t接近0或 π 时, 有 $|dx| \rightarrow 0$ 和 $|dy| \rightarrow b \cdot dt$, 这就是说, x几乎不增加的情况下, y有一定的增值, 这与椭圆两端所需要的趋势符合;

当t接近 $\pi/2$ 或 $3\pi/2$ 时, 有 $|dx| \rightarrow a \cdot dt$ 和 $|dy| \rightarrow 0$, 这就是说, y几乎不增加的情况下, x有一定的增值, 这与椭圆两侧所需要的趋势符合。

- 而且, 由于 $a > b$, 因此两侧的增值大于两端的增值, 做到了曲率大处点密, 曲率小处点稀。
t由0到 2π , 即可生成完整的椭圆。

3.3.3 抛物线生成算法

- 以坐标原点为顶点, 焦点在(0, p)的抛物线其标准方程为:

$$y=x^2/4p$$

在该表达式中, 自变量x的取值范围是任意的, 图形不封闭。但在实际绘图时, x的取值范围受图纸或者显示器上显示区域大小的限制。

- x的取值范围的确定

设显示区域限制在长为 XL 和高为 YH 范围内，
如图3.11所示。

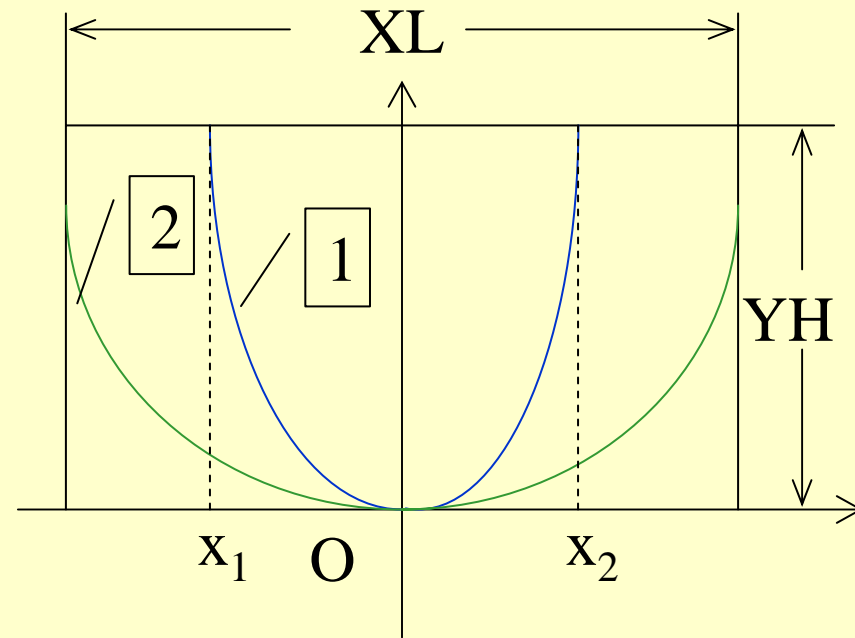


图3.11 确定 x 的取值范围

- 对于曲线2, 显然 x 的取值范围为 $[-XL/2, XL/2]$; 对于曲线1, 还必须计算相对于 $y=YH$ 时的 x_1 与 x_2 值, 这时 x 的取值范围为 $[x_1, x_2]$ 。
- 因此, 决定 x 的取值范围 $[x_1, x_2]$ 方法如下:
 - (1) 计算 $x=XL/2$ 时的 y 值;
 - (2) 若 $y \leq YH$, 则 $x_1=-XL/2, x_2=XL/2$;
否则, 将 $y=YH$ 代入抛物线方程解得 x_1 和 x_2 。
- 抛物线的生成
设定 x 的步进增量 dx , 从 $x=x_1$ 开始, 求出对应的 y 值, 直到 $x=x_2$ 为止。相邻点以直线段相连即可。

3.3.4 生成规则曲线时需注意的几点

- 首先, 建立对曲线生成有利的几何描述(数学表达式)。
- 用直线段逼近曲线时, 必须考虑逼近的精度, 选择合理的步进增量大小, 兼顾生成图形的质量和生成时效。
- 实际曲线总有一个预定的范围, 生成时注意不要发生坐标越界。

3.4 字符的生成方法

3.4.1 概述

3.4.2 矢量字符的存储和显示

3.4.3 点阵字符的存储和显示

3.4.1 概述

- 目前常用的字符有两种: ASCII码字符和汉字字符。另外,还有很多面向各种应用的符号,如电子线路中的电阻、电容符号;机械图纸中的加工精度、光洁度符号等。
- 1981年5月,我国国家标准局发布并实施的“信息、交换用汉字编码字符基本集”中,共收集字符7445个,其中国标一级汉字3755个,国标二级汉字3008个,其余符号682个。
- 常用的字符编码有国标(区位)码和电传(电报)码两种。

- 有两种基本的字符生成技术：
 - 一种是在计算机中用笔划(矢量)方式来表示, 然后通过扫描转换生成, 这是目前常用的方法, 生成的字符效果好, 但计算量大;
 - 另一种是在计算机内用位图(点阵)来表示, 存储在字符高速缓冲区(字符发生器)里, 显示时可以直接通过像素拷贝将其装入显示缓冲区中。这是传统的方法, 简单、速度快, 但不灵活。

3.4.2 矢量字符的存储和显示

- 1) 定义字符
- 2) 存储字符
- 3) 显示字符

1) 定义字符

用户定义矢量字符一般需要以下步骤：

- (1)写字模：字符写在一定大小的正方形网格(大小可选 16×16 , 32×32 , 64×64 , 128×128 等)中, 采集其每一笔划两端点相对网格左下角(0, 0)点的(x, y)坐标值。
- (2)确定字符代码：定义字符的编号(编号不能重复, 也不能越界)。
- (3)按一定结构保存字符各端点坐标及由前一点到此端点是否划线的标志。
- (4)将字模写入字符库中。

2) 存储字符

为节省存储空间, 查找迅速, 可采取以下措施:

- (1) 压缩存储: 如将每一端点的 x 、 y 坐标和划线标志3个信息用两个字节保存, x 、 y 坐标各占7位, 划线标志占一位。
- (2) 用定长记录存储字模: 这样可以加快查找速度。

3) 显示字符

- 在应用中, 用户必须指定字符的显示位置, 另外还可以按条件显示字符(字符尺寸、倾斜度、笔划的宽度等), 这些都可用坐标变换的方法实现。
- 显示过程包括:
 - (1)确定字符显示时字符界框的左下角点坐标(x,y);
 - (2)根据字符代码CODE, 取出该字符的存储记录;
 - (3)进行一系列坐标变换, 计算出该字符的每个端点在实际用户坐标系中的坐标, 并且根据端点间划线关系进行连接。

3.4.3 点阵字符的存储和显示

- 在光栅扫描显示系统中, 利用掩膜(mask)来定义字符, 保存在字符发生器中。所谓字符掩膜, 就是包含该字符的像素信息的一小块光栅。
- 当指定了字符掩膜的原点在显示缓存的对应位置(x_0, y_0)后, 就可将此字符掩膜中每个像素相对(x_0, y_0)平移后的值写入显示缓存, 从而在屏幕上显示此字符。
- 将字符掩膜相应的像素值置成背景色, 就可以擦除显示缓存中的该字符。

- 当字符写入显示缓存后，还可对字符掩膜进行修改，以获得不同字体或方向。

3.5 区域填充

3.5.1 平面多边形的阴影线填充

3.5.2 光栅图形的区域填充

3.5.3 图案填充

3.5.1 平面多边形的阴影线填充

为讨论方便起见,设多边形至多有一个内孔。

- 1) 多边形边框的表示
- 2) 阴影线的数学模型
- 3) 阴影线与多边形边框的求交运算
- 4) 交点处理及画阴影线段
- 5) 算法步骤

1) 多边形边框的表示

- 设多边形的总顶点数为 MN , 外环顶点数为 M , 则内孔顶点数 $N=MN-M$ ($N=0$ 时, 即无内孔)
- 定义二维数组 $P(2, MN)$ 存放多边形各顶点的 x 、 y 坐标。多边形的外环和内孔的顶点分别有序地排列在这个数组中。

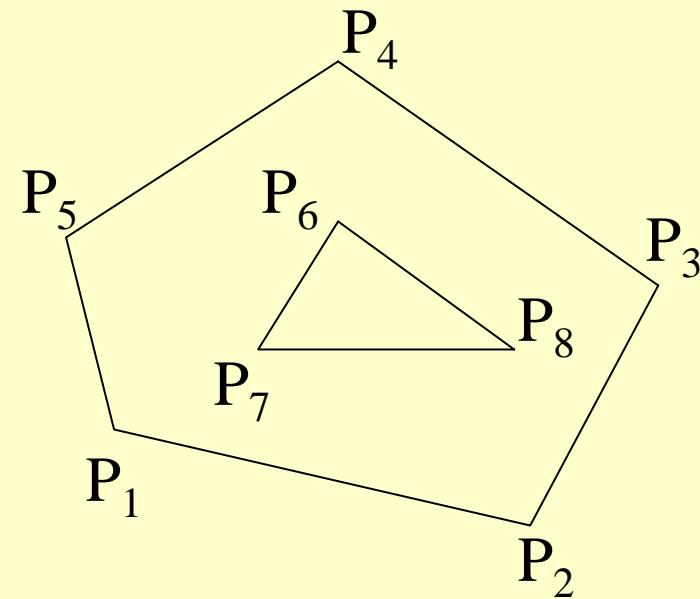


图3.12 带孔的平面多边形

2) 阴影线的数学模型

- 阴影线是一组平行的直线，通常用户指定它与X轴间的夹角 α 和相邻两阴影线之间的垂直间隔 h 。

- 阴影线的数学模型采用：

$$y=kx+b_j \quad (k=\tan\alpha)$$

- 截距的计算可以利用递推公式：

$$b_{j+1}=b_j+\Delta b$$

其中 $\Delta b=h/|\cos \alpha|$

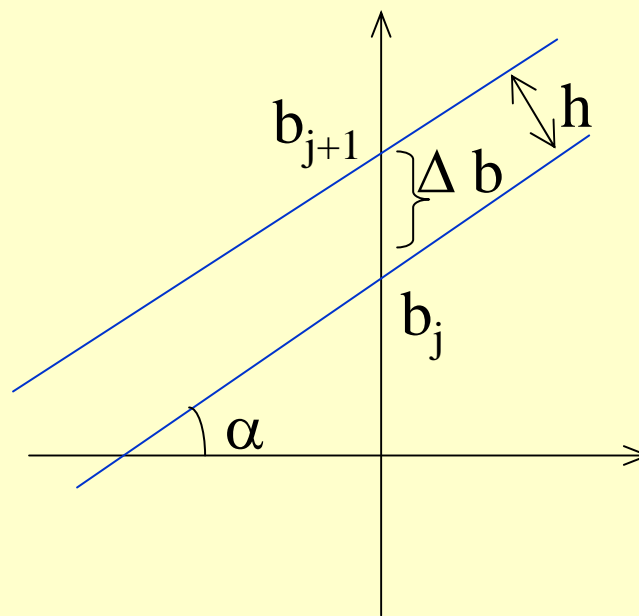
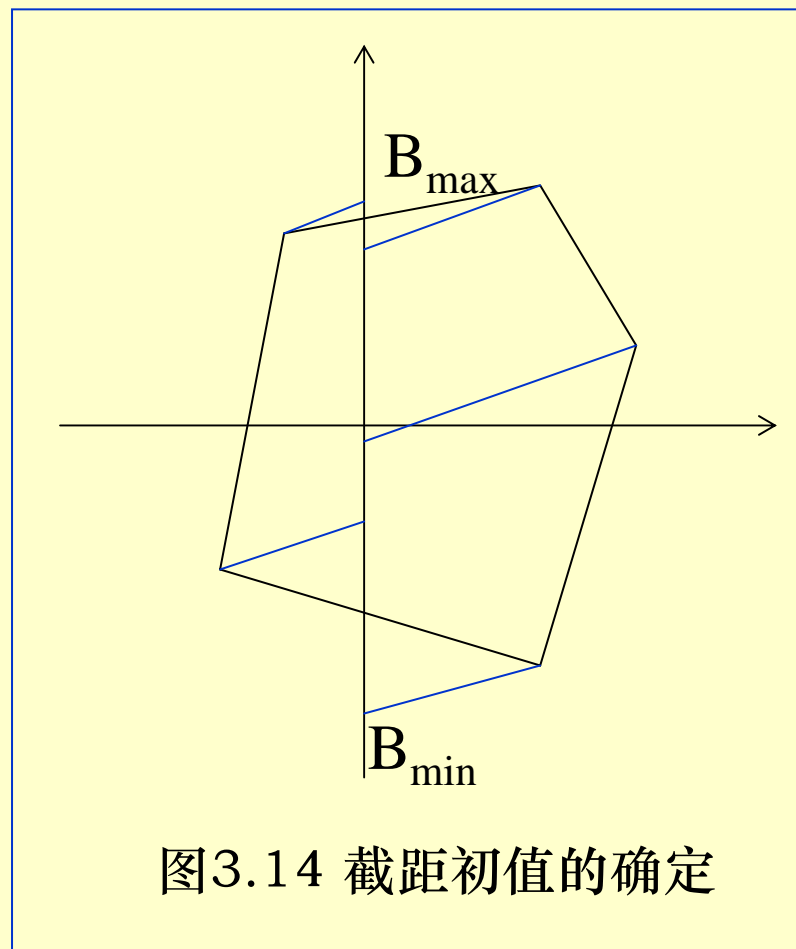


图3.13 截距差 Δb 的确定

- 截距初值 B_0 的确定：
经多边形外环各顶点按斜率 $\tan\alpha$ 引线，求出各引线的截距，设其最大值为 B_{\max} ，最小值为 B_{\min} ，则选取截距初值为：

$$B_0 = B_{\min}$$

所有填充的阴影线的截距范围是 (B_{\min}, B_{\max}) 。



3) 阴影线与多边形边框的求交运算

- 每一条阴影线只与多边形的部分棱边有交点, 设某条阴影线的截距为 b_j , 对于某一棱边而言, 设它的两个端点分别为 $P(x_p, y_p)$ 和 $Q(x_q, y_q)$, 从P和Q按阴影线斜率引线, 对应的截距为 b_p 和 b_q , 可用以下公式计算:

$$\begin{cases} b_p = y_p - kx_p \\ b_q = y_q - kx_q \end{cases}$$

当满足 $\min(b_p, b_q) \leq b_j < \max(b_p, b_q)$ 时, 该阴影线与该棱边有交点。

- 若阴影线与棱边的交点是 (x_l, y_l) , 则

$$\begin{cases} x_l = \frac{x_p y_q - y_p x_q + b_j (x_q - x_p)}{(y_q - y_p) - k(x_q - x_p)} \\ y_l = kx_l + b_j \end{cases}$$

- 因为不同的阴影线都需判断与每条棱边是否有交点,以决定相应的阴影线在填充时开始/终止于何处,因而有必要事先计算出所有棱边的两个端点按阴影线斜率引线的对应截距,方便每一次的判断,避免重复计算。
- 鉴于上述原因,定义数组**B(2,MN)**记录各棱边两端点按阴影线斜率引线产生的截距,并将两者中的较小值放在第一行。

4) 交点处理及画阴影线段

对每一条阴影线作以下处理：

- a. 将它与各棱边交点按X或者Y坐标值升序排列；
- b. 按顺序, 产生奇数点到偶数点之间的线段。

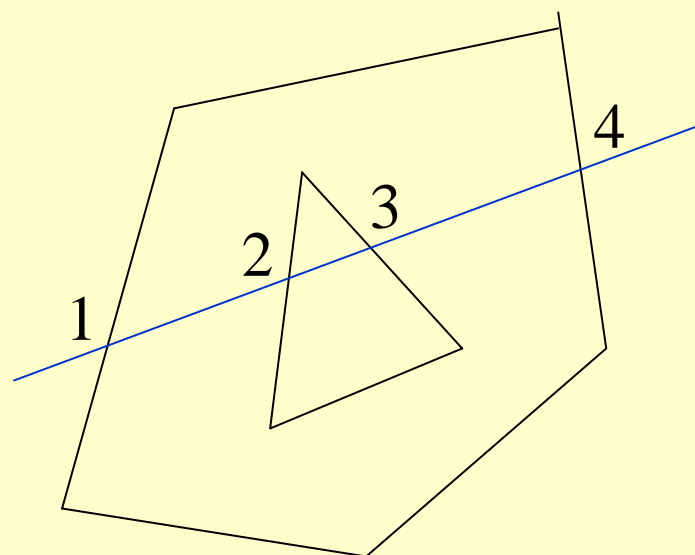


图3.15 交点排序

5) 算法步骤

- 1). 对于MN条棱边, 计算每一条棱边 L_i 的两个端点分别按阴影线斜率 $k=\tan\alpha$ 引线得到的截距, 其中较小值放在 $B(1,i)$ 中, 较大值放在 $B(2,i)$ 中。
- 2). 求出 $B_{\min}=\min B(1,i)$, $B_{\max}=\max B(2,i)$, $i=1,2,\dots,MN$
- 3). 取第一条阴影线截距为 $b=B_{\min}+\Delta b$, ($\Delta b=h/|\cos\alpha|$)
- 4). 初始化存放阴影线与各棱边交点的数组 $D(2,MN)$;
判断当前阴影线与各棱边是否有交点, 若存在则计算出交点坐标并存入D数组;
按X/Y坐标排列D数组中的交点并生成阴影线段;
取下一条阴影线 $b=b+\Delta b$, 若 $b\neq B_{\max}$ 转4)继续。

3.5.2 光栅图形的区域填充

3.5.2.1 相关概念

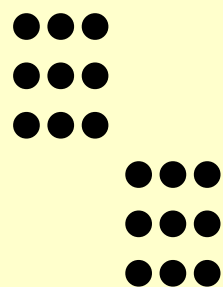
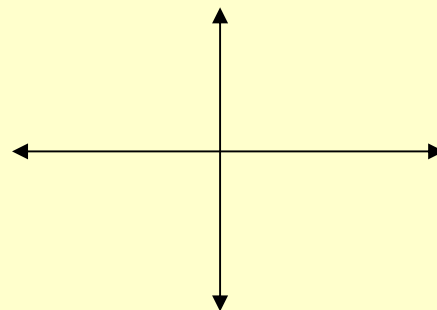
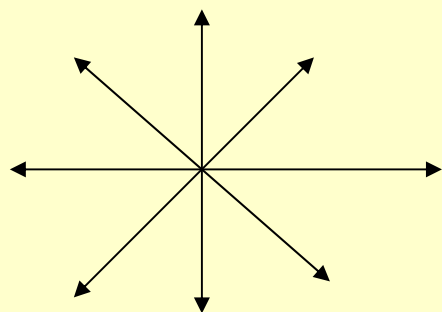
3.5.2.2 种子填充算法

3.5.2.3 多边形扫描转换算法

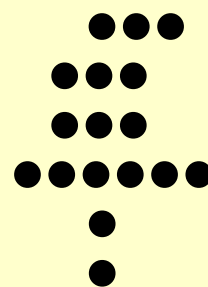
3.5.2.1 相关概念

- 区域：指一组相邻而又相连的像素，这些像素具有同样的属性。
- 区域的定义有两种方式：
 - 内点定义区域：区域内所有的像素具有相同的颜色/亮度值。
 - 边界定义区域：边界部分的像素具有相同的颜色/亮度值，区域内的像素具有不同于边界的颜色/亮度值。

- 区域的像素连通方式可以分为两类：
 - 4连通区域：取区域中的任何两个像素，从一像素出发，通过上、下、左、右4种运动，只经过该区域的点可以达到另一像素。
 - 8连通区域：取区域中任何两个像素，从一像素出发通过上、下、左、右、两条对角线方向共8种运动，只经过该区域的点可达到另一像素。
- 4连通区域和8连通区域的关系：
 - 4连通区域是8连通区域的一种特殊情况。4连通区域的边界必定是8连通式的；8连通区域的边界必定是4连通式的。



8连通区域



4连通区域

3.5.2.2 种子填充算法

- 1) 漫水法
- 2) 边界填充算法
- 3) 扫描线种子填充算法

1) 漫水法

- 适用对象：内点定义区域，把区域内的原像素值改变成另一种像素值。
- 算法步骤
 - a. 选择种子(x,y):在区域内且该像素值是原始值；
 - b. 改变种子像素值为新值；
 - c. 对该种子的 4/8 连通像素递归调用本算法。

(注: 8 连通区域填充算法适用于4连通区域,反之不然)

4连通区域(内点表示)填充漫水算法

```
Procedure flood-fill-4(x,y,old-color,new-color:integer)
begin
  if getpixel(framebuffer,x,y)=old-color
  then begin
    setpixel(framebuffer,x,y,newcolor);
    flood-fill-4(x, y+1, old-color,new-color);
    flood-fill-4(x, y-1, old-color,new-color);
    flood-fill-4(x+1, y, old-color,new-color);
    flood-fill-4(x-1, y, old-color,new-color);
  end
end
```

2) 边界填充算法

- **适用对象:**边界定义区域,把边界包围的区域内的原像素值改变成另一种像素值。
- **算法步骤**
 - a. 选择种子(x,y): 和边界值比较,确认该像素不属于边界;在区域内且该像素的值是原始值;
 - b. 改变种子像素值为新值;
 - c. 对该种子的 4/8 连通像素递归调用本算法。

4连通区域(边界表示)填充漫水算法

```
Procedure flood-fill-4(x,y,boundary-color,new-color:integer)
begin
  if getpixel(framebuffer,x,y)<>boundary-color and
    getpixel(framebuffer,x,y)<>new-color
  then begin
    setpixel(framebuffer,x,y,newcolor);
    flood-fill-4(x, y+1, boundary-color,new-color);
    flood-fill-4(x, y-1, boundary-color,new-color);
    flood-fill-4(x+1, y, boundary-color,new-color);
    flood-fill-4(x-1, y, boundary-color,new-color);
  end
end
```


3.5.2.3 多边形扫描转换算法

- 1) 适用对象
- 2) 基本原理
- 3) 利用边的连贯性简化求交点计算
- 4) 算法中的数据结构
- 5) 算法步骤
- 6) 实例
- 7) 算法特点

1) 适用对象

- 扫描转换填充算法适用于规则边界的区域。通常是将由顶点定义的多边形区域, 其内部用预期的像素值予以填充, 因此称为多边形的扫描转换。

2) 基本原理

- 这种算法不是孤立地去测试平面上的点是否在一个多边形之内, 而是利用一条扫描线上的像素存在着相关性这一特征。
- 在图3.16中, 对于 $y=8$ 这条扫描线, 可以求出交点A、B、C、D, 在AB和CD区段内的像素都位于多边形之内, 因此可以按每一条扫描线处理:
 - a. 与多边形各棱边求交, 若为水平边则跳过;
 - b. 交点按X坐标升序排列;
 - c. 填充每一对交点之间的所有像素。

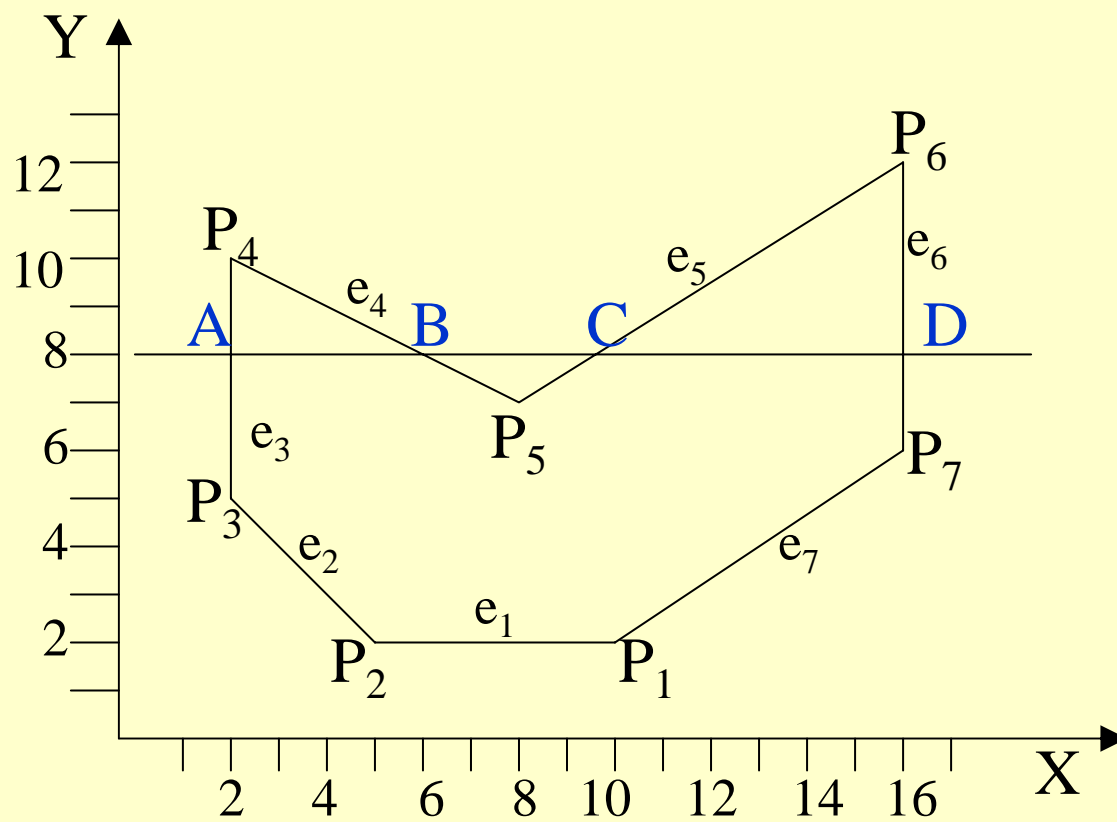


图3.16 多边形和扫描线

3) 利用边的连贯性简化求交点计算

- 每一条扫描线与多边形每条棱边求交点, 计算量是很大的。
- 设扫描线 $y=j$ 与多边形的某条棱边PQ的交点为 (x_j, j) , 若PQ与扫描线 $y=j+1$ 仍有交点存在, 则交点的X坐标应为:

$$x_{j+1} = x_j + 1/m$$

其中, m 是该棱边的斜率。

- 利用这种边的连贯性, 交点坐标可以简单递推得到。

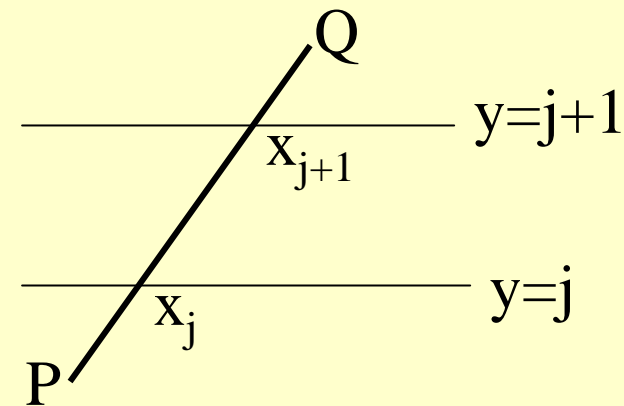


图3.17 简化交点计算

4) 算法中的数据结构

- **边的分类表ET**: 指针数组, 记录对多边形的所有棱边(水平边除外), 按下端点的纵坐标 y 的分类。下端点的纵坐标 $y=j$ 的边归入第 j 类, 有多少条扫描线, 就设多少类。同类中的边构成一个链。
- 链上的边元素由以下4个域组成:
 - y_{\max} : 该棱边的上端点的 y 坐标;
 - x : 该棱边的下端点的 x 坐标;
 - Δx : 该棱边的斜率 m 的倒数;
 - $next$: 指向下一条棱边的指针;同一类中, 边元素按 x (x 相同时按 Δx)值递增的顺序排列。

- 边的活化链表AEL：记录当前扫描线与棱边的交点序列。初值为空,在处理过程中利用ET表和求交点的递推关系不断刷新。
- 链表AEL的边元素由以下4个域组成：
 - y_{\max} ：该棱边的上端点的y坐标；
 - x：该棱边与当前扫描线交点的x坐标；
 - Δx ：该棱边的斜率m的倒数；
 - next：指向下一条棱边的指针。

5) 算法步骤

- a. 非极值奇点的预处理;
(极值奇点=2个交点; 非极值奇点=1个交点)
- b. 建立边的分类表ET;
- c. 取扫描线初始值 y =ET表中所列的最小 y 坐标值;
- d. 边的活化链表AEL初始化, 使其为空;
- e. 重复下列操作, 直至ET表和AEL表都变成空:
 - e1. 把ET表中纵坐标为 y 的链取下, 与AEL表合并, 并保持AEL表中元素按 x (x 相同时, 按 Δx)域值升序排列;
 - e2. 对于当前扫描线 y , 从左到右, 将AEL表中元素两两配对, 按每对两个 x 域定义的区段填充所需要的像素值。

- e3. 将AEL表中满足 $y_{\max}=y$ 的元素删除；
- e4. 对于仍留在AEL表中的元素, 求下一条扫描线与边的交点, 即 x 域累加 Δx : $x=x+\Delta x$;
- e5. 取下一条扫描线作为当前扫描线: $y=y+1$ 。

6) 实例(以图3.16所示多边形为例)

- 非极值奇点预处理

该多边形有两个非极值奇点 P_3 和 P_7 ，处理后各边的两个端点坐标如下：

e_1 : (10,2) 和 (5,2) 水平边,不考虑

e_2 : (5,2) 和 (2,5) $\Delta x = -1$

e_3 : (2,6) 和 (2,10) $\Delta x = 0$

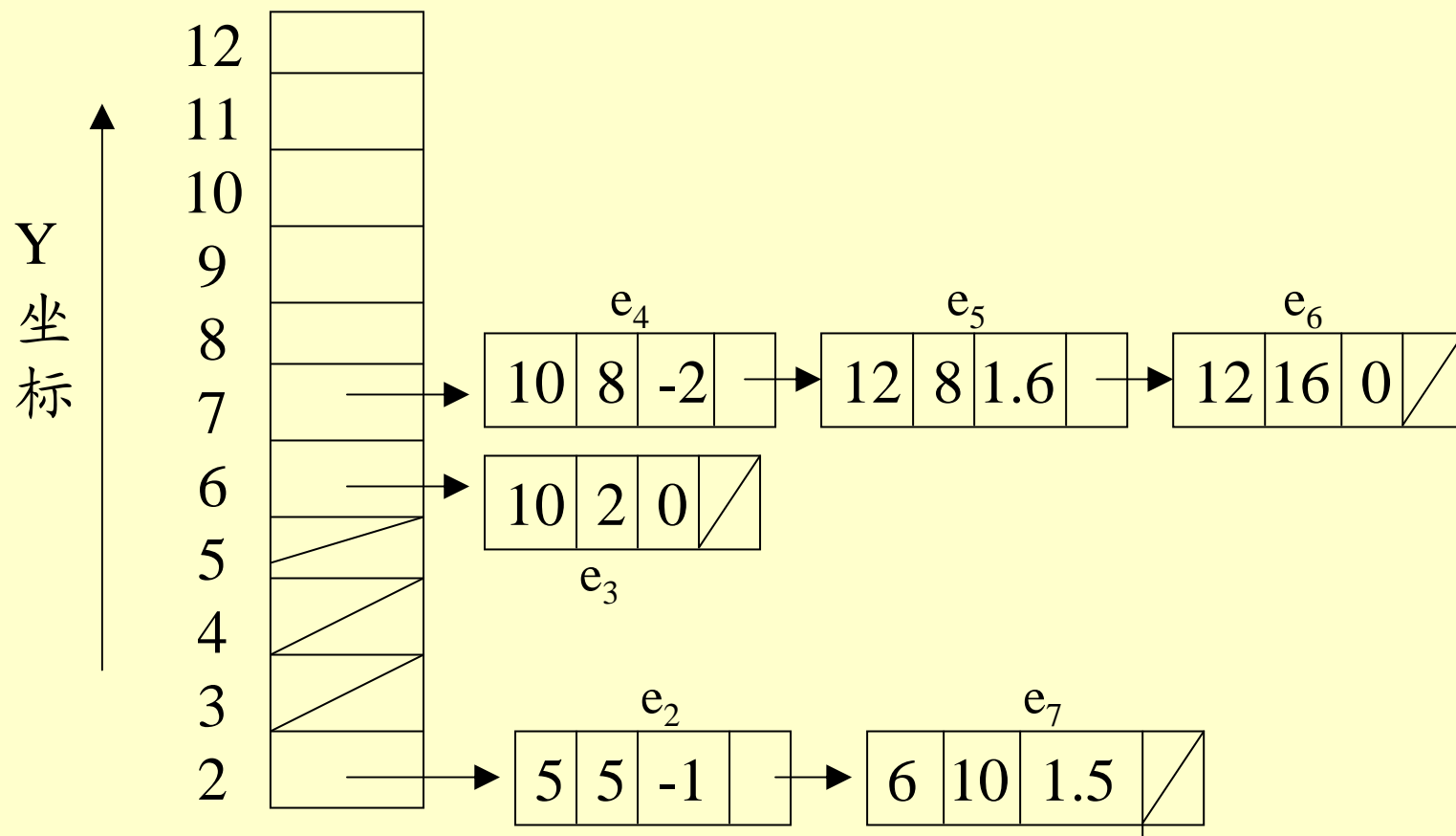
e_4 : (2,10) 和 (8,7) $\Delta x = -2$

e_5 : (8,7) 和 (16,12) $\Delta x = 1.6$

e_6 : (16,12)和 (16,7) $\Delta x = 0$

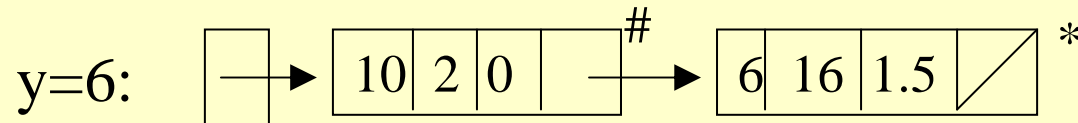
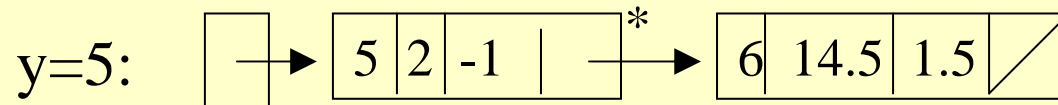
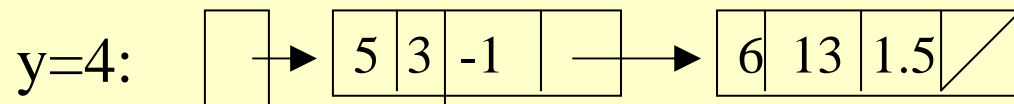
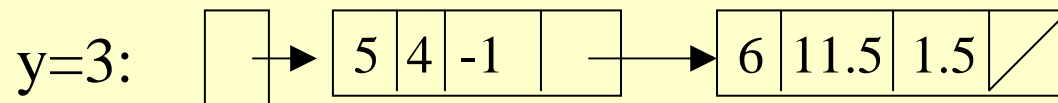
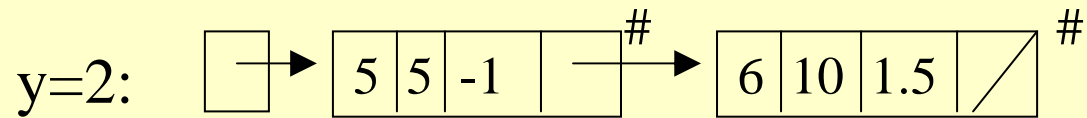
e_7 : (16,6) 和 (10,2) $\Delta x = 1.5$

- 建立边的分类表ET



(Ymax, X, Δx , next)

- AEL在不同扫描线y值时的状态



.....

注：标有*的边在填充处理之后应删除；标有#的边是新插入的。

7) 算法特点

- 本算法的数据结构和程序结构都比较复杂,但由于它充分利用了扫描线和边的相关性,避免了反复求交点运算,因此是一个效率较高的填充算法。

- 多边形扫描转换与区域填充两种着色法的比较:

扫描转换是将多边形顶点表示转换成点阵;区域填充只改变区域的颜色,不改变区域表示方法。(思想)

对多边形扫描转换中每条扫描线与多边形交点个数是偶数,不封闭的边界是允许的;而区域填充中边界必须为封闭的。

(边界要求)

区域填充要求指定区域一种子点,然后从这点开始对区域进行着色;多边形扫描转换无此要求。(出发点)

3.5.3 图案填充

- 图案填充是用一个图案模式来填充一个给定的区域，它是针对光栅扫描系统一种填充方式。
- 图案填充的两个主要过程是：
 - 1)定义图案
 - 2)填充区域

1) 定义图案

一个图案模式P通常定义为较小的 $n \times n$ 的像素阵列。 $P_{ij}(i=0,1,\dots,n-1; j=0,1,\dots,n-1)$ 代表在模式 (i,j) 处的颜色/亮度值。

2) 填充区域

在扫描线转换填充算法中, 增加一个相应的控制机构, 使之实际填充像素的颜色/亮度值是从图案模式中提取出来即可。

图案参照点的选择一般有如下两种方案:

- **相对定位方式:** 参照点选择在被填充区域的包围长方形的左下角 (x_{\min}, y_{\min})

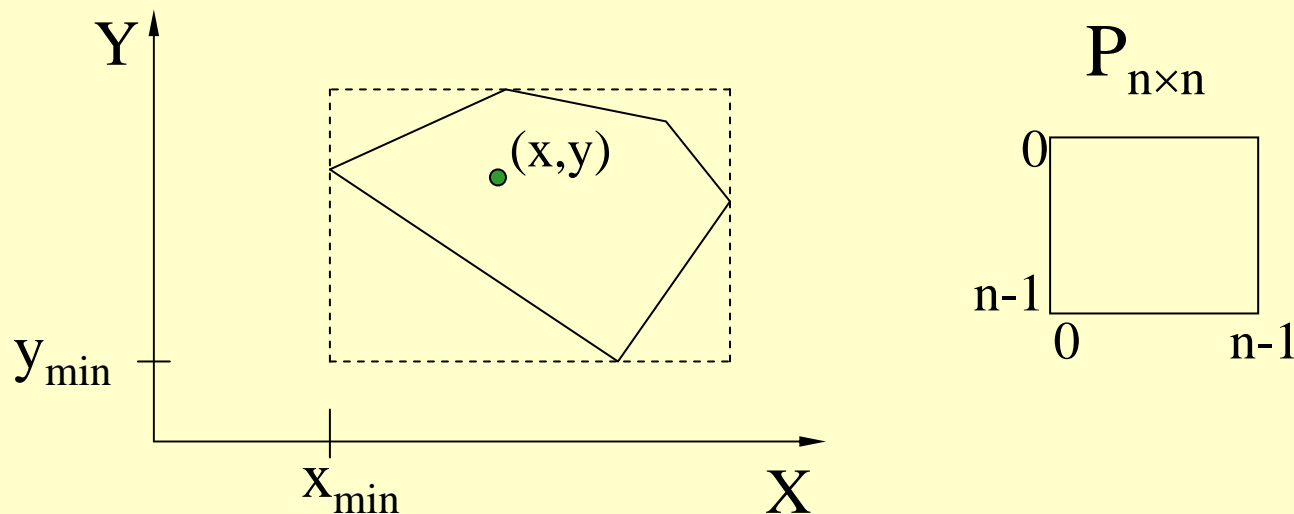


图3.18 填充图案的区域与模式示意

参照点与模式P的左下角元素($n-1, 0$)相对应,

则填充区域内的点(x,y)的像素值可从模式(i,j)处取得:

$$\begin{cases} i=(x-x_{\min}) \bmod n \\ j=(n-1)-[(y-y_{\min}) \bmod n] \end{cases}$$

这种方式,对于几何性较强的图案,即使填充区域移动,填充图案的视觉效果保持不变。

- **绝对定位方式:** 参照点选择在屏幕坐标原点。

屏幕坐标原点(0,0)与模式P的左下角元素(n-1,0)相对应,则填充区域内的点(x,y)的像素值可从模式(i,j)处取得:

$$\begin{cases} i = x \bmod n \\ j = (n-1) - (y \bmod n) \end{cases}$$

这种方式, 对于几何性较强的图案, 填充区域移动, 视觉效果可能是有差异的; 但是如果图案的几何方向性不是很强的话, 对视觉效果影响不大.

- 这种方式的好处在于提高了计算的效益, 而且用相同图案模式填充相邻或重叠区域时不会有接缝出现。

作业

1. 设计和实现一个图形函数库，具有绘制直线段、任意圆弧、椭圆弧、多边形区域的阴影填充和颜色填充等功能。（仅调用画点函数）

Windows API: `setpixel(hdc,x,y,color)`

第4章 图形变换与裁剪

内容：二维、三维图形的矩阵变换，三维图形的投影、透视变换，窗口到视图区的变换，裁剪

目的：让学生掌握基本的图形变换方法

本章内容

4.1 二维图形变换

4.2 三维图形变换

4.3 三维图形的投影变换

4.4 窗口到视区的变换

4.5 裁剪

4.1 二维图形变换

4.1.1 齐次坐标

4.1.2 变换矩阵

4.1.3 二维图形的几何变换

4.1.1 齐次坐标

- 用 $n+1$ 维向量表示 n 维点坐标的方法称为齐次坐标表示法。
- 例如,平面上一点 (x,y) 可以用齐次坐标 (hx, hy, h) ($h \neq 0$)来表示。
- 齐次坐标不是唯一的, h 可以为任意不为0的比例系数。但若用 h 去除齐次坐标各分量, 即得到 $(x,y,1)$, 这却是唯一的。
- 通常把 $h=1$ 的齐次坐标称为规格化齐次坐标, 而把使 $h=1$ 的变换方法称为齐次坐标规格化。

- 采用齐次坐标, 即是将 n 维空间的几何问题转换到 $n+1$ 维空间中去解决。从而找到一种统一的方法来处理各种图形的简单变换。
- 采用齐次坐标, 还能十分完美地表示 n 维空间的无穷远点。 $(x_1, x_2, x_3, \dots, x_n, 0)$ 即表示 n 维空间中的无穷远点。

4.1.2 变换矩阵

- 设二维平面上有一点(x,y), 经图形变换后成为另一点(x',y'), 则可用向量(x,y)乘上

一个变换矩阵 $\begin{bmatrix} a & d \\ b & e \end{bmatrix}$ 得:

$$[x' \ y'] = [x \ y] \begin{bmatrix} a & d \\ b & e \end{bmatrix} = [ax+by \ dx+ey]$$

- 若用齐次坐标表示, 则有

$$[x' \ y' \ u] = [x \ y \ 1] \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

$$= [ax+by+c \quad dx+ey+f \quad gx+hy+i]$$

我们称 $\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$ 为二维变换矩阵, 它可以分为

4个子矩阵: 其中 $\begin{bmatrix} a & d \\ b & e \end{bmatrix}$ 对图形进行比例、旋转、

对称、错切等变换; $[c \ f]$ 对图形进行平移变换;

$\begin{bmatrix} g \\ h \end{bmatrix}$ 对图形作投影变换; $[i]$ 对整体图形作比例变换。

4.1.3 二维图形的几何变换

- 1) 平移变换
- 2) 比例变换
- 3) 对称变换
- 4) 旋转变换
- 5) 错切变换
- 6) 复合变换

1) 平移变换

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ c & f & 1 \end{bmatrix} = [x+c \ y+f \ 1]$$

见图4.1(a)。

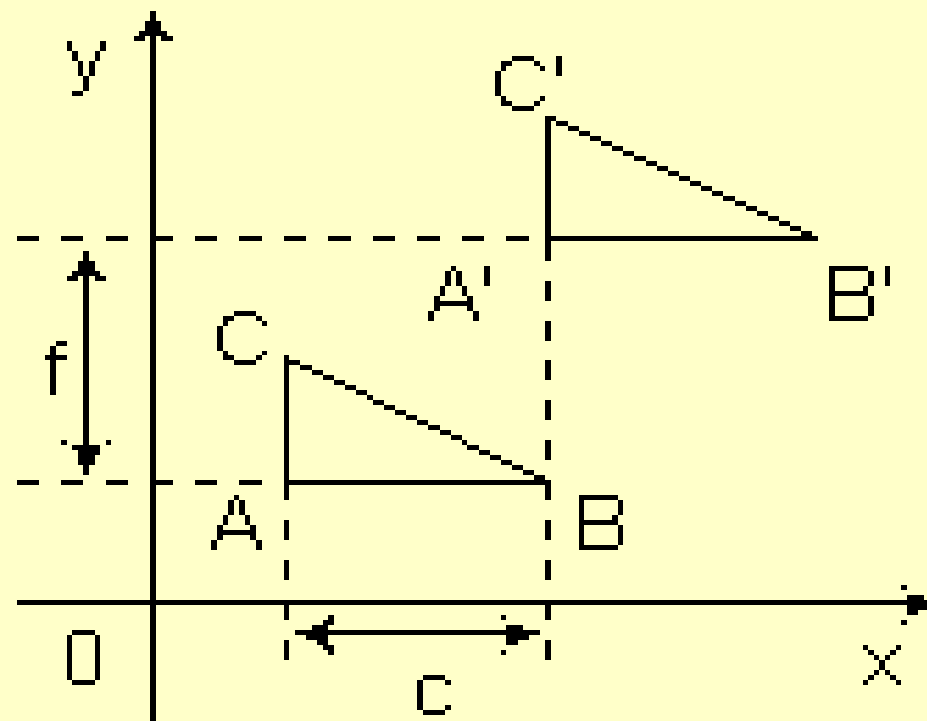


图4.1 平面图形二维几何变换
(a) 平移变换

2) 比例变换

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} a & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & 1 \end{bmatrix} = [ax \ ey \ 1]$$

1. 当 $a=e=1$ 时, 为恒等变换, 即图形不变;
2. 当 $a=e>1$ 时, 图形沿两个坐标轴方向等比例放大, 见图4.1(b);
3. 当 $a=e<1$ 时, 图形沿两个坐标轴方向等比例缩小, 见图4.1(c);
4. 当 $a \neq e$ 时, 图形沿两个坐标轴方向非均匀比例变换, 见图4.1(d);

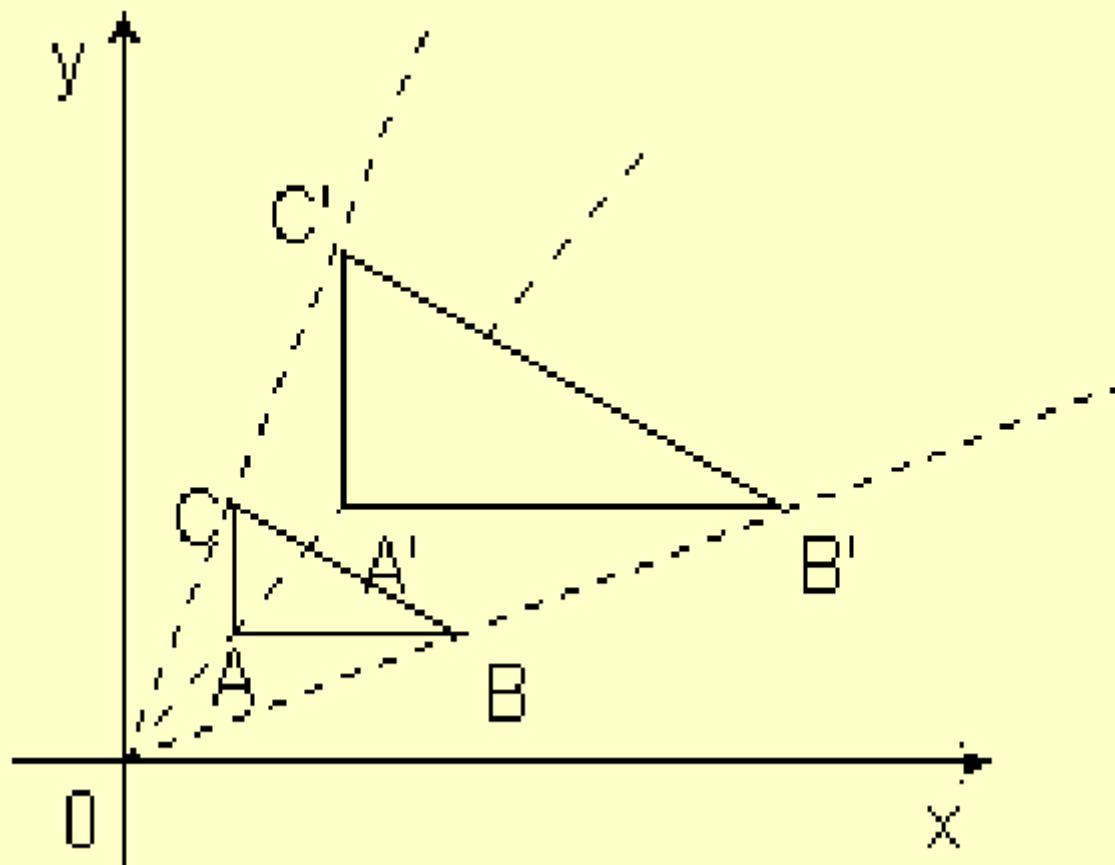


图4.1 平面图形二维几何变换
(b)比例变换 ($a=e>1$)

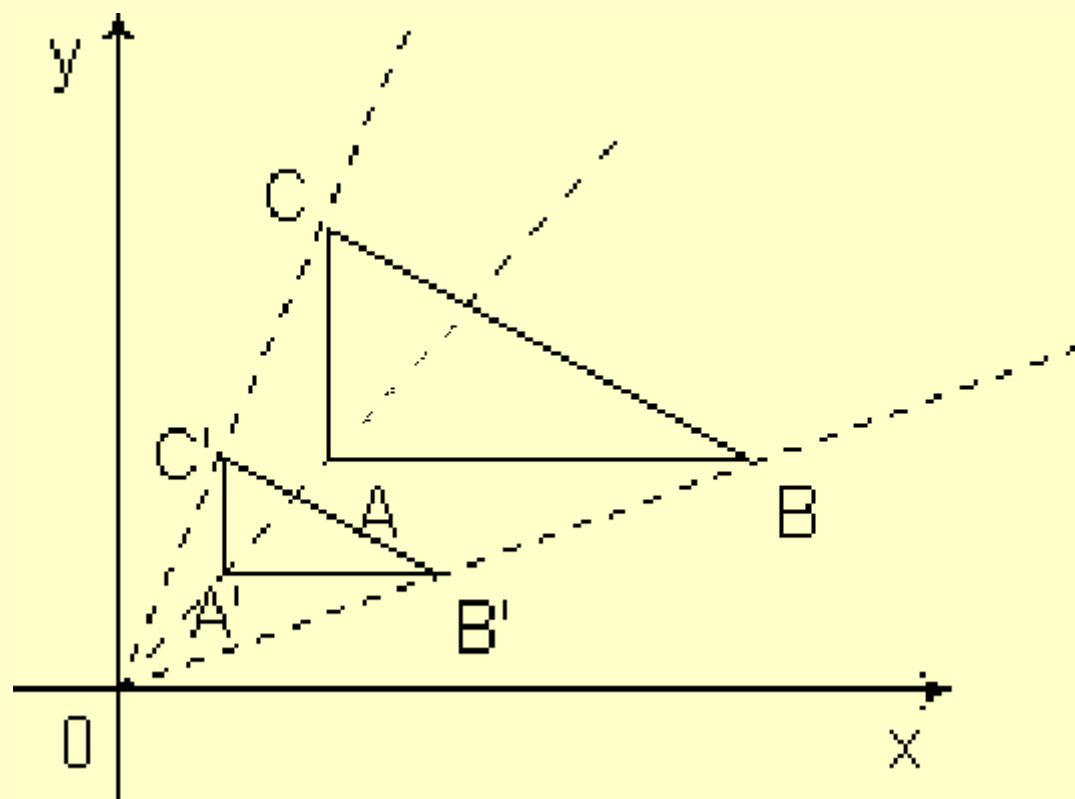


图4.1 平面图形二维几何变换
(c)比例变换($a=e<1$)

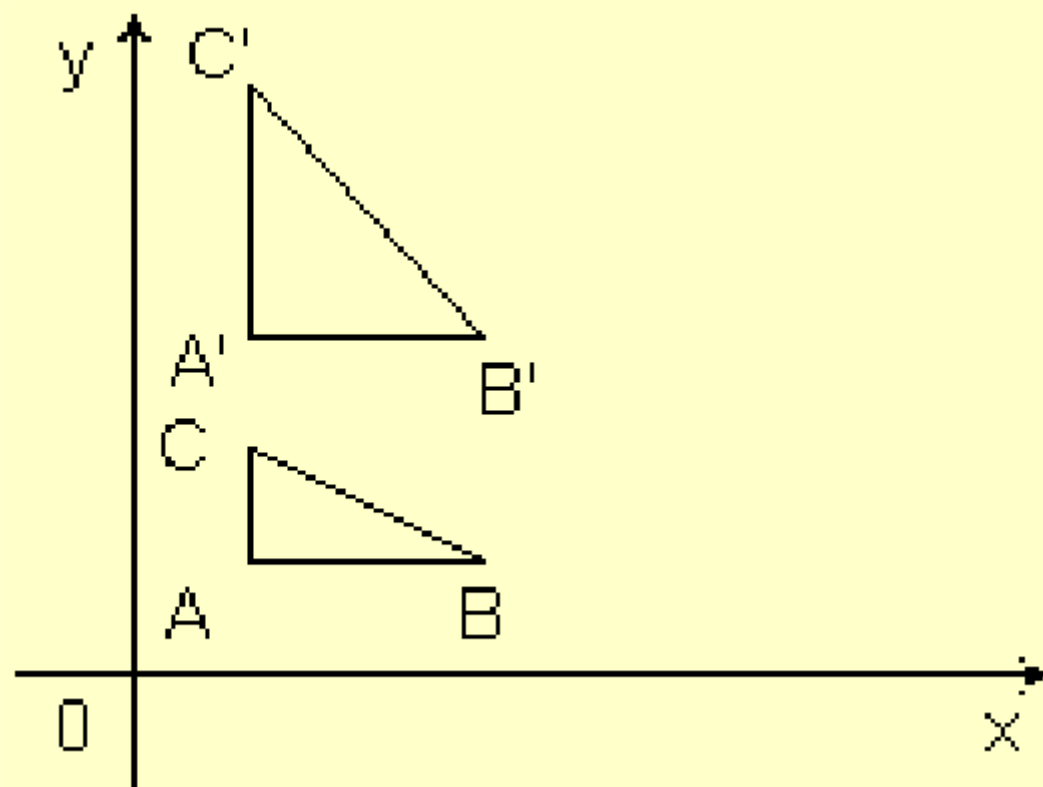


图4.1 平面图形二维几何变换
(d) 比例变换($a=1, e>1$)

3) 对称变换

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ 0 & 0 & 1 \end{bmatrix} = [ax+by \ dx+ey \ 1]$$

1. 当 $b=d=0, a=-1, e=1$ 时, 以 y 轴对称, 见图4.1(e);
2. 当 $b=d=0, a=1, e=-1$ 时, 以 x 轴对称, 见图4.1(f);
3. 当 $b=d=0, a=e=-1$ 时, 以原点对称, 见图4.1(g);
4. 当 $b=d=1, a=e=0$ 时, 以45度直线 $y=x$ 对称, 见图4.1(h);
5. 当 $b=d=-1, a=e=0$ 时, 以-45度直线 $y=-x$ 对称, 见图4.1(i) .

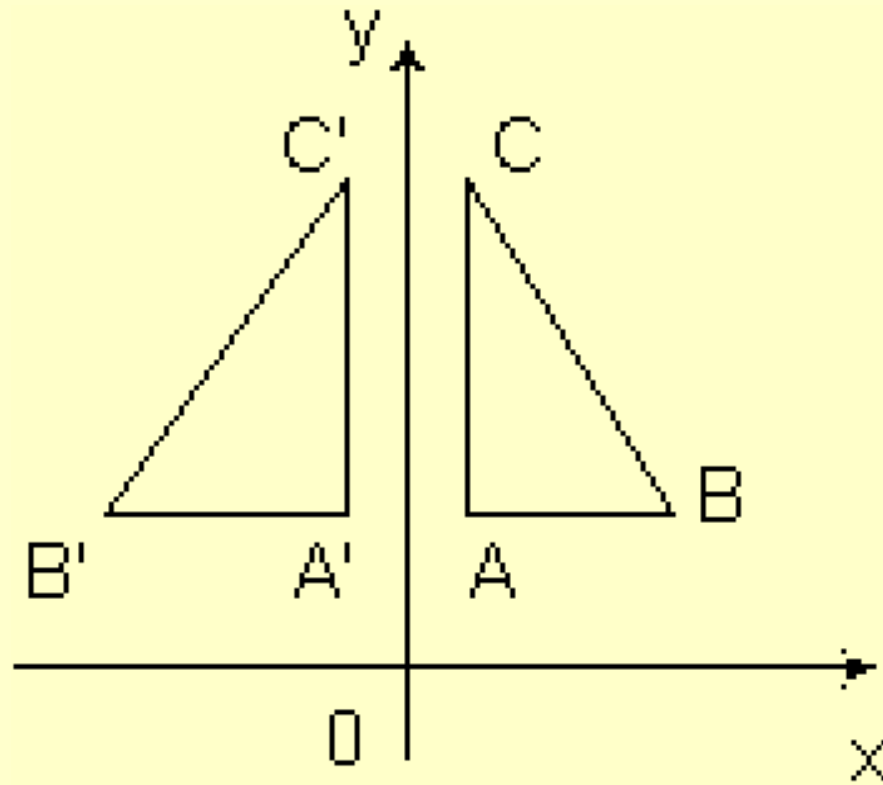


图4.1 平面图形二维几何变换
(e) 对称变换-y轴对称: $x'=-x$, $y'=y$

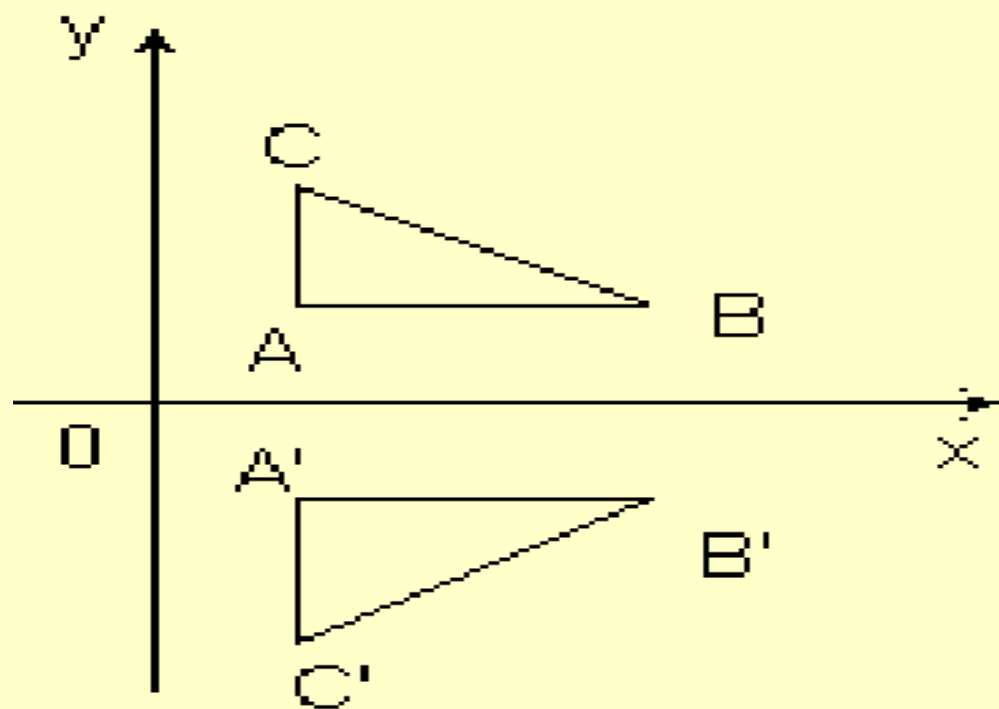


图4.1 平面图形二维几何变换

(f) 对称变换 --x轴对称: $x' = -x$, $y' = y$

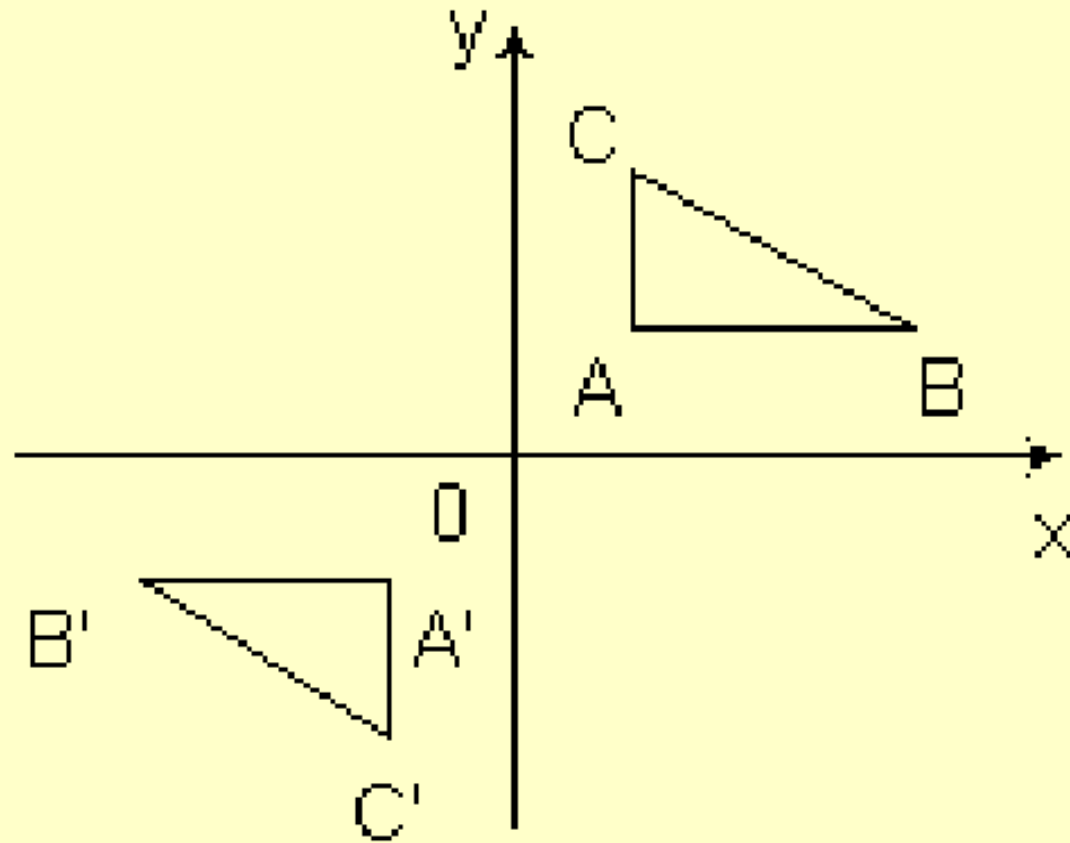


图4.1 平面图形二维几何变换
(f) 对称变换-原点对称($x'=-x$, $y'=-y$)

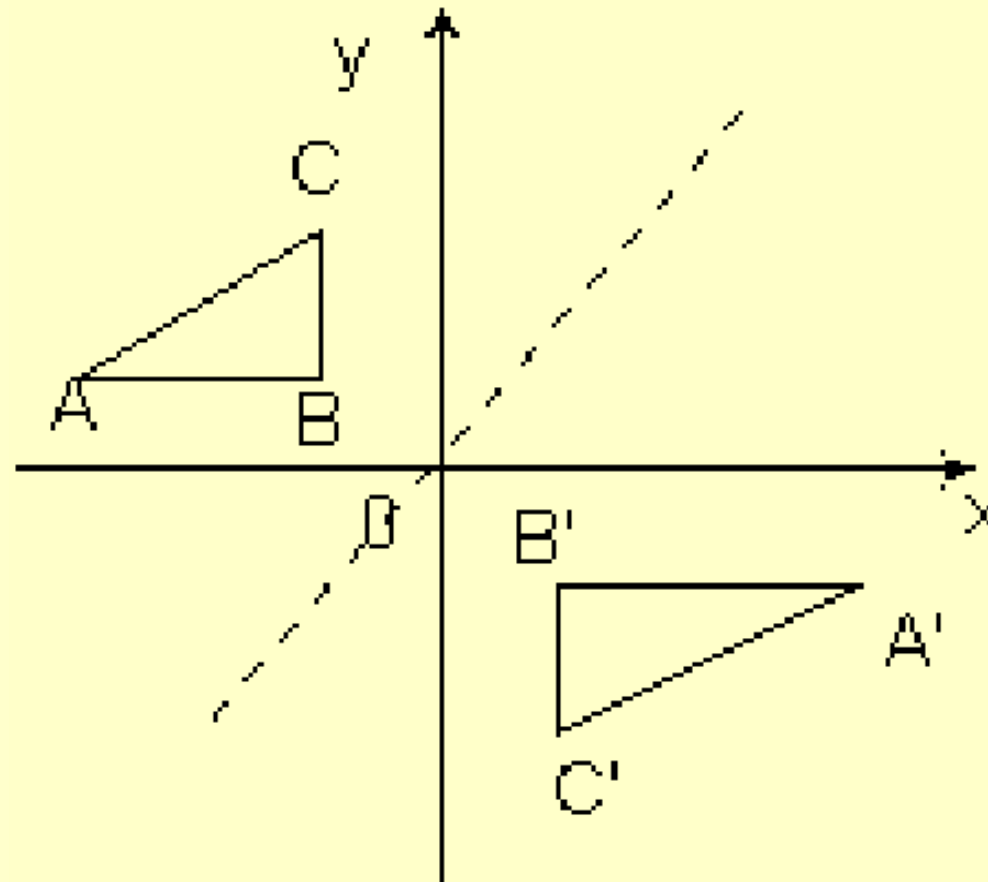


图4.1 平面图形二维几何变换

(h)对称变换-- $y=x$ 对称: $x'=y$, $y'=x$

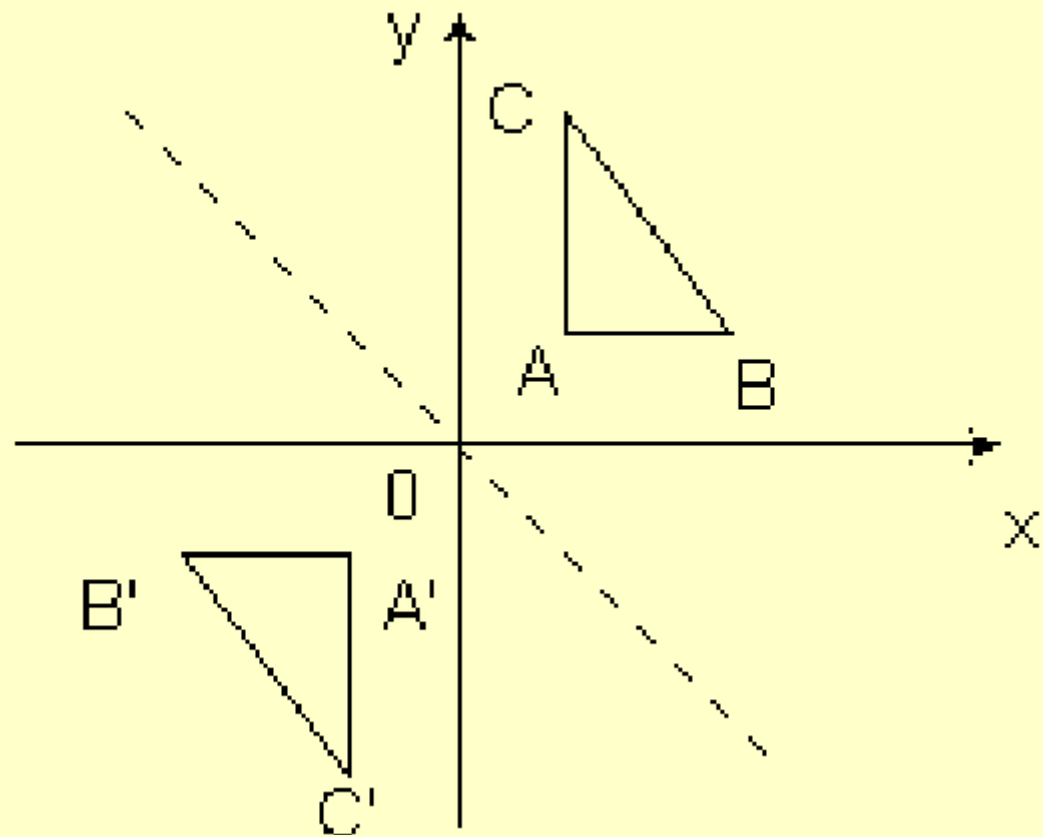


图4.1 平面图形二维几何变换
(i)对称变换 $--y = -x: x' = -y, y' = -x$

4) 旋转变换

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x \cos \theta - y \sin \theta \quad x \sin \theta + y \cos \theta \quad 1] \end{aligned}$$

此乃图形绕原点逆时针旋转 θ 角, 见图4.1(j)

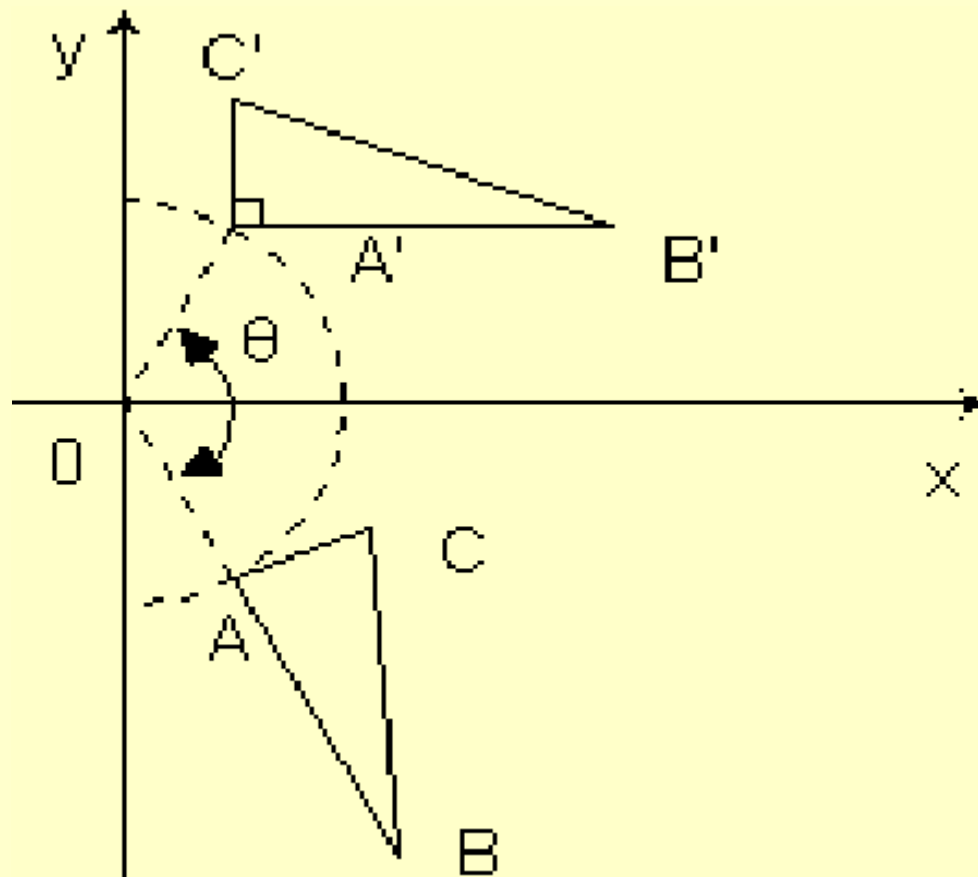


图4.1 平面图形二维几何变换
(j)旋转变换 θ 角

5) 错切变换

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & d & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x+by \ dx+y \ 1]$$

1. 当 $d=0$ 时, 图形沿 x 方向错切, 见图4.1(k);
2. 当 $b=0$ 时, 图形沿 y 方向错切, 见图4.1(l).

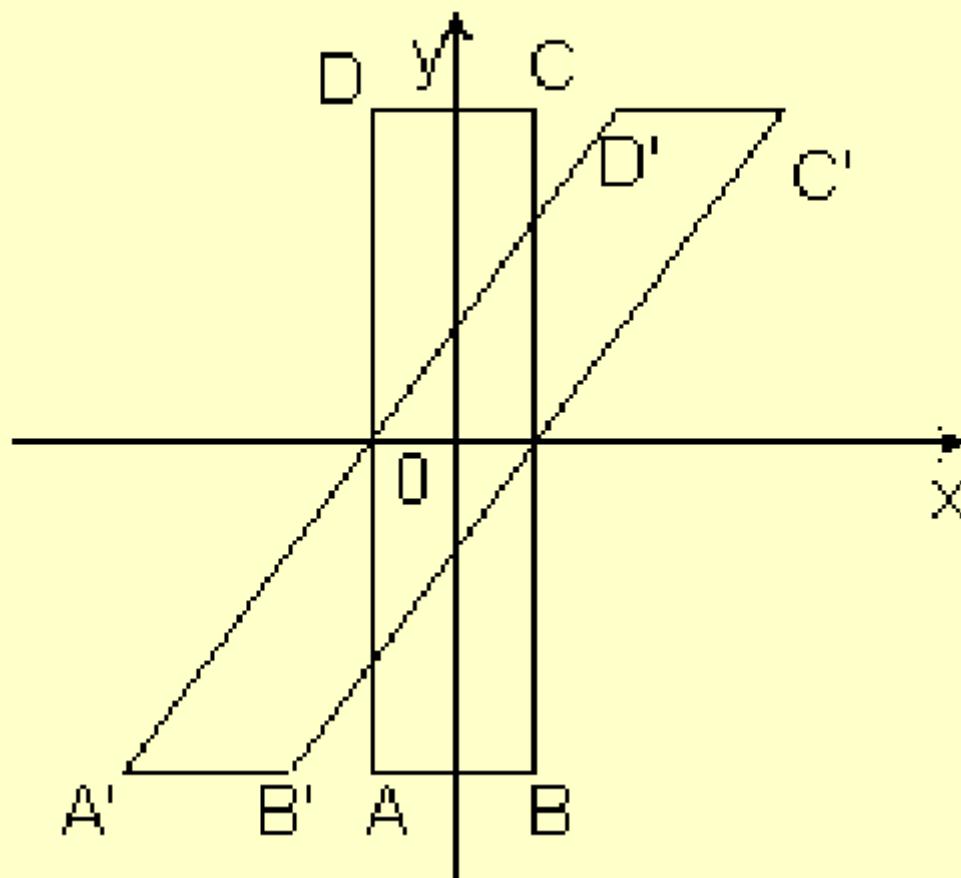


图4.1 平面图形二维几何变换
(k)沿x方向错切变换 ($b>0, d=0$)

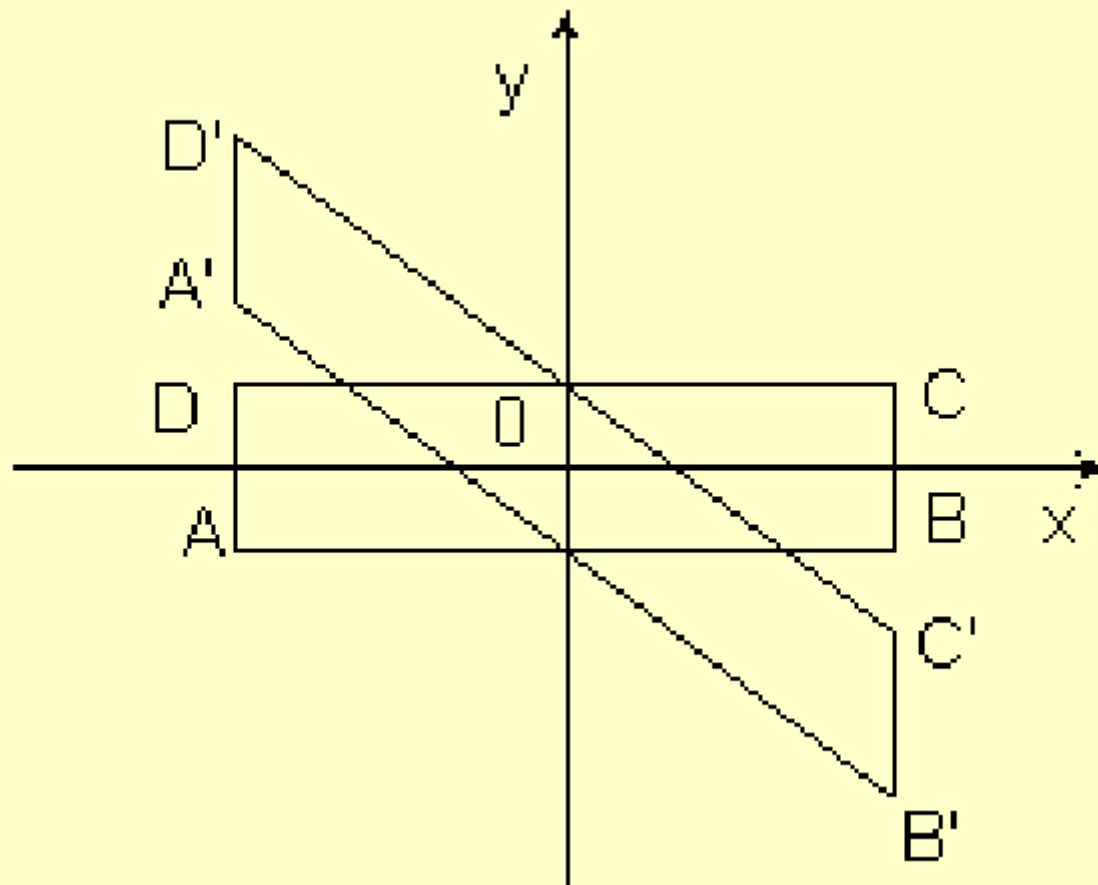


图4.1 平面图形二维几何变换

(1)沿y方向错切变换 ($d < 0, b = 0$)

6) 复合变换

- 有些复杂的变换单靠上面所列的某个简单变换是不可能实现的, 通常的做法是把多个简单变换联合起来作复合变换, 如以任意直线 $Ax+By+C=0$ 对称。见图4.1(m)
- 变换矩阵为:

$$T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ C/A & 0 & 1 \end{bmatrix}, \text{ 平移使D点与O点重合, 使}$$

直线过原点;

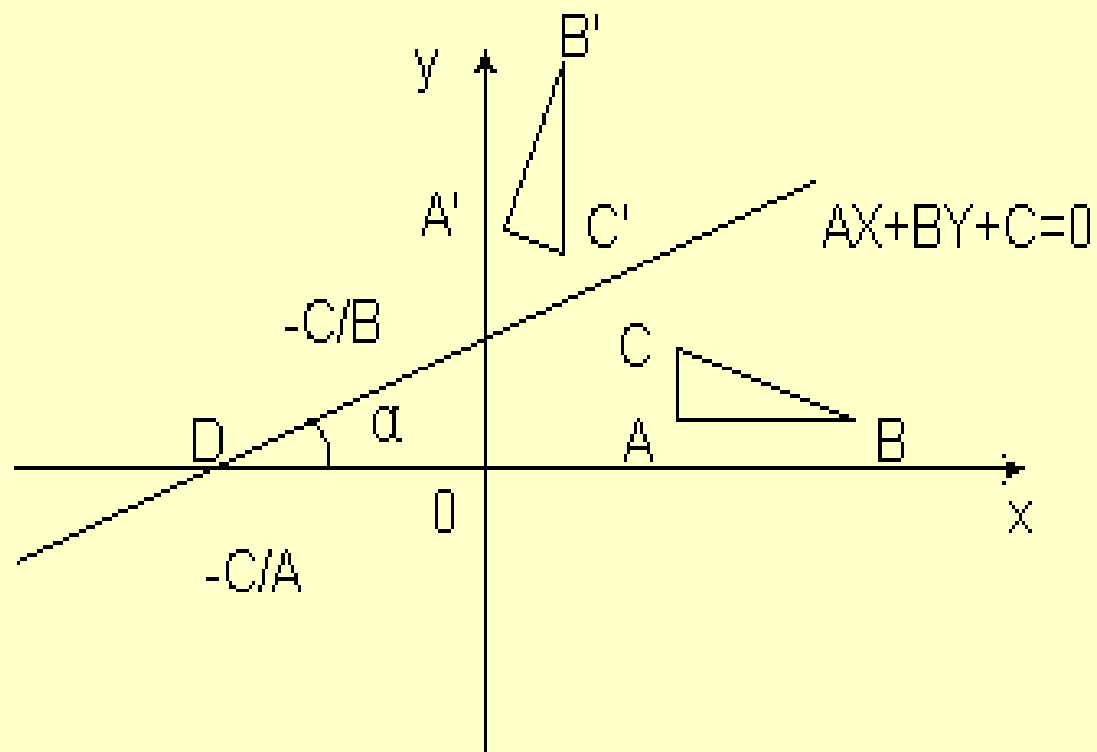


图4.1 平面图形二维几何变换

(m) 复合变换

$$T2 = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ 顺时针转动 } \alpha \text{ 角, 使直}$$

线与X轴重合;

$$T3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ 以X轴对称;}$$

$$T4 = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ 逆时针转动 } \alpha \text{ 角, 使直线}$$

恢复原斜率;

$$T_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -C/A & 0 & 1 \end{bmatrix}, \text{ 平移将D点移回原处, 恢复}$$

原直线;

$$T = T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 = \begin{bmatrix} \cos 2\alpha & \sin 2\alpha & 0 \\ \sin 2\alpha & -\cos 2\alpha & 0 \\ C(\cos 2\alpha - 1)/A & C \sin 2\alpha / A & 1 \end{bmatrix}$$

其中 $\alpha = \arctg(-A/B)$

4.2 三维图形变换

4.2.1 三维变换矩阵

4.2.2 三维图形的几何变换

4.2.1 三维变换矩阵

- 与二维变换矩阵相似, 采用齐次坐标的三维变

换矩阵为 $\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ \hline d & h & l & p \end{bmatrix}$, 它同样可以分为4个子

矩阵: $\begin{bmatrix} a & e & i \\ b & f & j \\ c & g & k \end{bmatrix}$ 对图像进行比例、旋转、错切、对称等几何变换; $[d \ h \ l]$ 产生平移变换; $\begin{bmatrix} m \\ n \\ o \end{bmatrix}$ 对图形进行投影变换; $[p]$ 产生整体比例变换.

4.2.2 三维图形的几何变换

1. 平移变换

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d & h & l & 1 \end{bmatrix} \\ &= [X+d \ Y+h \ Z+l \ 1] \end{aligned}$$

2. 比例变换

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [aX \ fY \ kZ \ 1] \end{aligned}$$

3. 对称变换

1) 以YOZ平面对称

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [-X \ Y \ Z \ 1] \end{aligned}$$

2) 以XOZ平面对称

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [X \ -Y \ Z \ 1] \end{aligned}$$

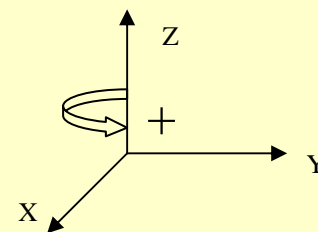
3) 以XOY平面对称

$$\begin{aligned}[X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [X \ Y \ -Z \ 1]\end{aligned}$$

- 以原点对称

$$\begin{aligned}[X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [-X \ -Y \ -Z \ 1]\end{aligned}$$

4. 旋转变换



- 用右手坐标系确定旋转角度正负(拇指与旋转轴正方向同向, 其余四指所指方向为正, 即逆时针方向为正)。

1) 绕X轴旋转

$$[X' \ Y' \ Z' \ 1] = [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [X \ Y \cos \theta - Z \sin \theta \ Y \sin \theta + Z \cos \theta \ 1]$$

2) 绕Y轴旋转

$$[X' \ Y' \ Z' \ 1] = [X \ Y \ Z \ 1] \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} X \cos \theta + Z \sin \theta & Y & -X \sin \theta + Z \cos \theta & 1 \end{bmatrix}$$

3) 绕Z轴旋转

$$[X' \ Y' \ Z' \ 1] = [X \ Y \ Z \ 1] \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} X \cos \theta - Y \sin \theta & X \sin \theta + Y \cos \theta & Z & 1 \end{bmatrix}$$

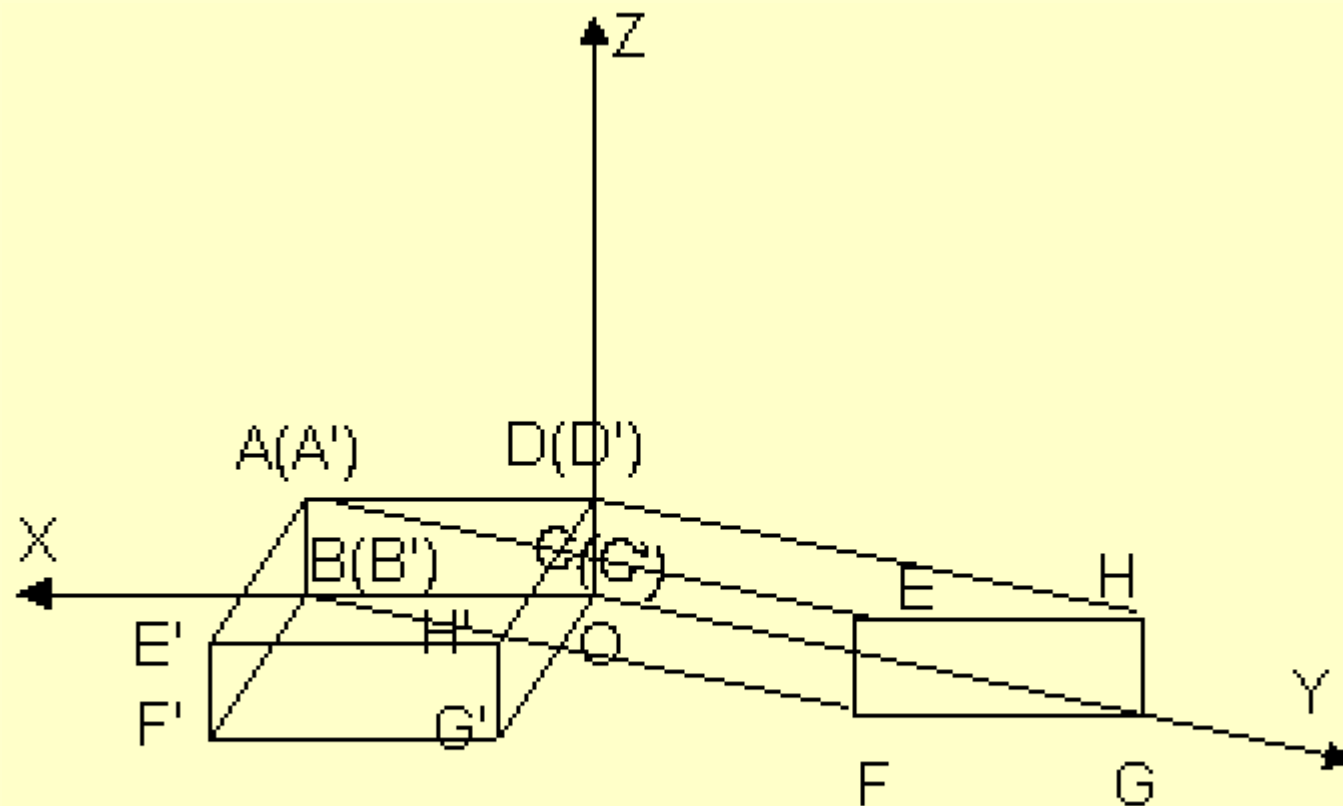
5. 错切变换

1) 沿X含Y错切, 见图4.2(a)。

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ b & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [X + bY \ Y \ Z \ 1] \end{aligned}$$

2) 沿X含Z错切, 见图4.2(b)。

$$\begin{aligned} [X' \ Y' \ Z' \ 1] &= [X \ Y \ Z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ c & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [X + cZ \ Y \ Z \ 1] \end{aligned}$$



(a)沿X含Y错切

图4.2 沿X轴错切

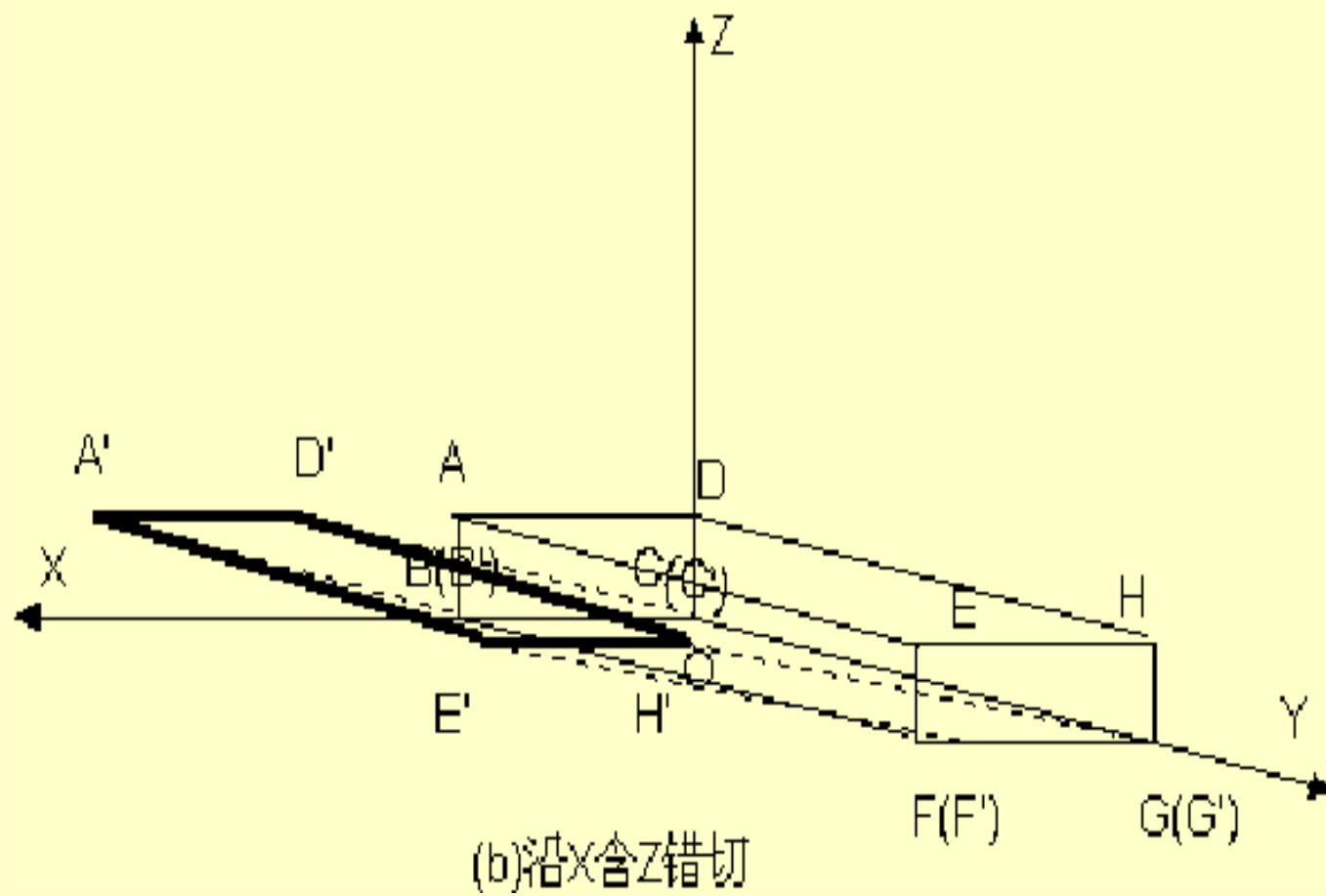


图4.2 沿X轴错切

6. 复合变换

与二维图形的几何变换类似, 三维图形的复杂变换也同样是多个简单的几何变换复合而成。

4.3 三维图形的投影变换

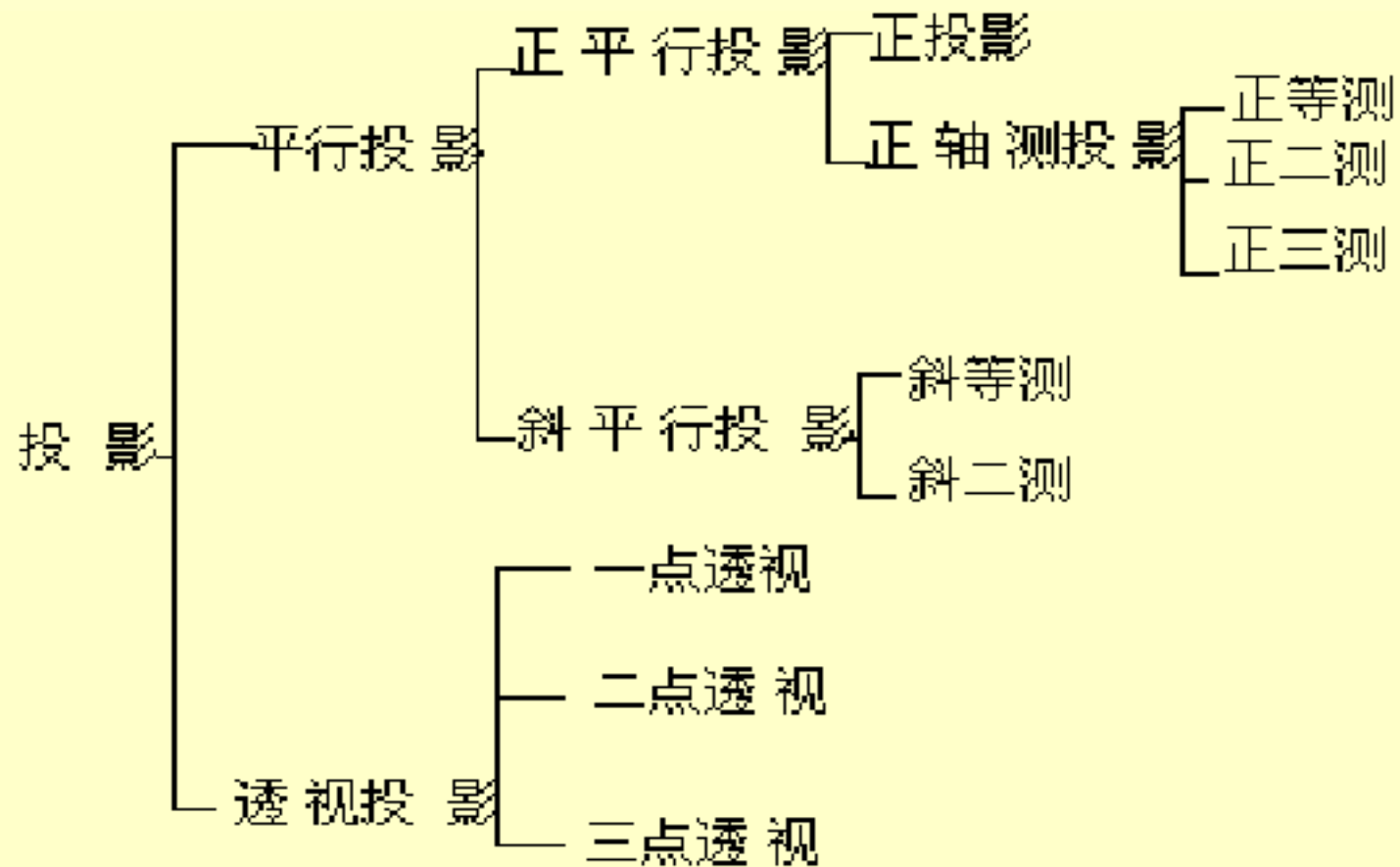
4.3.1 投影变换分类

4.3.2 正平行投影

4.3.3 透视投影

4.3.4 用户坐标系到观察坐标系的变换

4.3.1 投影变换分类



4.3.2 正平行投影

- 平行投影是指投影中心与投影平面之间的距离为无穷大, 投影线可以看成是平行的。
- 投影方向垂直于投影平面时称为正平行投影。它又包括正投影(三视图)和正轴测投影。

1. 三视图

我们通常说的三视图是指正视图、俯视图和侧视图, 投影平面分别与 x, y, z 轴垂直。见图4.3。

- 正视图(主视图) (YZ平面)

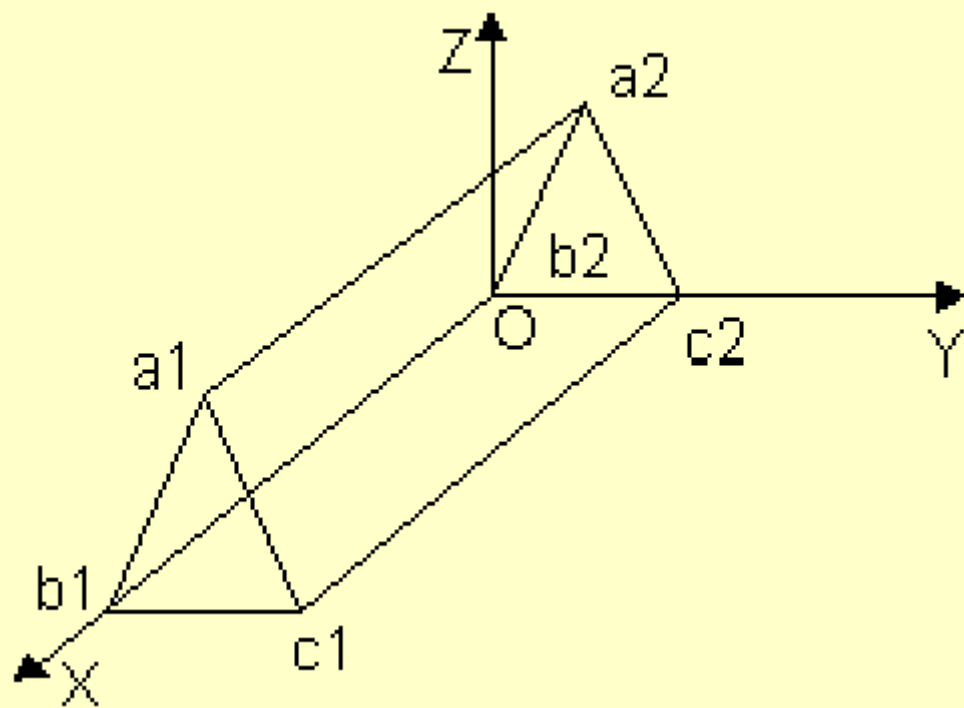
$$\begin{aligned} & \begin{matrix} & & & & |0 & 0 & 0 & 0| \\ [u & v & w & 1] = [x & y & z & 1] & |0 & 1 & 0 & 0| \\ & & & & |0 & 0 & 1 & 0| \\ & & & & |0 & 0 & 0 & 1| \end{matrix} \end{aligned}$$

- 俯视图 (XY平面)

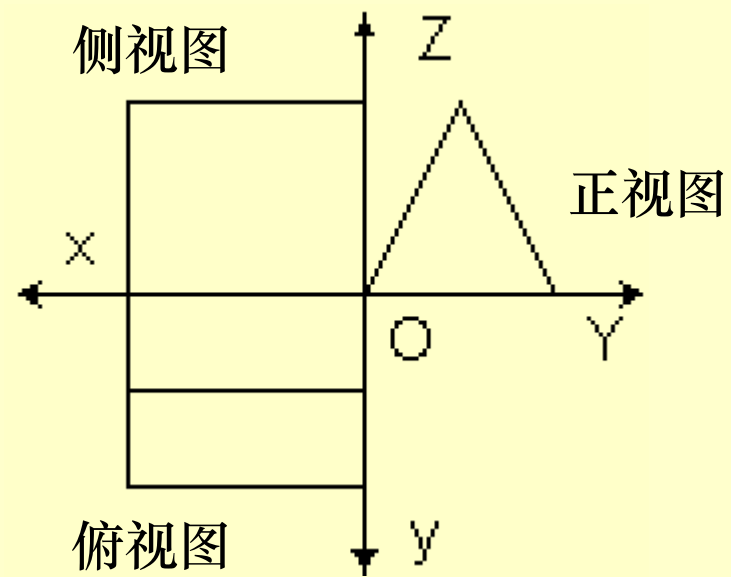
$$[u \ v \ w \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 侧视图 (XZ平面)

$$[u \ v \ w \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(a)



(b)

图4.3 三视图

2. 正轴测投影变换

- 投影面与坐标轴之间有夹角。设投影面与三个轴交于A,B,C, 投影方向与投影面垂直。
- 首先把物体及投影面绕Y轴顺时针旋转 φ (φ), 再绕X轴逆时针旋 θ 角, 使投影方向与Z轴重合, 如图4.4所示, 变换矩阵如下:

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

$$= \begin{vmatrix} \cos \varphi & -\sin \varphi \sin \theta & \sin \varphi \cos \theta & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ -\sin \varphi & -\cos \varphi \sin \theta & \cos \varphi \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- 最后做Z方向的正投影, 得到变换

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

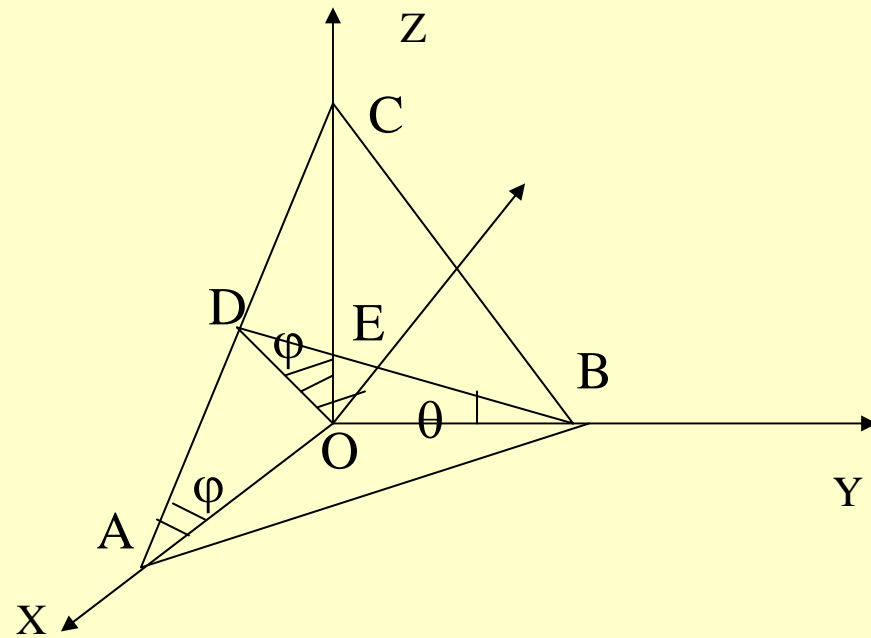


图4.4 正轴测投影图

按投影面与三个轴方向的夹角是否相等, 进一步分为:
正等测、正二测、正三测投影.

4.3.3 透视投影

- 透视投影的投影线是从视点(观察点)出发, 投影线是不平行的。
- 任何一束不平行于投影平面的平行线的透视投影将汇聚成一点, 称为灭点。灭点可以看作是无限远处的一点在投影面上的投影。灭点有无限多个。
- 在坐标轴方向上的灭点, 称为主灭点。
- 透视投影根据主灭点的个数分为一点透视、二点透视和三点透视。主灭点数是和投影平面切割坐标轴的数量相对应的。

- 一点透视

为了获得透视投影图，常将物体平移以后，再进行透视变换，然后再投影到正面(XOZ)，如图4.5所示。其变换矩阵为：

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

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & n \\ 0 & 0 & 1 & 0 \\ d & 0 & l & hn+1 \end{bmatrix}$$

通常取 $d>0$, $h<0$, $l<0$, $n<0$, 原点为灭点。

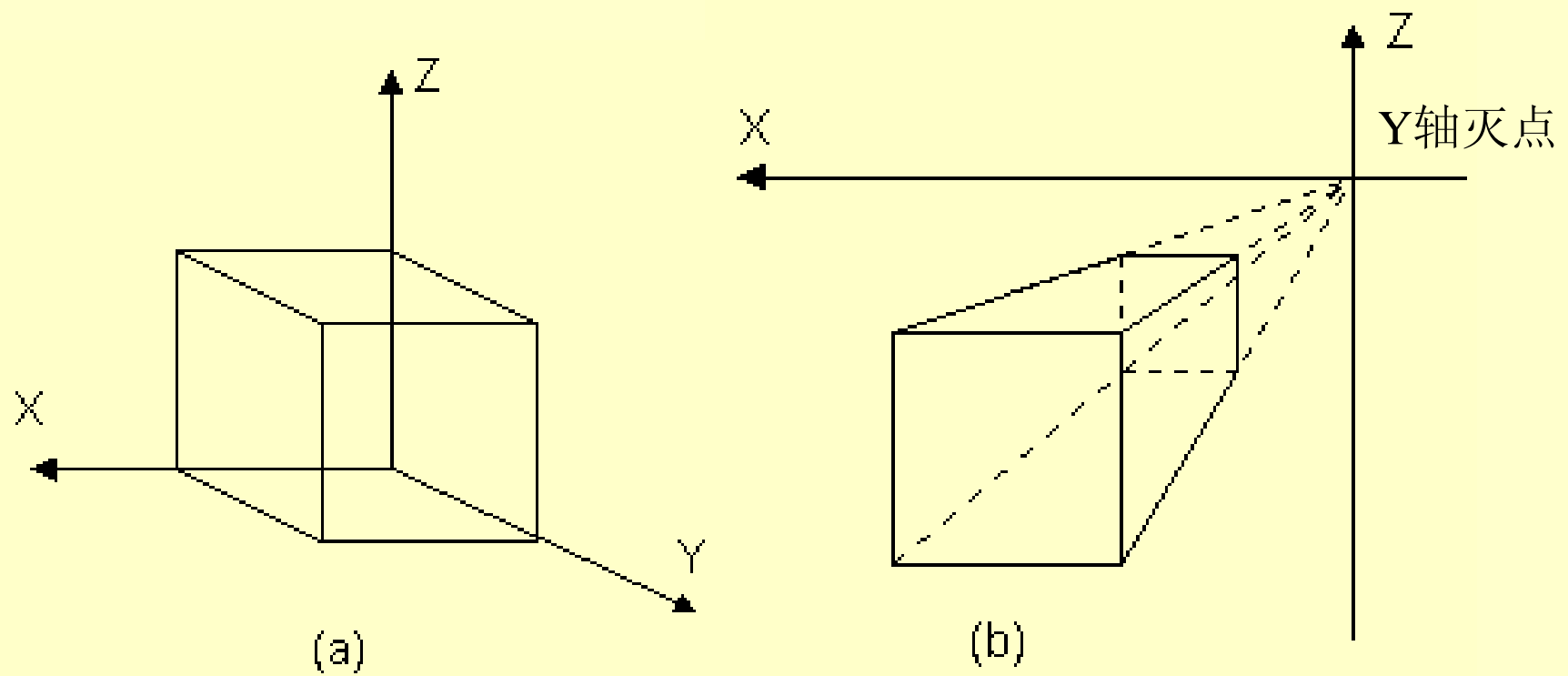


图4.5 一点透视投影图

- 两点透视

先将物体绕 Z 轴旋转 θ 角, 使物体上的两个坐标平面与投影面倾斜一个角度, 平移物体到正面 (XOZ) 之后、水平面之下, 再进行透视投影变换, 如图4.6所示。变换矩阵为:

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

$$= \begin{bmatrix} \cos \theta & 0 & 0 & n \sin \theta \\ -\sin \theta & 0 & 0 & n \cos \theta \\ 0 & 0 & 1 & 0 \\ d & 0 & l & hn+1 \end{bmatrix}$$

通常取 $d=0, h<0, l<0, n<0$, 两灭点在 X 轴上。

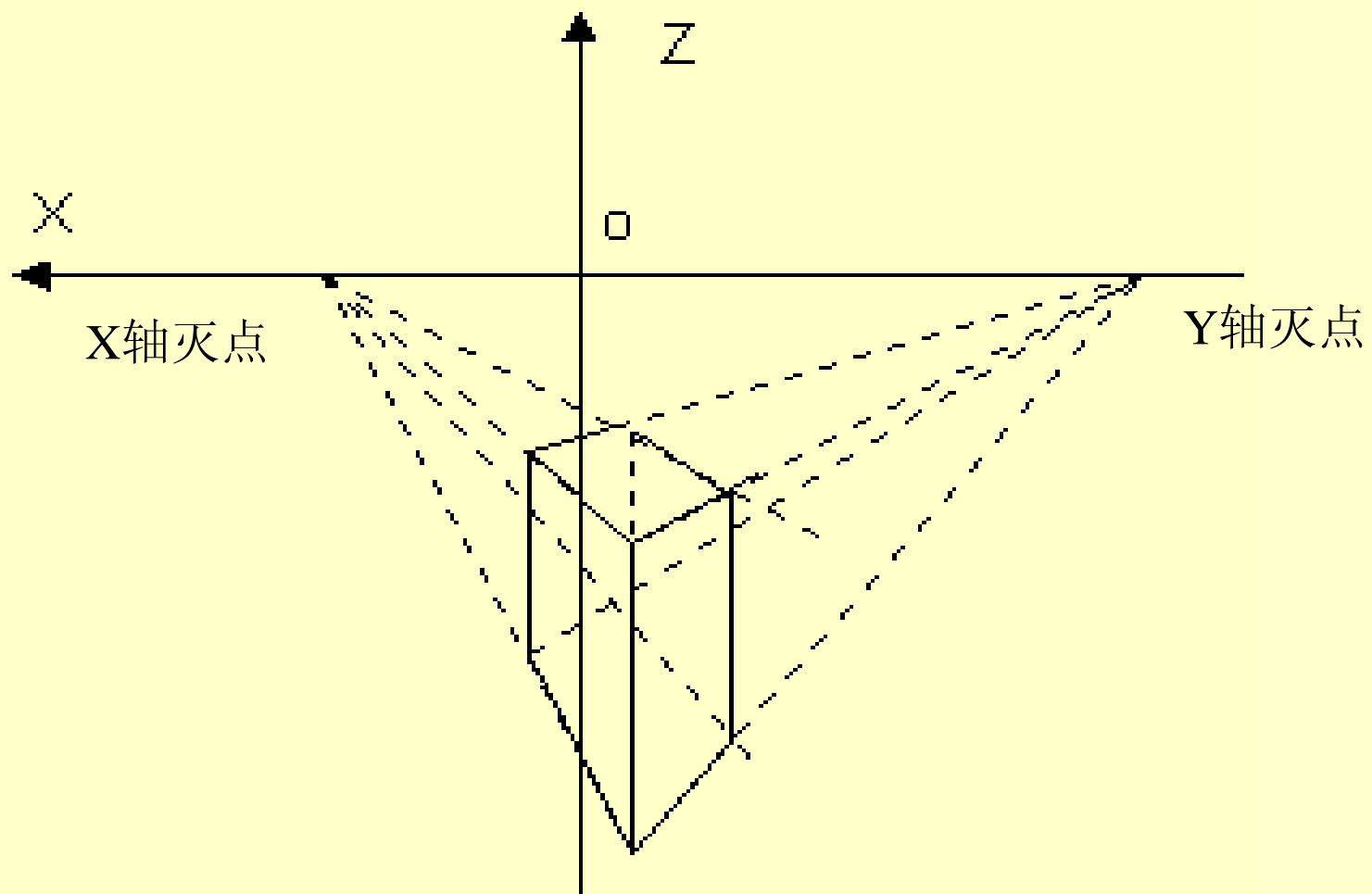


图4.6 两点透视投影图

- 三点透视

先将物体绕 Z 轴旋转 θ 角,再绕X轴旋转 ϕ 角,使物体上的坐标平面与投影面均倾斜一个角度,平移物体后,再进行投影变换,如图4.7所示。变换矩阵为:

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & n \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \sin \phi & n \sin \theta \cos \phi \\ -\sin \theta & 0 & \cos \theta \sin \phi & n \cos \theta \cos \phi \\ 0 & 0 & \cos \phi & -n \sin \phi \\ d & 0 & l & hn+1 \end{bmatrix}$$

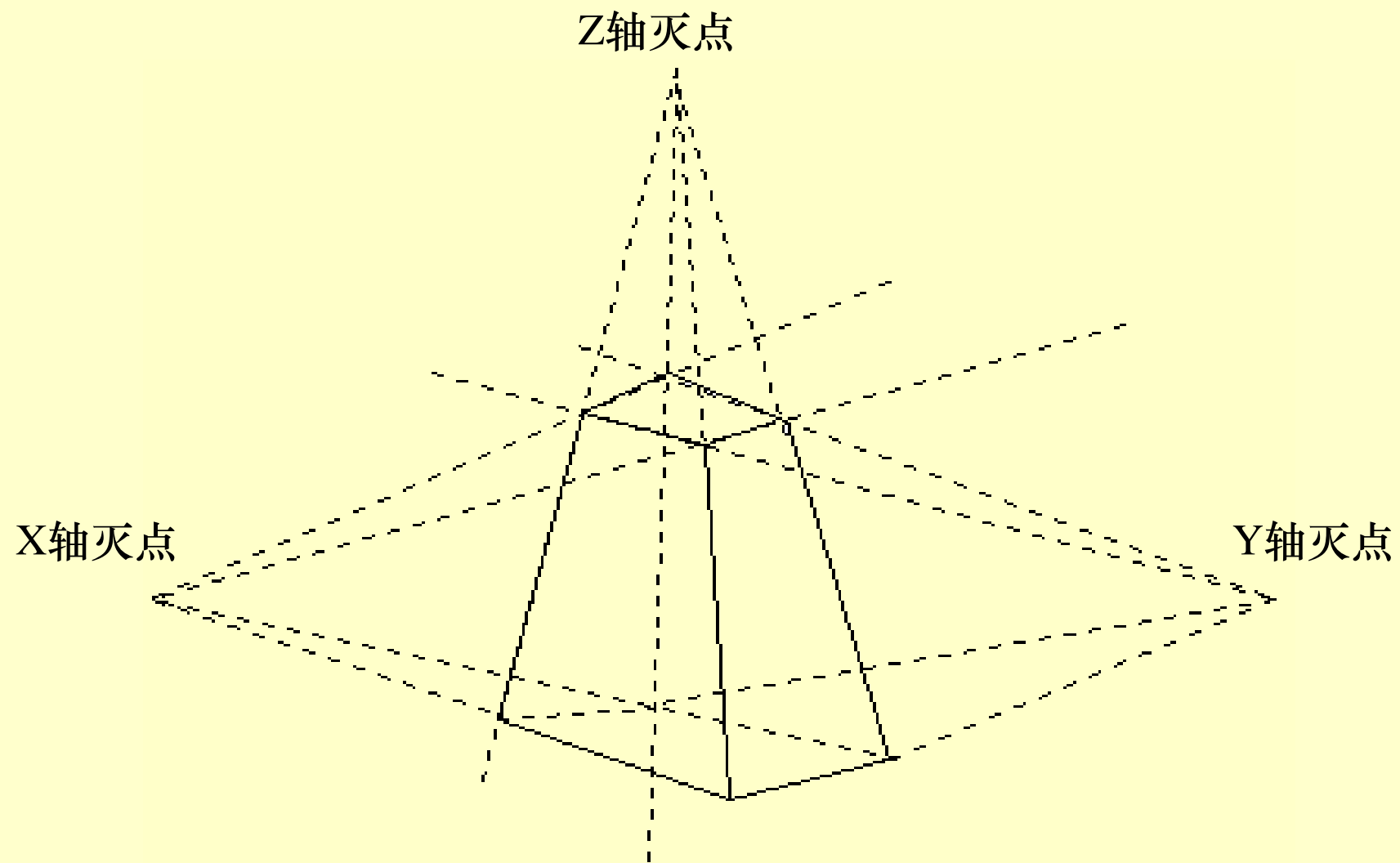
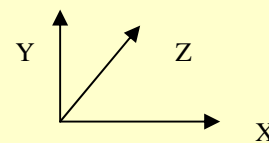


图4.7 三点透视投影图

4.3.4 用户坐标系到观察坐标系的变换

- 通常我们希望在透视投影时,物体不动,让视点在以形体为中心的球面上变化,观察形体的各个方向上的形象。
- 为解决这个问题,有效的方法是引进一个过渡坐标系,即观察坐标系。将用户在描述形体时的用户坐标系变换到观察坐标系。
- 观察坐标系采用左手坐标系, X 轴正向指向右, Y 轴正向指向上, Z 轴正向指向用户坐标系原点。
- 两个坐标系间的变换可以看成是以下几个变换的复合,如图4.8所示。



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

将用户坐标系原点平移到视点e, 设e在用户坐标系下的坐标为(a,b,c), 形体上的点相当于平移(-a,-b,-c).

- $T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90^\circ & -\sin 90^\circ & 0 \\ 0 & \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

令平移后的新坐标系绕 x' 轴旋转 90° , 则形体上的点相当于旋转 -90° 。

- $T_3 = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -b/v & 0 & a/v & 0 \\ 0 & 1 & 0 & 0 \\ -a/v & 0 & -b/v & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

其中 $V = \sqrt{a^2 + b^2}$, 再将新坐标系 y'' 转 $-\theta$ 角 (θ 大于 180°), 则形体上的点相当于旋转 θ 角。

- $T_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & \sin \phi & 0 \\ 0 & -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & v/u & c/u & 0 \\ 0 & -c/u & v/u & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

其中 $U = \sqrt{a^2 + b^2 + c^2}$, $V = \sqrt{a^2 + b^2}$, 再令新坐标系绕 x''' 轴转 $-\phi$ 角, 则形体上的点相当于旋转 ϕ 角。

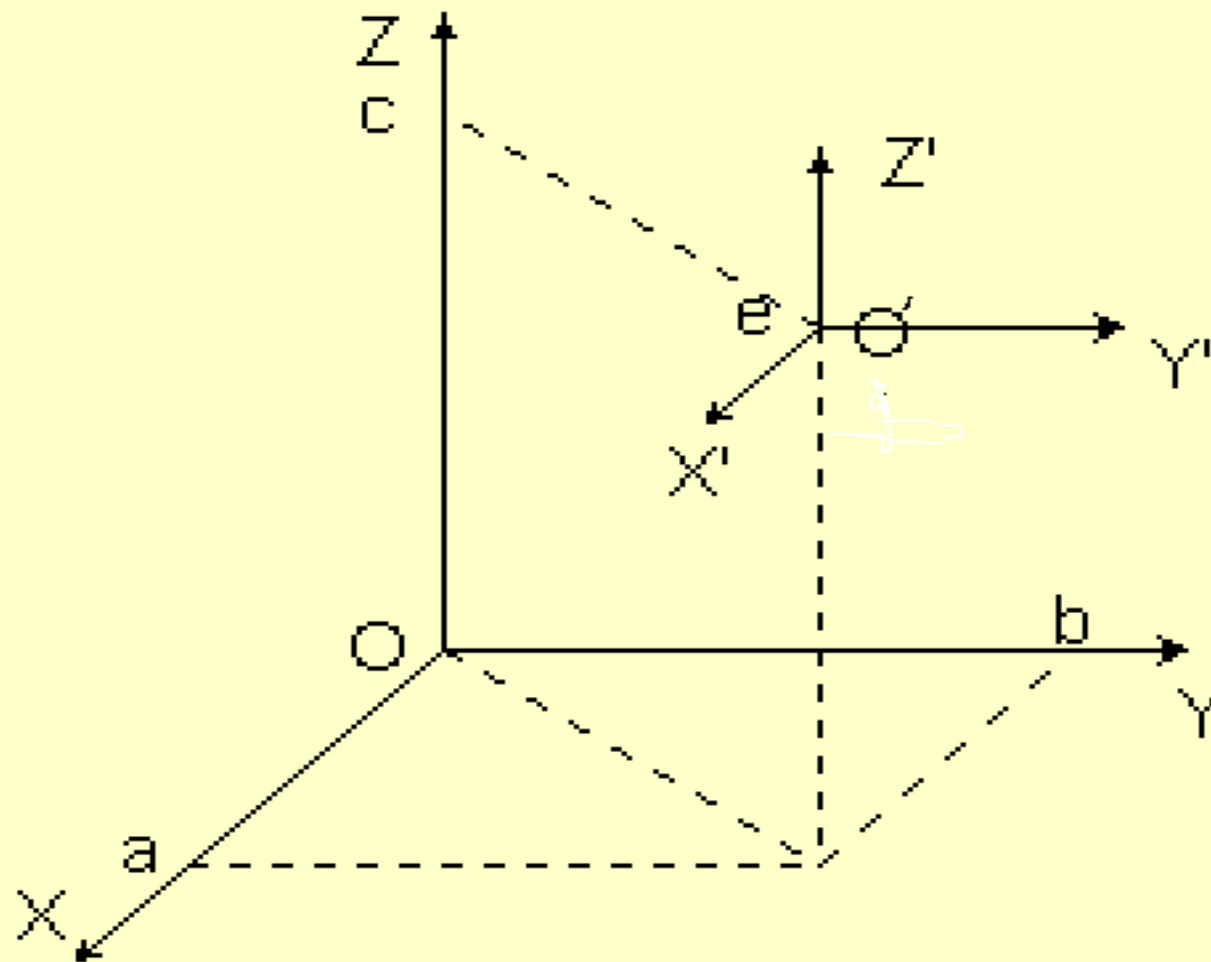
- $T_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

将 z'' 取反, 右手坐标系变成左手坐标系, 即得到观察坐标系 $e_x e_y e_z$ 。

- 复合变换矩阵为:

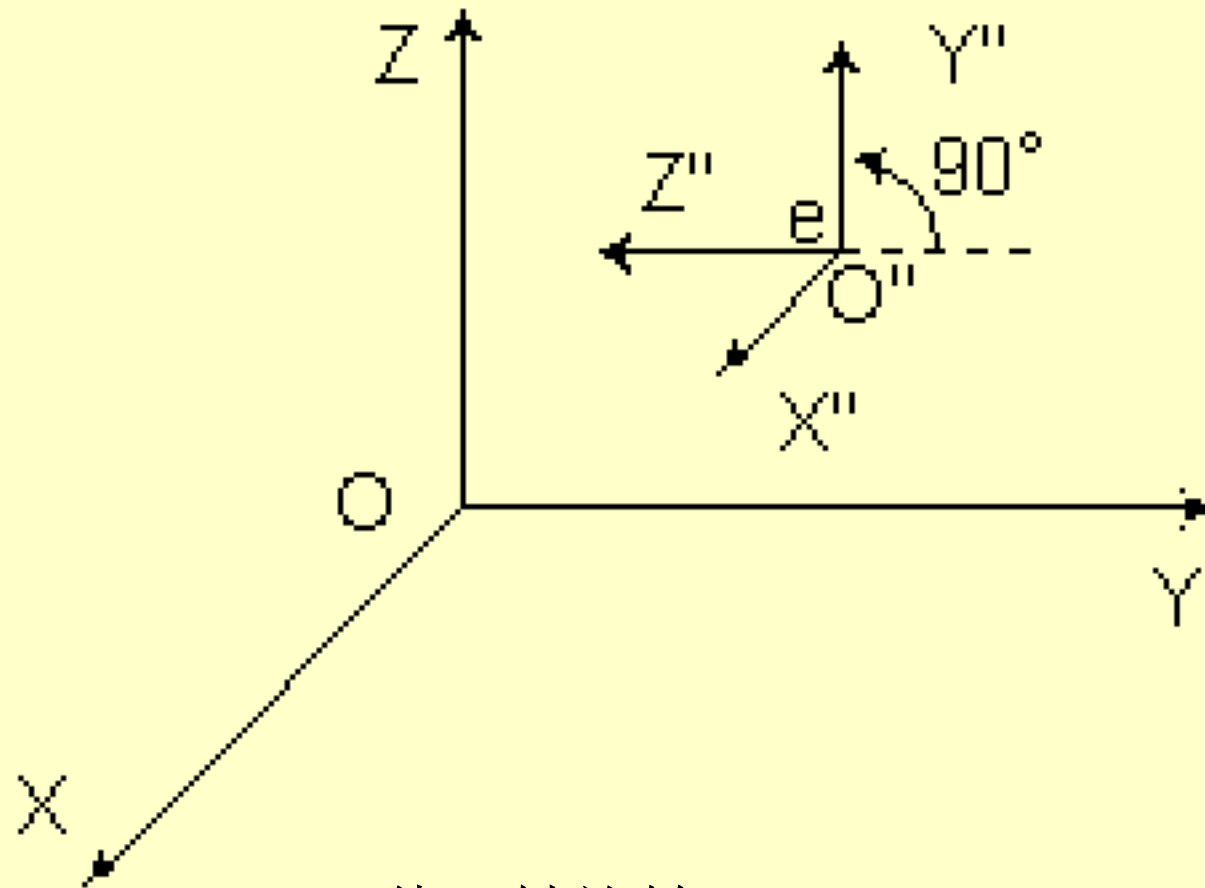
$$T = T_1 \cdot T_2 \cdot T_3 \cdot T_4 \cdot T_5 = \begin{bmatrix} -b/v & -ac/uv & -a/u & 0 \\ a/v & -bc/uv & -b/u & 0 \\ 0 & v/u & -c/u & 0 \\ 0 & 0 & u & 1 \end{bmatrix}$$

其中 $U = \sqrt{a^2 + b^2 + c^2}$, $V = \sqrt{a^2 + b^2}$ 。



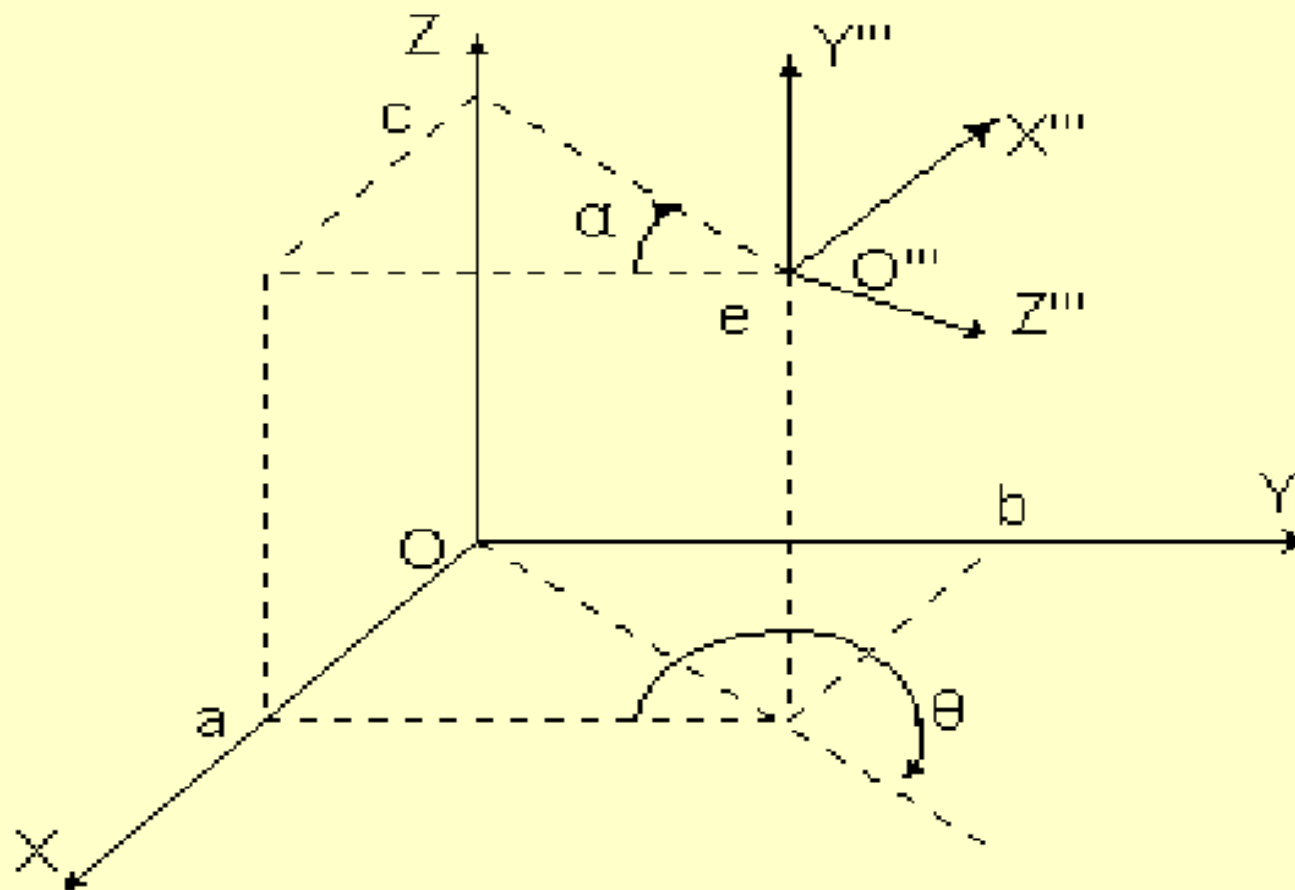
(a) 平移(视点 e 即 o')

图4.8 从用户坐标系到观察坐标系的变换过程



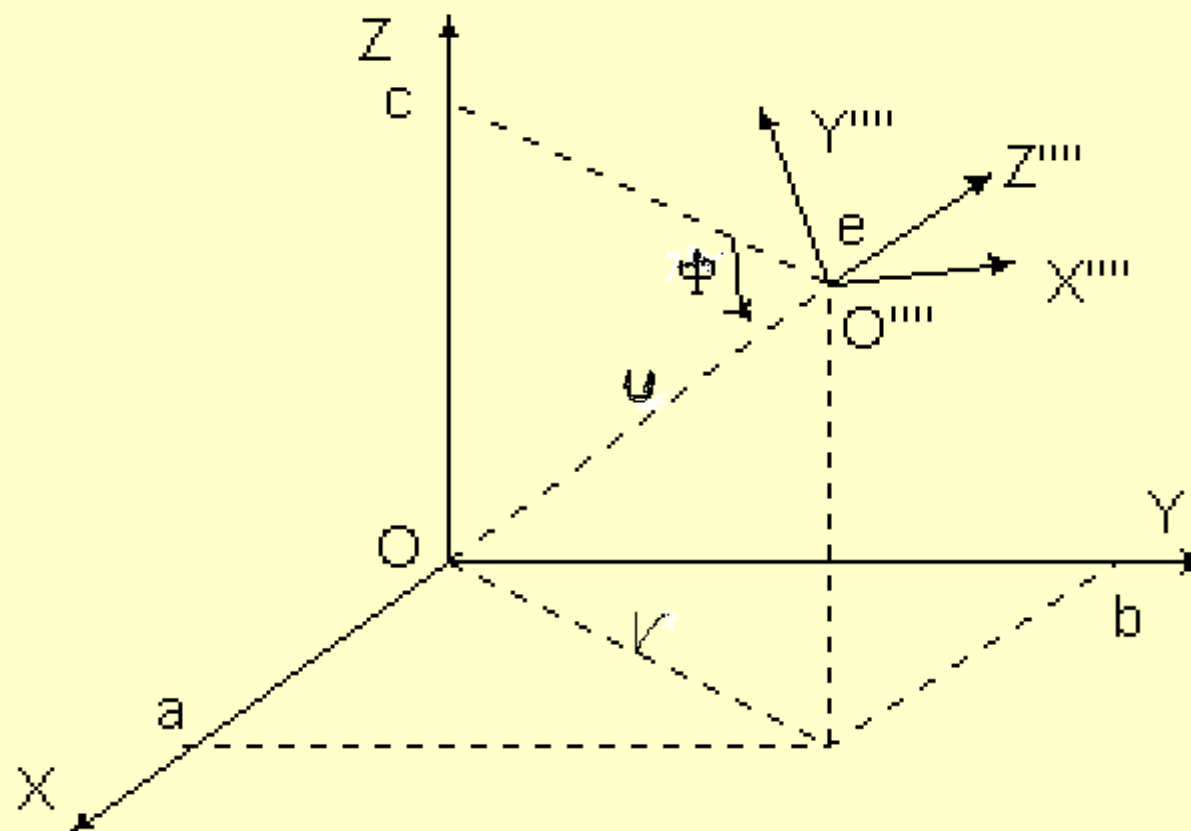
(b) 绕 x' 轴旋转 90°

图4.8 从用户坐标系到观察坐标系的变换过程



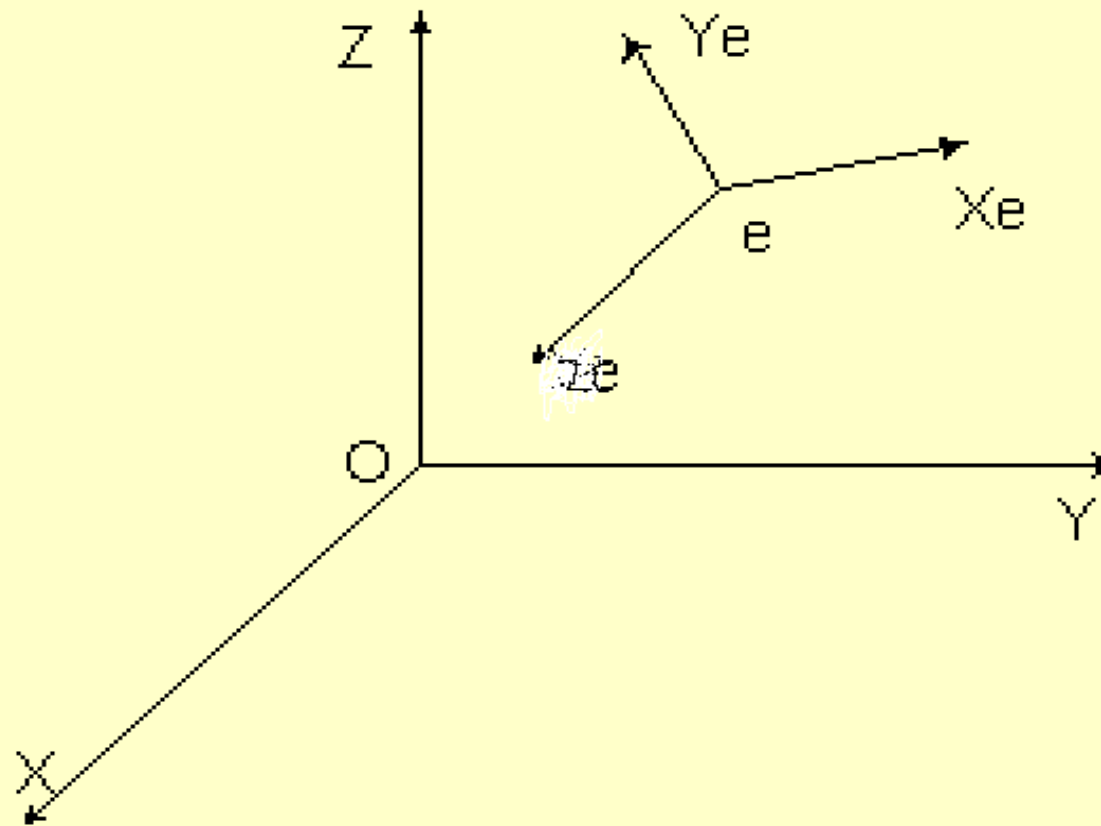
(c) 绕 y'' 轴旋转 θ 角

图4.8 从用户坐标系到观察坐标系的变换过程



(d) 绕 x''' 轴旋转 ϕ 角

图4.8 从用户坐标系到观察坐标系的变换过程



(e) z'''' 取反(指向 O), 即为观察坐标系

图4.8 从用户坐标系到观察坐标系的变换过程

4.4 窗口到视图的变换

4.4.1 用户域和窗口区

4.4.2 屏幕域和视图区

4.4.3 窗口区和视图区的坐标变换

4.4.4 从规格化坐标到设备坐标的变换

4.4.1 用户域和窗口区

- 用户域

用户域是指程序员用来定义图形的整个自然空间 wc (世界坐标)。用户域是一个实数域, wc 从理论上说是连续无限的。

- 窗口区

窗口区 w 是 wc 的一个子域, 小于或等于 wc , 通常是指用户指定的某个区域。窗口区通常是矩形域, 可以用左上角点和右下角点坐标来表示。窗口可以嵌套定义, 还可以定义圆形或多边形窗口。

4.4.2 屏幕域和视图区

- 屏幕域

屏幕域dc是设备输出图形的最大区域,是一个有限的整数域。如某图形显示器有1024×768个可编程像素,则dc可定为:

$$dc \in [0:1023] \times [0:767]$$

- 视图区

任何小于或等于屏幕域的区域都称为视图区,可由用户在屏幕域中用设备坐标来定义。视图区可嵌套定义,一般定义为矩形,也可定义为圆形或多边形。

4.4.3 窗口区和视图区的坐标变换

- 在用户坐标下, 由4条边WXL, WXR, WYB和WYT定义了一个窗口区, 相应在屏幕中也用4条边VXL, VXR, VYB, VYT定义视图区。

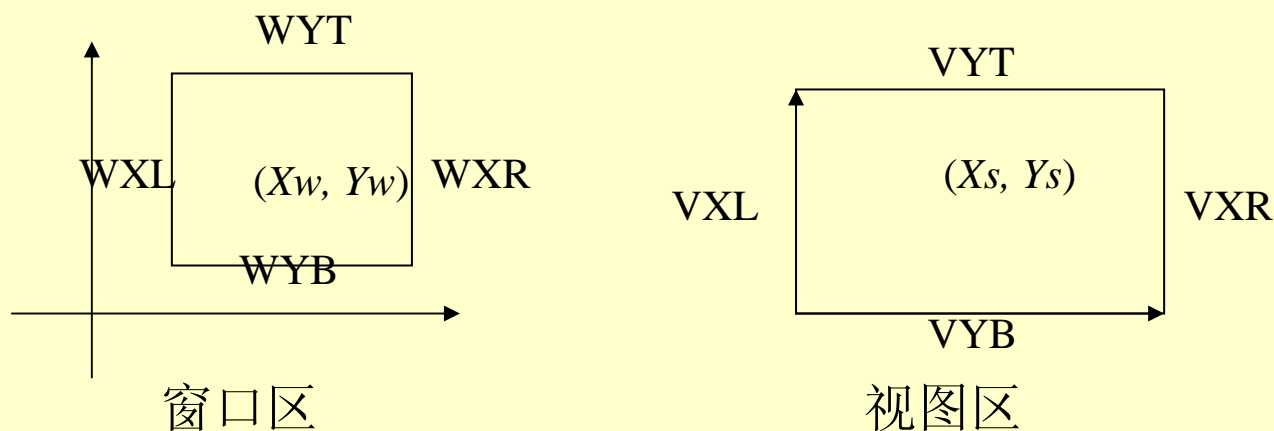


图4.9

- 要将窗口内图形对应应在屏幕视图区内显示出来, 亦即在用户坐标系下的点(X_w, Y_w)变换到视图区中的点(X_s, Y_s), 其变换公式为:

$$X_s = \frac{VXR - VXL}{WXR - WXL} \bullet (X_w - WXL) + VXL = a \bullet X_w + b$$

$$Y_s = \frac{VYT - VYB}{WYT - WYB} \bullet (Y_w - WYB) + VYB = c \bullet Y_w + d$$

其中: $a = \frac{VXR - VXL}{WXR - WXL}$

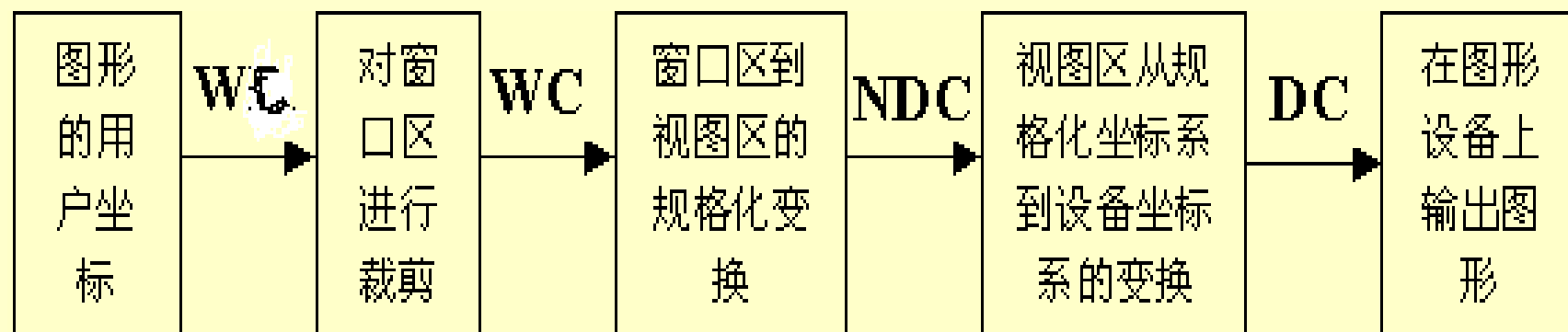
$$b = VXL - WXL \bullet \frac{VXR - VXL}{WXR - WXL}$$

$$c = \frac{VYT - VYB}{WYT - WYB}$$

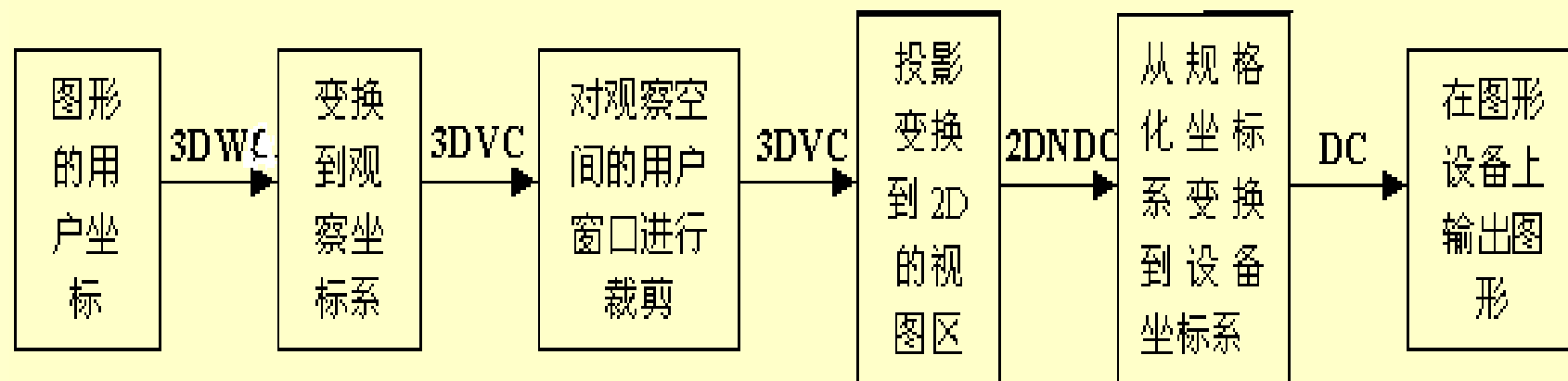
$$d = VYB - WYB \bullet \frac{VYT - VYB}{WYT - WYB}$$

相应的变换矩阵为:
$$\begin{bmatrix} a & 0 & 0 \\ 0 & c & 0 \\ b & d & 1 \end{bmatrix}$$

用户定义的图形从窗口区到视图区的输出过程如图4.10所示.



(a) 窗口---视图二维变换



(b)窗口---视图三维变换

图4.10 图形从窗口区到视图区的输出过程

4.4.4 从规格化坐标到设备坐标的变换

- 规格化坐标(NDC)也称为与设备无关坐标,它不考虑各种设备的不同,独立于设备而建立的一种坐标系.
- 在图形输出时,就必须将NDC转变为相应图形输出设备坐标(DC)一般要经过平移(dx, dy)和比例(Sx, Sy)变换后得到,通常采用的变换为:

$$\begin{cases} X_{DC} = S_x \cdot X_{NDC} + d_x \\ Y_{DC} = S_y \cdot Y_{NDC} + d_y \end{cases}$$

- 在NDC到DC的变换中,要注意以下问题:

NDC空间具有几何一致性,但在DC空间中却不一定成立,这是因为DC中的象素不一定是正方形;

还需要考虑 x,y 方向上的实际象素和在实际运用中NDC和DC的方向相反问题.

4.5 裁剪

4.5.1 二维图形的裁剪

4.5.2 三维图形的裁剪

4.5.1 二维图形的裁剪

一、裁剪的原理

- 在显示图形之前，组成图形的每一个基本元素都要经过裁剪，因此裁剪算法直接影响整个图形系统的效率。
- 裁剪的基本目的是判断图形元素是否在所考虑的区域內。如在区域內，则进一步求出在区域內的那一部分。因此裁剪处理包含两部分内容：
 - 1) 点在区域內外的判断；
 - 2) 计算图形元素与区域边界的交点。

二、二维线段的裁剪

1. 编码裁剪法(Sutherland-Cohen算法)

- 算法以图4.11所示的9个区域为基础, 其中 x_L , x_R , y_B , y_T 为窗口的4条边界。任何一条线段的端点, 根据其坐标所在的区域, 都可以赋予4位二进制代码。设最左边的位是第1位, 则其含义如下:

第1位为1, 表示端点在 y_T 上方;

第2位为1, 表示端点在 y_B 下方;

第3位为1, 表示端点在 x_R 右边;

第4位为1, 表示端点在 x_L 左边;

否则, 相应位置0。

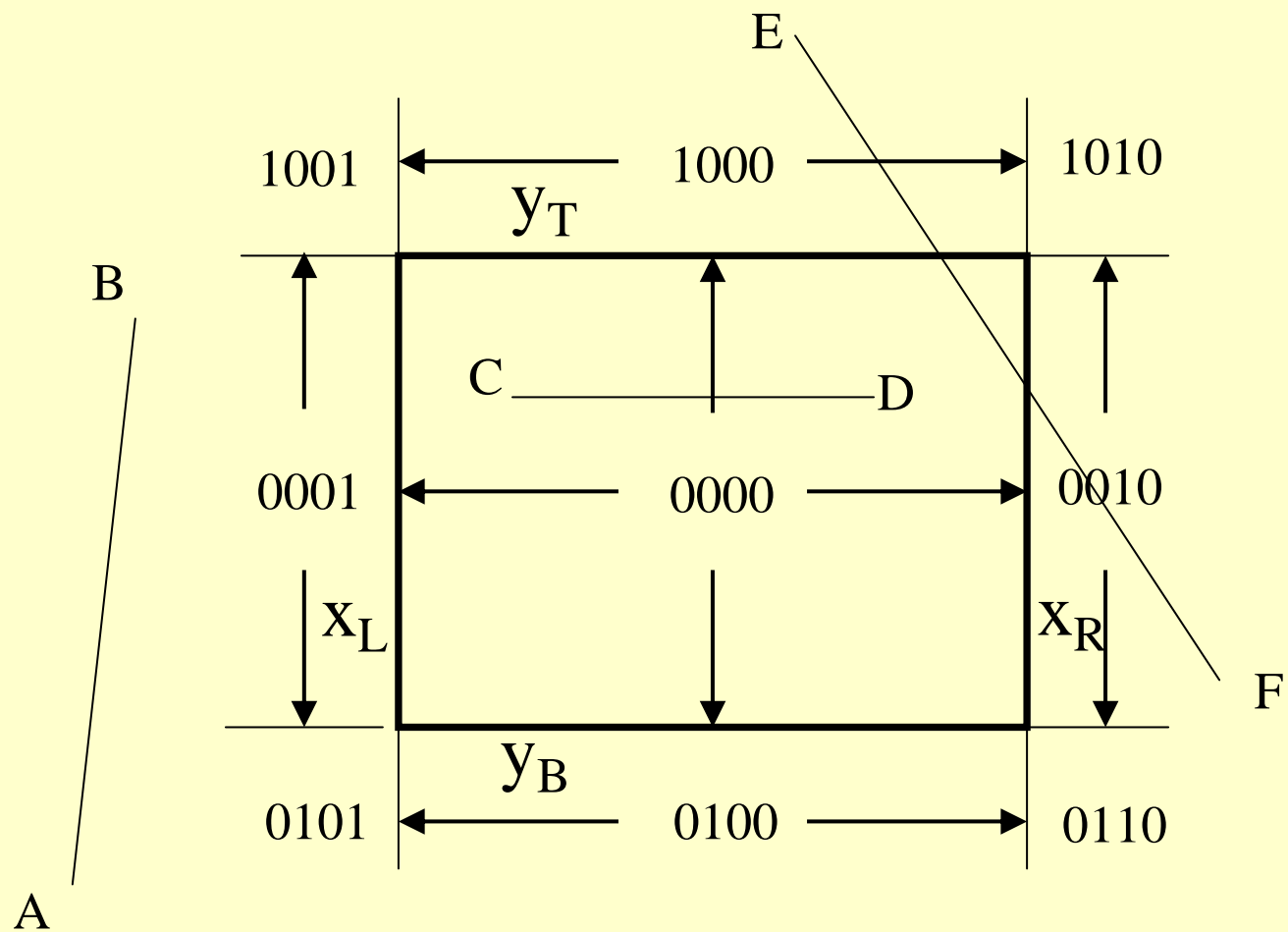


图4.11 窗口及线段端点的编码

- 如果线段两个端点的4位编码全为0, 则此线段全部在窗口内, 可直接接受; (如CD)
- 如果对线段两个端点的4位编码进行逻辑与运算, 结果为非0, 则此线段全部在窗口之外, 可直接舍弃; (如AB)
- 否则, 这一线段既不能直接接受, 也不能直接舍弃, 找到与窗口的一个边框的交点, 也赋予4位代码; (如EF)
- 对分割后的线段重复上述过程, 直到全部线段均被舍弃或被接受为止。

- 以下过程clip描述了这一算法：

```
var xl, xr, yt, yb: real;  
procedure clip(x0,y0,x2,y2: real);  
label return;  
type edge=[top, bottom, right, left];  
      outcode=set of edge;  
var c,c1,c2: outcode; x,y: real;  
procedure code(x,y:real; var c:outcode);  
begin  
  c:=[];  
  if x<xl then c:=[left]  
  else if x>xr then c:=[right];  
  if y<yb then c:=c+[bottom]  
  else if y>yt then c:=c+[top];  
end
```

```

begin
  code(x0,y0,c1); code(x2,y2,c2);
  while (c1<>[]) or (c2<>[])    // One is out of window
  begin
    if (c1 AND c2<>[]) then goto return; //all out of window
    c:=c1; if c:=[] then c:=c2; //c is out of window
    if left in c then //to find the point interacting with left
    begin
      y:=y0+(y2-y0)*(xl-x0)/(x2-x0);
      x:=xl
    end
    else if right in c then
    begin
      y:=y0+(y2-y0)*(xr-x0)/(x2-x0);

```

```

         $x := x_r$ 
    end
    else if bottom in c then
        begin
             $x := x_0 + (x_2 - x_0) * (y_b - y_0) / (y_2 - y_0);$ 
             $y := y_b$ 
        end
    else if top in c then
        begin
             $x := x_0 + (x_2 - x_0) * (y_t - y_0) / (y_2 - y_0);$ 
             $y := y_t$ 
        end;
    if c=c1 then
        begin

```

```
        x0:=x; y0:=y; code(x,y,c1)
    end
    else begin
        x2:=x; y2:=y; code(x,y, c2)
    end
end; //end of while
{*if we reach here, the line from (x0,y0) to (x2,y2) is visible*}
    showline(x0,y0,x2,y2);
return:
    end;
```

2. 中点分割裁剪法

- 算法思想

当一条线段既不能直接接受也不能直接舍弃, 欲求其与区域的交点时, 假设此交点在线段的中点, 如果估计错误, 则将直线分为两段, 并对该两段再分别加以测试。一直进行下去, 直到原来线段的一段被直接接受, 而另一段被直接舍弃。

- 设裁剪的区域是矩形, 要裁剪的线段为 P_1P_2 , 如图4.12所示。为求其可见部分AB, 算法可分为两个过程平行进行, 即:

- 1) 从 P_1 出发, 找出离 P_1 最近的可见点A;
- 2) 从 P_2 出发, 找出离 P_2 最近的可见点B;

此两点的连线AB, 即为原线段 P_1P_2 的可见部分。求 P_1 最近的可见点A的算法框图如图4.13所示。这里 ε 可取为一个像素的宽度。

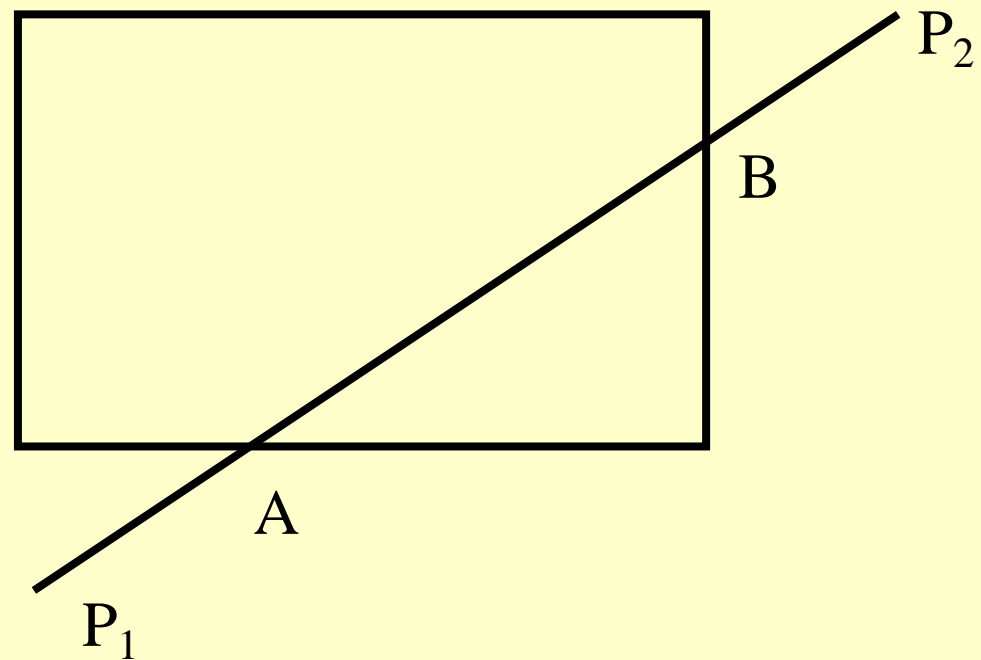


图4.12 中点分割裁剪示意图

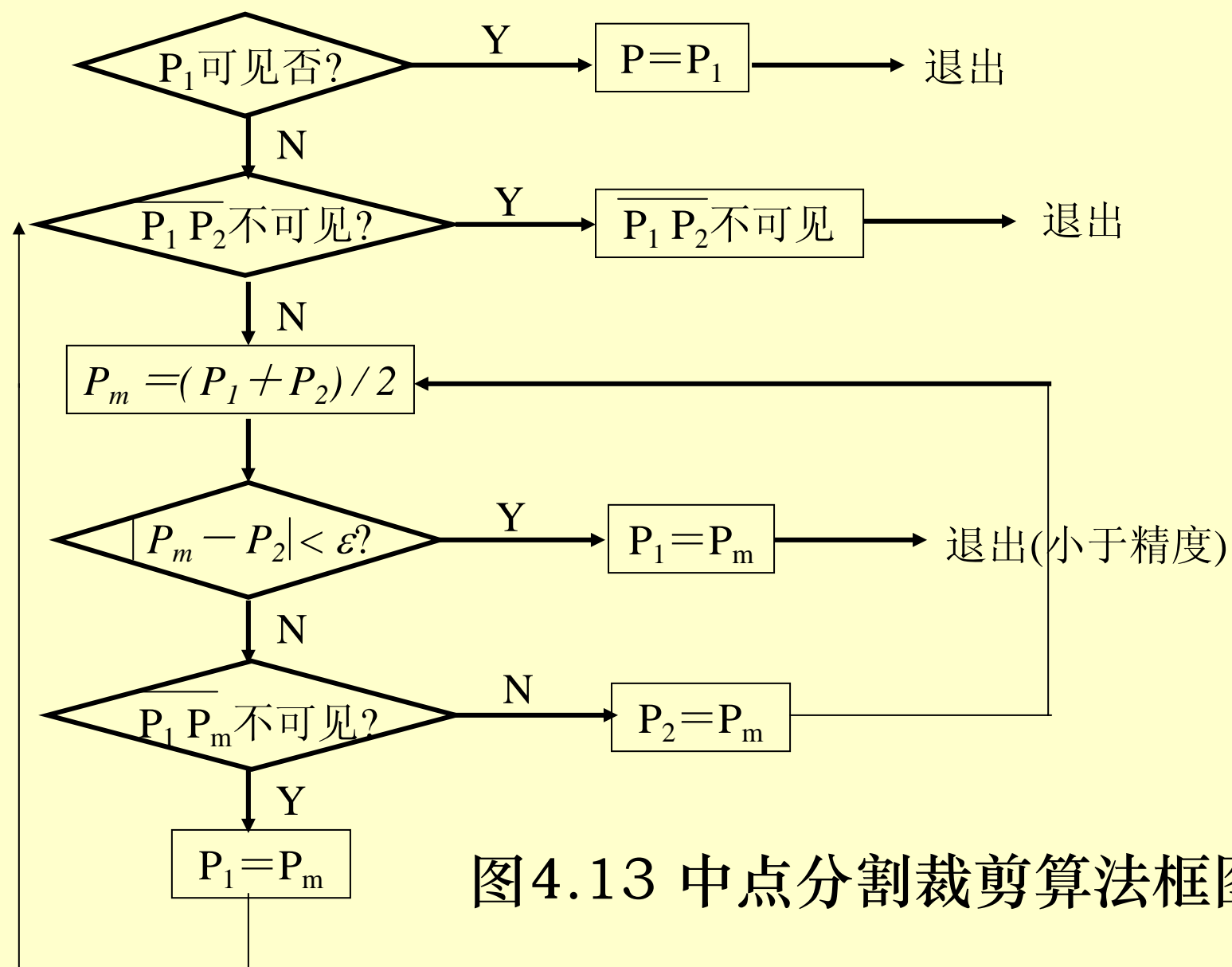
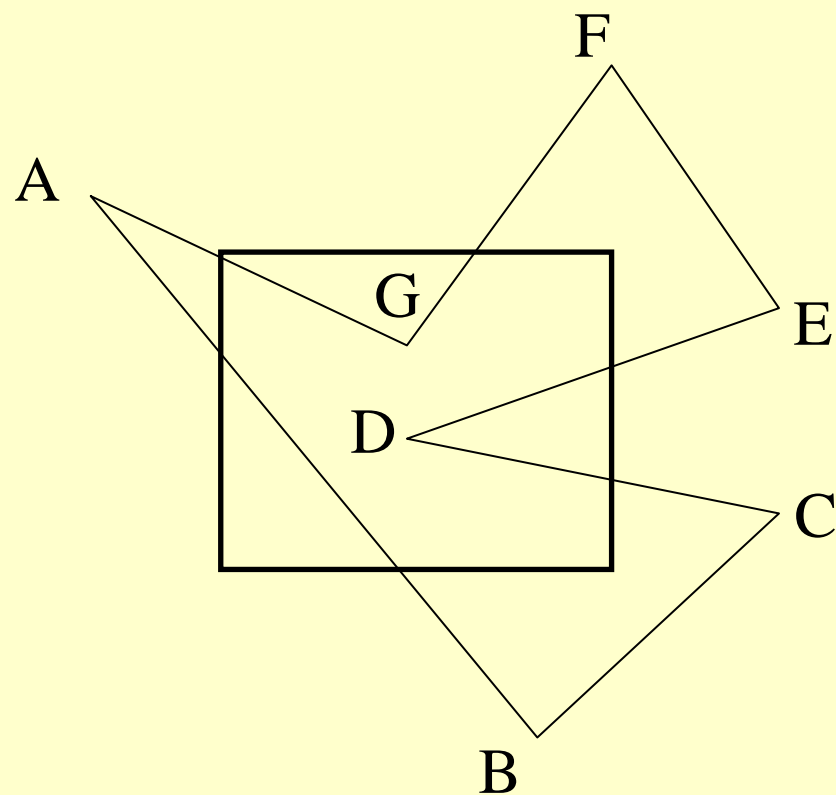


图4.13 中点分割裁剪算法框图

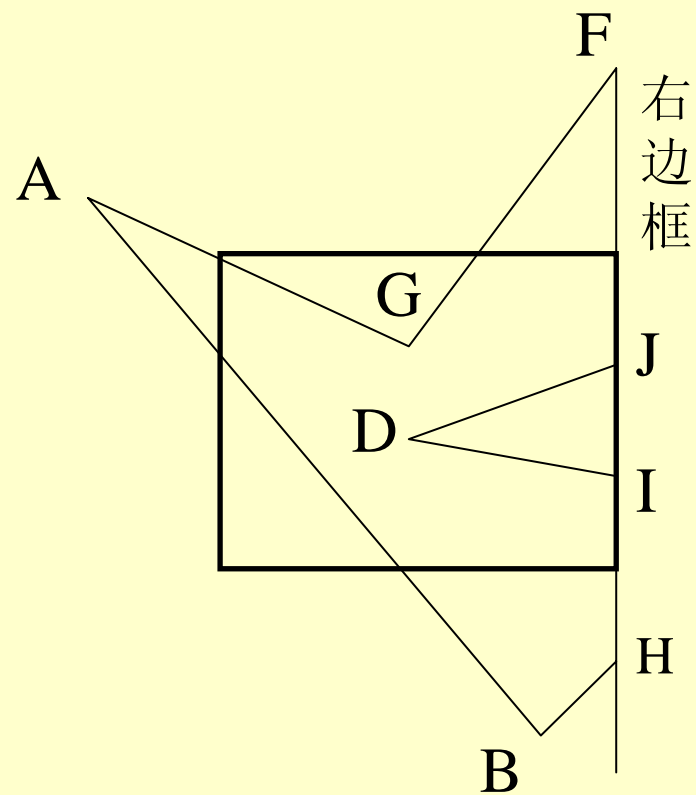
三、多边形的裁剪

1. 逐边裁剪法

- 每次用窗口的一条边界对要裁剪的多边形进行裁剪,把落在窗口外部区域的图形去掉,保留窗口内部区域的图形,并把它作为下一次待裁剪的多边形。
- 连续用窗口的4条边界对原始多边形进行裁剪,最后得到的多边形即为裁剪后的结果多边形。图4.14说明了这个过程。

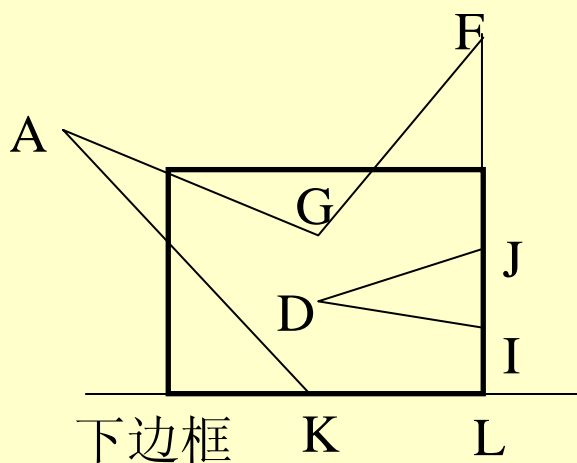


(a)原始多边形

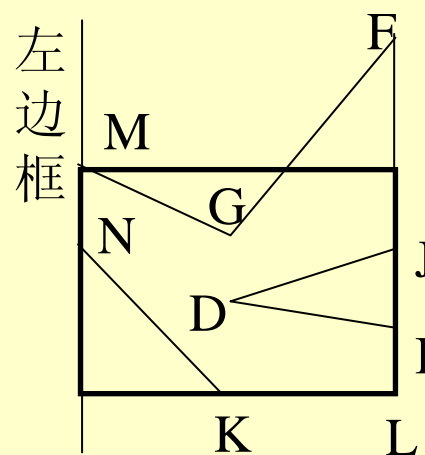


(b)右边界裁剪后的多边形

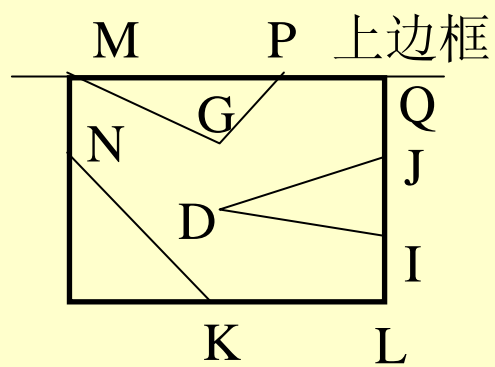
图4.14 多边形逐边裁剪算法执行过程



(c)下边界裁剪后的多边形



(d)左边界裁剪后的多边形



(e)上边界裁剪后的多边形

图4.14 多边形逐边裁剪算法执行过程

2. 双边裁剪法

- 设用户多边形为主多边形 P_s , 窗口为裁剪多边形 P_c 。同时设每一多边形按顺时针方向排列。
- 首先沿主多边形 P_s 任一点出发, 跟踪检测 P_s 的每一条边, 当 P_s 与 P_c 的有效边框相交时:
 - 1) 若 P_s 的边进入 P_c , 则继续沿 P_s 的边往下处理, 同时输出该线段;

- 2) 若 P_s 的边是从 P_c 中出来,则从此点(称前交点)开始,沿着窗口边框向右检测 P_c 的边,即用 P_c 的有效边框去裁剪 P_s 的边,找到 P_s 与 P_c 最靠近前交点的新交点,同时输出由前交点到此新交点之间窗口边上的线段;
- 3) 返回到前交点,再沿着 P_s 处理各条边,直到处理完 P_s 的每一条边,回到起点为止。

图4.15说明了双边裁剪算法的执行过程。

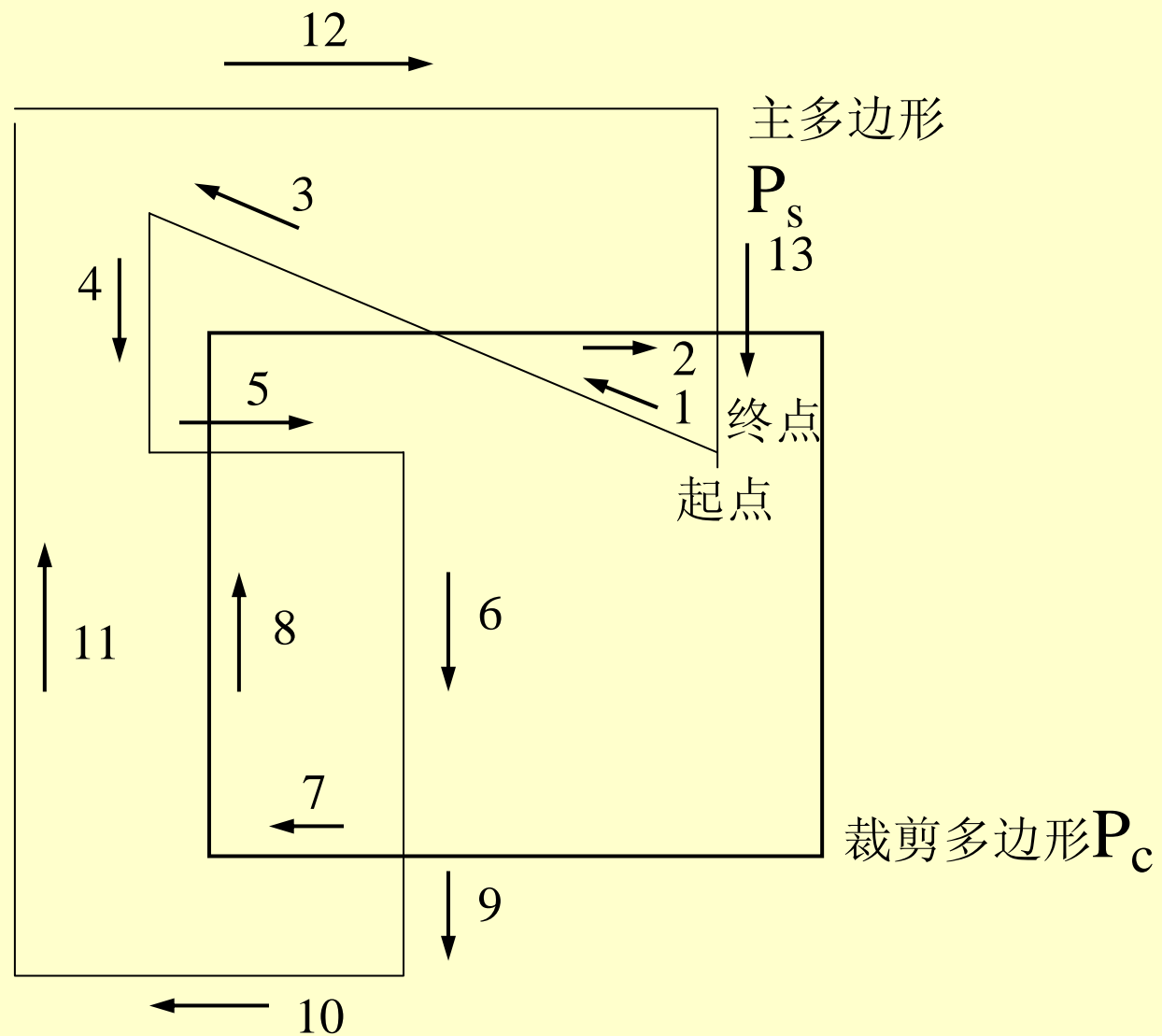


图4.15 多边形双边裁剪法执行过程图

4.5.2 三维图形的裁剪

- 前面我们介绍了三维投影空间的概念, 与设计二维裁剪算法的出发点相同, 在三维裁剪算法中, 裁剪操作应能迅速地接收完全可见线段, 舍弃完全不可见线段, 求出三维形体与裁剪窗口的交点, 从而得到形体内部分可见线段。
- 对应于立方体裁剪窗口6个面的方程(规格化)分别为:
 $x=-1; x=1;$
 $y=-1; y=1;$
 $z=-1; z=1$

- 空间任意一条直线段 $P_1(x_1, y_1, z_1)$ 和 $P_2(x_2, y_2, z_2)$, P_1P_2 端点和6个面的关系可转换为6位二进制代码表示:

第1位为1: 点在裁剪窗口的上面, 即 $y > 1$, 否则第1位为0;

第2位为1: 点在裁剪窗口的下面, 即 $y < -1$, 否则第2位为0;

第3位为1: 点在裁剪窗口的右面, 即 $x > 1$, 否则第3位为0;

第4位为1: 点在裁剪窗口的左面, 即 $x < -1$, 否则第4位为0;

第5位为1: 点在裁剪窗口的后面, 即 $z > 1$, 否则第5位为0;

第6位为1: 点在裁剪窗口的前面, 即 $z < -1$, 否则第6位为0。

- 如果一个点在窗口内,则两端点编码都为零。
- 对任意一条三维线段的参数方程为:

$$x=x_1+(x_2-x_1)t=x_1+pt$$

$$y=y_1+(y_2-y_1)t=y_1+qt$$

$$z=z_1+(z_2-z_1)t=z_1+rt$$

- 如求一条直线与裁剪空间上平面(如 $y=1$)的交点, 将 $y=1$ 代入, 得 $t = \frac{1-y_1}{q}$, 若 t 不在0到 1 的闭区间内, 则交点在裁剪空间外, 否则可求得:

$x = x_1 + \frac{1-y_1}{q}$, $z = z_1 + \frac{1-y_1}{q}$, 这时三维线段与裁剪窗口的有效交点为 $(x_1 + \frac{1-y_1}{q}, z_1 + \frac{1-y_1}{q})$;

- 类似可求出其它5个面与直线段的有效交点, 连续有效交点可得出落在裁剪窗口内的有效线段。
- 可以很方便得将二维裁剪中的Sutherland-Cohen算法与中点分割算法推广到三维裁剪。

第5章 人机交互技术

内容：交互的设备，交互任务，交互技术，交互过程及交互风格

目的：让学生掌握基本的人机交互方法

内容

5.1 交互的硬件设备

5.2 基本交互任务

5.3 高级的交互技术

5.4 输入过程基本处理模式

5.5 设计人机交互的一般风格及原则

5.1 交互的硬件设备

1. 定位设备

通过定位设备用户可在二维或三维坐标系统中给定点的位置,也就是使计算机能得到给定点在坐标系统中的坐标。

目前常用的鼠标器及其在屏幕上的光标便是最普遍的定位设备。

2. 键盘设备

计算机上最普遍使用的交互工具. 不仅可用于输入字母和数字, 而且可用来选择功能。

3. 取数设备

取数设备可用旋转或平移的电位器来实现,电位器输出的电压值可通过模数转换转成二进制数输入计算机。

4. 选择设备

用选择设备可选择功能或图形元素等。

5. 其它输入设备

语音识别器是另一种输入设备;还有人研制供画家用画笔绘画的输入板;在三维图形输入方面随着虚拟现实的发展也出现不少设备。

5.2 基本交互任务

1. 定位

- 此任务是确定平面上或空间的一个点: (x,y) 或 (x,y,z) 。一个直接的办法便是从键盘上输入二个或三个代表坐标的数。
- 在选定一个子图、菜单等操作时, 常常并不需要很精确的定位。这时, 我们可用在上一节中说明的各种定位设备和屏幕上的光标来实现。

2. 选择

- 选择任务是在某一选择集中选出一个元素。包括选择命令、设置参数、选择类别、选择属性、选择子图或图元等。
- 菜单功能是最普遍使用的一种交互方法。很多系统已有实现菜单的工具提供用户编程。菜单的元素可按英文字母次序来排,也可按功能目的来分类。如果选择的元素比较多,这时可根据它们内部分类形成一树状结构。例如,图5.1中是AutoCAD中选择用起点、圆心及终点绘图的选择过程。

File Edit View Insert Format Tool Draw Dimension Modify		
	Line Ray Construction Line	
	Multiline Polyline 3D Polyline Polygon Rectangle	
	Arc	3points
	Circle	Start,Center,End Start,Center,Angle Start,Center,Length

图5.1 AutoCAD选择绘制圆弧功能的过程

- 对话框也可提供选择功能。建立对话框的工具也可从各种软件的应用程序界面工具中找到。
- 在编辑已绘制的图形时, 选择已绘制的子图或图元便十分重要。一般可对每一个子图预先求一个边界盒或比边界盒稍大一点的 ε 边界盒(参见图5.2)。
- 所谓边界盒是边和坐标轴平行且包含该子图的最小的矩形。在选择时用户指定一个选择点。若这点仅在某一个子图的 ε 边界盒内, 则这子图便被选中。若有几个子图的 ε 边界盒被选中, 最短距离那一个子图便是被选中的。

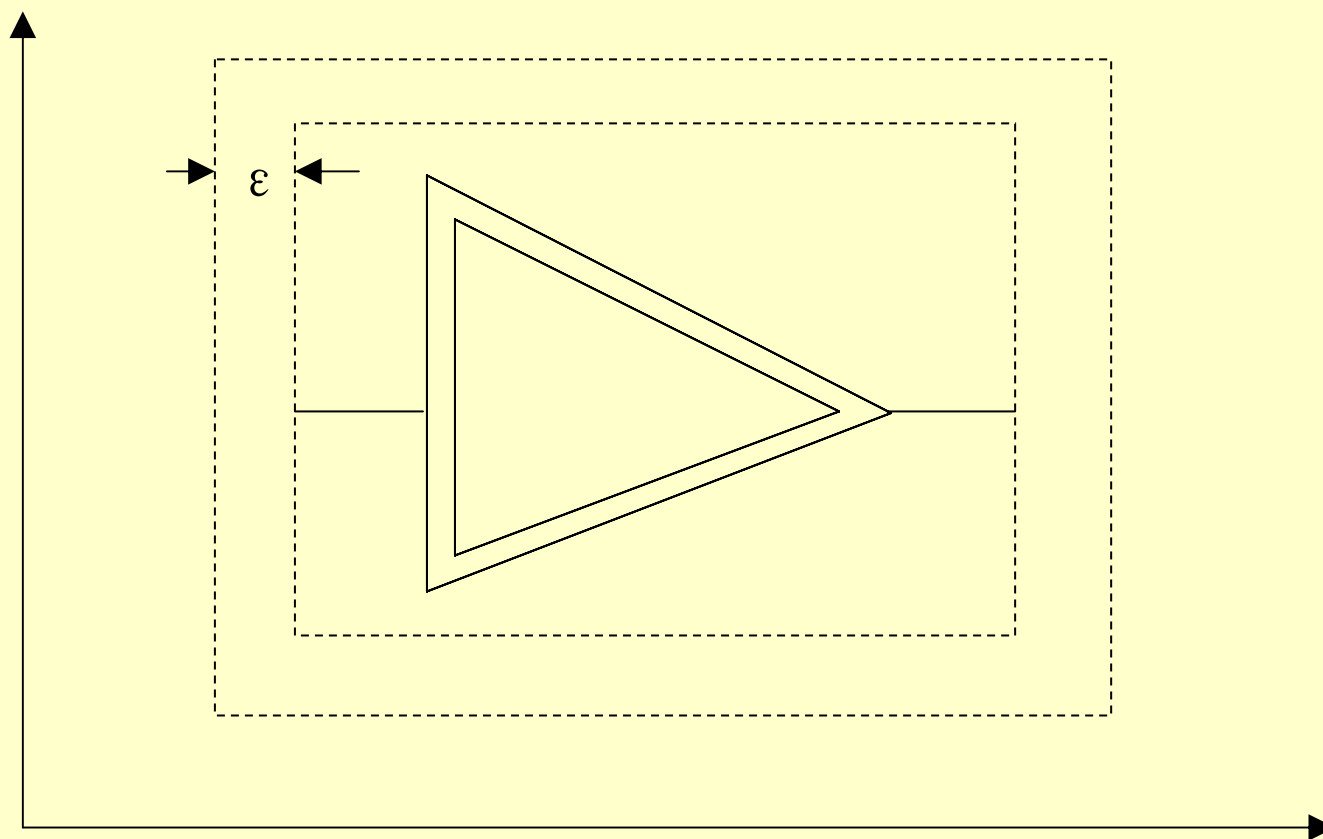


图5.2 ϵ 边界盒

3. 数量输入

数量输入是要在最大和最小二数值之间输入一个值。

4. 文本输入

文本是一个字符串。我们可以用键盘输入字符串,也可用一类似图形输入板的书写板去输入字符。语音输入也是文本输入以及功能选择的一种输入方法。

5. 三维交互任务

- 在三维空间中定位、选择及令对象旋转等称为三维交互任务。
- 在三维空间绘制了一个对象后, 常常希望从各个角度去观察这对象, 或让这对象在屏幕上旋转一定的角度。为此可用经度方向的角度 α 及纬度方向的角度 β 去控制对象的旋转(参见图5.3)。这两个值可用二维空间中的鼠标去输入, 如 x, y 方向的增量分别看作 α 及 β 的增量。

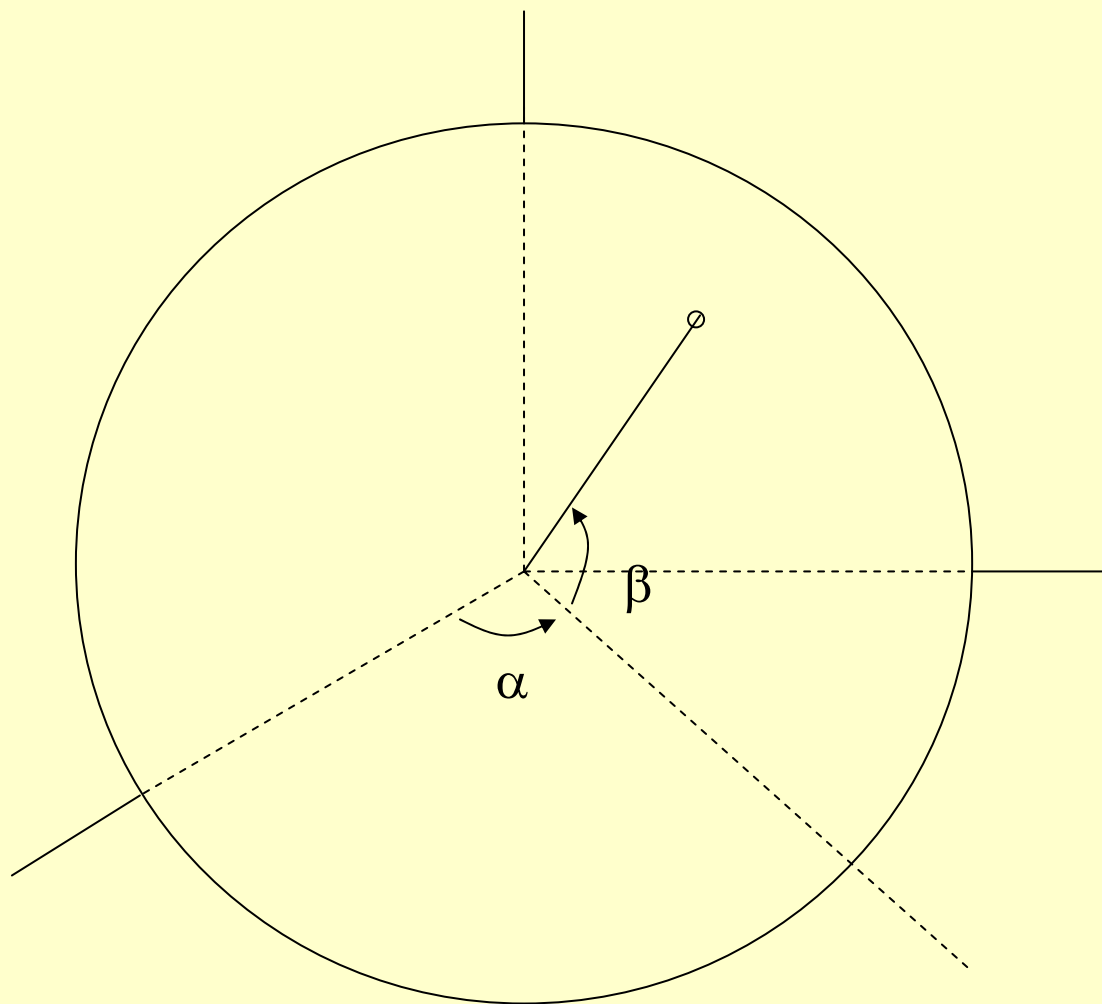


图5.3 控制对象旋转的角 α 及 β

5.3 高级的交互技术

1.几何约束

- 第一种几何约束是定位约束。在屏幕上定义一个不可见的网格,有时也可用点的形式显示网格结点的位置。对用任何方式输入一个点都用离它最近的一个网格结点来代替。
- 第二种几何约束为方向约束。例如,要绘制的线只能是垂直或水平两个方向,当给定的起点和终点连线和水平线的交角小于 45° 时,便绘出一条水平线,否则就绘垂直线。如图5.4所示。

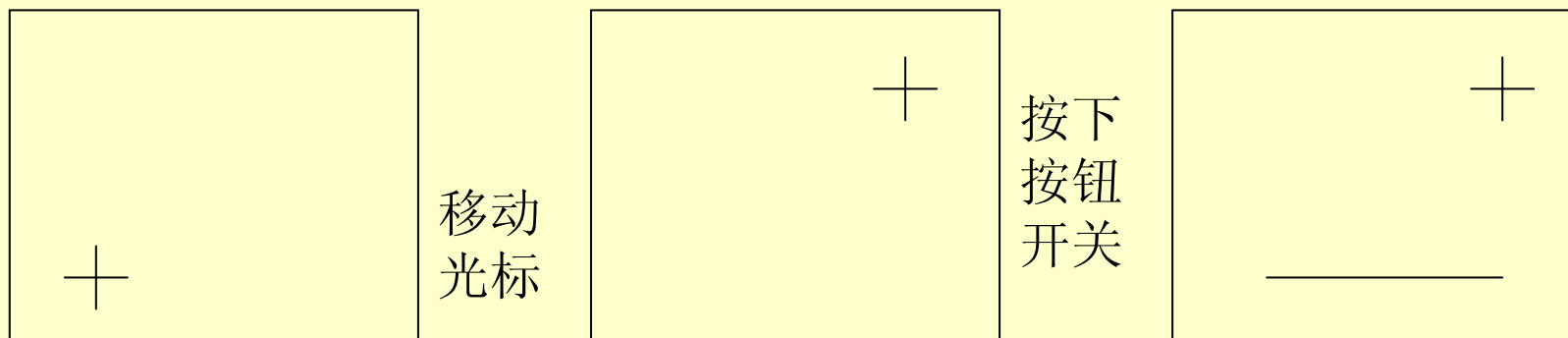


图5.4 方向约束

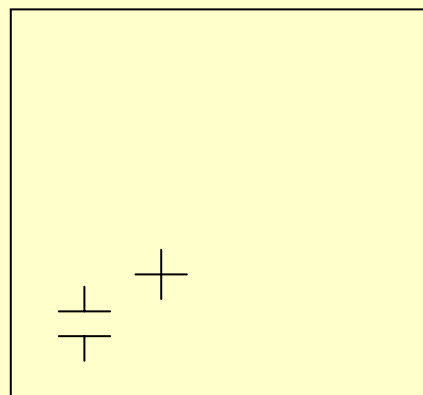
2. 拖动

当要把一个子图放到一个新位置时,如果不是简单地用光标指定新位置的一个点,而是随着鼠标器引导光标移动的同时,子图也跟着拖到新的位置。这将使用户感到更直观并可把这子图的位置定得更恰当。(参见图5.5)

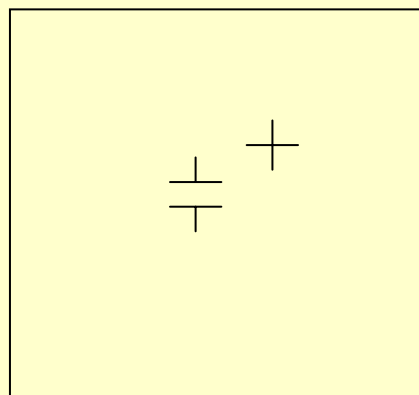
3. 橡皮筋技术

在平面上确定一条直线时,在起点确定后,光标移出去定终点时,在屏幕上始终显示一条连结起点和光标中心的直线,这条直线随着光标中心位置的变动而变动。如图5.6画圆的切线。

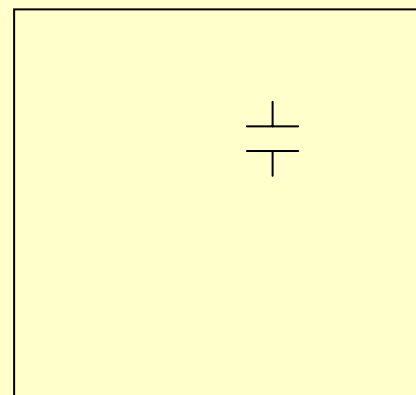
橡皮筋也可以做成矩形,如图5.7所示,这时用一个矩形代替直线。



光标选图元



拖动图元



到目的地
按一下按钮开关

图5.5 拖动

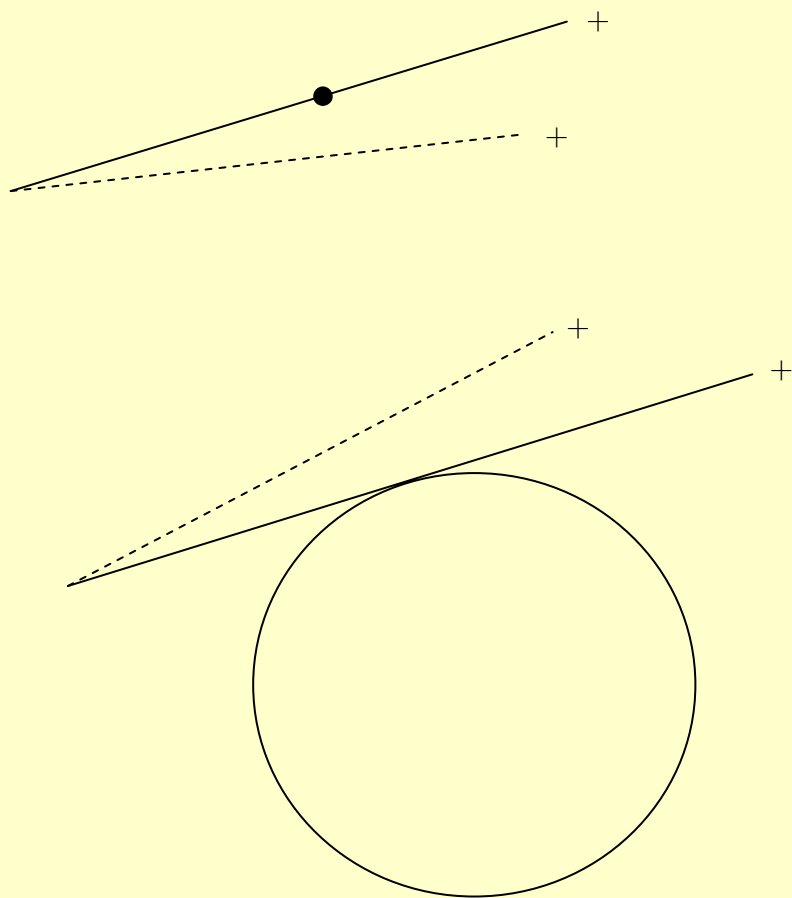


图5.6 橡皮筋技术

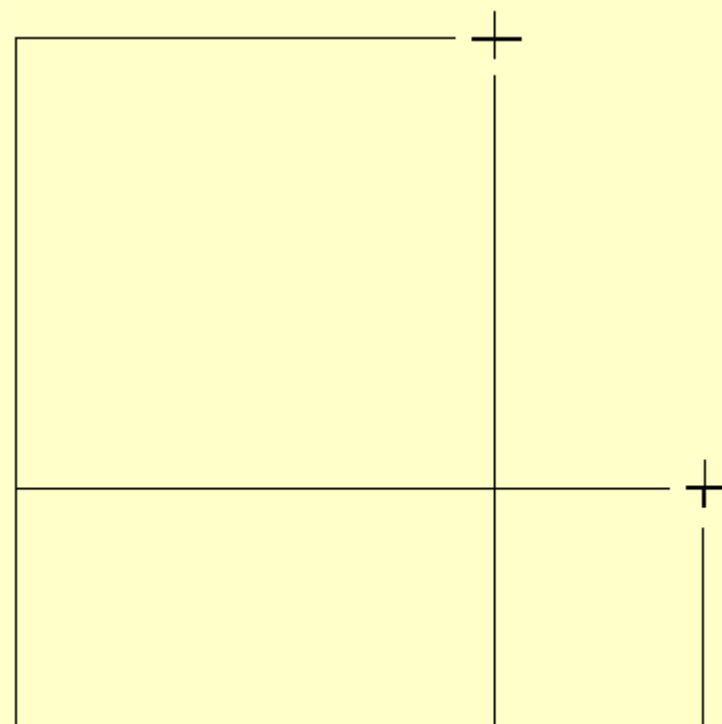


图5.7 矩形橡皮筋

4. 图形变比

为了把屏幕上显示的图的某部分放大或缩小,可用图形变比(zoom)。

5. 引力场效果

在用光标选图时,要把光标中心移到图段上去,如果被选的图形只是一个点或一根线段,对准该点或线段比较困难。可在每一条线段的周围假想有一个区域,这个区域像一根香肠或哑铃,如图5.8所示,光标中心落在这个区域内时,就自动地被直线上最近的一个点代替。如同被吸引一样。

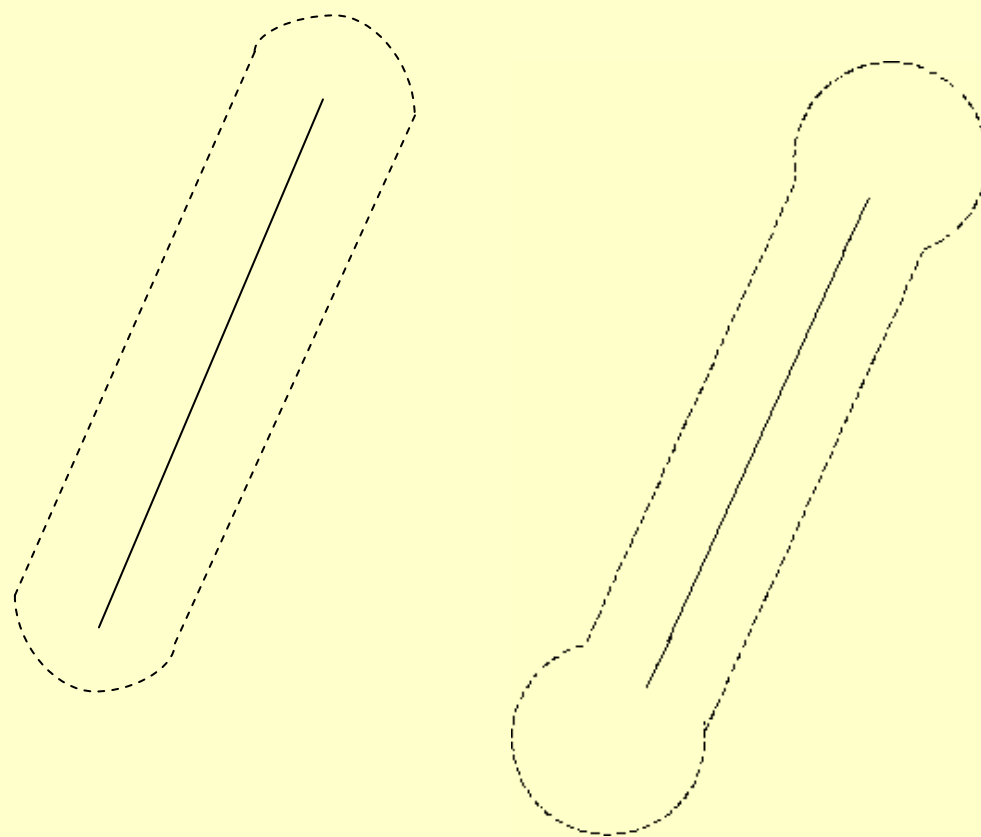


图5.8 引力场

6. 标尺和导向线

- 为了能用比较准确的尺寸来绘图,可以在屏幕上使用标尺。用户可用适当的命令在屏幕上显示标尺,如图5.9,利用标尺来决定长度或位置。
- 图5.10说明导向线的用法。图中希望两个矩形的左端能对齐,用户命令系统显示一导向线,达到目的后用命令去掉导向线。

7. 坐标显示

为了能把点定位定得准确,可以在坐标附近显示出点的坐标值,如图5.11。

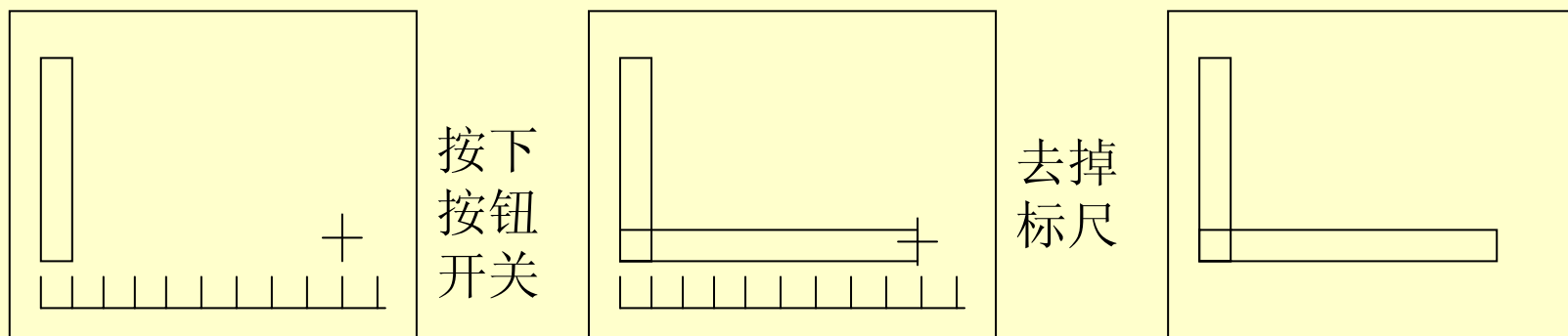


图5.9 标尺

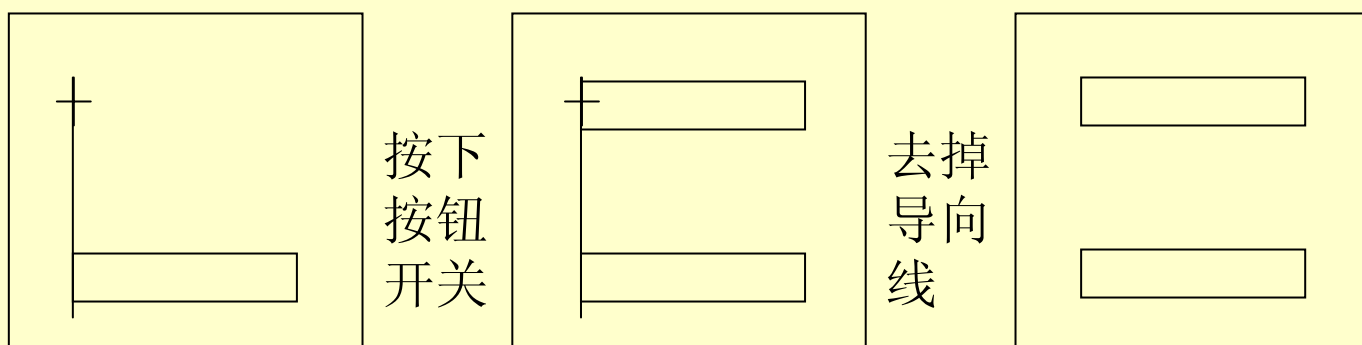


图5.10 导向线

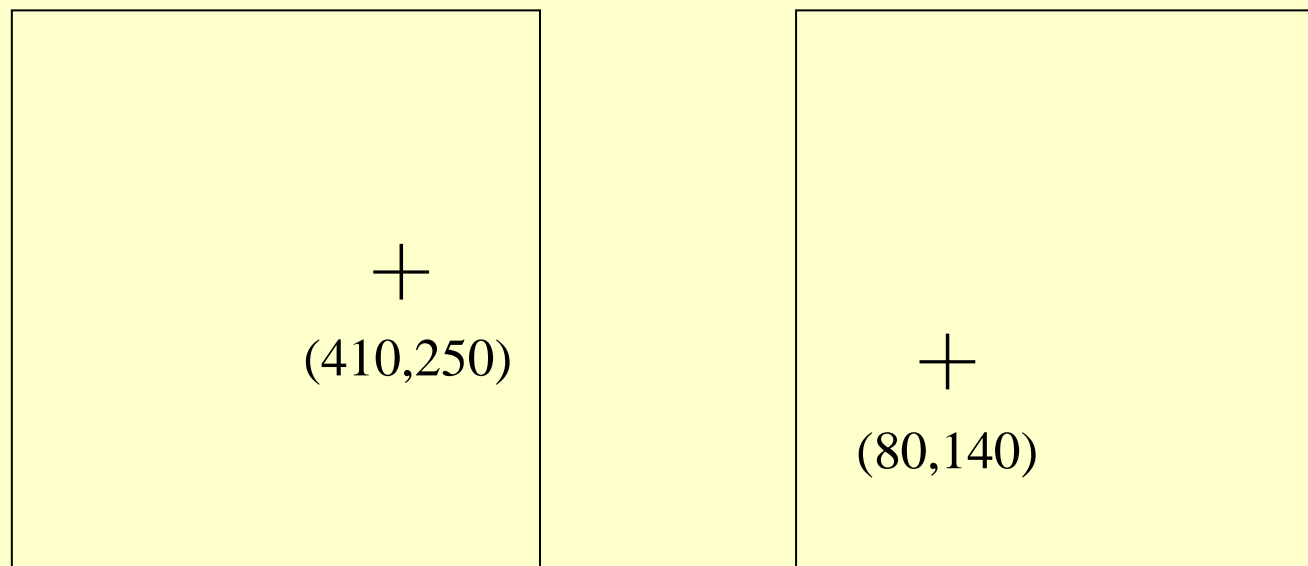


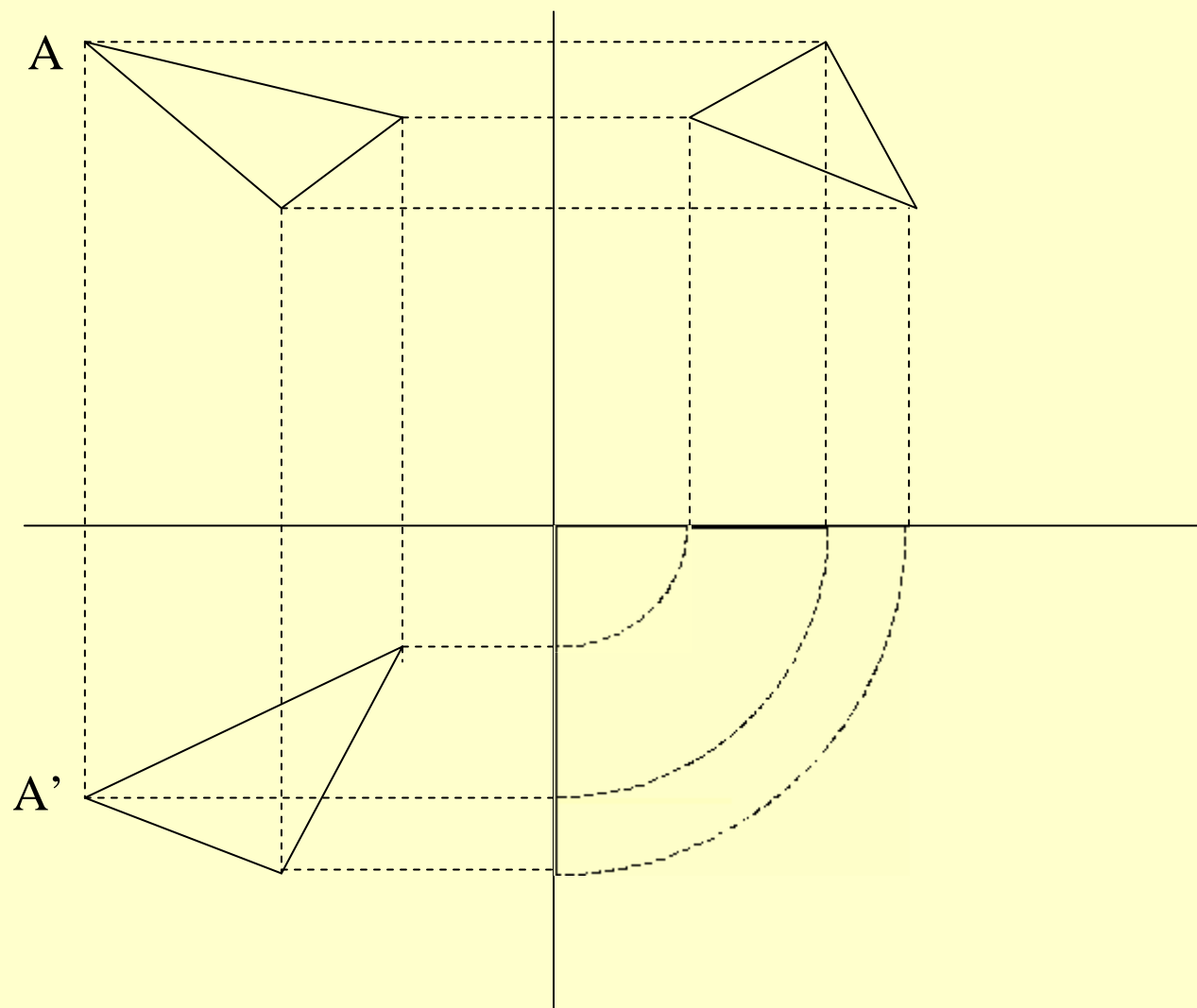
图5.11 坐标显示

8. 在三视图上作三维输入

在输入一个立体图时,常是一个个点,一条条线或一个个面单独输入的。对于输入一个点来说,只要在两个视图上把点的对应位置指定后便唯一确定了三维空间中的一个点(见图5.12)。这样把直线段上两端点在三视图上输入后便可决定三维空间的一条直线。同理输入面、体。

9. 推移

有很多图是“二维半”的,如图5.13中的柱体和旋转体。对这种物体可先输入一个二维图形,例如柱体的底面或旋转体的截面(参见图5.14)。对柱体还要给出一个高度 h ,把输入的底面沿法线方向延伸。对于旋转体,把输入的二维图形绕指定的对称轴旋转一个指定的角度,便可生成图5.13中的旋转体。



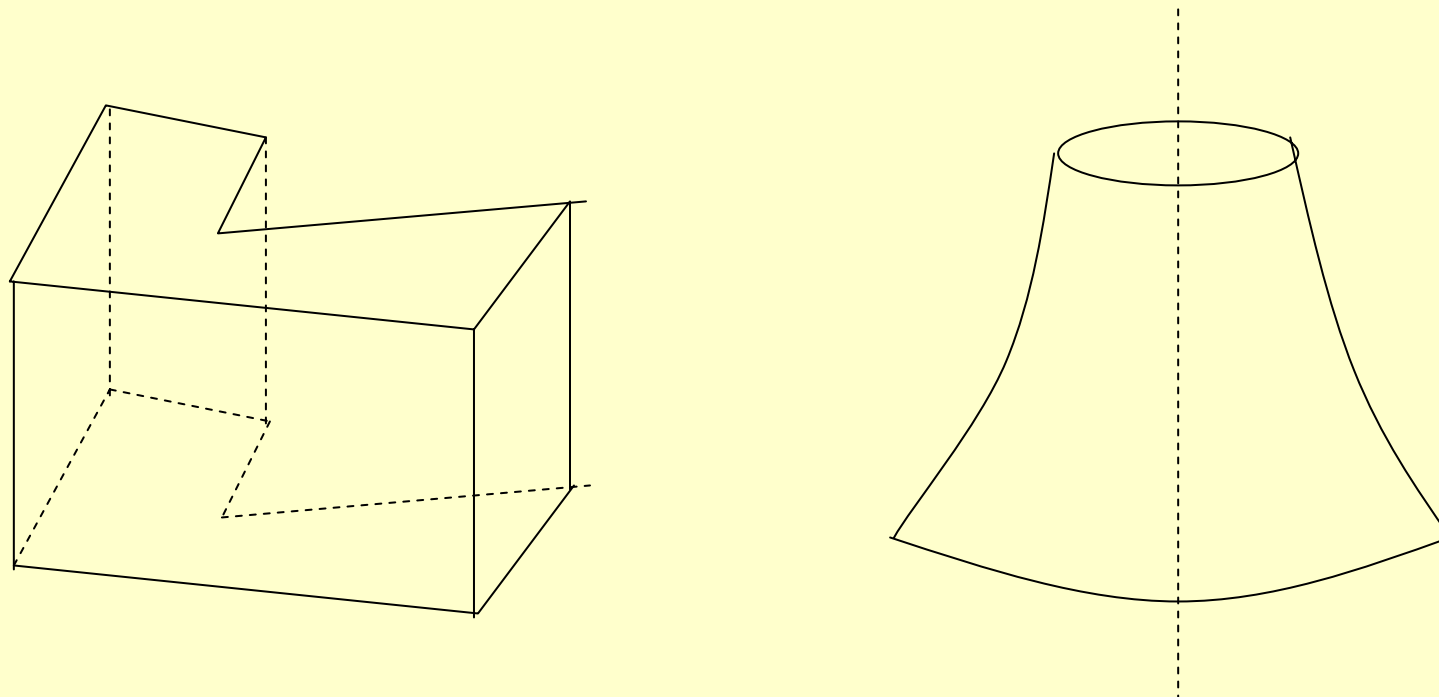


图5.13 二维半图形

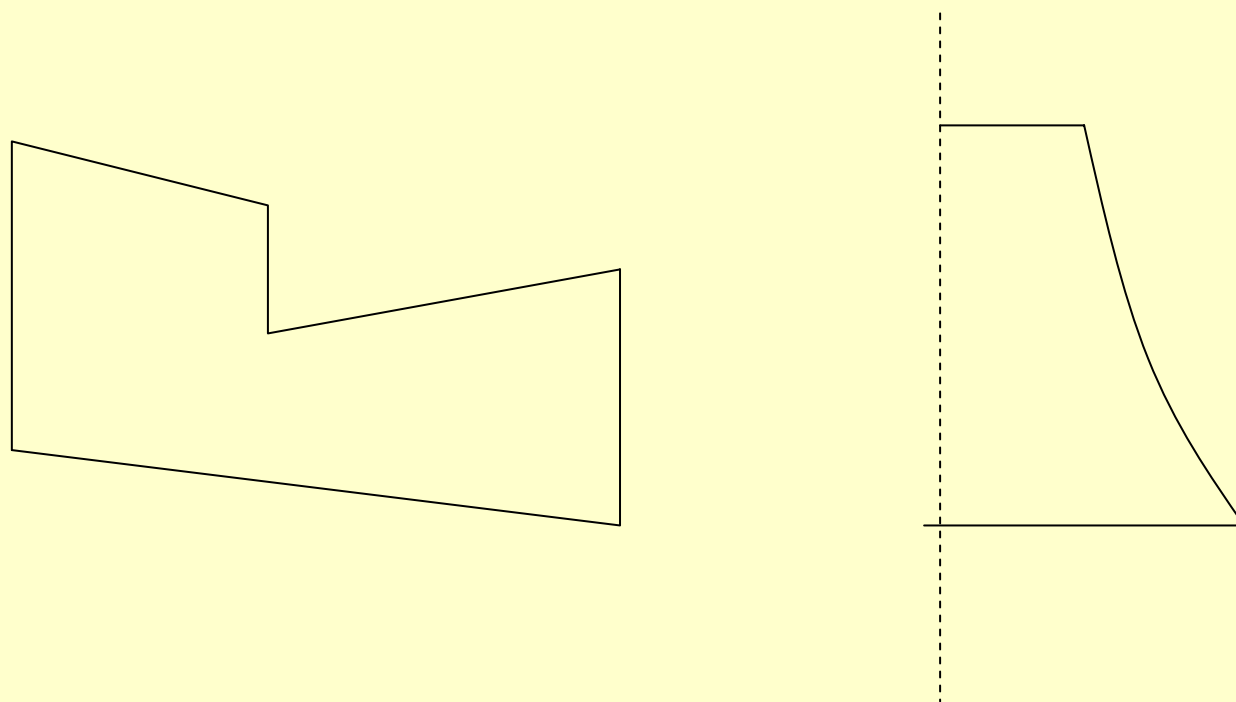


图5.14 用二维图形生成二维半图形

10.结构平面

- 输入立体图形常是由简到繁地构造。例如要生成图5.17的图形, 先要生成图5.15中的二维半图形, 再在平面ABCD上用推移方法拉伸出一个柱体来。
- 在这里要把平面ABCD旋转到它的法线恰好指向用户的位置, 如图5.16所示, 然后在ABCD平面上绘一个圆, 在通过拉伸操作, 在ABCD上加上一个柱体, 如图5.17所示。
- 这个ABCD平面便称为结构平面。

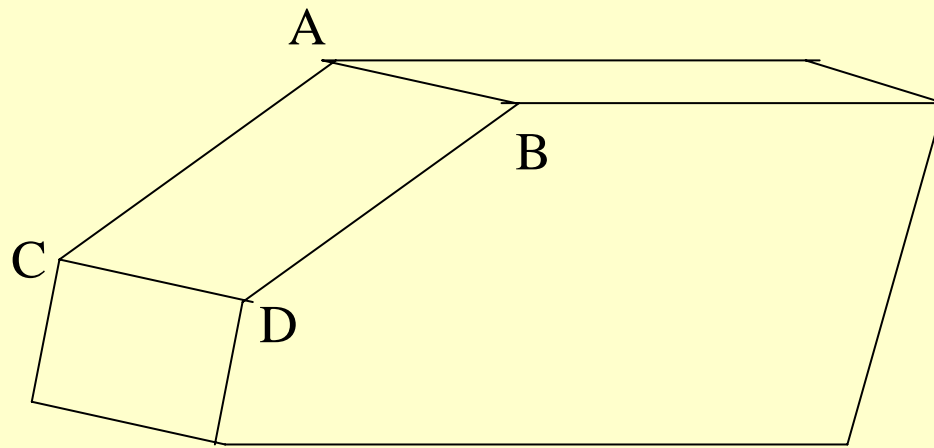


图5.15 指定结构平面
ABCD

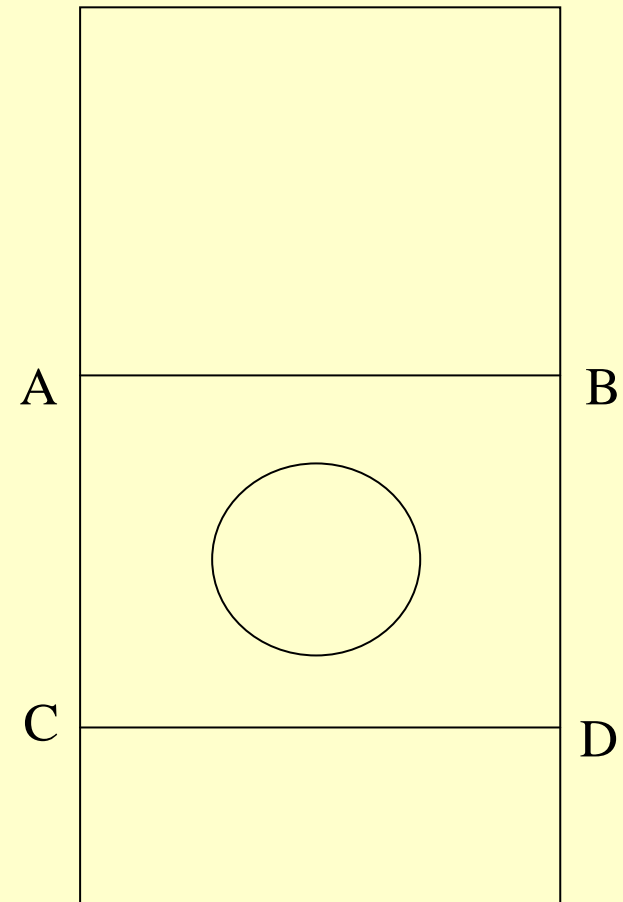


图5.16 把结构平面转
成面向用户

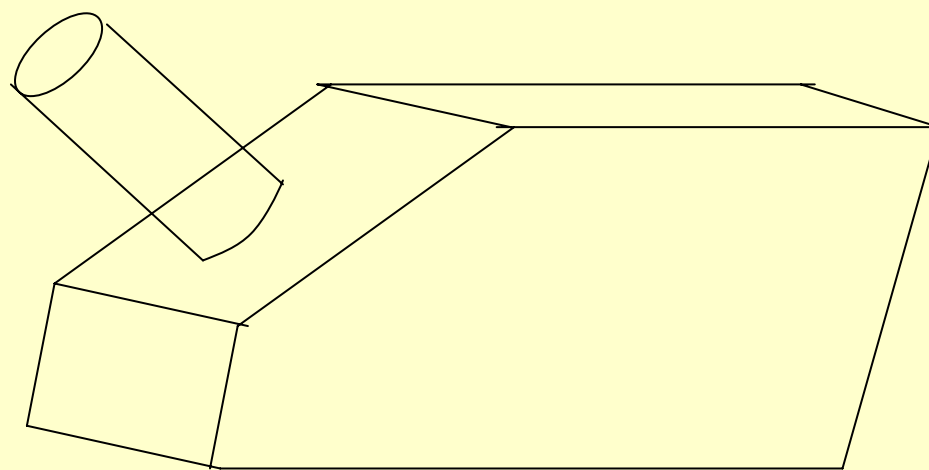


图5.17 形成要输入的图

5.4 输入过程基本处理模式

- 为了实现上两节中的各种交互功能, 必须把从输入设备输入的信息和应用程序有机地结合好。
- 由于输入设备是多种多样的, 而且对一个应用程序而言可以有多个输入设备。这就要求输入过程的处理要有一些合理的办法。
- 现在常用的3种基本处理模式是:

1.请求方式(Request mode)

- 用请求方式时, 输入设备的初始化是在应用程序中安排的。
- 例如, 当应用程序执行到某点时需要输入一个点, 在应用程序的对应位置便安排一条输入命令。这命令初始化输入设备并等待用户输入。而应用程序也暂停在这点上, 一直到用户输入了一个信息(如把光标移到一定位置并按一下鼠标器上的按键), 这时输入命令才返回到应用程序, 应用程序才继续执行下去。

2. 取样方式(Sample mode)

当把一台或多台输入设备定义为取样方式后,这些设备立即会把信息输入进来,和应用程序的输入命令无关。

3. 事件方式(Event mode)

当设备设置成事件方式后,输入设备和程序同时工作。当用户在输入设备上发生一个表示输入的动作(如按按钮)便产生一个事件,输入信息及设备编号等便会存放到一个事件的队列中去。不同的应用程序可到队列中来检索和取走属于该应用程序有关的事件(参见图5.18)。

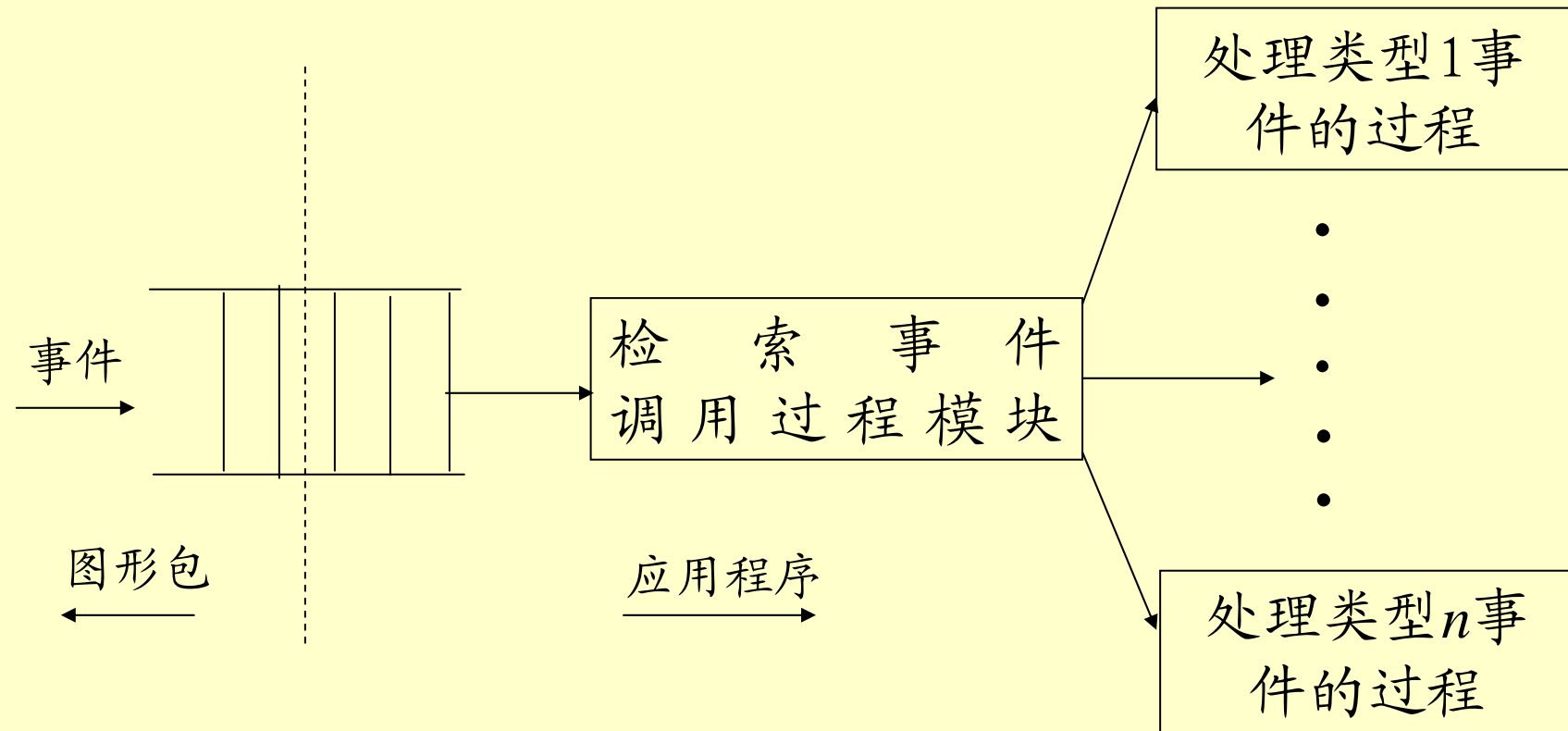


图5.18 事件方式

5.5 设计人机交互的一般风格及原则

- 现在计算机系统的人机界面一般具有下列风格, 即: “所见即所得”(what you see is what you get), 直接操作(direct manipulate)及菜单和图形符号(icon)驱动。
 - “所见即所得”在交互式图形系统中一般都能做到, 即在屏幕上所见到的设计结果和用硬拷贝所得的输出结果是一致的。
 - 直接操作是对对象、特性及关系等操作时用户可得到一种直观及形象的表示, 以说明这个操作是正确地被执行了。

- 图形符号驱动的目的是要用户不需要专门学习及记忆便可借助于菜单选择来运行系统。要做到这一点最主要的是要设计好图形符号，使它一看便知道它代表什么操作。
- 用命令语言作交互手段也在一些人机交互界面中使用。
- 自然语言也是一种有希望的交互手段。
- 人机界面设计的一些基本原则：
 - (1) 简单易学
 - (2) 提供反馈
 - (3) 对错误操作容易纠正 Undo
 - (4) 设计一致性 功能布局、颜色、代号等应一致

第6章 曲线曲面的表示

知识点: Bezier曲线、曲面和B样条曲线、曲面以及Coons曲面的表示方法

教学目的: 让学生掌握不规则曲线、曲面的表示方法

本章内容

6.1 曲线和曲面的基本概念

6.2 Bezier曲线曲面

6.3 B样条曲线曲面

6.4 Coons曲面

6.1 曲线和曲面的基本概念

- 曲线 描述物体轮廓, 运动轨迹
- 方法
 - 规则 代数方程定义, 如:
$$x^2 + y^2 + z^2 = R^2, y=0$$
 - 不规则 给定控制点拟合
- 曲面 描述物体形状的一种方法, 即3D物体可描述为一张曲面包含的3D空间
- 方法
 - 规则的平面、圆柱面、圆锥面、球面

- 拟合的或构造的：

- Bézier曲面、B样条曲面、孔斯曲面

- 方法的特点

- 形状不依赖坐标系

- 人机交互方便

- 易离散生成、构造灵活、易于拼接等

6.2 Bézier曲线曲面

6.2.1 Bézier曲线

6.2.2 Bézier曲面

6.2.1 Bézier曲线

一、Bézier曲线的定义

- 给定空间 $n+1$ 个点的位置矢量 $P_i(i=0,1,\dots,n)$, 则称下面参数曲线为 n 次Bézier曲线:

$$P(t) = \sum_{i=0}^n P_i J_{i,n}(t), 0 \leq t \leq 1$$

见图6.1。其中 $J_{i,n}(t)$ 是Bernstein基函数:

$$J_{i,n}(t) = C_n^i t^i (1-t)^{n-i}, i = 0, 1, \dots, n$$

$$C_n^i = \frac{n!}{i!(n-i)!}$$

$$J_{0,n}(t) = (1-t)^n, J_{n,n}(t) = t^n$$

- 折线 $P_0P_1 \dots P_n$ 为 $P(t)$ 的控制多边形, P_0, P_1, \dots, P_n 各点为 $P(t)$ 控制顶点。

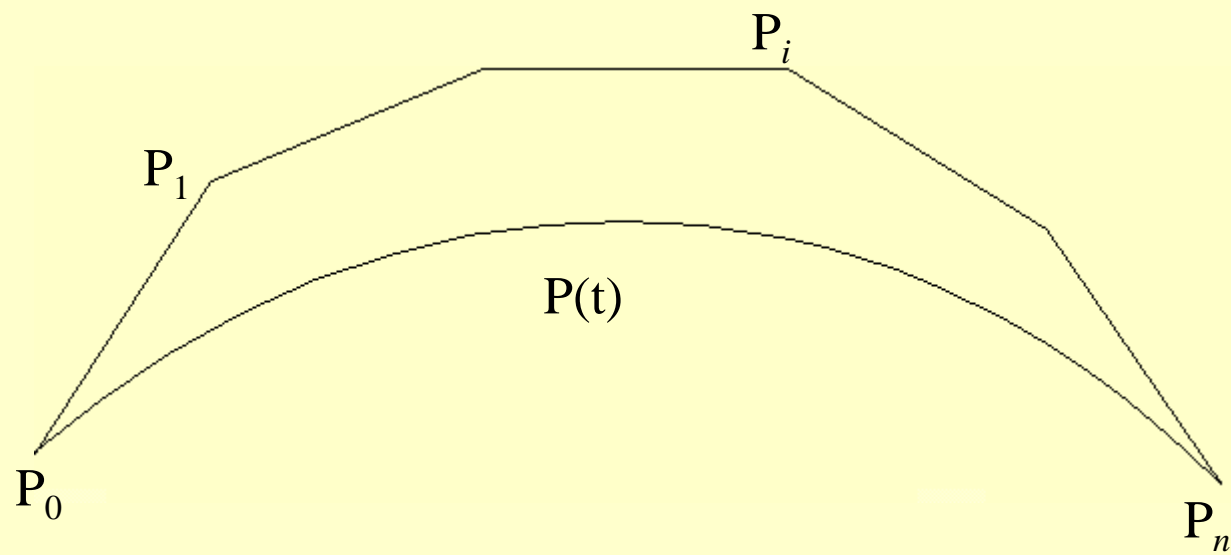


图6.1 Bézier曲线

二、Bézier曲线的性质

(1)端点位置

P_0, P_n 为 $P(t)$ 的端点: $P(0)=P_0, P(1)=P_n$

(2)端点切线

$P(t)$ 在 P_0 点与边 P_0P_1 相切, 在 P_n 点与边 $P_{n-1}P_n$ 相切:

$$P'(0)=n(P_1-P_0), P'(1)=n(P_n-P_{n-1})$$

(3)端点的曲率

$P(t)$ 在 P_0, P_n 两端点曲率分别为:

$$K(0)=\frac{n-1}{n} \cdot \frac{|\overrightarrow{P_0P_1} \times \overrightarrow{P_1P_2}|}{|\overrightarrow{P_0P_1}|^3}$$
$$K(1)=\frac{n-1}{n} \cdot \frac{|\overrightarrow{P_{n-2}P_{n-1}} \times \overrightarrow{P_{n-1}P_n}|}{|\overrightarrow{P_{n-1}P_n}|^3}$$

(4)凸包性

Bézier曲线位于其特征多边形构成的凸包之中。

$$\because J_{i,n}(t) \geq 0,$$

$$\sum_{i=0}^n J_{i,n}(t) \equiv 1 \quad (0 \leq t \leq 1)$$

即, $P(t)$ 为 P_0, P_1, \dots, P_n 各点的凸线性组合。曲线 $P(t)$ 必在点 P_0, P_1, \dots, P_n 凸包内(凸多边形)。

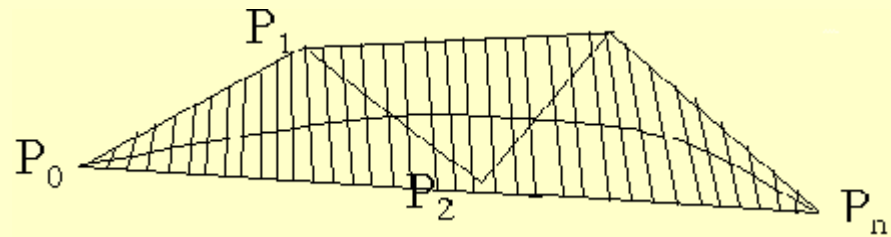


图6.2

(5)几何不变性

几何特性不随一定的坐标系变换而变化的性质称几何不变性。Bézier曲线的形状仅与各顶点 P_i 位置有关, 与坐标系选择无关。

(6) 交互能力

改变 $P(t)$ 的形状, 只要改变 P_0, P_1, \dots, P_n 位置, 把控制多边形作为输入和人机交互的手段。

(7) 保凸性

如果平面上的凸控制多边形能导致所生成的曲线为凸曲线, 称这个生成曲线的方法具有保凸性. Bézier曲线有保凸性。

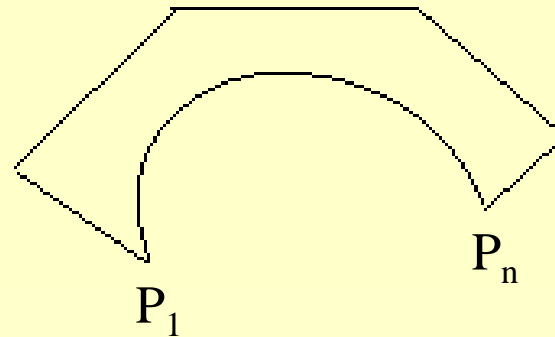


图6.3 保凸性

6.2.2 Bézier曲面

一、定义

在空间给定 $(n+1) \times (m+1)$ 个点 $P_{ij}(i=0,1,\dots,n; j=0,1,\dots,m)$, 称下列张量积形式的参数曲面为 $n \times m$ 次Bézier曲面:

$$P(u, v) = \sum_{i=0}^n \sum_{j=0}^m P_{ij} J_{i,n}(u) J_{j,m}(v), 0 \leq u, v \leq 1$$

一般称 P_{ij} 为 $P(u, v)$ 的控制顶点, 把由两组多边形

$$P_{i0} P_{i1} \dots P_{im} \quad (i=0,1,\dots,n)$$

$$P_{0j} P_{1j} \dots P_{nj} \quad (j=0,1,\dots,m)$$

组成的网称为 $P(u, v)$ 的控制网格, 记为 $\{P_{ij}\}$, 如图6.4。

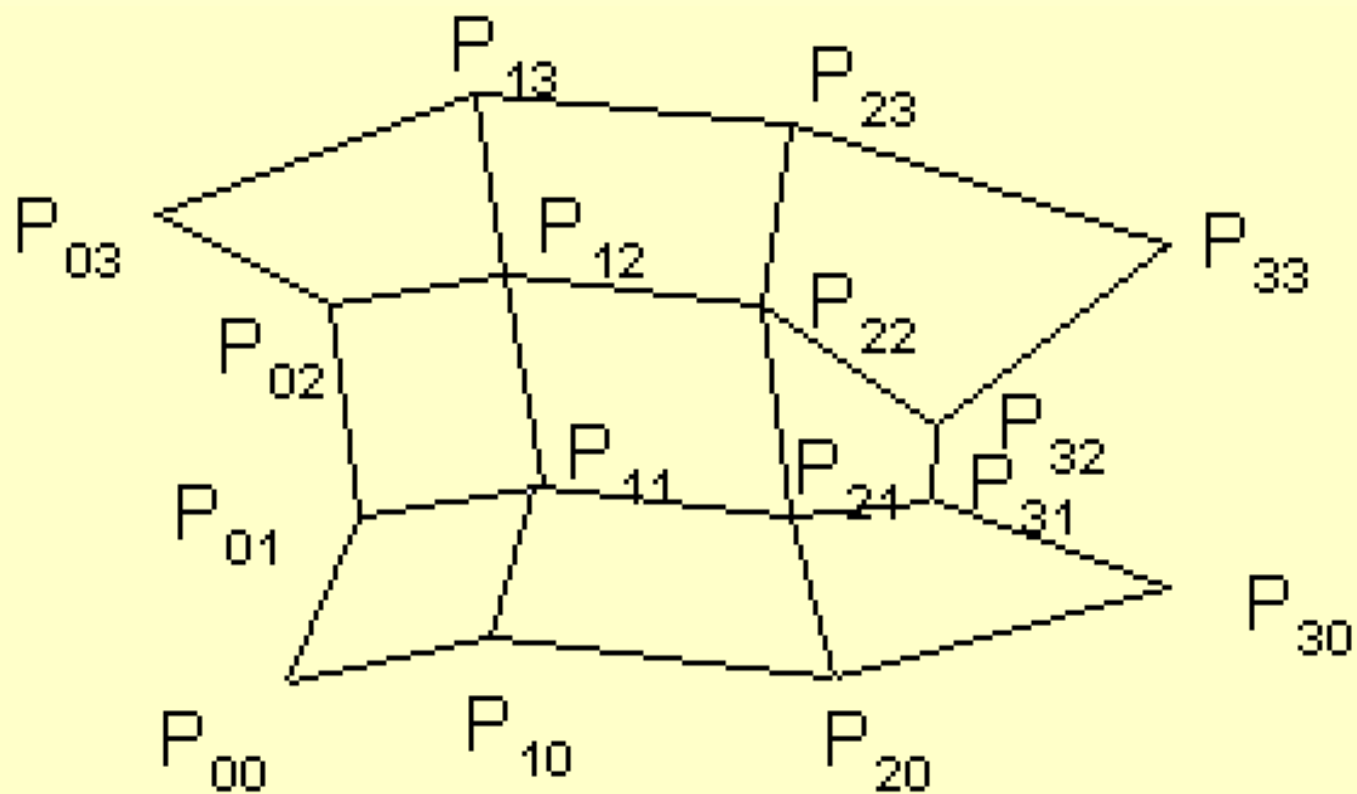


图6.4 Bézier曲面控制网格

二、 $P(u,v)$ 的性质

(1) 端点位置

$$P_{00} = P(0,0), \quad P_{0m} = P(0,1)$$

$$P_{n0} = P(1,0), \quad P_{nm} = P(1,1)$$

(2) 边界线位置

$P(u,v)$ 的4条边界线 $P(0,v)$, $P(u,0)$, $P(1,v)$, $P(u,1)$ 分别是以 $P_{00}P_{01}P_{02}\dots P_{0m}$, $P_{00}P_{10}P_{20}\dots P_{n0}$, $P_{n0}P_{n1}\dots P_{nm}$, $P_{0m}P_{1m}\dots P_{nm}$ 为控制多边形的Bézier曲线, 见图6.5。

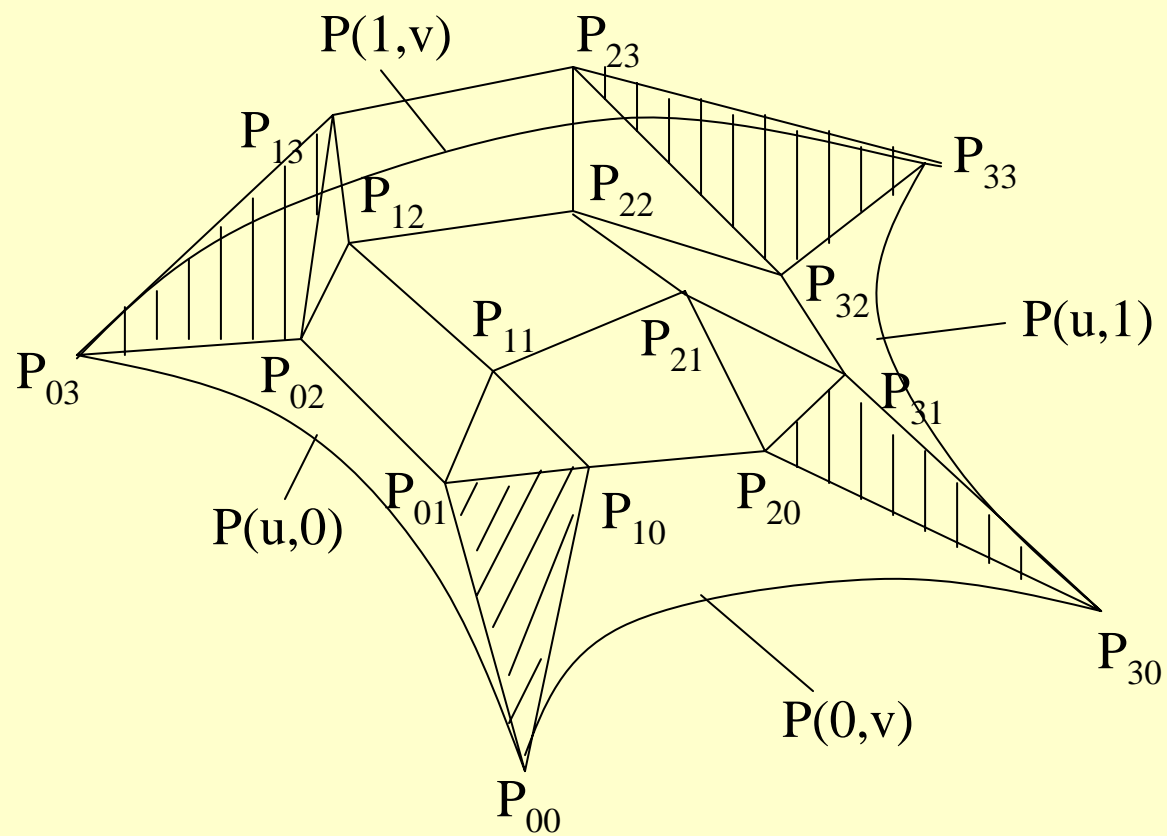


图6.5 Bézier曲面端点的切平面和边界

(3) 端点的切平面

易知端点和其相邻两个控制点构成的三角形(如 $P_{00} P_{10} P_{01}$)与曲面 $P(u, v)$ 相切。

(4) 凸包性

曲面 $P(u, v)$ 位于控制顶点 P_{ij} ($i=0, 1, \dots, n; j=0, 1, \dots, m$)凸包内。

(5) 几何不变性

曲面 $P(u, v)$ 的形状和位置与坐标系的选取无关, 仅和点 $P_{00}, P_{01}, \dots, P_{nm}$ 的位置有关。

(6) 交互能力

$P(u, v)$ 的控制网格 $\{P_{ij}\}$ 可以作为曲面的输入和人机交互的良好手段。

另外, 易拼接性、易离散性都较好。

6.3 B样条曲线曲面

6.3.1 B样条曲线

6.3.2 B样条曲面

6.3.1 B样条曲线

一、B样条基函数

- 给定参数 t 轴上一个分割 $T = \{t_i\}_{-\infty}^{+\infty}$ ($t_i \leq t_{i+1}$, $i=0, \pm 1, \pm 2, \dots$), 由下列关系所定义的 $B_{i,k}(t)$ 称为 T 的 k 阶($k-1$ 次)B样条基函数。

$$B_{i,1}(t) = \begin{cases} 1, & \text{若 } t_i \leq t \leq t_{i+1} \\ 0, & \text{其它} \end{cases}$$

$$B_{i,k}(t) = \frac{(t - t_i)B_{i,k-1}(t)}{t_{i+k-1} - t_i} - \frac{(t_{i+k} - t)B_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}}$$

$-\infty < t < \infty$ (如遇 $0/0$, 取结果为 0)

- 常称T为节点向量, t_i 为节点, 如果 $t_{j-1} < t_j = t_{j+1} = \dots = t_{j+l-1} < t_{j+l}$, 则称上式中除 t_{j-1} 和 t_{j+l} 以外的每一节点为T的 l 重节点。
- 由 $B_{i,k}(t)$ 的定义, 它具有以下性质:

(1)局部性

$$B_{i,k}(t) \begin{cases} > 0, t_i < t < t_i + k & \text{(此区间为正)} \\ = 0, t \leq t_i \text{或} t \geq t_i + k & \text{(其它为0)} \end{cases}$$

(2) 权性

$$\sum_{i=-\infty}^{+\infty} B_{i,k}(t) \equiv 1, \quad -\infty < t < +\infty$$

$$\text{或者 } \sum_{i=j+1-k}^{j+q-1} B_{i,k}(t) \equiv 1, \quad t_j < t < t_{j+q} \quad (q > 0)$$

(3) 分段多项式

$B_{i,k}(t)$ 在每一长度非零的区间 (t_j, t_{j+1}) 上都是次数不高于 $k-1$ 次的多项式。

(4)连续性

$B_{i,k}(t)$ 在 l 重节点处的连续阶不低于 $k-1-l$ (阶连续),
 $B_{i,k}(t)$ 的求导公式如下:

$$B'_{i,k}(t) = (k-1) \left[\frac{B_{i,k-1}(t)}{t_{i+k-1} - t_i} - \frac{B_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}} \right]$$

二、B样条曲线

- 给定空间 n 个点 $P_1 P_2 \dots P_n$ ($n \geq k$), 称下列参数

曲线 $P(t) = \sum_{i=1}^n B_{i,k}(t) P_i, t_k \leq t \leq t_{n+1}$ 为 k 阶

(或 $k-1$ 次)B样条曲线, 折线 $P_1 P_2 \dots P_n$ 为 $P(t)$ 的控制多边形, 称点集 $\{P_i\}$ 为 $P(t)$ 的控制顶点。如图6.6所示。

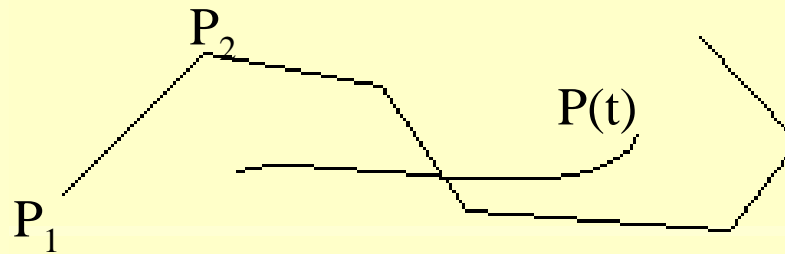


图6.6

- 由 $B_{i,k}(t)$ 的定义, 可知定义的B样条曲线具有以下性质:

(1) 凸包性

B样条曲线在区间 (t_k, t_{i+1}) , $k \leq i \leq n$ 上的部分位于 k 个点 $P_{i-k+1}, P_{i-k+2}, \dots, P_i$ 的凸包内。整条曲线 $P(t)$ 则位于各凸包的并集之内。见图6.7。

(2) 分段参数多项式

$P(t)$ 在每一区间 (t_i, t_{i+1}) , $k \leq i \leq n$ 上都是次数不高于 $k-1$ 的参数 t 的多项式, 即 $P(t)$ 是参数 t 的 $k-1$ 次的分段多项式曲线。

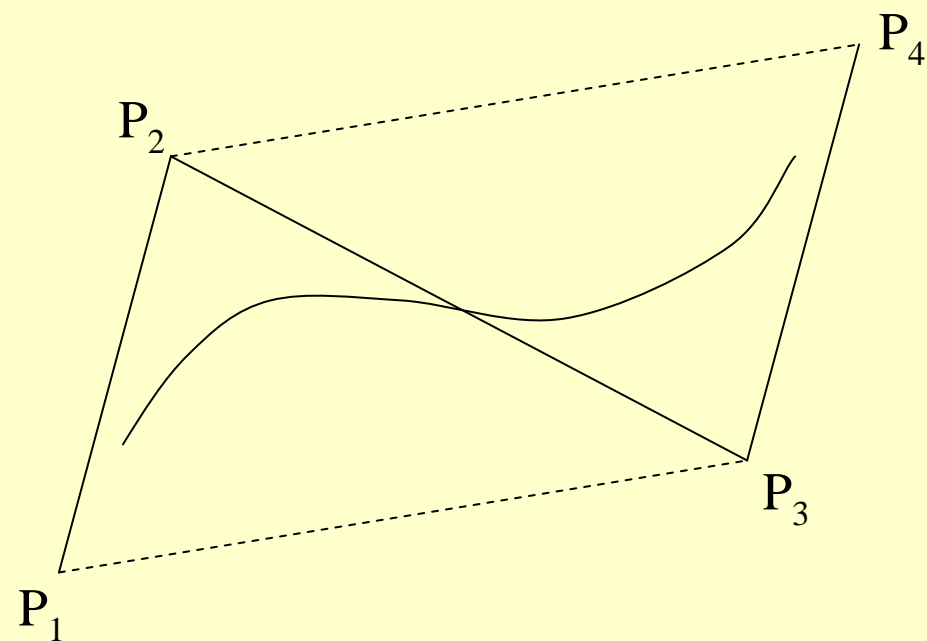


图6.7 B样条曲线在控制多边形凸
包 $P_1 P_2 P_3 P_4$ 内部

(3) 连续性

$B_{i,k}(t)$ 在 l 重节点 $t_i(k+1 \leq i \leq n)$ 处的连续阶不低于 $k-1-l$, 整条曲线 $P(t)$ 的连续阶不低于 $k-1-l_{max}$, 其中 l_{max} 为 (t_k, t_{n+1}) 内节点最大重数。

(4) 几何不变性

$P(t)$ 的形状和位置与坐标系的选择无关。

(5) 保凸性

B样条曲线和Bézier曲线一样,也具有保凸性, 即当 $P_1 P_2 \dots P_n$ 形成一个平面凸的闭多边形时, $P(t)$ 是一平面凸曲线, 参见图6.8。

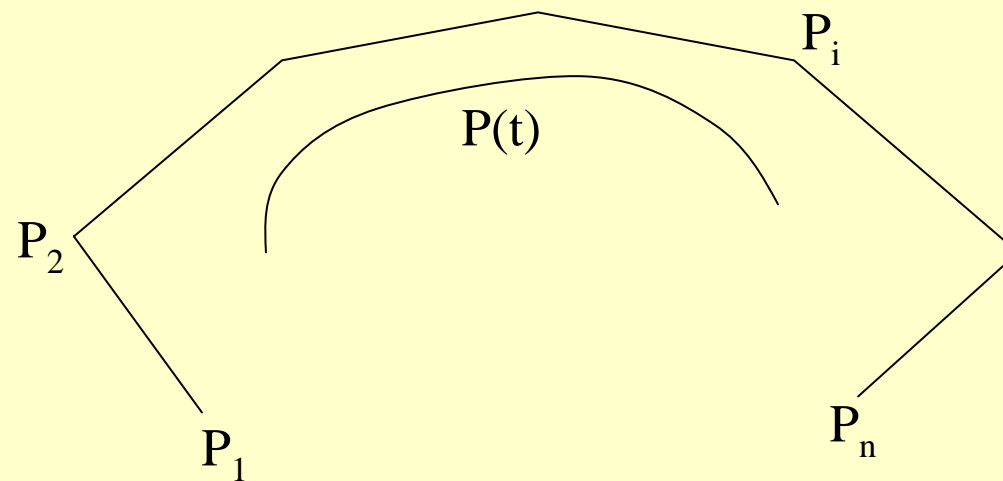


图6.8 B样条曲线保凸性

(6)局部调整性

由于B样条基函数的局部性, 如果只变动某一个控制点 P_i ($1 \leq i \leq n$), 曲线 $P(t)$ 上只有一段发生变化(对应 $\max(t_1, t_i) \leq t \leq \min(t_{n+1}, t_{i+k})$ 的一段)。 $P(t)$ 的其它部分均不变动, 这就为设计曲线时修改某一局部的形状带来很大的方便。

(7)造型灵活性

在设计曲线时, 有时希望在曲线某一点处形成一角点或在某一段形成直线段, 或要求曲线与某一直线相切, B样条曲线提供了这种手段。

可用控制多边形大致勾画B样条曲线形状, 再通过局部修改使曲线设计满足要求。

如为使三次B样条曲线 $P(t)$ 对应 (t_{i+3}, t_{i+4}) 一段为直线段, 只要取 $P_i P_{i+1} P_{i+2} P_{i+3}$ 位于一直线上。

6.3.2 B样条曲面

一、定义

- 设节点向量

$$U = \{u_i\}_{-\infty}^{+\infty}, V = \{v_i\}_{-\infty}^{+\infty} \quad (u_i \leq u_{i+1}, v_i \leq v_{i+1})$$

分别是对参数 uv 平面上的 u 轴和 v 轴的分割,见图6.9。
称下列张量积的形式参数曲面为 $k \times h$ ($k \leq n, h \leq m$)
阶的B样条曲面:

$$P(u, v) = \sum_{i=1}^n \sum_{j=1}^m P_{ij} B_{i,k}(u) B_{j,h}(v)$$

$$(u_k \leq u \leq u_{n+1}, v_h \leq v \leq v_{m+1})$$

其中, P_{ij} 是空间中给定的 $n \times m$ 个点; $B_{i,k}(u)$ 和 $B_{j,h}(v)$ 分别是关于节点向量 u, v 的 k 阶和 h 阶的B样条基函数。

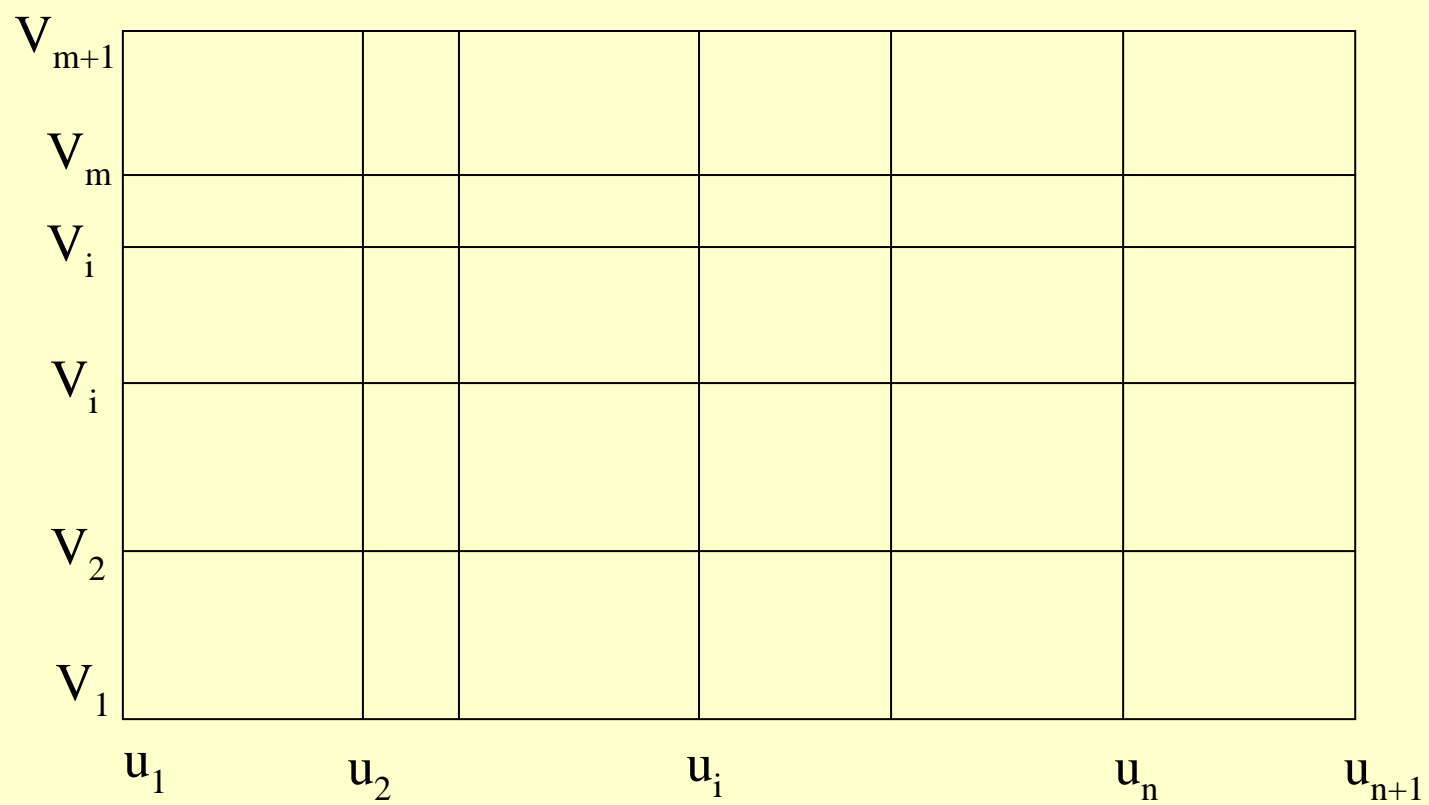


图6.9 uv 平面的分割

- 通常称 P_{ij} 为 $P(u,v)$ 控制顶点, 把两组多边形 $P_{i1} P_{i2} \dots P_{im}$ ($i=0,1,\dots,n$)和 $P_{1j} P_{2j} \dots P_{mj}$ ($j=0,1,\dots,m$)组成的网格称为 $P(u,v)$ 的控制网格 $\{P_{ij}\}$ (见图6.10)。
- 与Bézier曲面类似, $\{P_{ij}\}$ 是 $P(u,v)$ 大致形状的勾画, $P(u,v)$ 是对 $\{P_{ij}\}$ 的逼近。

二、B样条曲面的 性质

它的凸包性, 几何不变性, 局部调整性和人机交互性与B样条曲线情况类似。

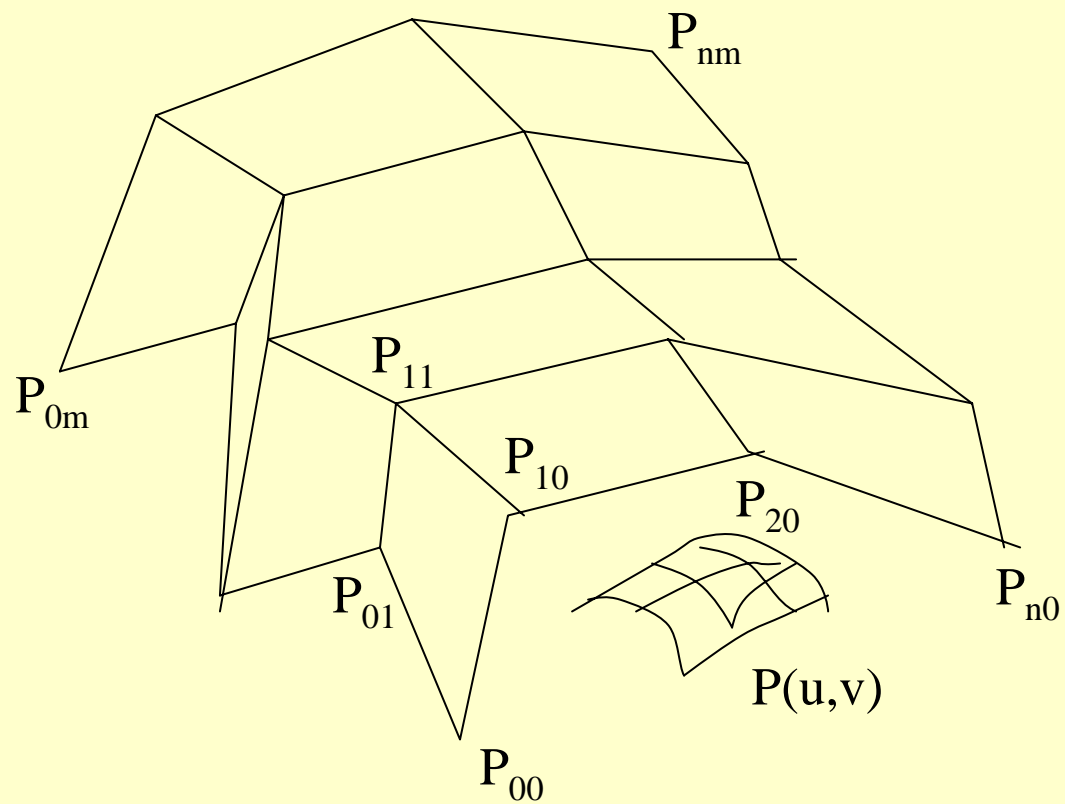


图6.10 B样条曲面及其控制网格

6.4 Coons曲面

一、双三次孔斯曲面(1964)

Bezier曲面和B样条曲面的特点是曲面逼近控制网格; Coons曲面特点是插值, 它构造满足给定边界条件的曲面。现考虑下面插值问题。

- $H(u,v)$ 是在区域 $[0,1; 0,1]$ 上给定的曲面, 现要求作一曲面 $P(u,v)$ 使得

$$P(u, v) = H(u, v), \quad u = 0, 1, v = 0, 1$$

$$P'_u(u, v) = H'_u(u, v), \quad u = 0, 1, v = 0, 1$$

$$P'_v(u, v) = H'_v(u, v), \quad u = 0, 1, v = 0, 1 \quad (6.1)$$

$$P''_{uv}(u, v) = H''_{uv}(u, v), \quad u = 0, 1, v = 0, 1$$

这要求 $P(u, v)$ 的端点、端点导矢、端点扭矢都要与 $H(u, v)$ 相同。

- 解决方法之一是作双三次多项式的插值。取埃尔米特基函数 F_0, F_1, G_0, G_1 作为混合函数。它们分别为：

$$F_0(t) = 2t^3 - 3t^2 + 1$$

$$F_1(t) = -2t^3 + 3t^2$$

$$G_0(t) = t^3 - 2t^2 + t$$

$$G_1(t) = t^3 - t^2$$

现规定行向量 $[F_0 F_1 G_0 G_1] = [F_0(u) F_1(u) G_0(u) G_1(u)]$

$$\text{列向量} \begin{bmatrix} F_0 \\ F_1 \\ G_0 \\ G_1 \end{bmatrix} = \begin{bmatrix} F_0(v) \\ F_1(v) \\ G_0(v) \\ G_1(v) \end{bmatrix}$$

则下式定义的双三次曲面 $P(u, v)$ 满足上述的四方面插值条件, 称为双三次孔斯曲面。

$$P(u, v) = [F_0 F_1 G_0 G_1] \begin{bmatrix} H(0,0) & H(0,1) & H'_v(0,0) & H'_v(1,0) \\ H(1,0) & H(1,1) & H'_v(1,0) & H'_v(1,1) \\ H'_u(0,0) & H'_u(0,1) & H''_{uv}(0,0) & H''_{uv}(0,1) \\ H'_u(1,0) & H'_u(1,1) & H''_{uv}(1,0) & H''_{uv}(1,1) \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ G_0 \\ G_1 \end{bmatrix} \quad (6.2)$$

二、孔斯曲面的性质

(1) 端点位置

$H(0,0)$, $H(0,1)$, $H(1,0)$, $H(1,1)$ 是曲面 $P(u,v)$ 的端点。

(2) 边界线

在 $v=0$ 处曲面的边界线为

$$P(u,0) = H(0,0)F_0(u) + H(1,0)F_1(u) + H'_u(0,0)G_0(u) \\ + H'_u(1,0)G_1(u), 0 \leq u \leq 1$$

这是以式(6.2)中的矩阵的第一列元素为系数的三次埃尔米特曲线。其余的边界线 $P(u,1)$, $P(0,v)$, $P(1,v)$ 分别是以该矩阵中的第2列, 第1行和第2行的元素为系数的三次埃尔米特曲线。

(3) 跨界导矢

由式(6.1)可得 $v=0$ 处的跨界导矢为

$$P'_v(u,0) = H'_v(0,0)F_0(u) + H'_v(1,0)F_1(u) + H''_{uv}(0,0)G_0(u) \\ + H''_{uv}(1,0)G_1(u), 0 \leq u \leq 1$$

这是以式(6.2)中的矩阵的第3列元素为系数的三次埃尔米特曲线。

其余各条边界的跨界导矢 $P'_v(u,1), P'_u(0,v), P'_u(1,v)$ 分别是以该矩阵中的第4列, 第3行和第4行的元素为系数的三次埃尔米特曲线。

另外,双三次孔斯曲面与双三次Bezier曲面的可相互转化。

第7章 三维实体的造型

知 识 点：边界表示法、体素构造表示法
及其它常用造型方法简介

教学目的：了解三维立体造型的方法

本章内容

7.1 概述

7.2 边界表示法

7.3 体素构造表示法

7.4 八叉树表示

7.5 其它常用造型方法简介

7.1 概述

- 三维几何造型
 - 曲面造型: 如何在计算机内描述物体的表面, 对它的外部形状显示与控制。
常用方法:
 - 规则曲面—代数曲面与参数曲面
 - 拟合曲面—Bezier曲面、B样条曲面、Coons曲面
 - 立(实)体造型: 如何在计算机内定义、表示一个三维物体。计算和分析物体的整体性质。

- 常用的立体造型系统, 一般采用下述几种物体表示方法
 - 基本体例表示法(Pure primitive instancing)
 - 空间位置枚举法(Spatial occupancy enumeration)
 - 单元分解法(Cell decomposition)
 - 推移表示法(Sweep representation)
 - 体素构造表示法(Constructive solid geometry)
 - 边界表示法(Boundary representation scheme)

- 不论选择采用哪一种表示法,必须考虑两点:
 - 表示法的覆盖域,即用这种表示法所能定义的物体范围的大小。
 - 表示法蕴涵信息的完整性,即用这种表示法所决定的数据结构是否唯一地描述了现实生活中的一个三维立体。

7.2 边界表示法

- 用顶点、棱边、表面等物体的边界信息来表示物体。边界就是物体内部点与外部点的分界面。举例见图7.1。
 - 边界表示法特点
 - 包含两方面信息：
 - 几何信息 – 物体大小、尺寸、位置、形状等
 - 拓扑信息 – 物体上所有顶点、棱边、表面间连接关系
- 几何信息和拓扑信息分开表示。

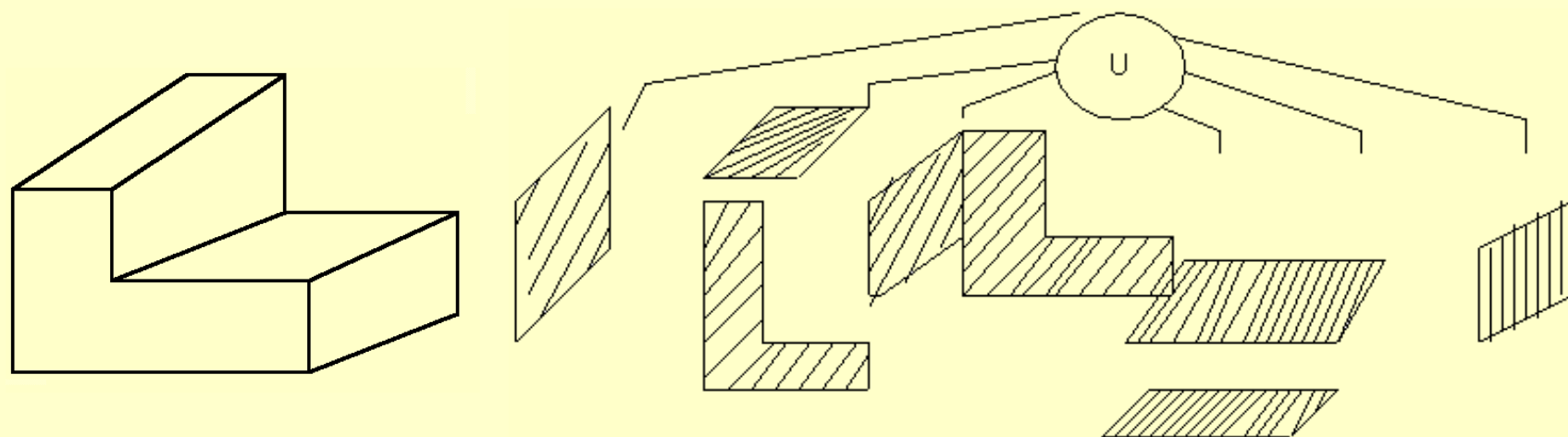


图7.1 边界表示法举例

- 优点:

- (1) 便于具体查询物体中各元素、获取它们的信息;
- (2) 容易支持对物体的各种局部操作;
- (3) 对于具有相同拓扑结构而只是大小、尺寸不同的物体,可以用统一的数据结构加以表示;
- (4) 便于在数据结构上附加各种非几何信息。

- 翼边结构

一种典型的边界表示数据结构

- 在顶点、棱边、表面等组成物体的三要素中, 翼边结构以边为中心来组织数据, 如图7.2所示。

- 棱边e的数据结构中包含:

2个点指针, 分别指向e的两个端点--顶点 P_1 和 P_2 ;

2个环指针, 分别指向e所邻接的两表面上的环--
Loop左, Loop右;

4个边指针(相邻棱边):

E_{rcc} — e在右环中沿逆时针所连的下一条棱边;

E_{rcw} — e在右环中沿顺时针所连的下一条棱边;

E_{lcw} — e在左环中沿顺时针所连的下一条棱边;

E_{lcc} — e在左环中沿逆时针所连的下一条棱边; ⁹

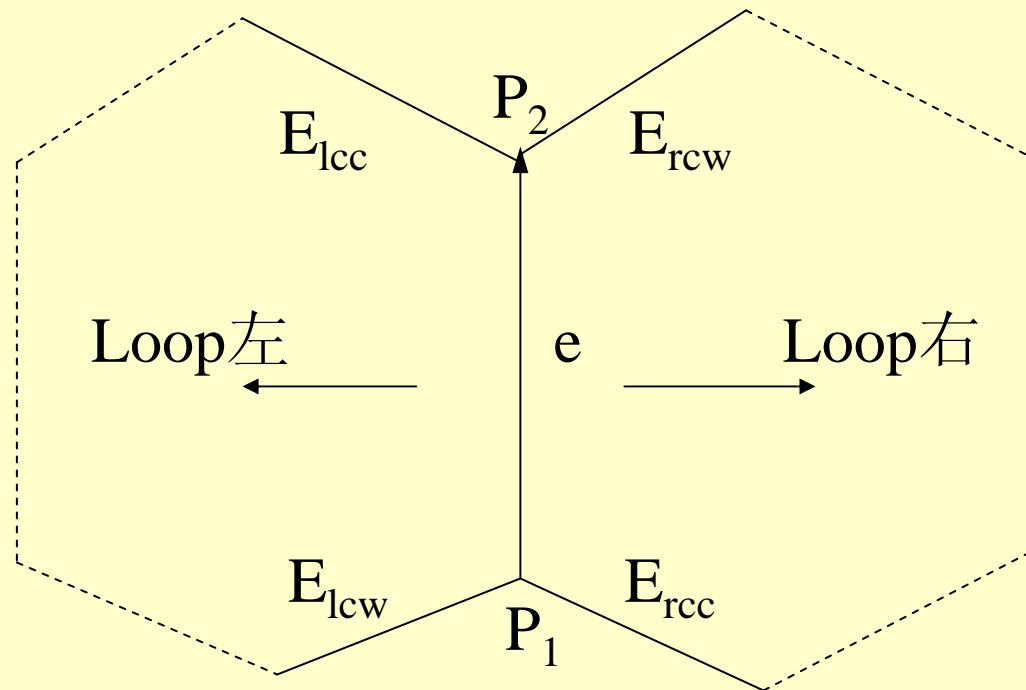


图7.2 翼边结构

p1	loop左	Ercc,Erclw
p2	loop右	Elcc,Elclw

图7.3 翼边结构中边的数据结构

- 多面体拓扑关系的表示形式
 - 多面体的顶点、棱边、表面之间的拓扑关系可以用9种形式表示。如图7.4。
 - 实际上每种表示可由其它任何一种关系经过适当运算导出, 所以在具体系统中往往根据情况由一种或几种关系的组合来表示物体。

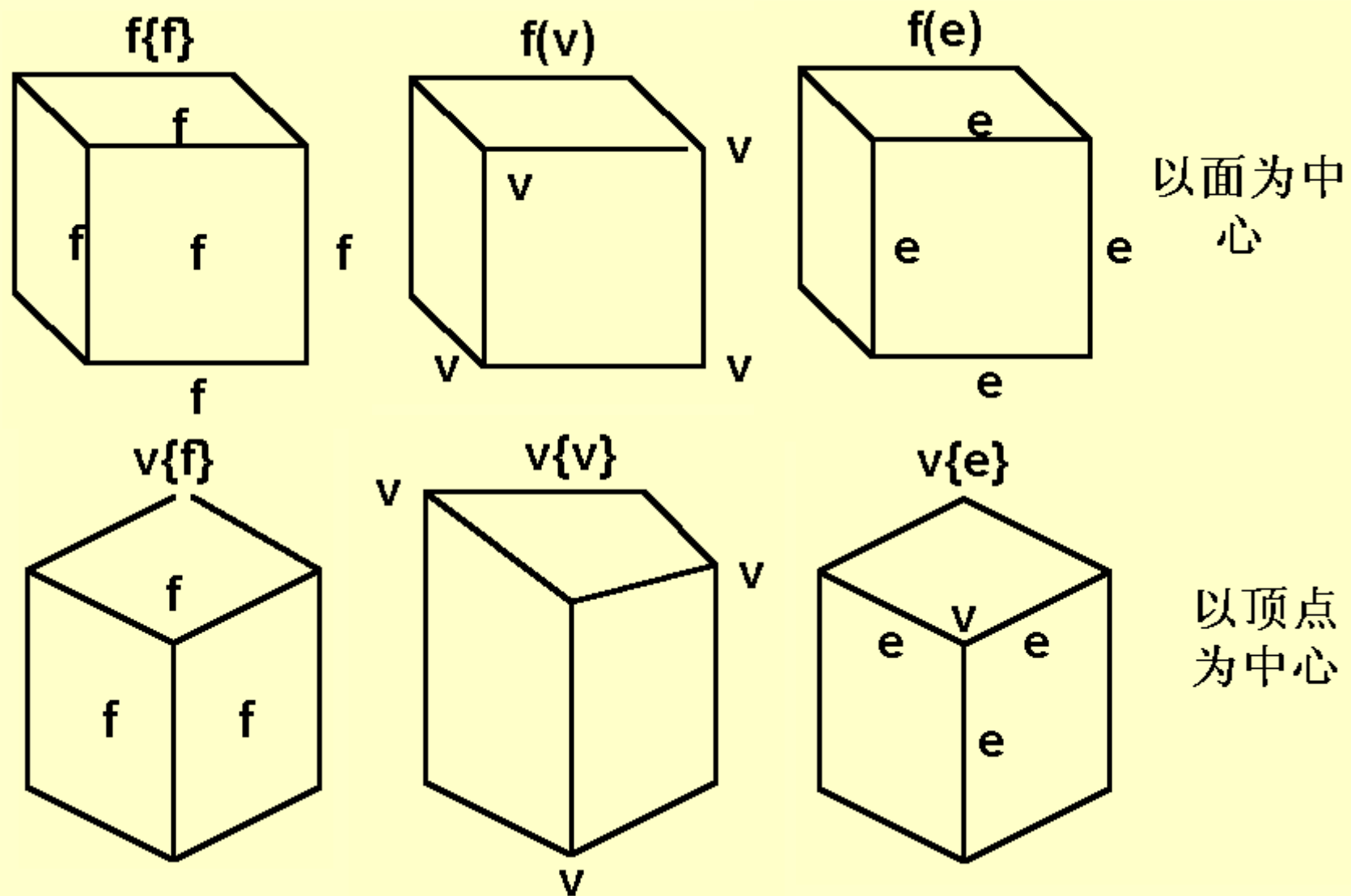


图7.4 拓扑关系的表示形式(1)

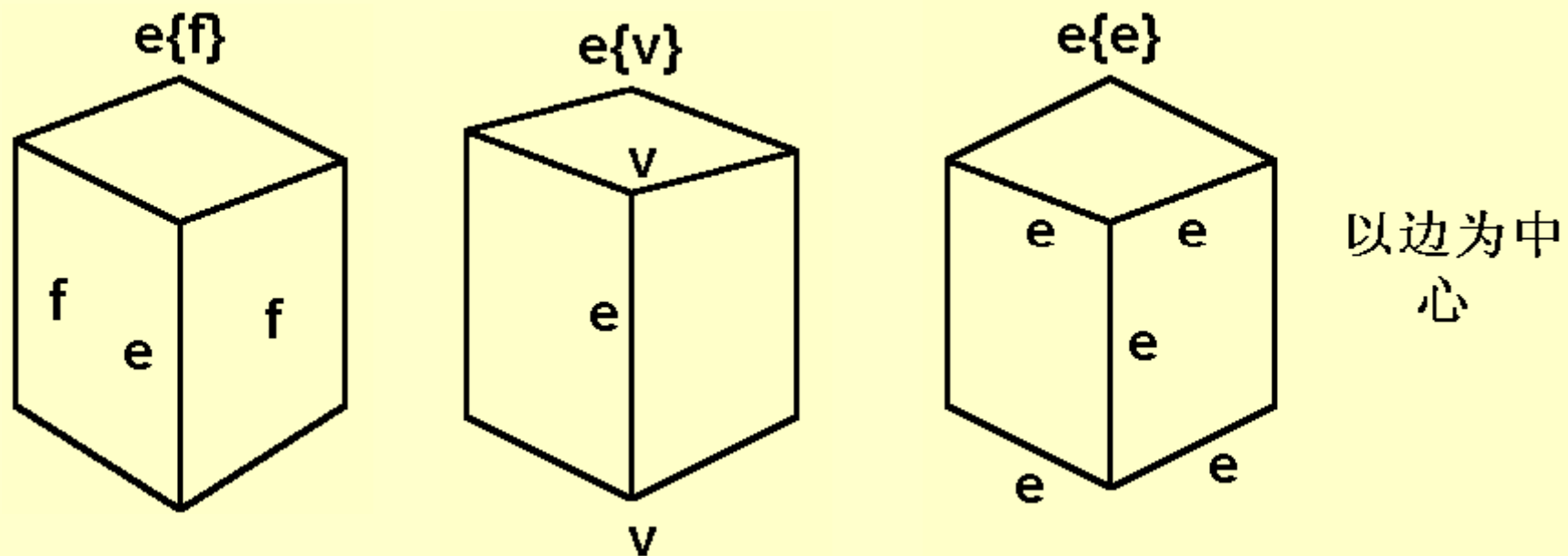


图7.4 拓扑关系的表示形式(2)

- 欧拉运算

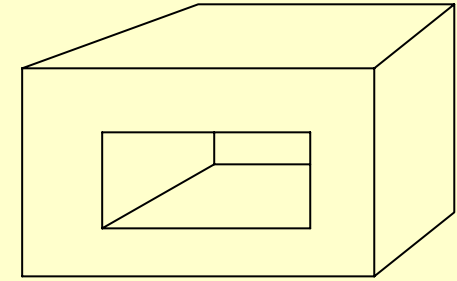
- 给用户提供了直接使用顶点、棱边、表面等基本元素构造三维立体的手段。用户构型通过输入点,再建立边,构成面,形成体。
- 任何数目顶点、棱边、表面并不能构成一个体。它们之间须满足拓扑一致性和几何一致性。
- 几何一致性由用户输入几何信息时保证,系统应提供检查输入信息几何一致性功能。

- 拓扑一致性用欧拉公式表示:

$$v - e + f - r = 2(s - h)$$

v, e, f 表示物体顶点,棱边,表面数; r, s, h 分别表示物体表面边界的内环数,不相连的物体个数以及物体上通孔数目。

例图: $v=16, e=24, f=10, r=2, s=1, h=1$



- 如将 v, e, f, r, h, s 分别看作独立的变量, 则上式定义了高维空间中的一张平面, 该平面通过原点。由于上述变量均是大于或等于0的整数, 故任一实际的三维物体只对应此高维平面上的网格点。
- 造型的过程就是从原点(null object)开始一步一步地走到给定的位置, 构造所需的立体。只需选定5种位移作为基本走法, 就可以多达高维平面上的任一网格点。

- 设计5种位移(走法) 应考虑
 - (1) 每一步只在所涉及的坐标轴上移动单位距离;
 - (2) 每种走法有明显的几何意义(有效).

7.3 体素构造表示法

7.3.1 物体定义

- 我们将物体定义为三维欧氏空间 E^3 中点的正则集合 $rX=kiX$ 。式中 X 表示 E^3 中一个点集, r 表示正则化算子, k, i 分别表示点集的闭包和内部。 bX 是 X 的边界, cX 是 X 的外部。
- 正则化的含义是取点集内部并用一张边界把它包裹起来。
- 由以上定义可得下述推论:
 - (1) 一个物体只能占据有限空间。它是封闭的, 具有一定的体积。
 - (2) 物体中不允许存在孤立点、悬挂的线或面。见图7.5。

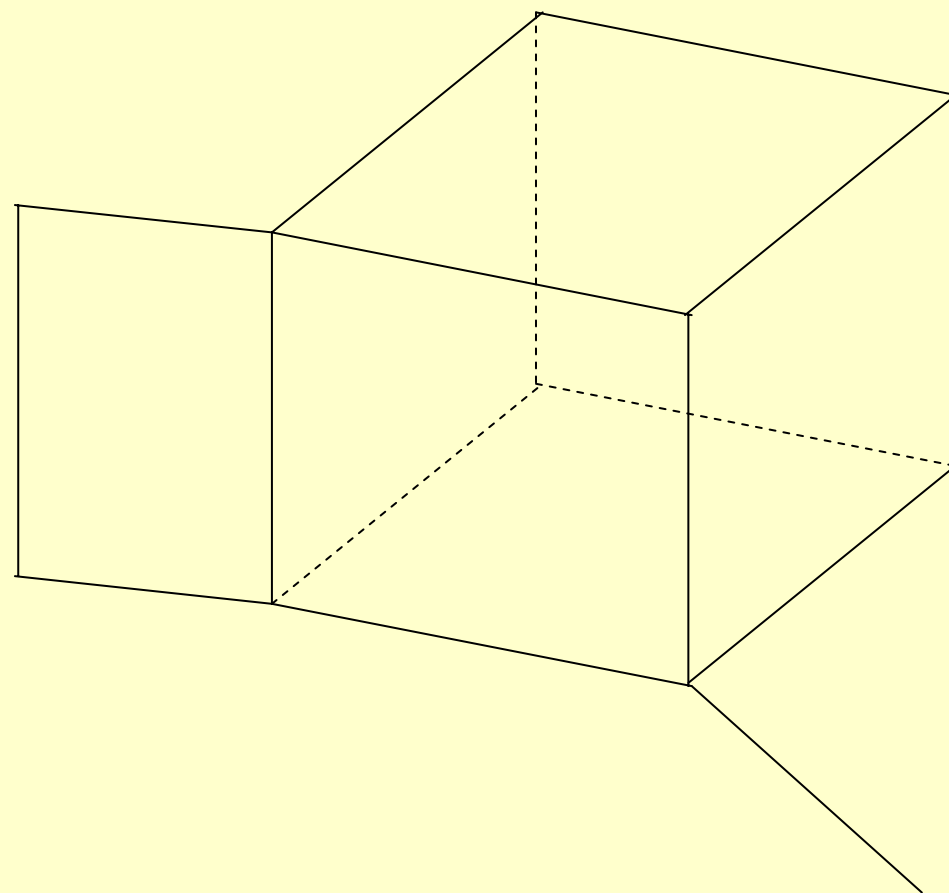


图7.5 非有效体

7.3.2 物体间的正则集合运算

- 传统的点集之间的并、交、差运算可能改变点集的正则性质, 有必要对传统的点的集合运算施加一定的限制。

- 为此, 对点集的正则集合运算作下述定义:

正则并运算 $A \cup^* B = r(A \cup B)$

正则交运算 $A \cap^* B = r(A \cap B)$

正则差运算 $A -^* B = r(A - B)$

其中, r 表示正则化算子。正则物体经正则运算结果仍为正则物体。

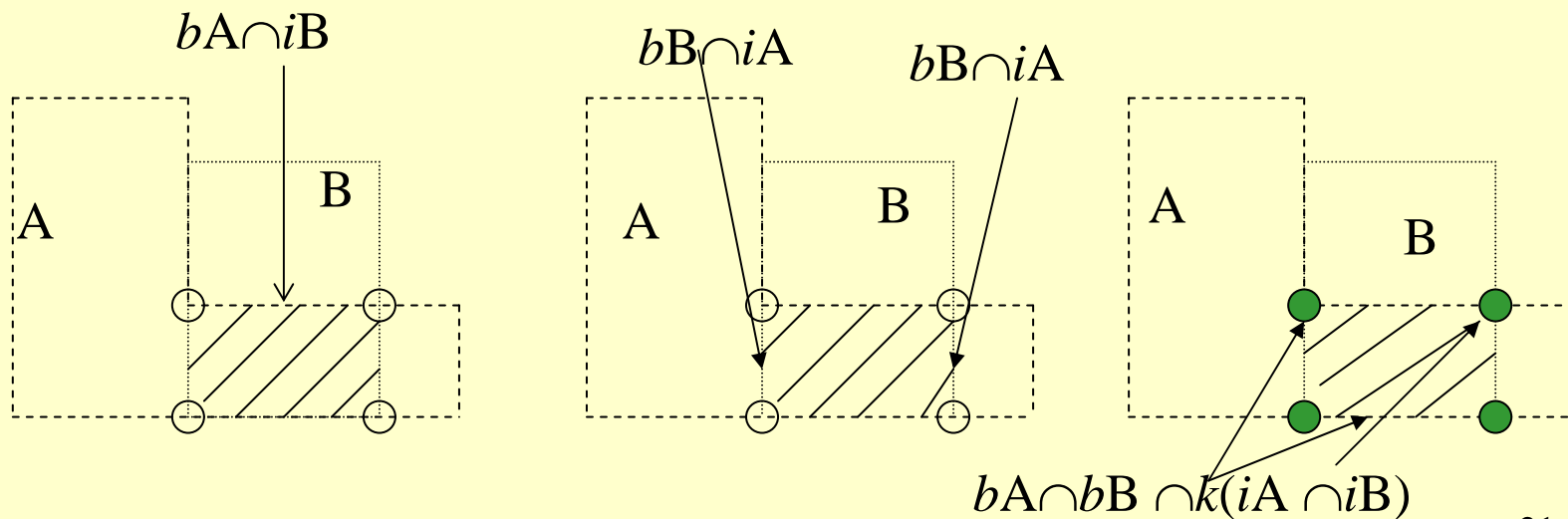
- 正则集合运算后产生的新物体的边界是原两拼合物体边界的一个子集, 即:

$$b(A \langle op \rangle B) \subset (bA \cup bB)$$

- A, B为两个物体, 由于A, B为正则点集, 所以

$$A = bA \cup iA$$

$$B = bB \cup iB$$



边界公式:

$$b(A \cap^* B) = (bA \cap iB) \cup (bB \cap iA) \cup (bA \cap bB \cap k(iA \cap iB))$$

$$b(A \cup^* B) = (bA \cap cB) \cup (bB \cap cA) \cup (bA \cap bB \cap k(cA \cap cB))$$

$$b(A -^* B) = (bA \cap cB) \cup (bB \cap iA) \cup (bA \cap bB \cap k(iA \cap cB))$$

7.3.3 物体的CSG树表示

- 一个复杂物体可由一些比较简单、规则的物体经过布尔运算而得到(如图7.6)。这种结构可描述为一棵树, 即CSG(Constructive Solid Geometry)树.
- 其中叶结点为基本体素(立方体, 圆柱, 圆锥等); 中间结点为正则集合运算结点

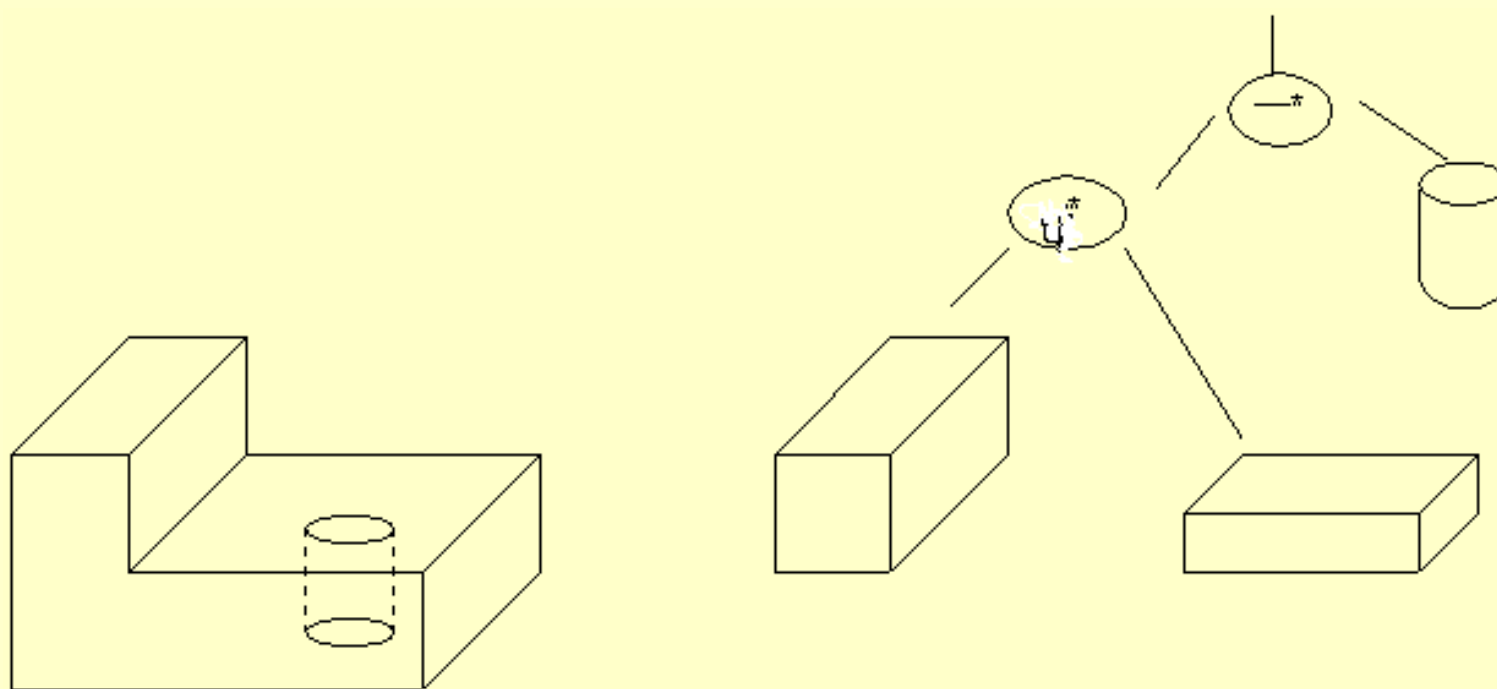


图7.6 CSG树表示

—数据结构:

图7.7给出CSG树结点数据结构的一种组织方式。每一个结点由操作码、坐标变换域、基本体素指针、左子树、右子树等5个域组成。除操作码外,其余域均以指针形式存贮。操作码按约定方式取值。其中每一复杂结点的坐标变换域存贮该结点所表示物体在进行新的集合运算前所作坐标变换的信息。

OP-Code (操作码)		OP-Code
transform (坐标变换)	primitive (基本体素)	=0 基本体素
left—subtree (左子树)	right—subtree (右子树)	=1 求并
		=2 求差
		=3 求交

图7.7 CSG树结点的数据结构

- CSG树只定义了它所表示物体的构造方式。它既不存贮顶点、棱边、表面等物体的有关边界信息,也未显式定义三维点集与所表示物体在空间的一一对应关系。
- CSG树表示一个复杂形体常比较简洁。它所表示的物体的有效性是由基本体素的有效性和集合运算的正则性而自动得到保证的。
- **优点:** 将复杂物体表示转换为简单物体之间运算,也可递归求出物体性质;
- **缺点:** 方法有局限性,物体复杂时,这种表示不太适应。

- CSG树表示最适宜采用“分治”算法(divide-and-conquer): 复杂问题分解为若干小问题分别求解; 然后再将求得的小问题的解合成整个问题的解。这是一个递归模式。
- 设F是关于一个CSG树所定义的几何性质(重心、点与物体的包含关系)的函数。依上述“分治”方法, 计算F的算法:

$$F(\text{tree}, \text{data}) = \text{if } (\text{tree-is-a-leaf}) \text{ then } \text{PRIM-F}(\text{tree}, \text{data})$$
$$\text{else } \text{COMBINE}(F(\text{left-subtree}, \text{data}), F(\text{right-subtree}, \text{data}), \text{node}(\text{tree}))$$

*PRIM-F*为计算基本体素的几何性质*F*的函数,
*COMBINE*为合成函数

7.4 八叉树表示

1. 空间位置枚举法

- 使用该方法时, 先将空间分割成均匀的立方体网格, 然后根据物体所占据的网格位置来定义物体的形状和大小。
- 它所使用数据结构为三维数组, 如图7.9. 这样每个数组元素对应一空间位置:

$$s(i,j,k)=\begin{cases} 1 & \text{位置}(i,j,k) \text{ 被物体所占据} \\ 0 & \text{否则} \end{cases}$$

- 数组的大小取决于空间分辨率。一般来说, 所描述物体的形状越复杂, 细节越丰富, 精度要求愈高, 则选取的空间分辨率也较高。
- 优点-- 适合所有形状三维物体表示, 容易实现物体的并交差及整体性质计算(对应着数组运算)
- 缺点-- 没有明确给定物体边界信息, 占据存储量大.
- 一种有效的表示法—八叉树

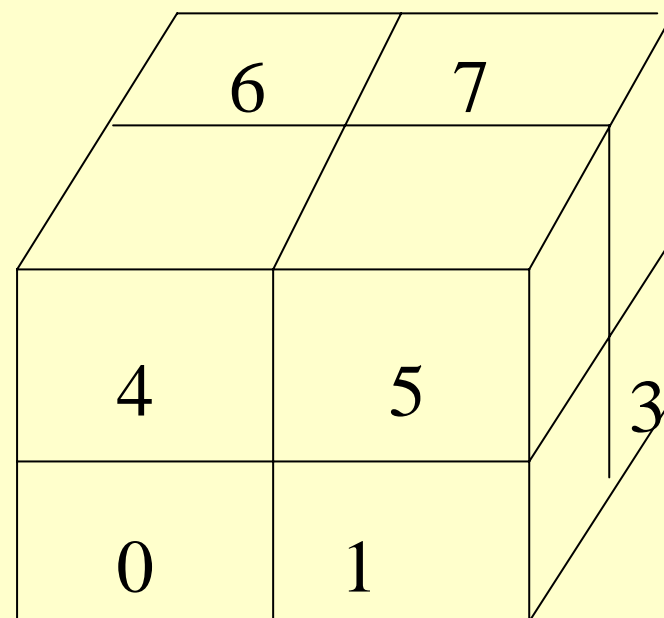
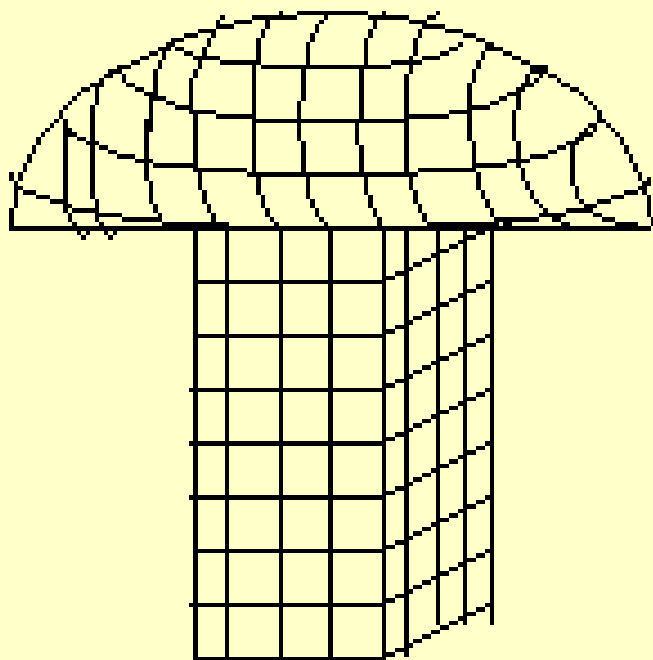


图7.9 空间位置枚举法表示 八叉树

2. 八叉树表示方法

- (1) 定义一个能够包含所表示物体的立方体, 3条棱边长为 2^n .
- (2) 若立方体空间能被物体完全占据, 则物体可用这个立方体予以表示。否则, 将立方体等分为8个小块, 边长为原来的1/2, 编号为0, 1, ..., 7.
- (3) 若某一小立方体内空间全部被物体占据, 此立方体标识为“FULL”; 若某一小立方体内空间与物体无交, 此立方体标识为“EMPTY”;
否则标识为“PARTIAL”。
- (4) 继续对非终结点分割, 直到一小立方体边长为单位长时分割即告终止。此时, 应将标识为“PARTIAL”的小立方体重标为“FULL”。 (如图7.9)

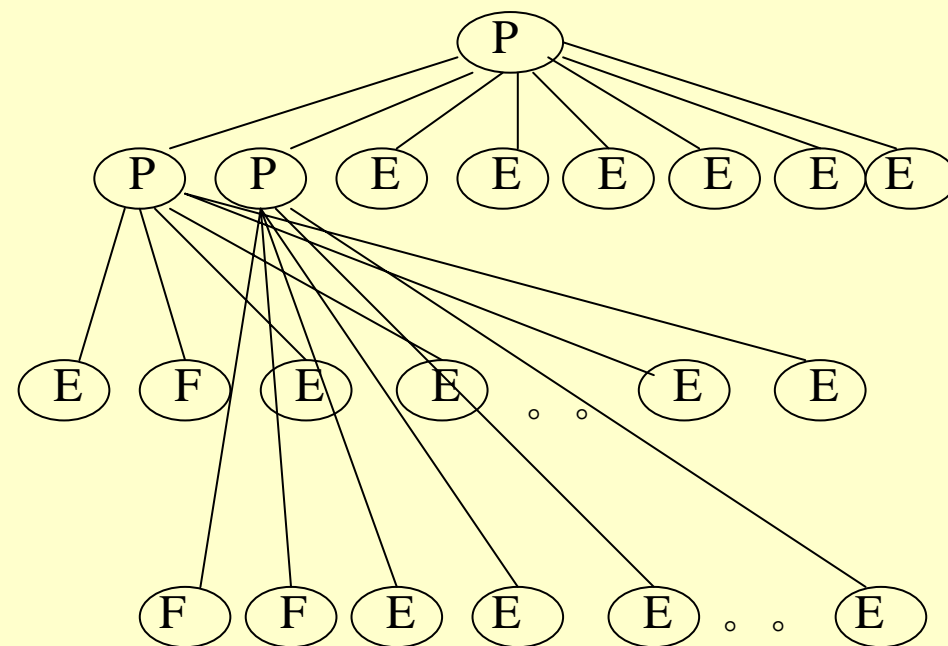
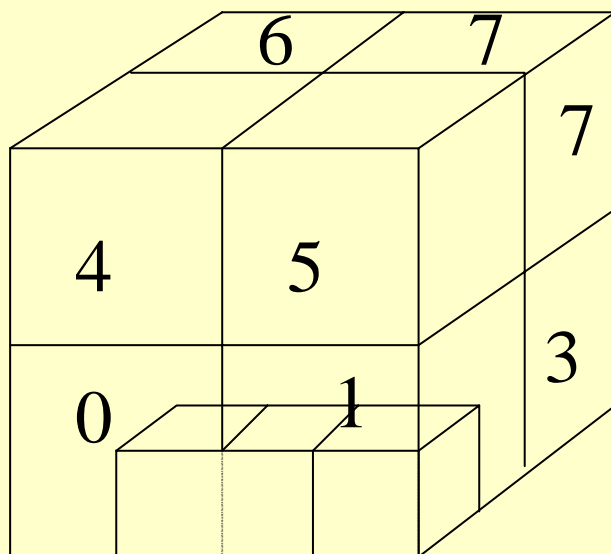


图7.10 八叉树举例

3.八叉树优点:

- 物体之间集合运算在八叉树中十分简单
物体并—两物体一共占有的空间;
物体交—两物体共同占有的空间。
运算时只需同时遍历参加集合运算两物体相应的八叉树。
- 简化了隐藏线和隐藏面的消除
核心是排序(按离观察点远近排序);
八叉树中物体元素已按空间位置排成一定顺序,
同一层的八叉树结点通过优先级表示。
- 计算物体的性质(体积、质量)更简单
对物体的各项操作=» 体元的操作。

4. 八叉树缺点

- 占用存贮很多
 - (1) 物体表示复杂;
 - (2) 每个结点除去描述该结点性质外, 还需存储指向父节点及8个子树的指针。(10个域)
- 运算时只需同时遍历参加集合运算两物体相应的八叉树。
- 八叉树表示以存储空间换取了算法的效率。

5. 线性八叉树

为减少八叉树表示所需空间的存储量, 用一可变长度的一维数组存储八叉树(仅存终结点)。每个节点在八叉树中位置可用一个八进制数表示。(化为八进制运算)

$$Q = q_{m-1} 8^{m-1} + q_{m-2} 8^{m-2} + \dots + q_0 8^0$$

q_0 是该节点在兄弟间序号;

q_1 是其父节点在同辈间序号;

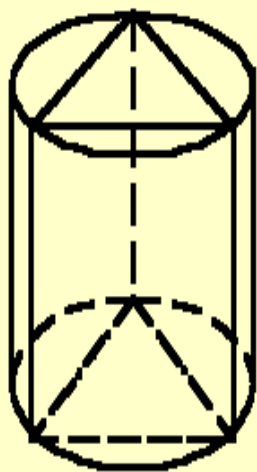
○ ○ ○ ○ ○

7.5 其它常用的立体造型方法

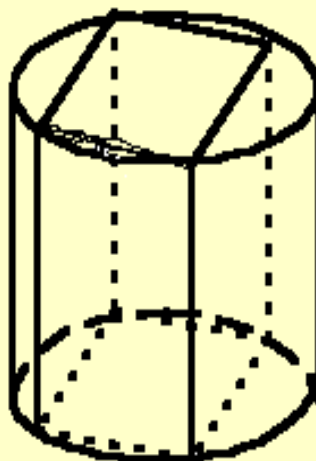
1. 基本体例表示

用一组参数来表示一族形状类似, 但大小不同物体。

- 例: 一个正 n 棱柱, 可用参数组 (n, R, H) 定义, 其中 R, H 分别为圆柱体的半径和高。(如图7.8)。
- 优点--适宜用来表示工业上已定型的标准件.
缺点--有很大的局限性。



$n=3$
 $R=1$
 $H=2$



$n=4$
 $R=2$
 $H=2$

图7.8 基本体例表示示例

2.单元分解法

- 该方法克服空间枚举法缺点,允许将物体表示成一些形状相同但大小不同的基本体积单元的组合。(单元不一定是立方体)
- 优缺点
该方法节省存储量,但导致了复杂的数据结构。同一个物体由于它的空间方位不同,数据结构也不同。(如图7.10)

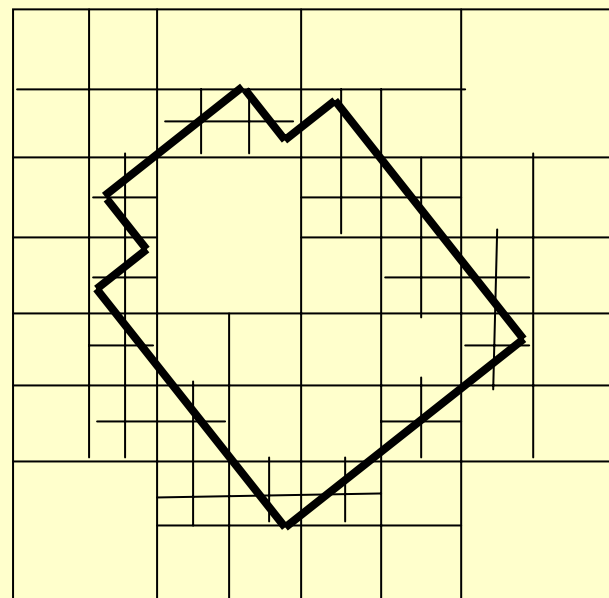
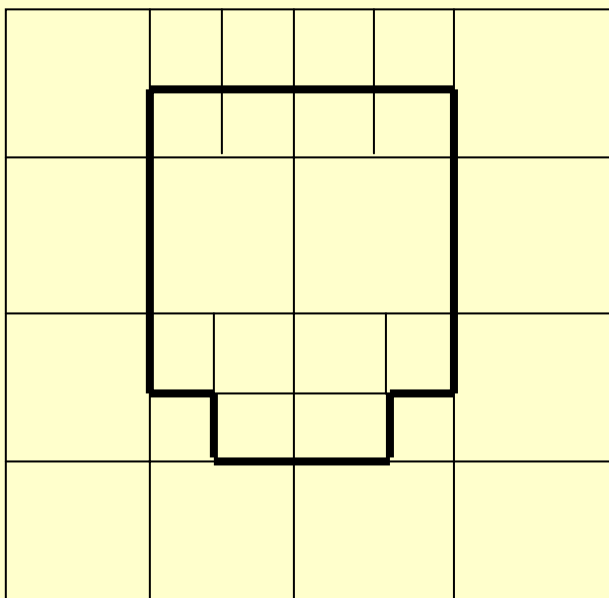
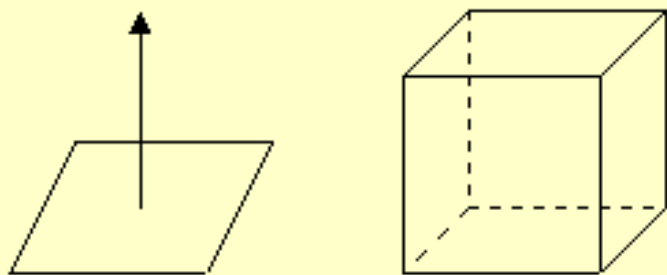


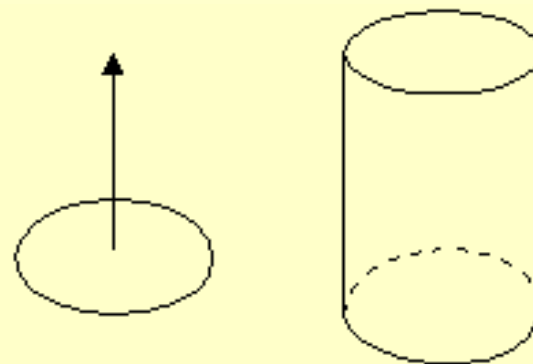
图7.10 单元分解法

3. 推移(SWEEP)法

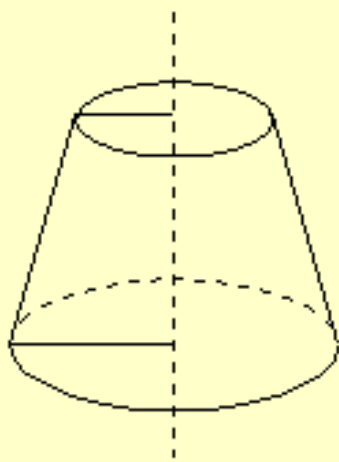
- 又称扫移法. 根据2D或3D物体沿某一曲线(一般直线或圆弧)推移时的外轮廓的轨迹来定义一物体的方法称为推移法. (如图7.11)
- 优点: 特别适合生成工业上常见柱面体、旋转体, 在主体造型系统中常用作简单的造型输入手段.
- 缺点: 表示物体有局限性.



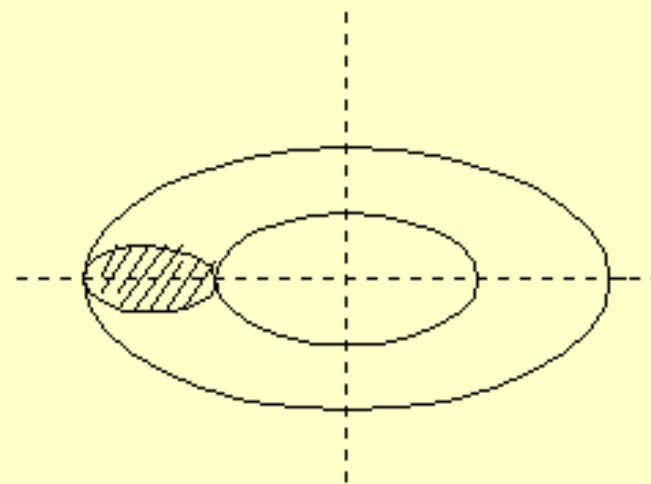
(a) 平移式扫移形成长方体



(b) 平移式扫移形成圆柱



(c) 旋转式扫描形成圆台



(d) 旋转式扫描形成环

图7.11 推移法

第8章 消隐技术

知 识 点：消隐的概念、区域子分算法、深度缓存算法、扫描线算法等常用消隐方法

教学目的：了解图形消隐的基本方法

本章内容

8.1 基本概念

8.2 画家算法

8.3 区域子分算法

8.4 深度缓存算法

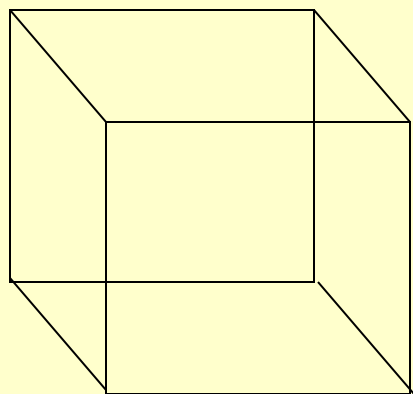
8.5 扫描线算法

8.6 区间扫描线算法

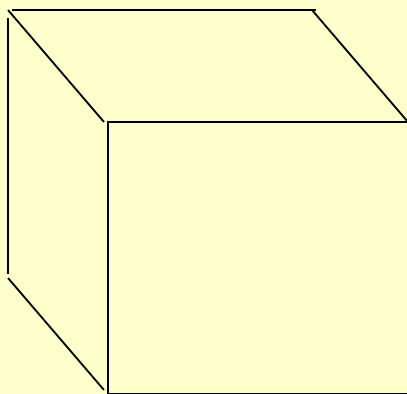
8.7 光线投射算法

8.1 基本概念

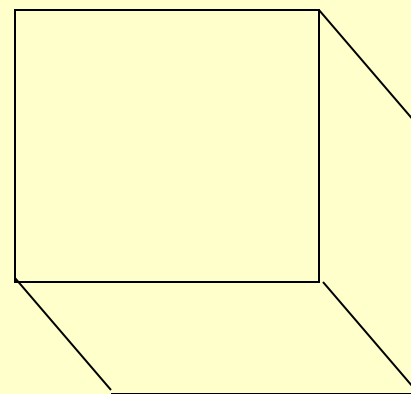
- 在用户坐标系下定义的物体, 经过空间变换、裁剪, 可以投影到观察平面, 再变换或映射到视图区并显示出来, 这时的图形是物体的线框图。
- 图8.1(a)的图形是一个立方体的线框图。但很难确定, 图8.1(a)所示的立方体是代表(b)中的立方体, 还是代表(c)中的立方体。
- 找出并消除物体中不可见的部分, 称为消隐。经过消隐得到的图形称为消隐图。图8.1(b)(c)都是立方体的消隐图。



(a)



(b)



(c)

图8.1 一个立方体的线框图和消隐图

- 消隐的对象是三维物体,三维物体在计算机中可用其表面上的平面多边形表示,形成三维物体的体面边点四表结构。
- 根据消隐对象的不同,消隐算法分为两类:
若消除的是物体上不可见的线段,称为线消隐。
若消除的是物体上不可见的面,称为面消隐。
- 消隐不仅与消隐对象有关,还与观察物体的方式,即投影方式有关. 随着观察点、观察方向或投影面的改变,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。

- 消隐是在规格化投影空间或屏幕坐标空间中进行的(一般投影平面指OXY, 视点在Z轴的正无穷大处, 视线方向为负Z轴)。
- 根据消隐空间的不同,消隐算法又可分为3类:
 - 物体空间的消隐算法(光线投射)

物体空间是指物体所在的空间, 即规范化投影空间。(逐面比较, 决定每个面棱之间的遮挡关系)
 - 图象空间的消隐算法(区域子分、深度缓存、扫描线)

图象空间是相对于物体空间而言的, 在消隐算法中, 图象空间就是屏幕坐标空间。(逐像素判断, 决定哪个多边形在该像素可见)
 - 物体空间和图象空间的消隐算法(画家算法)

在物体空间中预先计算面的可见性优先级, 再在图象空间中生成消隐图。

- 消隐算法种类繁多,但必然都涉及排序和相关性两个基本原则:
 - 排序是为了确定消隐对象之间的遮挡关系。大多数消隐算法都是先在Z方向排序,确定体、面、边、点相对于观察点的距离,然后再在X方向和Y方向进一步排序,以确定这种遮挡关系。
 - 所谓相关性,是指所考察的物体或视图区内的图象局部保持不变的一种性质。相关性利用得越充分、越巧妙,消隐算法的效率也就越高。

8.2 画家算法

算法思想：

类似油画作家作画的过程，先画远景，再画中景，最后画近景。

算法原理：

- 先把屏幕置成背景色；
- 再把物体的各个面按其离视点的远近进行排序，排序结果存在一张深度优先级表中；
- 然后按照从远到近的顺序逐个绘制各个面，后显示的图形取代先显示的画面，相当于消除隐藏面。

- 算法关键

- 如何对场景中的物体按深度排序。

- 深度重叠测试

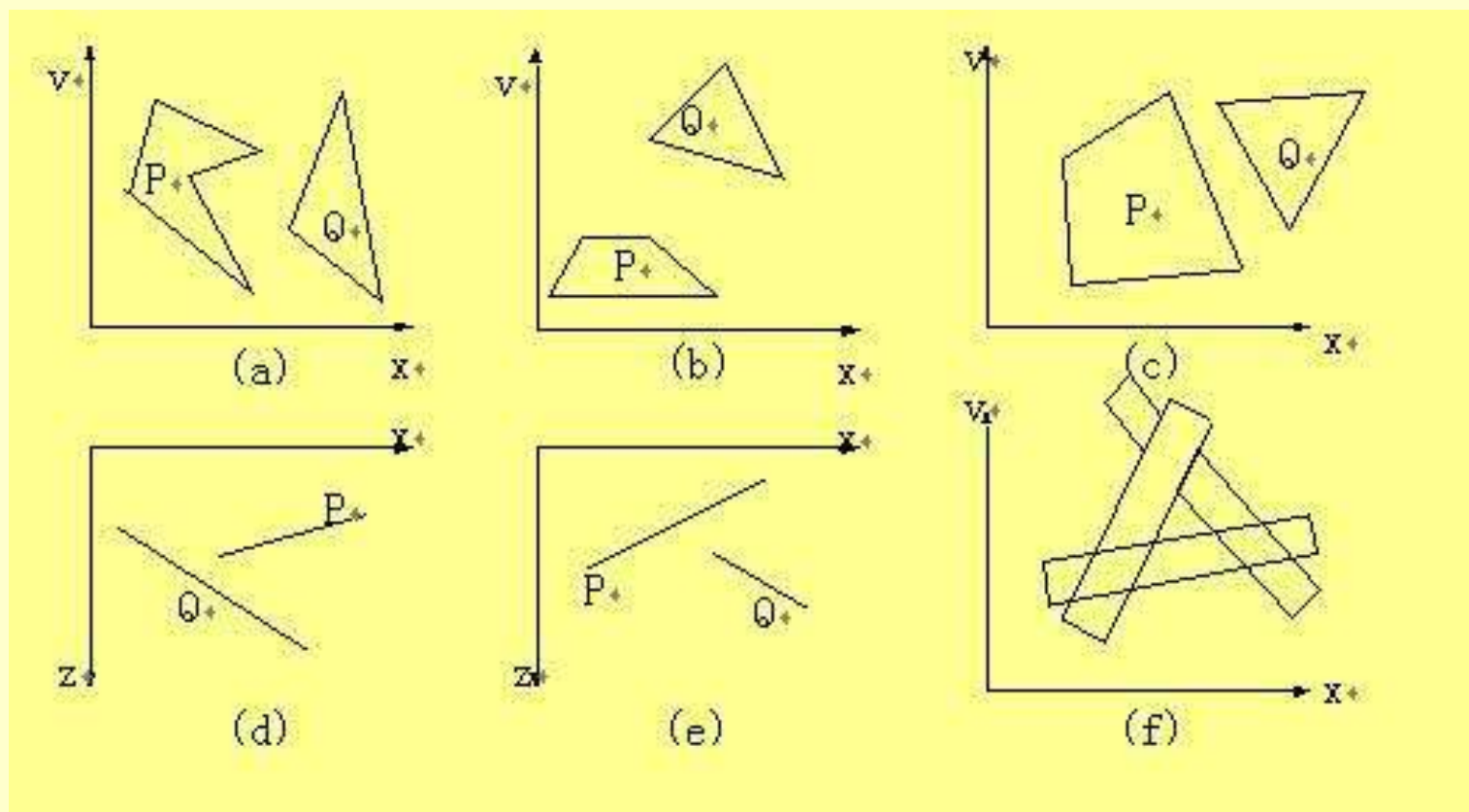
- 先根据每个多边形顶点z坐标的极小值 z_{\min} 的大小把多边形作初步排序；
- 设 z_{\min} 最小的多边形为P，序列中其他多边形记为Q。显然， $z_{\min}(P) < z_{\min}(Q)$ ，若 $z_{\max}(P) < z_{\min}(Q)$ ，则P肯定不能遮挡Q。
- 若 $z_{\max}(P) > z_{\min}(Q)$ ，则需进一步做投影重叠判断。

- **投影重叠判断**

- P和Q在oxy平面上投影的包围盒在x方向上不相交;
- P和Q在oxy平面上投影的包围盒在y方向上不相交;
- P和Q在oxy平面上的投影不相交;
- P在Q之后, 即P的各顶点均在Q的远离视点的一侧;
- Q在P之前, 即Q的各顶点均在P的靠近视点的一侧。
- 上面5项只要有1项成立, P就不遮挡Q。否则, 需要做精确的重叠测试。

- **精确的重叠测试**

- 以上测试失败, 必须对两个多边形在oxy平面上的投影作求交运算。计算时不必具体求出重叠部分, 在交点处进行深度比较, 只要能判断出前后顺序即可。
- 若遇到多边形相交或循环重叠的情况, 还必须在相交处分割多边形, 然后进行判断。



P不遮挡Q的各种情况(a, b, c, d, e) 及互相遮挡(f) ([Http://cg.cs.tsinghua.edu.cn](http://cg.cs.tsinghua.edu.cn))

- 优点

- 原理简单，关键是如何对场景中的物体按深度排序。

- 缺点

- 只能处理互不相交的面，而且深度优先级表中面的顺序可能出错。在两个面相交，三个以上的面重叠的情形，用任何排序方法都不能排出正确的序。这时只能把有关的面进行分割后再排序。

8.3 区域子分算法

8.3.1 算法思想

- 区域子分算法是针对光栅扫描式图象显示器上填色产生图形的。它是一种所谓分而治之的算法。
- 整个屏幕称为窗口，每一次把矩形的窗口等分成4个相等的小矩形，分成的矩形也称为窗口，见图8.2。

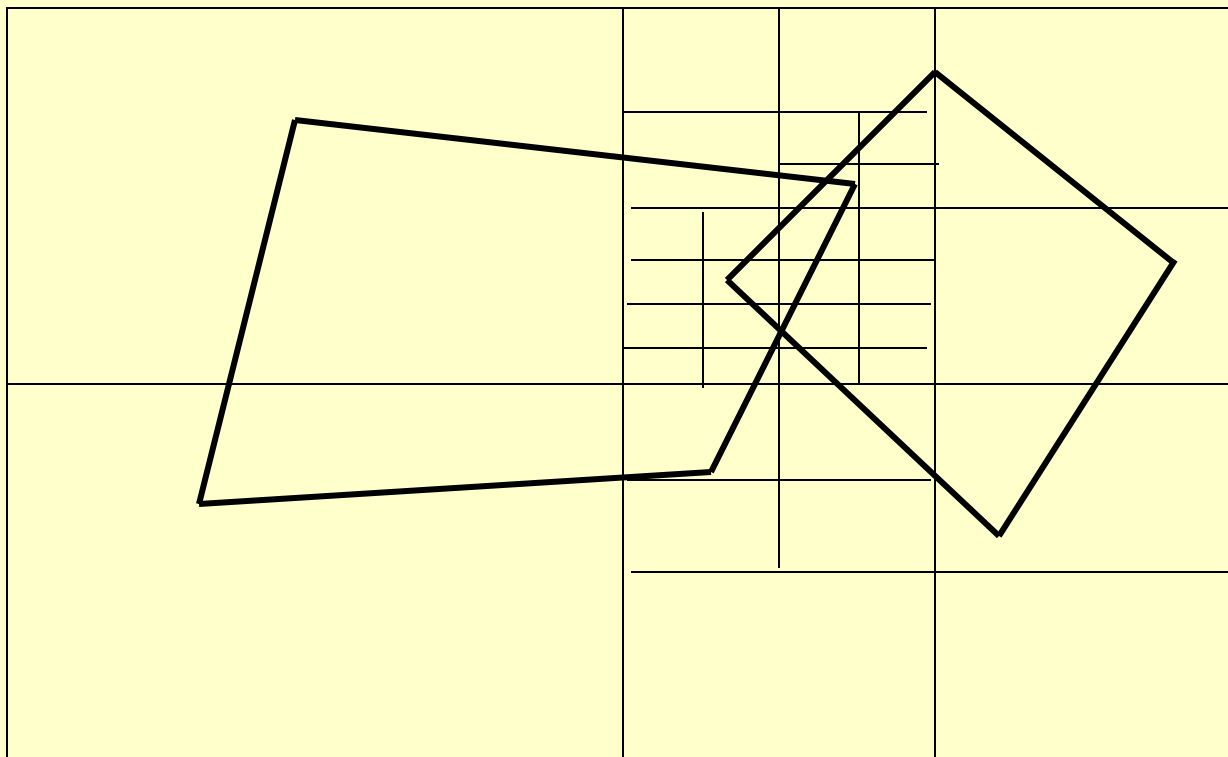


图8.2 子分的过程

- 每一次子分,均要把要显示的多边形和窗口的关系做一次判断。这种关系有以下4种:
 - 多边形包围了窗口(图8.3中情况1);
 - 多边形和窗口相交(图8.3中情况2);
 - 窗口包围了多边形(图8.3中情况3);
 - 窗口和多边形分离(图8.3中情况4)。

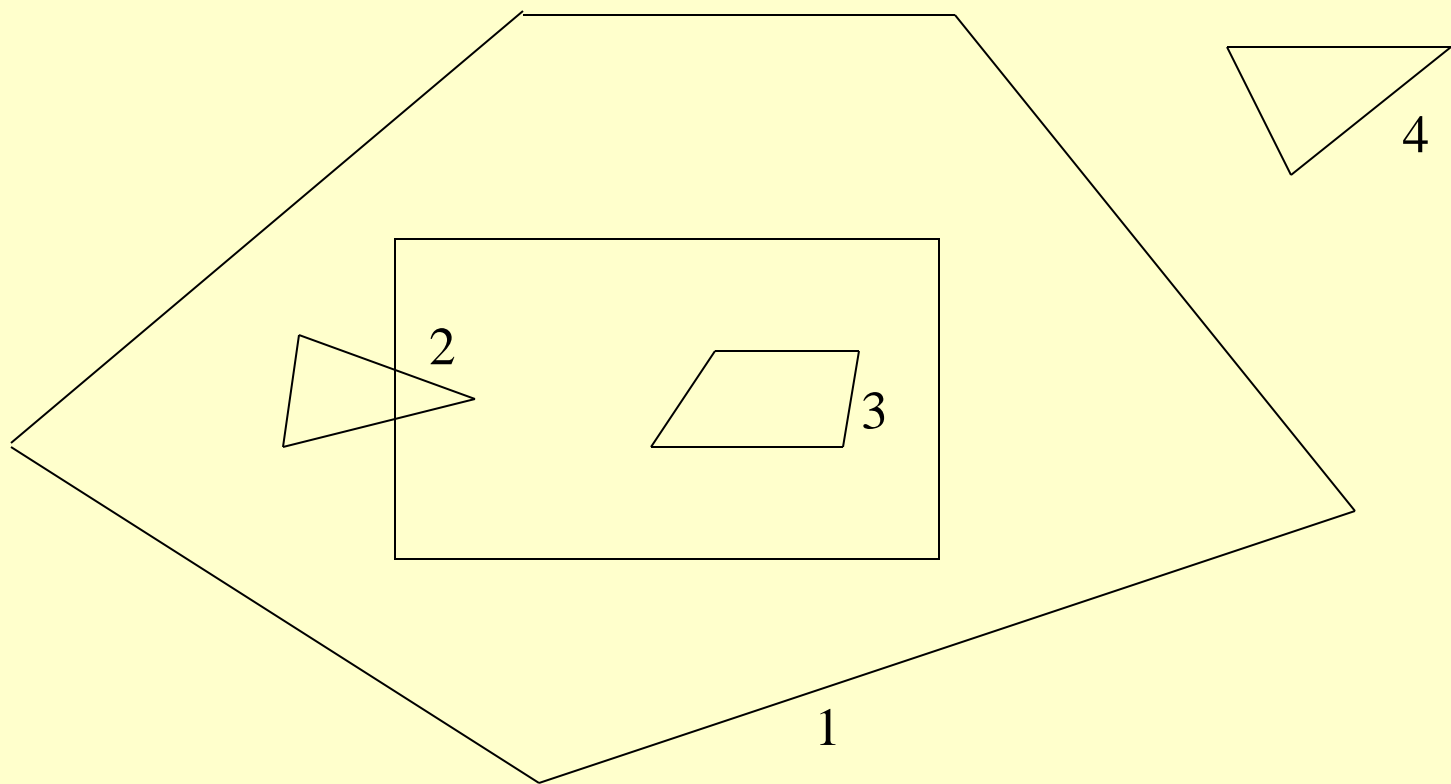


图8.3 多边形和窗口的4种关系

- 窗口和每个多边形的关系确定之后, 有些窗口内的图形便可显示了, 它们属于下列情况:
 - (1)所有多边形都和窗口分离。这时只要把窗口内所有的像素填上背景颜色。
 - (2)只有一个多边形和窗口相交, 或这个多边形包含在窗口内。这时先对窗口内每一像素填上背景颜色, 再对窗口内多边形部分用扫描线算法填色。
 - (3)只有一个多边形和窗口相交, 这个多边形把窗口整个包围在内; 或虽有几个多边形和窗口相交, 但离观察者最近的一个多边形包围了整个窗口。这时把整个窗口填上离观察者最近的那个多边形的颜色。

- 对上述3种情况不成立的窗口再一分为四,见图8.2。分得的窗口重复上述的处理。
- 重复处理后,窗口的边长越分越短,分了若干次后,窗口的边长就和一个象素的宽度一样了。这时,这个窗口对应的象素的颜色可取成最靠近观察者的多边形的颜色,或和这个窗口相交的多边形颜色的平均值。

- 算法步骤

- 该算法把初始窗口取作屏幕坐标系的矩形，将场景中的多边形投影到窗口内。

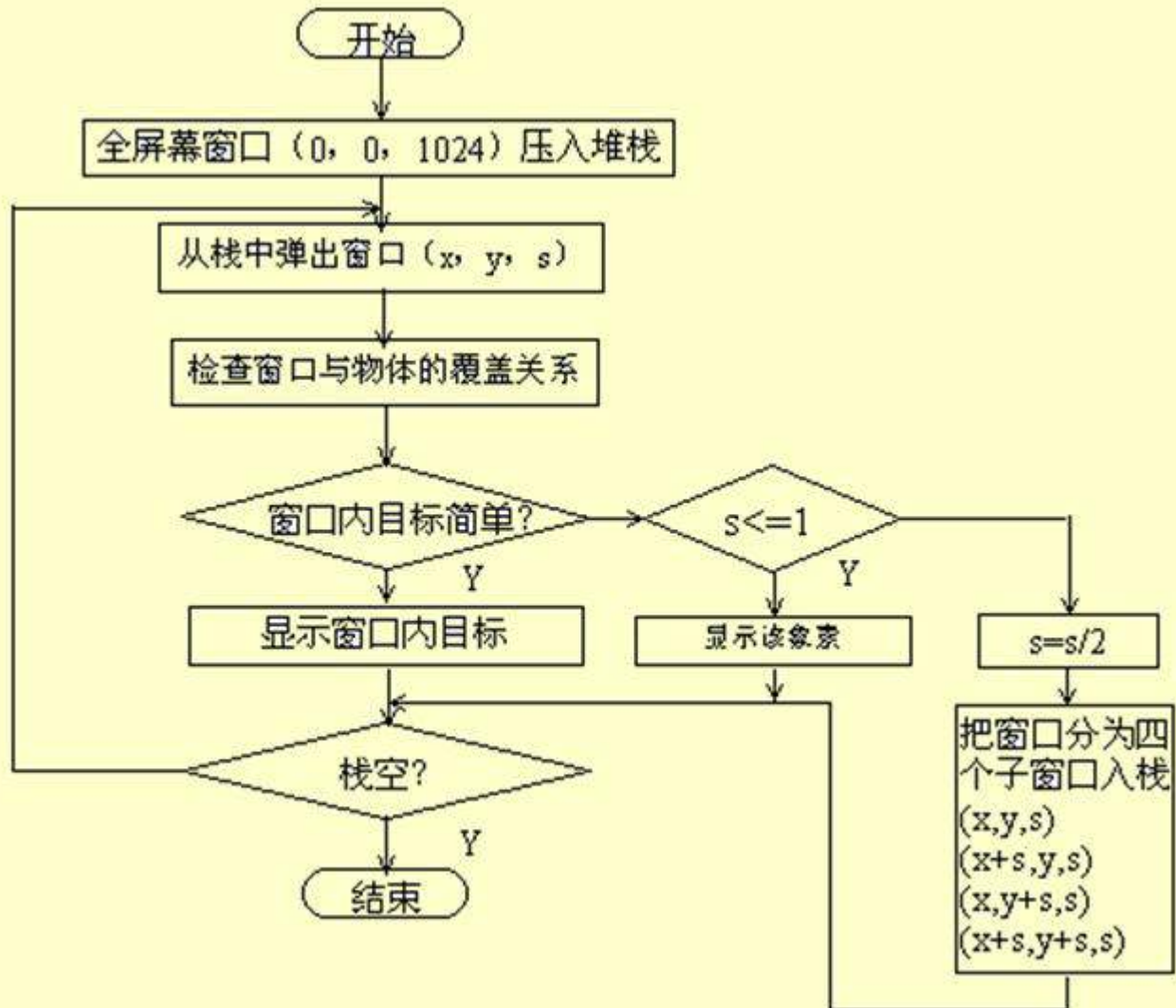
- 如果窗口内没有物体，则按背景色显示；

- 若窗口内只有一个面，则把该面显示出来。

- 否则，窗口内含有两个以上的面，则把窗口等分成四个子窗口。对每个小窗口再做上述同样的处理。这样反复进行下去。

- 如果到某个时刻，窗口仅有像素那么大，而窗口内仍有两个以上的面，这时不必再分割，只要取窗口内最近的可见面的颜色或所有可见面的平均颜色作为该像素的值。

- 算法流程图



8.3.2 改进算法效率的方法

- 一个窗口中可能含多个多边形, 对所有的多边形按其顶点的 z 坐标最大值来排序。对一个具体窗口来说, 随着窗口不断地被细分, 它的多边形序列的元素将越来越少。
- 窗口变小了, 就可能被一个多边形包含在内。这样, 在窗口内, 比这个多边形远离观察者的多边形都会被遮挡。因此, 这些被遮挡的多边形也可从窗口多边形序列中去掉。
- 用边界盒的办法就可判定一些多边形和指定窗口是无交的。因此, 这些多边形可从窗口多边形序列中排除, 从而提高排序效率。

边界盒方法

- 该方法可用于判断两个线段或多边形是否相交。
多边形的边界盒是包含这个多边形的平行于坐标轴的最小矩形, 这个矩形由4个参数 x_{min} , y_{min} , x_{max} , y_{max} 来确定, 见图8.4。
- 如果两个边界盒的参数为

$$x_{min,i}, y_{min,i}, x_{max,i}, y_{max,i}, i=1, 2$$

当它们满足

$$(x_{min,1} > x_{max,2}) \text{ 或 } (y_{min,1} > y_{max,2}) \text{ 或 } (x_{min,2} > x_{max,1}) \\ \text{或 } (y_{min,2} > y_{max,1})$$

时, 这两个边界盒不相交, 因而原来的两个多边形也不会相交。

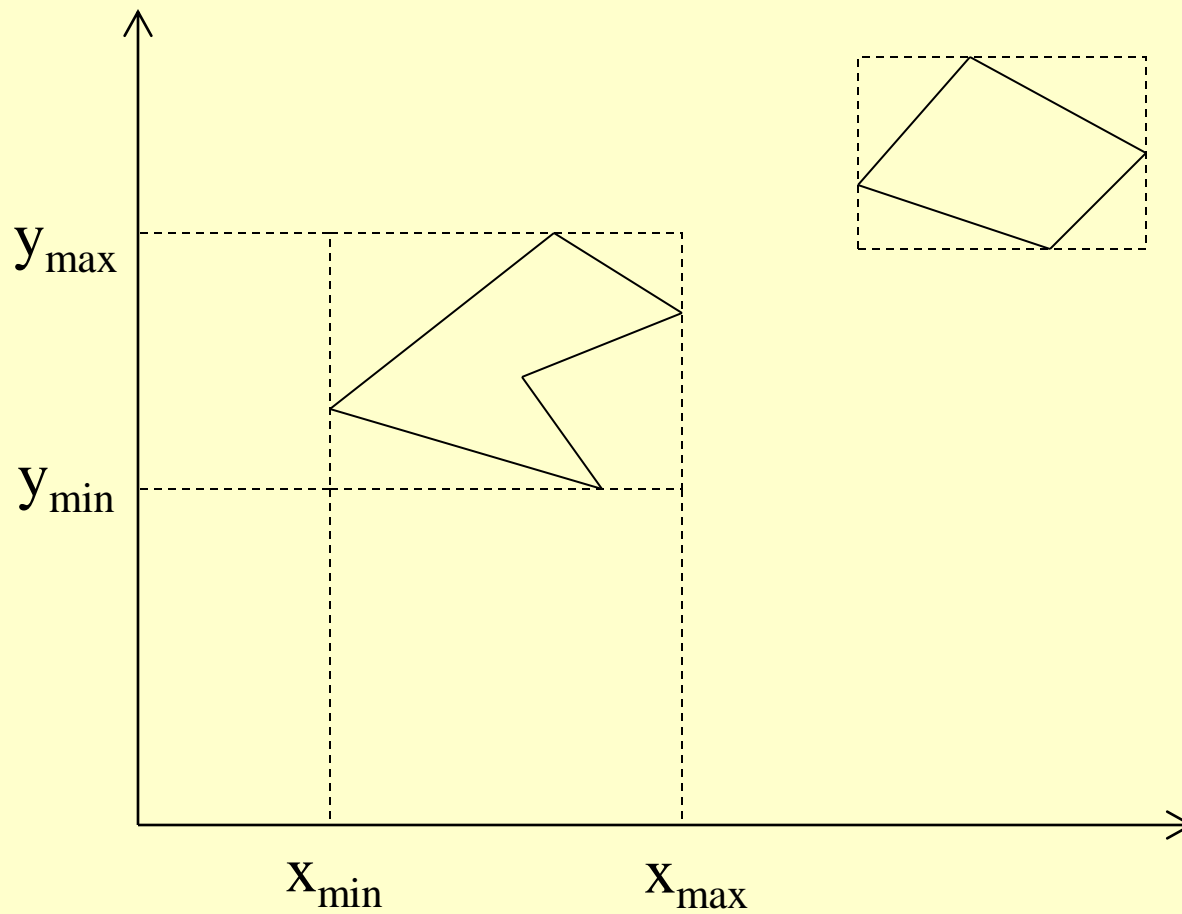


图8.4 多边形的边界盒

- 对于不能用边界盒办法判断和窗口不相交的多边形, 则要经过计算去确定它们之间的关系。这时可采用类似(第4章)多边形裁剪的方法去确定多边形和窗口的边界是否有交点。有交点时, 说明多边形和窗口有交。无交点时, 还要去确定它们是分离还是包含关系。
- 在找到了包围所考虑窗口的一个多边形后, 就要把它和多边形序列中其它多边形离观察者的远近进行比较, 把被它遮挡的多边形从序列中去掉。

遮挡判断

- 可把窗口4个顶点坐标的 x, y 值代入那个包围窗口的多边形的平面方程, 求出对应4个顶点处该平面的 z 坐标值, 这4个 z 坐标的最小值记为 z_{min} 。
- 对序列中第 i 个多边形, 求各顶点 z 坐标值的最大值 $z_{max,i}$, 若 $z_{min} \geq z_{max,i}$, 则第 i 个多边形被遮挡, 如图8.5中多边形CD被多边形AB所遮挡。
- 其它情况, 如图8.5中多边形EF, 可在窗口内多边形EF上任找一点 $G(x_G, y_G, z_G)$, 把 x_G, y_G 代入多边形AB所在的平面方程, 求出H点处的坐标 z 值 z_H , 若 $z_H > z_G$, 则AB在窗口内遮挡了EF。

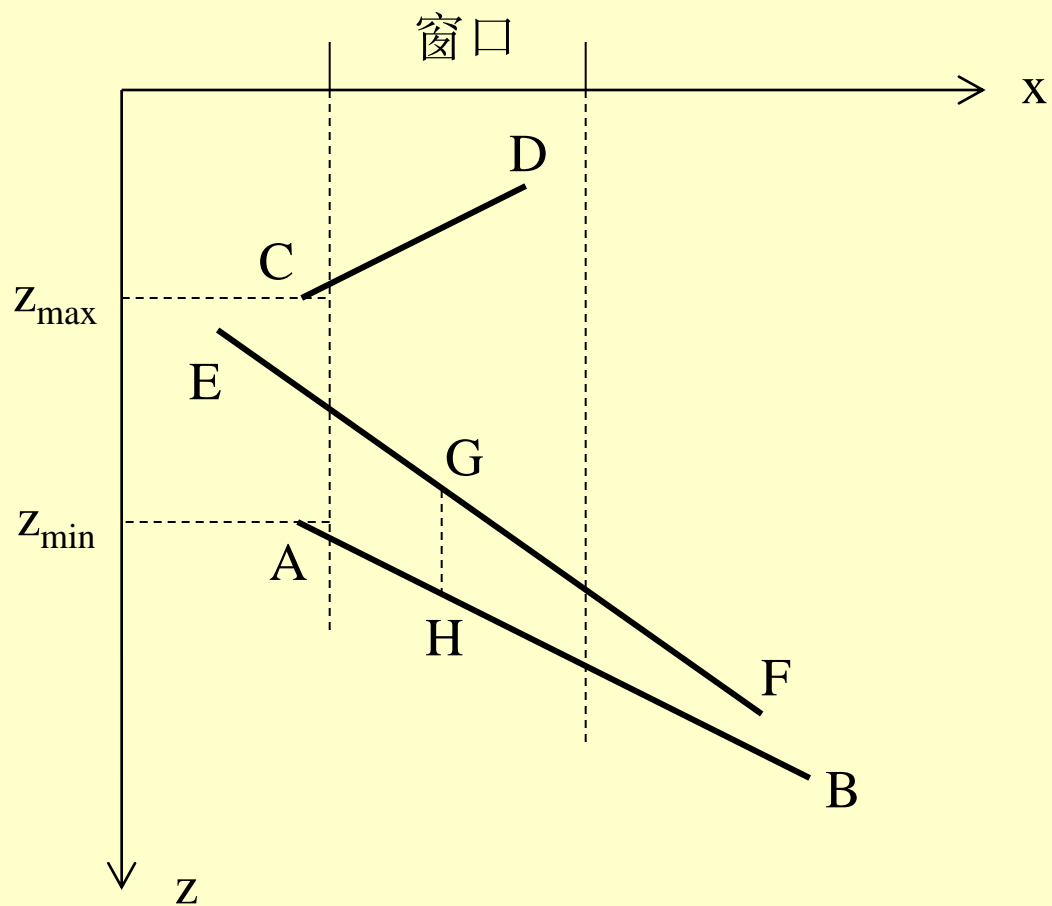


图8.5 确定被遮挡多边形的办法

8.4 深度缓存算法

8.4.1 算法思想

- 深度缓存算法(Z-Buffer)是一种最简单的图象空间面消隐算法, 既适应于多边形面也适用其它曲面。
- 它需要一个深度缓存数组ZB, 其大小与屏幕上象素点的个数相同, 也与显示器的帧缓存FB的单元个数相同, 彼此一一对应。
- 如图8.6, 在屏幕坐标系中, 过屏幕上任一象素点 (i, j) 作平行于Z轴的射线 R , 与物体表面上的多边形相交于 p_1 和 p_2 点。
- 比较 p_1 和 p_2 的Z值, 将最大Z值存入深度缓存数组ZB, 最大Z值所对应点的颜色存入显示器的帧缓存FB。

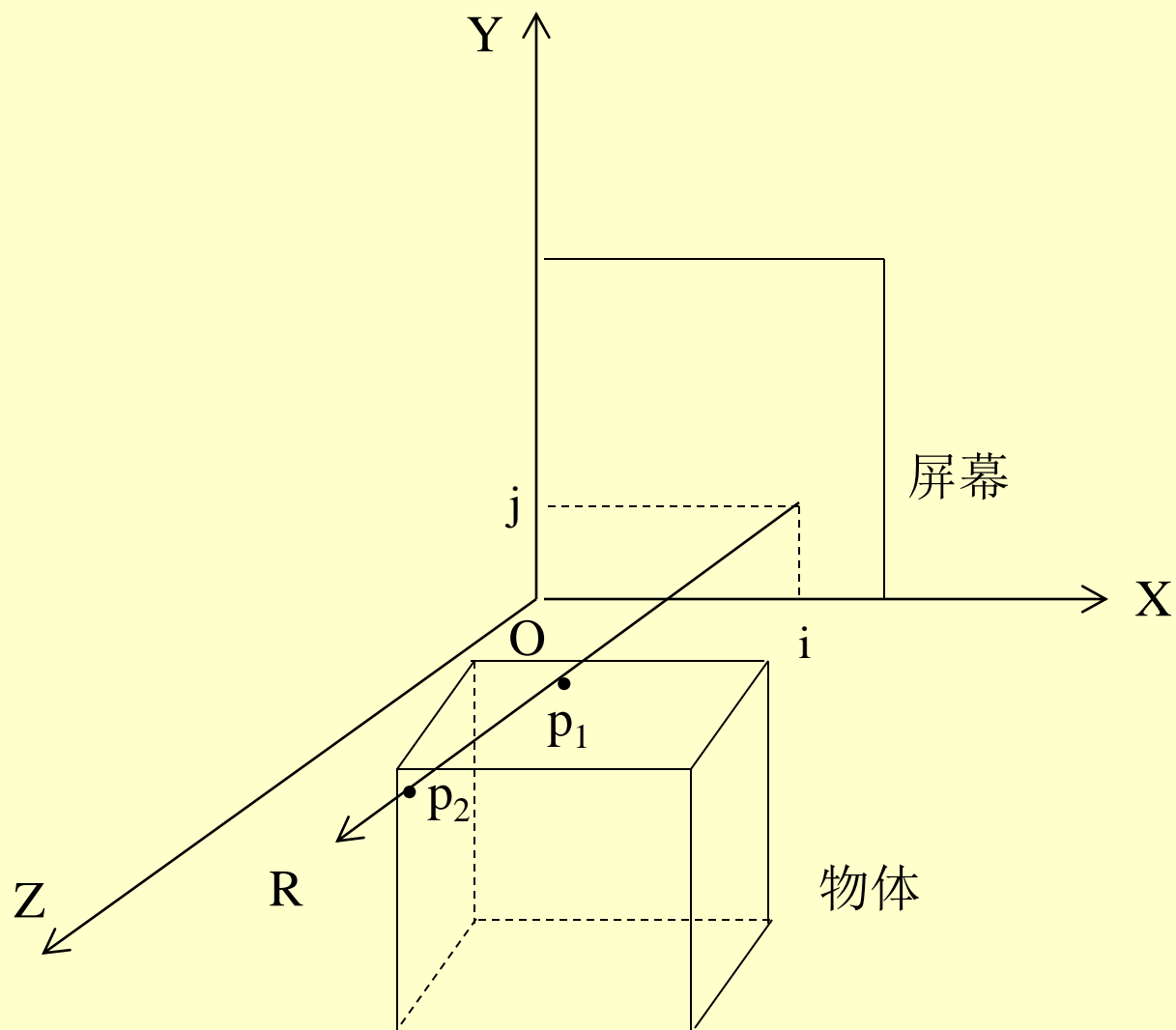


图8.6 射线R与物体相交于 p_1, p_2 点

8.4.2 算法描述

- 若有 N 个多边形, 屏幕上象素点个数为 $m \times n$, 则算法的基本步骤如下:

(1) 初始化 ZB 和 FB , 使

$ZB(i,j)=z$ 的极小值, $FB(i,j)=$ 背景色;

$(i=1,2,\dots,m, j=1,2,\dots,n)$

(2) $FOR\ j=1,n$ /*第 j 根扫描线*/

$FOR\ i=1,m$ /*第 j 根扫描线上第 i 个象素点*/

$FOR\ k=1,N$ /*多边形 P_k */

- (2. 1) 令 $Z_{ij}=Z$ 的极小值;
- (2. 2) 判断点 (i,j) 是否落在多边形 P_k 在XOY面上的投影多边形内;
- (2. 3) 若 (i,j) 在 P_k 的投影多边形内, 则计算多边形 P_k 在点 (i,j) 处的深度值 Z_{ij} ;
- (2. 4) 比较 Z_{ij} 与 $ZB(i,j)$ 的大小, 若 $Z_{ij}>ZB(i,j)$, 则令 $ZB(i,j)=Z_{ij}$,
 $FB(i,j)=$ 多边形 P_k 的颜色。

- 实现上述算法要解决如下2个问题：
 - (1).判断点 (i, j) 是否落在多边形 P_k 在XOY平面上的投影多边形内;
 - (2).计算多边形 P_k 在点 (i, j) 处的深度值 Z_{ij} 。
- 解决问题(1)可采用射线法或弧长法。
- 计算多边形 P_k 在 (i, j) 处的深度值, 方法如下:
设多边形 P_k 的平面方程: $ax+by+cz+d=0$
若 $c \neq 0$, 则把 (i, j) 代入面方程, 得

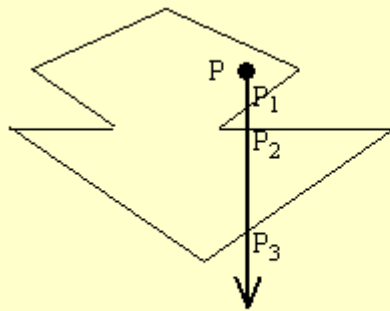
$$Z_{ij} = -\frac{ai + bj + d}{c}$$

若 $c = 0$, 则说明多边形 P_k 在XOY的投影为一条直线, 算法不考虑这种多边形。

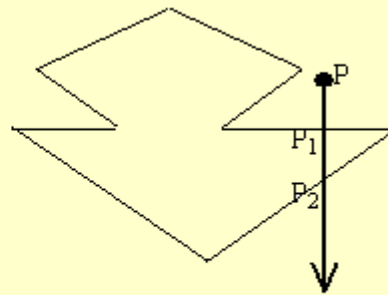
- 射线法

- 由被测点P处向 $y=-\infty$ 方向作射线

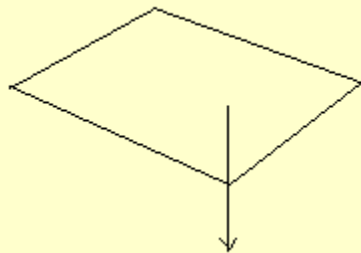
- 交点个数是奇数，则被测点在多边形内部；
 - 交点个数是偶数，则被测点在多边形外部。



(a) 奇数交点



(b) 偶数交点



- 若射线正好经过多边形的顶点，则采用“左开右闭”的原则来实现。即：当射线与某条边的顶点相交时，若边在射线的左侧，交点有效，计数；若边在射线的右侧，交点无效，不计数。

- 弧长法

- 要求多边形是有向多边形，一般规定沿多边形的正向，边的左侧为多边形的内域。
- 以被测计算点为圆心作单位圆，将全部有向边向单位圆作径向投影，并计算其在单位圆上弧长的代数和。
 - 代数和为0，点在多边形外部；
 - 代数和为 2π ，点在多边形内部；
 - 代数和为 π ，点在多边形边上。
- 计算边的弧长费时，可利用多边形的顶点符号，以及边所跨越的象限计算弧长代数和。

8.5 扫描线算法

- 深度缓存算法的优点是简单、可靠,不需要对显示对象的面预先进行排序;
- 缺点是要很大的Z缓冲器,显示对象的表面和像素对应的每一个点处都要计算它的Z值,因而工作量较大。
- 为了克服上述缺点,引入面向多边形场景的扫描线算法。处理时考虑扫描线内多边形面的相关性,以及扫描线间的多边形面的相关性,省略大量Z值计算。处理后,帧缓冲器中这一行的值便反应了消隐后的图形。

- 在处理当前扫描线时，开一个一维数组作为当前扫描线的Z-buffer。首先找出与当前扫描线相关的多边形，以及每个多边形中相关的边对。在求交点时，利用边的相关性。
- 对每一个边对之间的小区间的各像素，计算其深度，并与Z-buffer中的值比较，找出各像素处的可见平面。在计算深度时，充分利用图形的连贯性，采用增量算法计算深度。
- 计算颜色，写帧缓存。

- 如图8.7所示, 利用边的相关性可容易计算多边形P在指定象素 (i,j) 的深度值Z。

改进方法如下:

(1)计算投影多边形P'的边L与扫描线的交点

- 设L的方程为

$$px+qy+r=0$$

则在 $y=j$ 这根扫描线上, 边与扫描线交点的 x 坐标为:

$$x_j=-(qj+r)/p$$

在 $y=j+1$ 这根扫描线上, 交点的 x 坐标为:

$$x_{j+1}=-(q(j+1)+r)/p=x_j-q/p= x_j+\Delta x$$

- 上式表明, 将边与前一条扫描线的交点 x_j 做一次加法就可得到下一条扫描线的交点

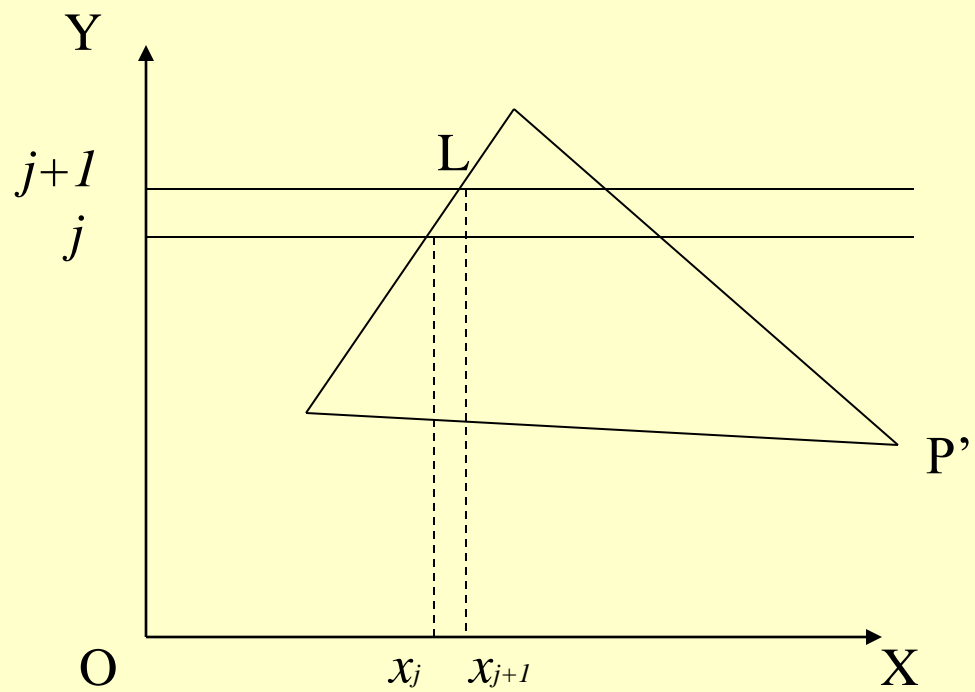


图8.7 边的相关性

(2)计算多边形在其边界的深度值

- 设多边形 $P(ax+by+cz+d=0)$ 的边 L 与扫描线 $y=j$ 的交点为 x_j ,则多边形 P 在 (x_j,j) 处的深度为

$$z_j=-(ax_j+bj+d)/c$$

- 在扫描线 $y=j+1$ 上,边 L 与扫描线的交点坐标为 $(x_{j+1},j+1)$,则多边形 P 在交点处的深度为

$$z_{j+1}=-(ax_{j+1}+b(j+1)+d)/c$$

$$z_{j+1}=z_j+aq/(cp)-b/c= z_j+ \Delta z_x\Delta x+\Delta z_y$$

Δz_x 为多边形在X方向的深度增值, Δz_y 为多边形在Y方向的深度增值。

- 上式表明,求多边形在边与扫描线 $j+1$ 的交点处的深度值 z_{j+1} ,可基于边与前一个扫描线 j 的交点深度值做一次乘法二次加减法就可得到。

(3)计算多边形在点*(i,j)*处的深度值

- 设*(i,j,z_{ij})*为多边形P内的一点, *z_{ij}*为多边形P在*(i,j)*处的深度值,则

$$z_{ij} = -(ai + bj + d)/c$$

- 在点*(i+1,j)*,则多边形P的深度为

$$z_{i+1,j} = -(a(i+1) + bj + d)/c$$

$$z_{i+1,j} = z_{ij} - a/c = z_{ij} + \Delta z_x$$

Δz_x 为多边形在X方向的深度增值。

- 上式表明,求多边形P在*(i,j)*处的深度值*z_{ij}* 后,用一个加法就可得到多边形在同一条扫描线*j*上后续点*(i+1,j)*处多边形P的深度*z_{i+1,j}*。

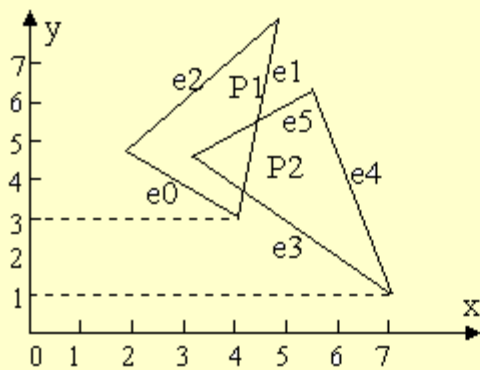
- 数据结构

- 多边形Y表，活化多边形表APT，边表ET，活化边对表AET。

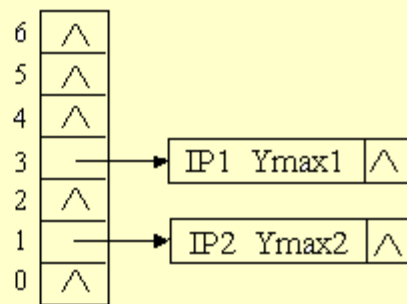
- 多边形Y表：将所有多边形存在多边形Y表中。

根据多边形顶点中最小的y坐标值，插入多边形Y表中的相应位置。

多边形Y表中只保存多边形的序号和其顶点的最大y坐标。根据序号可以从定义多边形的数据结构中取多边形信息：多边形所在面的方程 $ax+by+cz+d=0$ 的系数，多边形的边，顶点的坐标和颜色等。



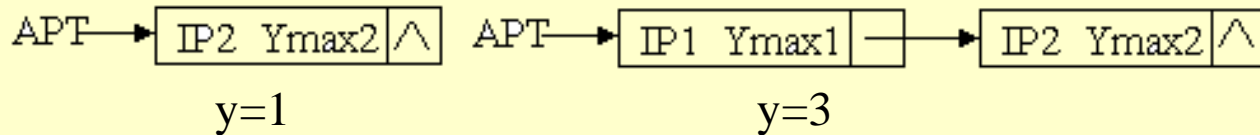
待消隐对象



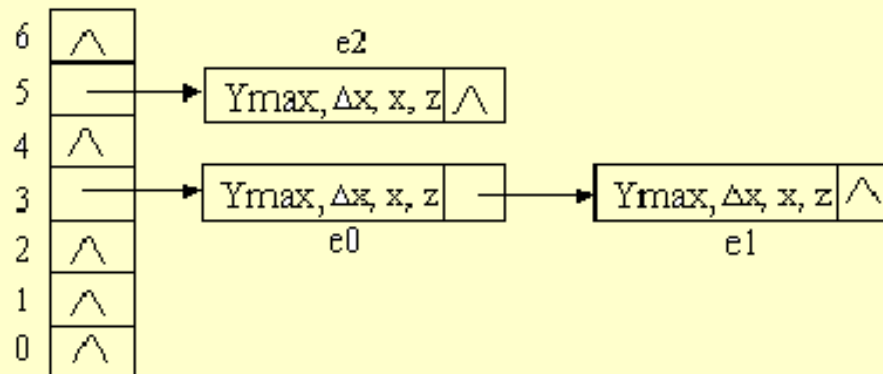
多边形Y表

- 数据结构

- 活化多边形表APT：与当前扫描线相交的多边形存在APT中，是一个动态的链表。



- 边表ET：活化多边形表中的每一个多边形都有一个边表。下图给出了多边形 P_1 的边表。边表中，存放了每条边端点中较大的y值，增量 Δx ，y值较小一端的x坐标和z坐标。



• 数据结构

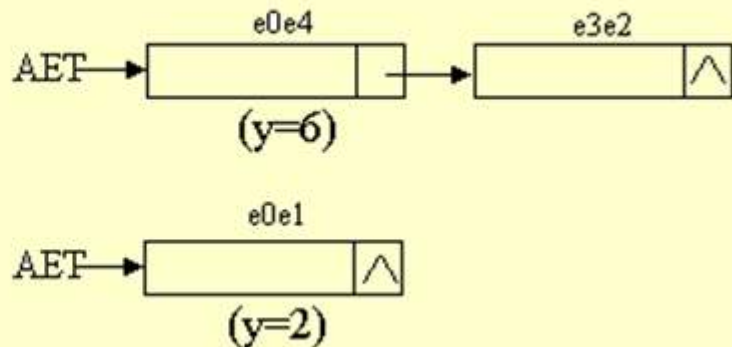
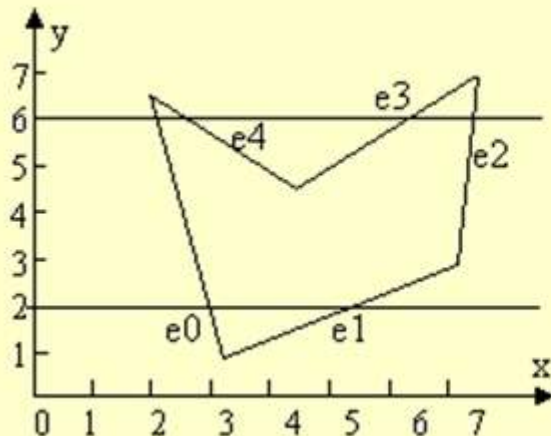
– 活化边对表AET

- 在一条扫描线上，同一多边形的某两条边构成一个边对。
活化边对表中存放当前多边形中与当前扫描线相交的各边对的信息。

$x_l \Delta x_l y_{lmax}$: 左侧边与扫描线交点的x坐标；左侧边在扫描线加1时的x坐标增量；左侧边两端点中最大的y值。

$x_r \Delta x_r y_{rmax}$

$z_l IP \Delta z_x \Delta z_y$: 左侧边与扫描线交点处的多边形深度值；多边形序号；沿扫描线方向（x方向）增加1个像素时，多边形所在平面的z坐标增量，为 $-a/c$ ；沿y方向扫描线加1时，多边形所在平面的z坐标增量，为 $-b/c$ 。



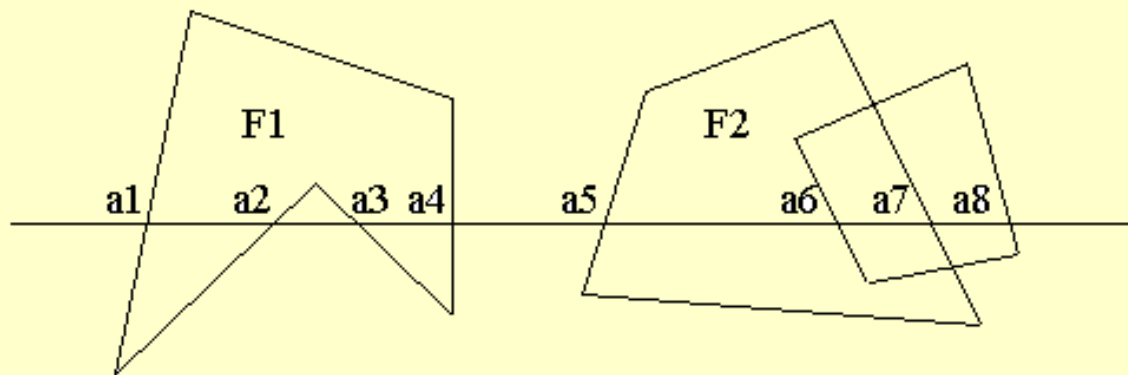
扫描线Z-buffer算法 ()

```
{ 建立多边形y表; 对每一个多边形根据顶点最小的y值, 将多边形置入多边形y表。  
 活化多边形表APT, 活化边对表AET初始化为空。  
  for (每条扫描线i, i从小到大)  
  { 1. 帧缓存CB置为背景色。  
    2. 深度缓存ZB(一维数组)置为负无穷大。  
    3. 将对应扫描线i的, 多边形y表中的多边形加入到活化多边形表APT中。  
    4. 对新加入的多边形, 生成其相应的边表。  
    5. 对APT中每一个多边形, 若其边表中对应扫描线i增加了新的边, 将新边配对,  
      加到活化边对表AET中。  
    6. 对AET中的每一对边:  
      6.1 对 $x_l < j < x_r$ 的每一个像素, 按增量公式 $z = z + \Delta z_x$ 计算各点深度depth。  
      6.2 与ZB对应量比较,  $depth > ZB(j)$ , 则 $ZB(j) = depth$ , 并计算颜色值, 写帧缓存。  
    7. 删除APT中多边形顶点最大y坐标为i的多边形, 并删除相应的边。  
    8. 对AET中的每一个边对, 作如下处理:  
      8.1 删除 $y_{lmax}$ 或 $y_{rmax}$ 已等于i的边。  
      8.2 用增量公式计算新的 $x_l, x_r$ 和 $z_l$ 。  
           $x_l = x_l + \Delta x_l$   $x_r = x_r + \Delta x_r$   $z_l = z_l + \Delta x_l \Delta z_x + \Delta z_y$   
  }  
}
```

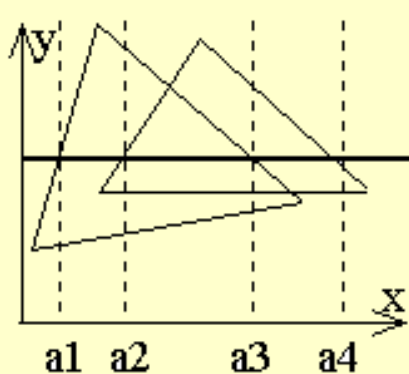

8.6 区间扫描线算法

- 与深度缓存算法相比，扫描线算法做了两点改进：
 - 将整个绘图窗口内的消隐问题分解到一条条扫描线上解决，使所需的Z缓冲器大大减少；
 - 计算深度值时，利用了面连贯性，只用了一个加法。
- 问题
 - 在每个像素处都计算深度值，进行深度比较。
 - 被多个多边形覆盖的像素区处需进行多次计算，计算量仍很大。
- 区间扫描线算法
 - 在一条扫描线上每个区间只计算一次深度值，不需要Z缓冲器。
 - 把当前扫描线与各多边形在投影平面的投影的交点进行排序后，将扫描线分成若干个子区间。
 - 在区间任一点处找出在该处z值最大的一个面，这个区间上的每一个像素就用这个面的颜色来显示。
 - 优点：将逐点(像素)计算改为逐段计算，效率大大提高。

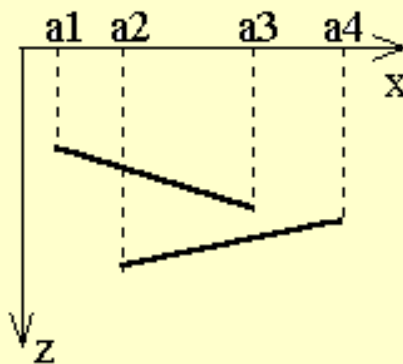
- 如何确定小区间的颜色可分为三种情况：
 - 小区间上没有任何多边形，该小区间用背景色显示。
 - 小区间上只有一个多边形，用对应的多边形在该处的颜色显示。
 - 小区间上存在两个或两个以上的多边形，必须通过深度测试判断哪个多边形可见。



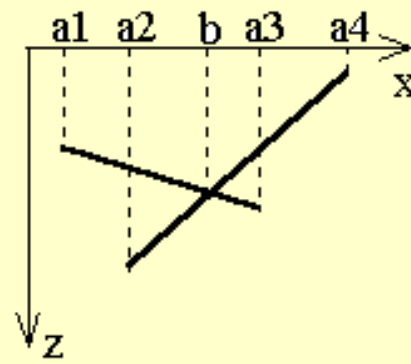
- 若允许物体表面相互贯穿，还必须求出它们在扫描平面(zx 平面)的交点，用这些交点把该小区间分成更小的子区间(称为间隔)，在这些间隔上决定哪个多边形可见。例如将 $[a2, a3]$ 区间分成 $[a2, b]$ 和 $[b, a3]$ 两个子区间。



两个多边形在屏幕上的投影



无贯穿的情形



相互贯穿的情形

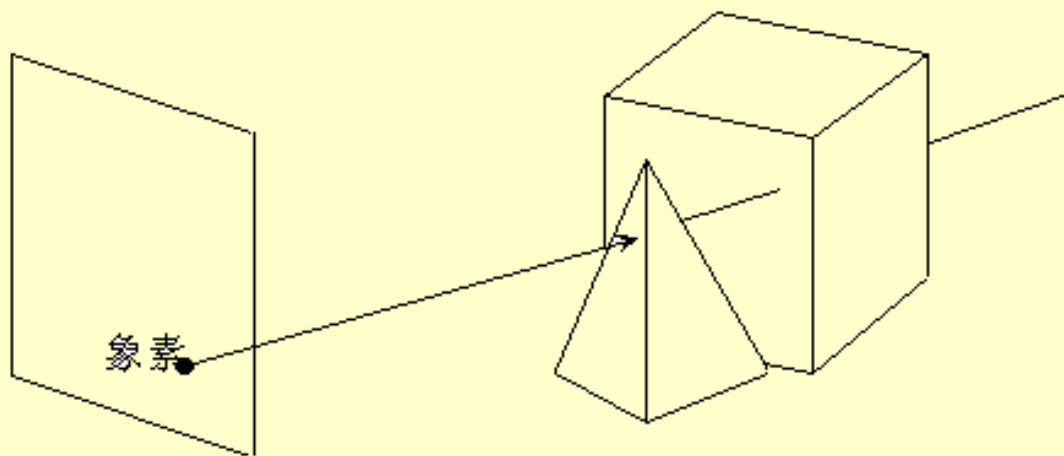
- 为了确定某间隔内哪一个多边形可见，可在间隔内任取一采样点，分析该点处哪个多边形离视点最近，该多边形即为在该间隔内可见的多边形。

8.7 光线投射算法

- 基本思想

- 考察由视点出发穿过观察屏幕的一像素而射入场景的一条射线，则可确定出场景中与该射线相交的物体；
- 在计算出光线与物体表面的交点之后，离像素最近的交点所在面片的颜色为该像素的颜色；
- 如果没有交点，说明没有多边形的投影覆盖此像素，用背景色显示它即可。

- 将通过屏幕各像素的投影线与场景中的物体表面求交



```
for 屏幕上的每一像素
{
    形成通过该屏幕像素(u, v)的射线;
    for 场景中的每个物体
        将射线与该物体求交;
    if 存在交点
        以最近的交点所属的颜色显示像素(u, v)
    else
        以背景色显示像素(u, v)
}
```

- 光线投射算法与Z缓冲器算法相比，算法复杂度类似。
- 区别在于光线投射算法不需要Z缓冲器。
- 为了提高本算法的效率，可使用包围盒技术，空间分割技术以及物体的层次表示方法来加速。

第9章 真实感图形技术

知 识 点： 光照模型，明暗处理，光线跟踪，纹理等真实感技术。

教学目的： 使学生了解真实感图形生成的基本原理和基本方法。

本章内容

9.1 概述

9.2 简单光照模型

9.3 多边形表示物体的光滑明暗处理

9.4 整体光照模型

9.5 光线跟踪技术

9.6 纹理

9.1 概述

- 真实感图形

综合运用数学、物理学、计算机科学与其它科学知识，在计算机设备上生成象彩色照片那样的真实感图形。

- 真实感图形生成步骤

- 用数学方法建立所需三维场景的几何描述，并将它们输入计算机；
- 将三维几何描述转换为二维透视图；
- 确定场景中的所有可见图(消隐)；
- 计算场景中可见面的颜色。（本章重点）

- 光强(Intensity of light)或称光亮度
既能表示光能大小又能表示其色彩组成的物理量。一般表示为(R,G,B).
- 光照模型
即光强的计算公式。根据光学的有关定律计算画面上景物表面各点投射到观察者眼中的光线的光亮度的公式。

9.2 简单光照模型

9.2.1 基本原理

- 光照射到物体上后分解为：反射光、透射光、被物体表面吸收转换成热。其中反射光和透射光的强弱决定了物体表面的明暗程度。(参见图9.1)
- 不同光源, 不同表面材料对光亮度有很大影响。

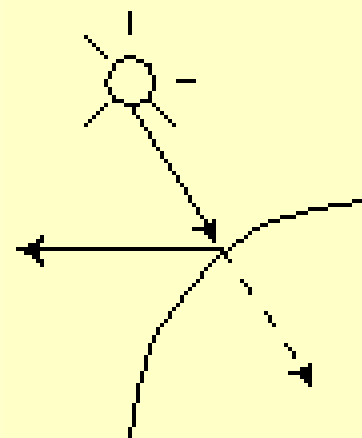


图9.1

- 简单光照模型(不考虑透射光, 仅有反射光)

而反射光又分成3个分量, 即:

反射光=环境反射+漫反射+镜面反射

- 环境反射

物体除了受特定光源照射外, 还受到周围环境来的反射光照射。该分量用 I_{pa} 表示, 取常数。

- 漫反射

表示特定光源在景物表面的反射光中那些向空间各方向均匀反射出去的光。

漫反射可使用朗伯定律计算。

- 朗伯定律

对于一个漫反射体, 表面的反射光亮度和光源入射角的余弦成正比。

$$I = I_{pd} \cos i$$

I 为表面反射光的光亮度;

I_{pd} 为光源垂直入射时反射光的光亮度;

i 为光源入射角(入射光角和表面法向夹角).

(参见图9.2)



- 镜面反射

对某些材质的物体,在光照射下出现“高光现象”。镜面反射的计算一般遵守反射定律。实际上,光滑表面镜面反射光分布于表面镜面反射方向的周围:

$$I = I_{ps} \cos^n \theta$$

I_{ps} -- 镜面反射方向上镜面反射光亮度;

θ -- 镜面反射方向和视线方向的夹角;

n -- 会聚指数。

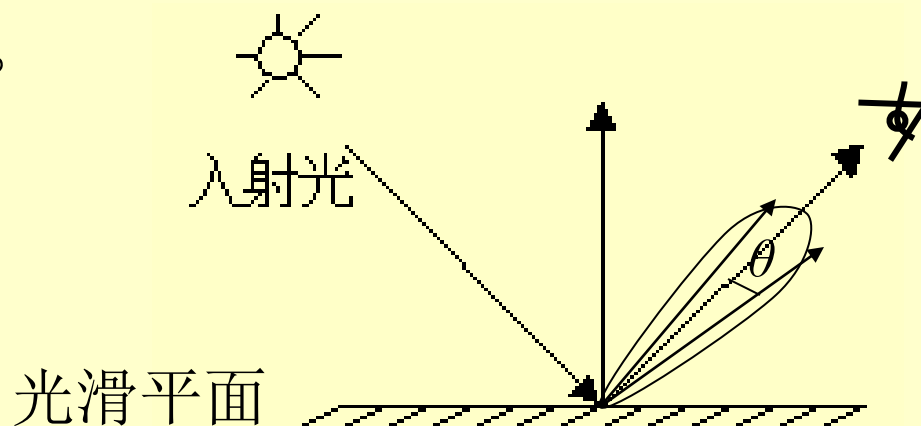


图9.3

- Phong模型

考虑反射光3个分量所建立的光照模型, 表示为:

$$I = k_a I_{pa} + \sum [k_d I_{pd} \cos i + k_s I_{ps} \cos^n \theta]$$

k_a, k_d, k_s 分别为环境反射、漫反射和镜面反射分量系数, 即材质系数 ($k_d + k_s = 1$), \sum 表示对所有光源求和。

- 计算机实现时, 用R、G、B三分量计算, 表示为:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = k_a \begin{bmatrix} r_{pa} \\ g_{pa} \\ b_{pa} \end{bmatrix} + \sum \left[k_d \begin{bmatrix} r_{pd} \\ g_{pd} \\ b_{pd} \end{bmatrix} \cos i + k_s \begin{bmatrix} r_{ps} \\ g_{ps} \\ b_{ps} \end{bmatrix} \cos^n \theta \right]$$

9.2.2 算法实现

- 首先确定 $\cos i$ 和 $\cos \theta$

$$\begin{cases} \cos i = (\vec{L}_0 \cdot \vec{N}_0) \\ \cos \theta = (\vec{R}_0 \cdot \vec{V}_0) \approx (\vec{N}_0 \cdot \vec{H}_0) \end{cases}$$

其中 $\vec{L}_0, \vec{N}_0, \vec{R}_0, \vec{V}_0$ 是L, N, R, V相应的单位向量,
 \vec{H}_0 为沿 \vec{L} 和 \vec{V} 的角平分线
的单位向量。

- 使用Phong模型的扫描
线绘制算法

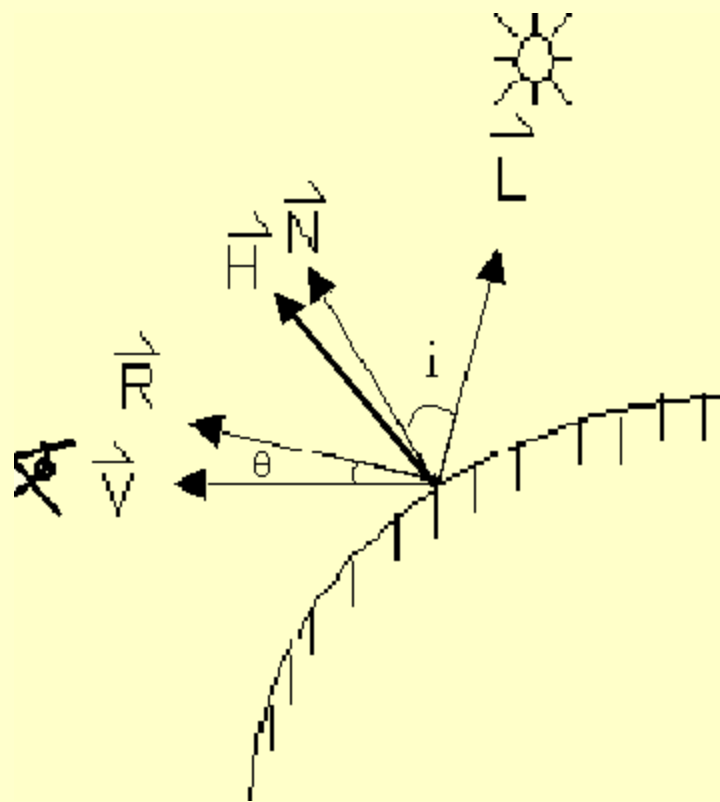


图9.4

算法:

begin

for 屏幕上每条扫描线y do

begin

将数组color初始化成y扫描线的背景颜色值;

for y扫描线上每个可见区间段s中的每点(x,y) do

begin

设(x, y)对应的空间可视点为p;

求出p点的单位法向量 \vec{N}_o , 单位入射光

向量 \vec{L}_o , 单位视线向量 \vec{V}_o ;

求出 \vec{L}_o 在P点的单位镜面反射向量 \vec{R}_o ;

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = k_a \begin{bmatrix} r_{pa} \\ g_{pa} \\ b_{pa} \end{bmatrix} + \sum \left[k_d \begin{bmatrix} r_{pd} \\ g_{pd} \\ b_{pd} \end{bmatrix} (\vec{L}_0 \cdot \vec{N}_0) + k_s \begin{bmatrix} r_{ps} \\ g_{ps} \\ b_{ps} \end{bmatrix} (\vec{R}_0 \cdot \vec{V}_0)^n \right]$$

且置color(x, y) := (r, g, b)

end

显示color

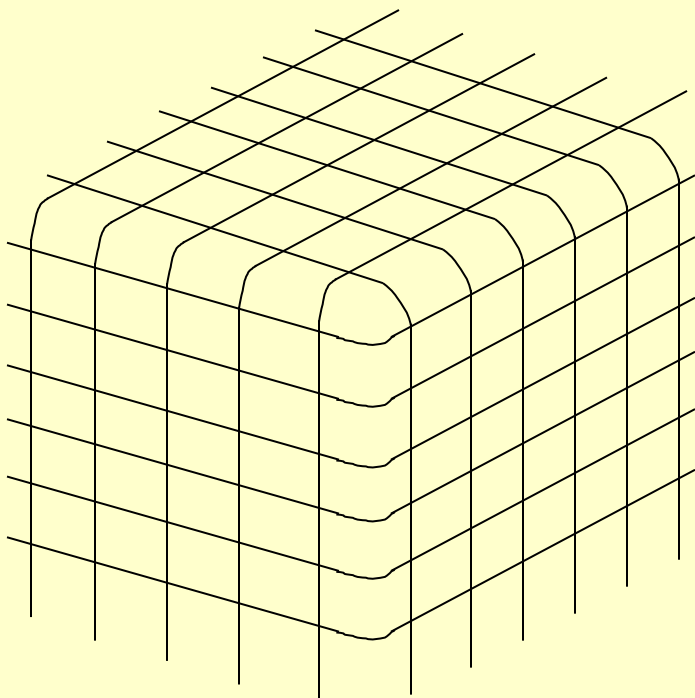
end

end

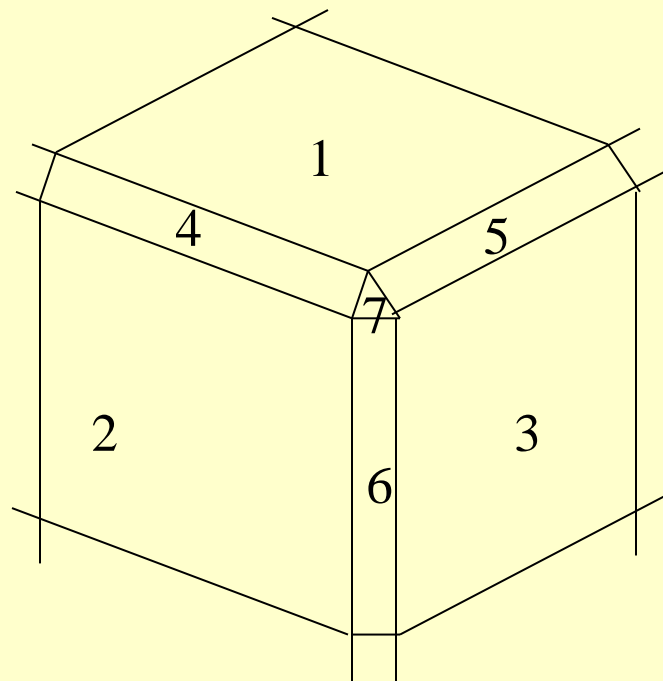
9.3 多边形表示物体的光滑明暗处理

9.3.1 问题

- 在计算机图形学中,光滑表面常用多边形予以逼近和表示,使处理变得容易。如图9.5(a)的图形常用(b)来近似。
- 但是,直接使用Phong模型计算时,不同平面块之间存在不连续的法向量跳跃,生成的图形失去原有曲面光滑性,呈多面体状。



(a)



(b)

图9.5 磨光立方体的多边形表示

9.3.2 解决方法

一、 Gouraud明暗处理技术(Gouraud Shading)

- 思想：对离散的光亮度采样作双线性插值以获取连续的光亮度函数。
- 过程：
 - a.计算出多边形顶点处的光亮度值, 作为插值采样点；
 - b.对多边形顶点的光亮度插值计算出多边形内任一点的光亮度。

- 如图9.6, 设三角形面3个顶点处光亮度为 I_1, I_2, I_3 , 为计算P点光亮度 I_p

$$I_a = uI_1 + (1-u)I_2 \quad (u = av_2/v_1v_2)$$

$$I_b = vI_1 + (1-v)I_3 \quad (v = bv_3/v_1v_3)$$

$$I_p = tI_a + (1-t)I_b \quad (t = Pb/ab)$$

同时, 相邻两点 P_1, P_2 可用增量法计算光亮度:

$$I_{p2} = I_{p1} + \Delta I$$

- 优缺点**

效果尚好, 计算量小;

不能正确模拟高光, 会产生Mach带效应(光亮度变化率不连续的边界处呈现亮带或黑带)。



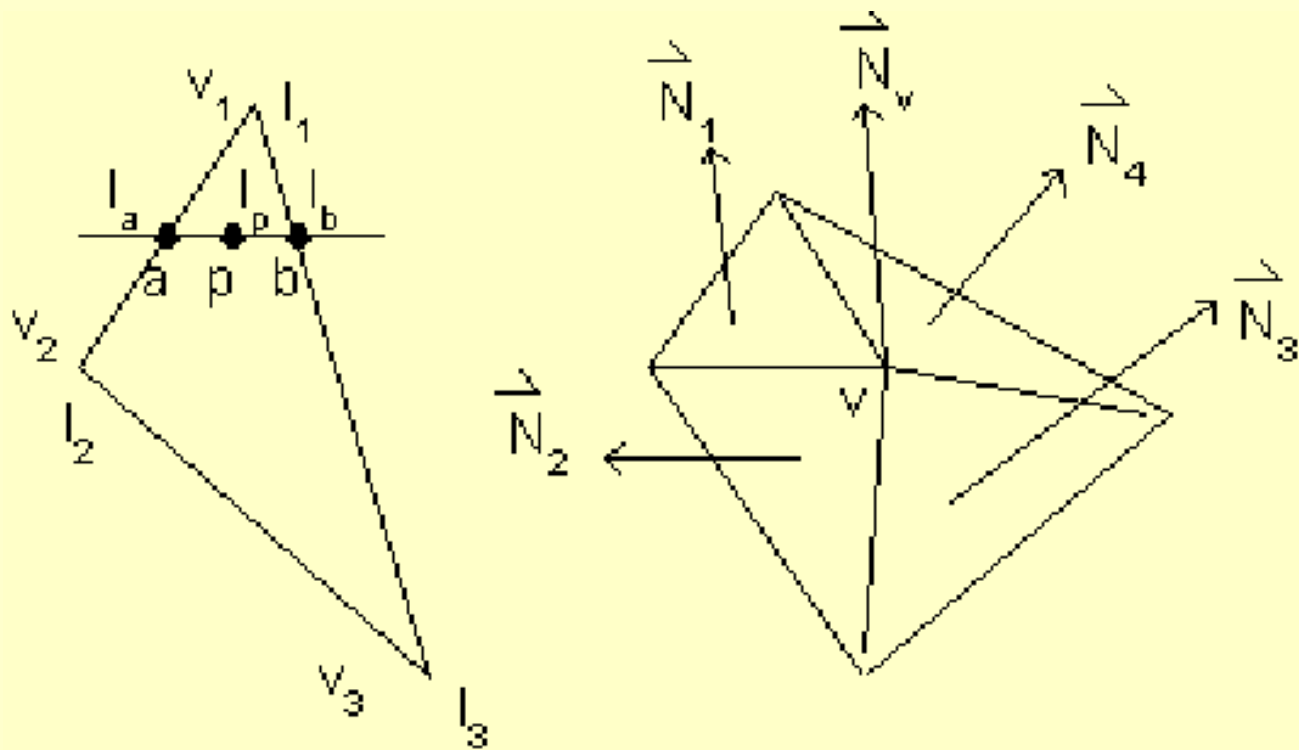


图9.6 光亮度插值及其顶点法向量计算

二、Phong明暗处理技术(Phong Shading)

- 思想: 对离散的法向量采样作双线性插值, 构造一个连续的法向量函数, 将这个连续的法向量插值函数代入光亮度计算公式, 即得到一个非线性的光亮度插值公式。
如图9.7所示, 任一点P处法向按插值方法由各顶点处法向推出。
- 优点: 大大减少了马赫带效应;
产生真实的高光效果。
- 缺点: 由于对每一像素光亮度计算还需使用光照模型, 故计算量大。

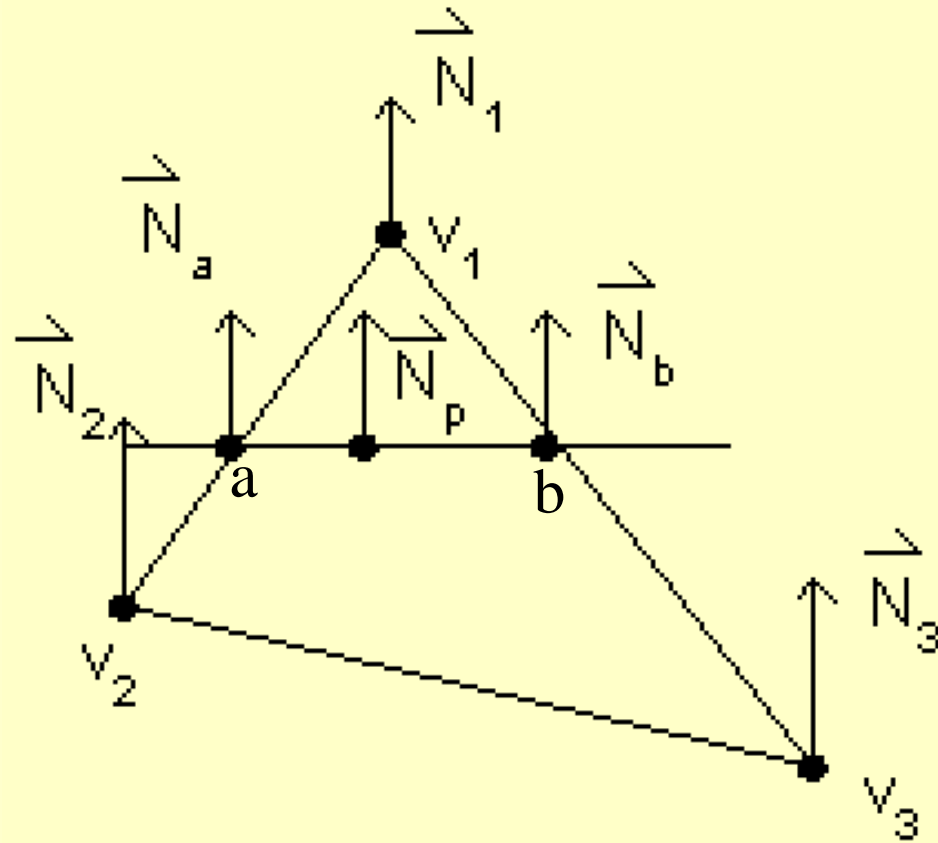


图9.7 法向双线性插值

9.4 整体光照模型

- Phong模型仅考虑光源直接照射在景物表面产生的反射光能,因而是一种局部光照模型。
- 局部光照模型忽略了光能在环境景物之间的传递,很难生成高质量真实感图形。
原因: 未能考虑环境的漫射、镜面反射和规则透射对景物表面产生的整体照明效果。



9.4 整体光照模型

- 整体光照模型

Whitted在Phong模型中增加了环境镜面反射光亮度 I_s 和环境规则透射光亮度 I_t , 从而模拟周围环境的光透射在景物表面上产生的理想镜面反射和规则透射现象。

- Whitted模型

如图9.8, 假设景物表面向空间某方向 \mathbf{V} 辐射的光亮度 I 由3部分组成:

- (1)由光源直接照射引起的反射光亮度 I_c ;
- (2)由 \mathbf{V} 的镜面反射方向来的环境光 I_s 投射在光滑表面上产生的镜面反射光;
- (3)由 \mathbf{V} 的规则透射方向来的环境光 I_t 通过透射在透明体表面上产生的规则透射光。

- Whitted模型表示公式

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

k_s : 反射系数, $0 \leq k_s \leq 1$; k_t : 透射系数, $0 \leq k_t \leq 1$.
 I_c 可用Phong模型计算, 而 I_s 和 I_t 可转化为其它物体表面朝P点方向辐射的光亮度, 仍利用Whitted模型计算。因而Whitted模型是一递归计算模型。

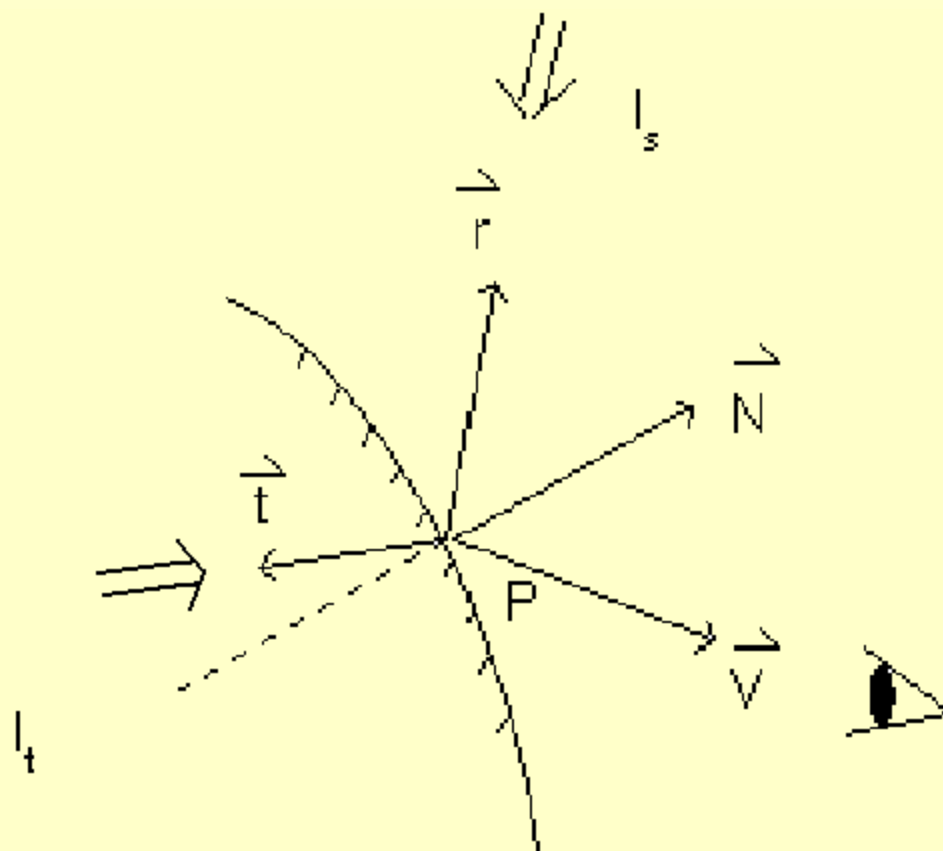


图9.8 Whitted模型示意图

9.5 光线跟踪技术(Ray Tracing)

9.5.1 基本原理

- 光线跟踪技术是为了求解Whitted模型而提出一种高度真实感图形绘制技术。
- 光线投射(ray casting)

假设从视点V通过屏幕像素e向场景投射一光线交场景中的景物于 $P_1, P_2 \dots P_m$ 点, 那么离视点最近的 P_1 点就是画面在像素e处的可见点, 像素e的光亮度应由 P_1 点向 $\overrightarrow{P_1V}$ 方向辐射的光亮度决定的。这样, 对屏幕上每一像素都投射光线以求得与场景的第一个交点, 并置像素的光亮度为交点处的光亮度。(如图9.9)

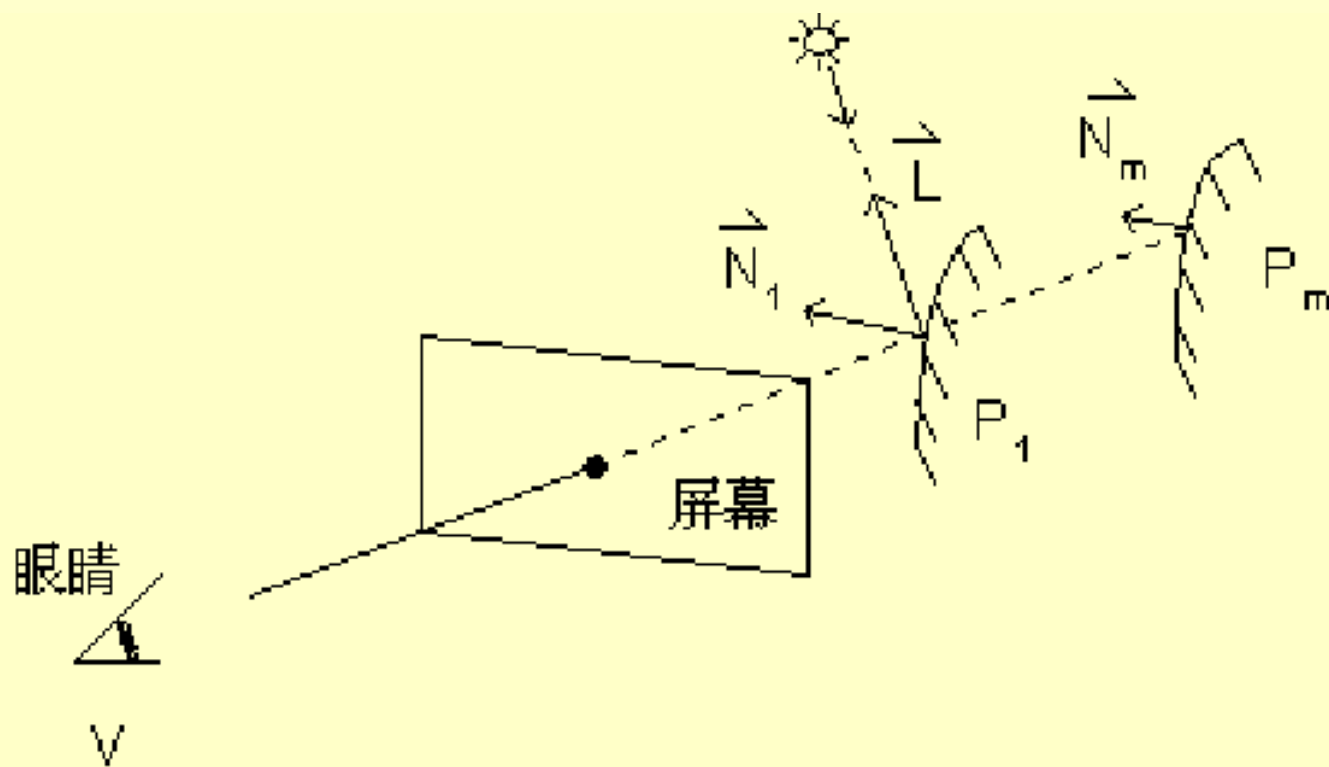


图9.9 光线投射过程

- 光线投射方法评价

优点: 不必单独消隐, 算法简单;

局限性: 仅实现局部光照效果。

- 光线跟踪技术原理

Whitted在求解其整体光照模型时将光线投射发展改进为光线跟踪技术。根据Whitted模型:

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

光线跟踪示意图如图9.10。

- 光线跟踪求解任一像素点e亮度 I 过程如下:

(1)先从视点V出发, 通过像素e发射一根光线求得它与场景的最近交点 P_1 , 容易算出 P_1 点的局部光照亮度 I_c (使用Phong模型)

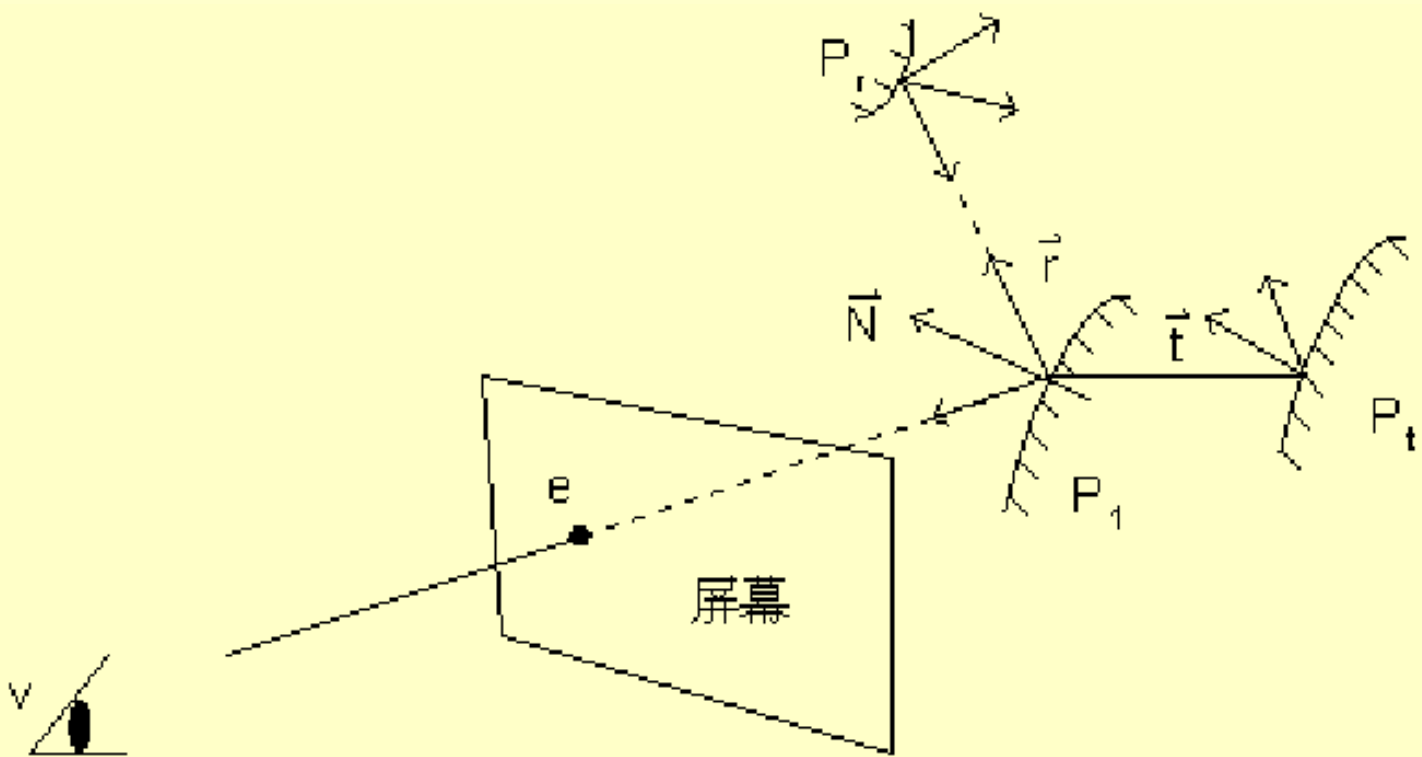


图9.10 光线跟踪示意图

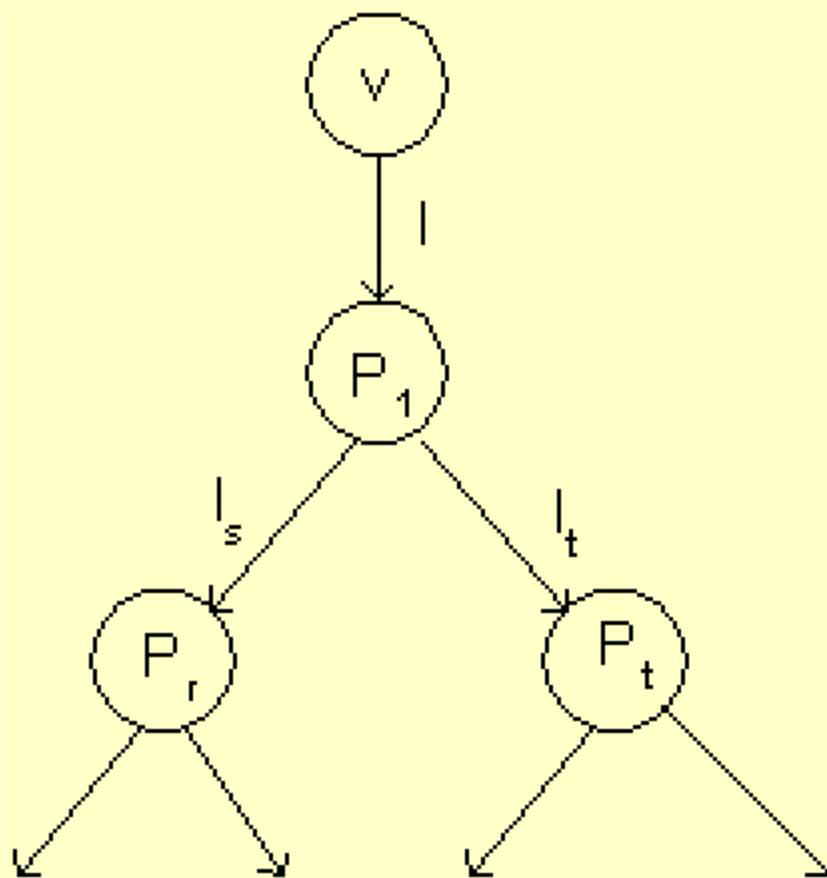


图9.11 光线树

- (2)为了计算整体环境向 P_1 点发射的镜面反射光和规则透射光,从 P_1 点出发向 \vec{r} 方向和 \vec{t} 方向发射两条光线求得它们与景物交点 P_r 和 P_t ,计算 P_r 和 P_t 点向 P_1 点辐射的光亮度 I_s 和 I_t ,代入Whitted模型,即可计算出总的亮度 I ;
- (3)计算 P_r 和 P_t 处光亮度时,除了由于光源直接照射引起的局部光照效果外,还要考虑周围环境对这两点产生的整体光照效果,于是需要连续从 P_r 和 P_t 点出发向相应 \vec{r} 方向和 \vec{t} 方向发射光线;
- (4)上述跟踪过程当跟踪的光线射出画面或跟踪深度达到给定层次时,应停止发射光线。也可通过跟踪光线对显示像素亮度 I 的贡献是否少于一阈值 k/G 来动态跟踪深度。

9.5.2 光线跟踪算法

算法描述

begin

for 每个像素e do

begin

确定通过视点V和像素e的光线R;

ray_tracing(R, I, 1);

置e的光亮度为I;

end

end

procedure ray_tracing(R,I,A)

/*R—跟踪光线,I—跟踪光线光亮度; A—I对总光亮度的贡献系数*/

```

begin
  if  $A < k/G$  then  $I := 0$ 
  else
    begin
      R与景物求交,返回可见点 $P_1$ ;
      计算 $P_1$ 的局部光照明亮度 $I_c$ ;
      若 $P_1$ 所在表面为光滑镜面,确定 $P_1$ 的镜面反射光线 $R_r$ ;
      ray_tracing( $R_r, I_s, k_s A$ );
      若 $P_1$ 所在表面为透明面,确定 $P_1$ 的规则透射光线 $R_t$ ;
      ray_tracing( $R_t, I_t, k_t A$ );
       $I = I_c + k_s I_s + k_t I_t$ ;
    end
  end
end

```

- 光线跟踪算法评价

优点：能够模拟环境的镜面反射和规则透射，模拟整体软影和透明阴影等高度真实感效果。

局限性：计算量大。

9.5.3 光线跟踪求交加速算法

(1)包围盒技术(如图9.12)

- 场景分层次表示: 将场景中的所有表面按景物组成和景物间的相对位置分层次组织成一棵景物树。
- 包围盒: 指用几何形状相对简单的封闭表面将复杂景物包裹起来, 若被跟踪的光线与包围盒不交, 则与它所含所有景物表面均无交。
- 包围盒与场景分层次表示技术结合使用, 大大减少求交工作量。

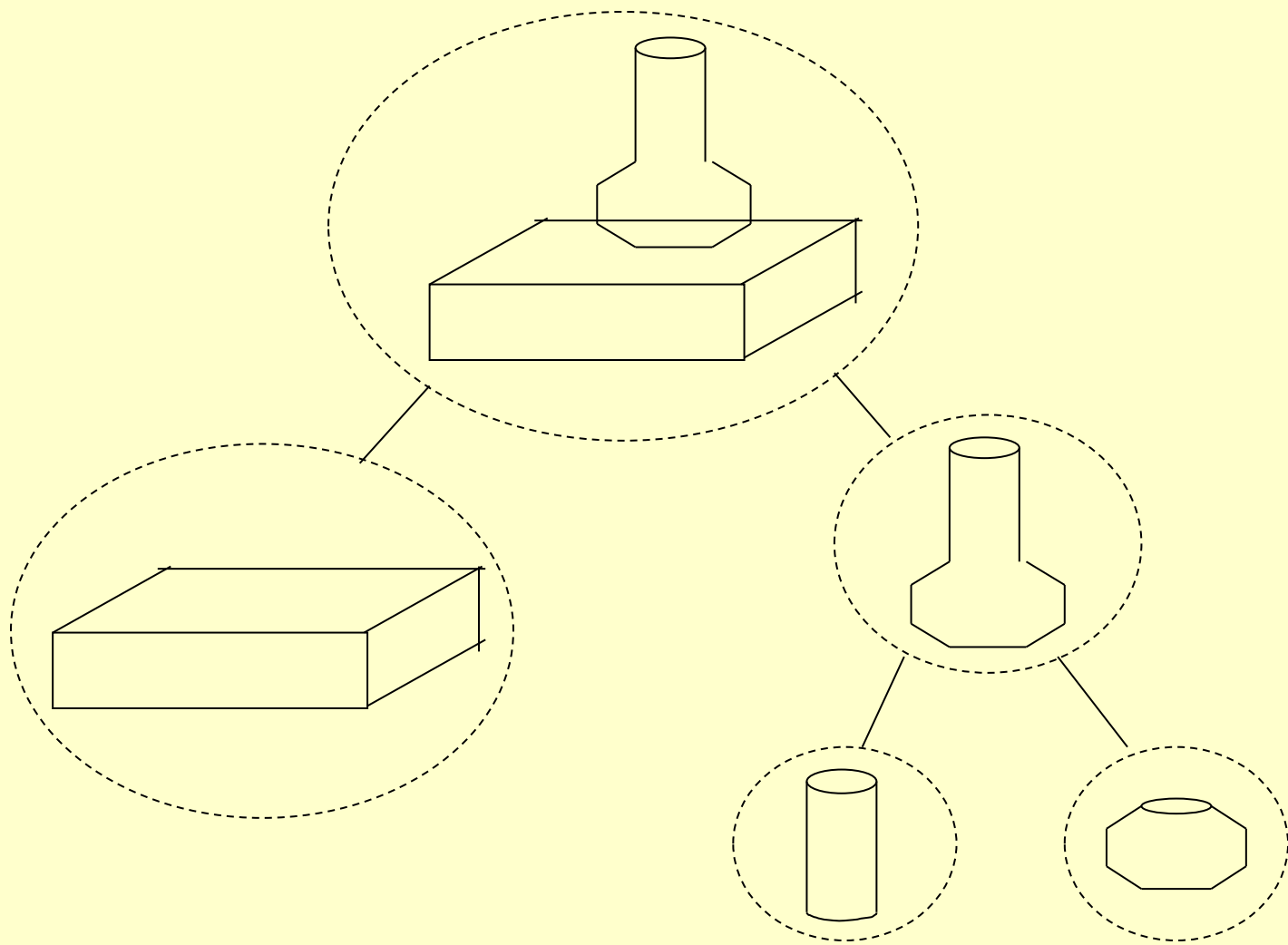


图9.12 景物树及其包围球

```
Procedure intersection (ray, scene)
  begin
    if (ray与scene的包围盒有交点) then
      begin
        if (scene是终结点) then
          begin
            ray与scene求交;
            若有交,则交点置入交点表中
          end
        else for (scene的所有儿子child_of_scene) do
            intersection (ray, child_of_scene)
          end
        end
      end
    end
  end
```

(2) 景物空间分割技术

- 此方法是将景物所在空间分割成网格单元, 利用相邻网格单元的空间连贯性, 求得它与景物的交点。
- 空间网络分割方式
 - a. 均匀分割成大小相同立方体网格单元
 - 优点: 用三维DDA算法, 计算简单;
 - 缺点: 较大存储需求。
 - b. 八叉树式空间分割
 - 优点: 减少空间网格单元数目;
 - 缺点: 算法复杂。

9.7 纹理(Texture)

一、基本概念

- 纹理：物体表面所呈现的表面细节。
如本质物体木纹、装饰图案、机器表面文字说明等等。
- 纹理分类
 - (1)颜色纹理：通过色彩或明暗度的变化体现出来的表面细节(如布纹)。
 - (2)几何纹理：由物体表面不规则细小凹凸造成(如桔子皱折表皮)。

- 纹理映射(texture mapping)

生成颜色纹理的方法。其过程是:

在一平面区域(纹理空间)上预先定义纹理图案;
然后建立物体表面的点与纹理空间的点之间的
对应关系(即映射)。

- 几何纹理生成方法

主要有法向扰动法、分形生成技术等。

二、纹理函数

- 最常用的纹理为二维光亮度函数, 可用数学模型定义或一幅平面图象表示。这种数学模型定义在单位正方形区域, 称纹理空间, 定义在纹理空间的函数即纹理函数。
- 实际上, 纹理函数往往采用一些特殊函数。

$$\text{例1: } g(u,v) = \begin{cases} b, \lfloor u \times 8 \rfloor + \lfloor v \times 8 \rfloor \text{ 为奇数} \\ a, \lfloor u \times 8 \rfloor + \lfloor v \times 8 \rfloor \text{ 为偶数} \end{cases}$$

其中 $0 < a < b < 1$, $\lfloor X \rfloor$ 表示小于 X 的最大整数。

$g(u,v)$ 用于模拟国际象棋棋盘上的黑白相间方格。

例2: $f(u,v)=A(\cos(pu)+\cos(qv))$

其中A为[0,1]上的随机变量, p,q 为频率系数, u,v 为函数参数。

$f(u,v)$ 常用于模拟粗布纹理。

三、纹理映射

- 离散法定义纹理

通过一个二维数组来定义各种图形、字符位图和数字化图象构成的纹理。

- 纹理映射

为把二维纹理图案映射到三维的物体表面, 必须建立物体空间坐标 (x, y, z) 与纹理空间坐标 (u, v) 之间的对应关系——物体表面参数化。

- (1) 如果物体表面是参数曲面, 对应是显然的;
- (2) 如果物体表面是平面多边形或隐函数定义二次曲面, 须求出映射关系。

- 二次曲面纹理映射

可用极坐标把曲面的隐式方程改写为参数方程。

例1：圆柱面 $x^2+y^2=1(0\leq z\leq 1)$ 改写为：

$$\begin{cases} x = \cos(2\pi u) \\ y = \sin(2\pi u) \\ z = v \end{cases} \begin{cases} 0 \leq u \leq 1 \\ 0 \leq v \leq 1 \end{cases}$$

给定 u, v , 可以求出 x, y, z 。

反之, 给定圆柱面上一点 (x, y, z) , 可用下列公式求出参数 u, v 。

$$(u, v) = \begin{cases} (y, z) & \text{当 } x = 0 \\ (x, z) & \text{当 } y = 0 \\ \left(\frac{\text{atan2}(y, x)}{2\pi}, z \right) & \text{其它情况} \end{cases}$$

例2：球面 $x^2+y^2+z^2=1$ 可改写为

$$\begin{cases} x = \cos(2\pi u) \cos(2\pi v) \\ y = \sin(2\pi u) \cos(2\pi v) \\ z = \sin(2\pi v) \end{cases} \begin{pmatrix} 0 \leq u \leq 1 \\ 0 \leq v \leq 1 \end{pmatrix}$$

其逆公式：

$$(u, v) = \begin{cases} (0,0) & \text{当}(x,y)=(0,0) \\ \left(\frac{\text{atan2}(y, x)}{2\pi}, \frac{\text{asin}(z)}{2\pi} \right) & \text{其它情况} \end{cases}$$

- 一般平面多边形表面的纹理映射
可以用透视变换建立点坐标与参数的关系
齐次坐标公式表示为：

$$(xw \ yw \ zw \ w) = (u \ v \ 1) \begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \end{bmatrix}$$

在此变换下, 所有坐标分量都具有

$$\begin{cases} x = (Au + Bv + c) / (Ju + Kv + L) \\ y = (Du + Ev + F) / (Ju + Kv + L) \\ z = (Gu + Hv + I) / (Ju + Kv + L) \end{cases}$$

形式。

四、几何纹理

- 通常采用法向扰动法, 通过对表面法向量进行扰动, 来产生凹凸不平效果。

定义 $F(u,v)$ 为纹理函数;

$P(u,v)$ 为理想光滑表面;

- 若物体表面每一点 $P(u,v)$ 都沿该点处法向量移动 $F(u,v)$ 个单位长, 则新的表面位置为:

$$P'(u,v) = P(u,v) + F(u,v) * N(u,v)$$

$N(u,v)$ 是 $P(u,v)$ 在 u, v 处的法向量。

(如图9.13所示)

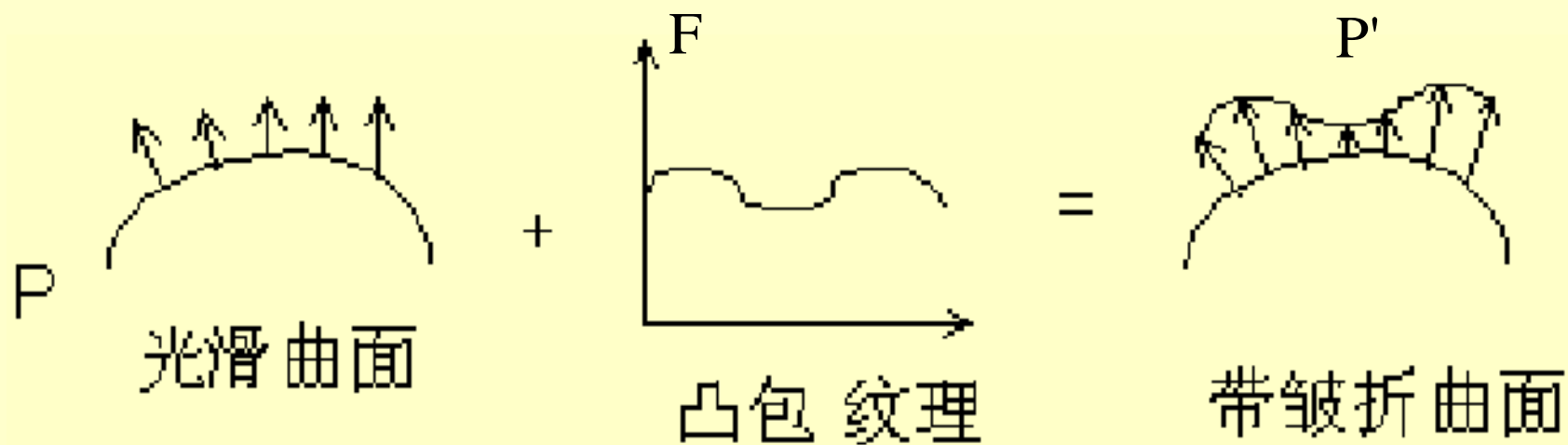


图9.13 法向扰动法

- 求新表面 P' 法向量 N'

N' 可通过两个偏导数求叉积求得: $N' = P_u' \times P_v'$

这里: $P_u' = dP'/du = d(P + FN)/du = P_u + F_u N + FN_u$

$$P_v' = dP'/dv = d(P + FN)/dv = P_v + F_v N + FN_v$$

- 由于粗糙表面凹凸高度相对于表面尺寸一般小得多, 故 F 相对于式中其它量很小, 可忽略不计。

$$P_u' \approx P_u + F_u N, P_v' \approx P_v + F_v N$$

所以新表面法向量近似为:

$$N' \approx (P_u + F_u N) \times (P_v + F_v N)$$

$$= P_u \times P_v + F_u (N \times P_v) + F_v (P_u \times N) + F_u F_v (N \times N)$$

$$= N + D$$

其中, $D = F_u(N \times P_v) + F_v(P_u \times N)$ 为扰动向量。 D 是 P 的切平面上的一个向量, 它与 N 之和改变了原曲面法向量, 即产生扰动。

N '单位化后用于明暗度计算, 产生纹理效果。

- 几何(凸包)纹理函数

其定义方法与颜色纹理函数定义方法相同。
几何(凸包)纹理由函数 P 和凸包位移函数定义。

第10章 OPEN GL及其实践

知 识 点: Open GL 基本功能和编程应用

教学目的: 让学生了解常用的图形库系统
Open GL使用方法

本章内容

10.1 概述

10.2 OPEN GL程序设计入门

10.3 基本几何元素

10.4 坐标变换

10.5 光照明模型处理

10.1 概述

- Open GL是在SGI公司推出的三维图形库GL的基础上开发而成的一个三维图形软件。
- 提供了很强的功能使三维对象用真实感的图形显示出来。可以显示线框图,也可绘制有明暗处理的图。
- 面向虚拟现实、可视化及动画的应用。

10.1.1 Open GL的特点

- 从程序开发人员的角度看, OpenGL实际上是一种3D程序接口(即常说的3D API),它是3D图形加速卡硬件和3D图形应用程序之间一座非常重要的桥梁。
- OpenGL的API集提供了物体描述、平移、旋转、缩放、光照、纹理、材质、像素、位图、文字、交互以及提高显示性能等方面的功能,基本涵盖了开发二、三维图形程序所需的各个方面。

- OpenGL具有两个主要特点

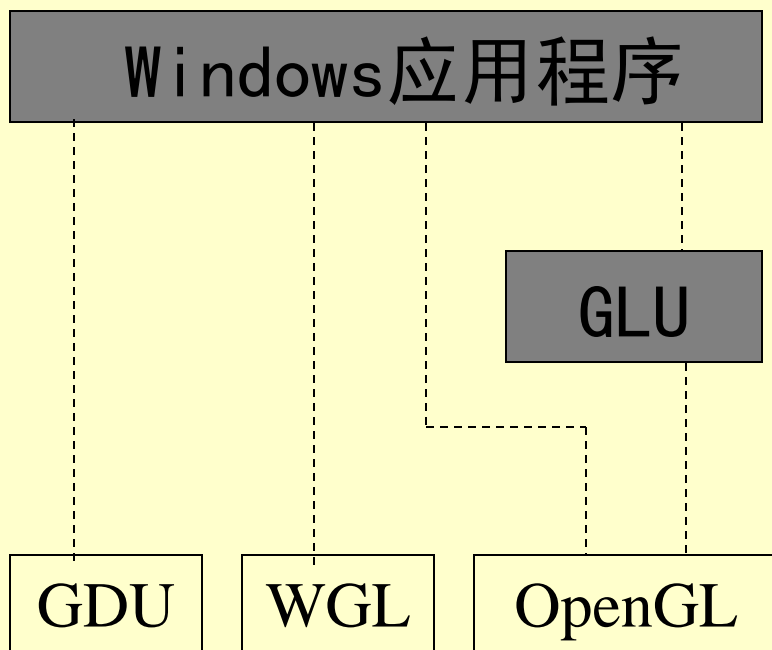
- (1) 它是与硬件无关的软件接口,可以在不同的平台之间进行移植。

- (2) 可以在客户机/服务器系统中工作,即具有网络功能,这一点对于制作大型3D图形、动画非常有用。

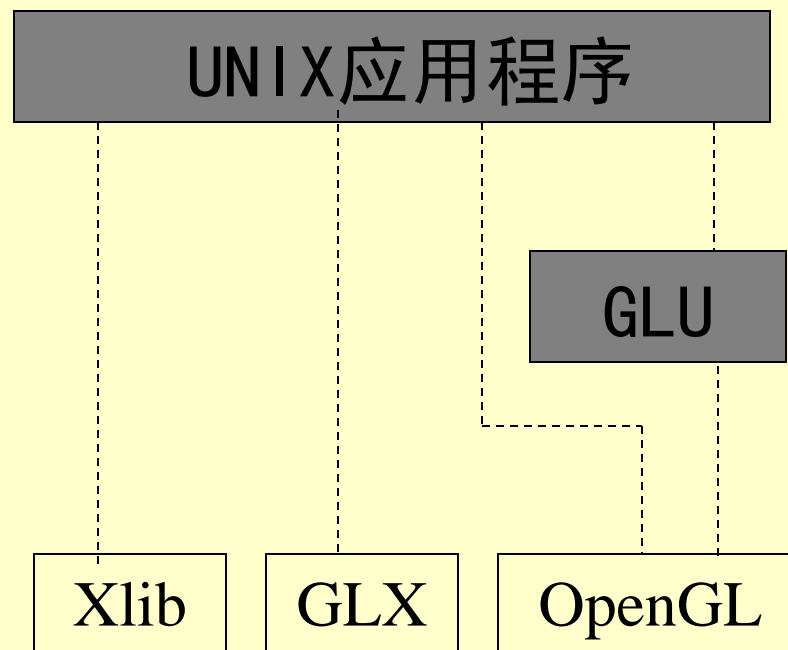
- 另外,在OpenGL的基础上还有Open Inventor、Cosmo3D、Optimizer等多种高级图形库,适应不同应用。

10.1.2 OpenGL的API结构

- 图10.1(a)是Win32平台上OpenGL API的结构简图;
(b)是UNIX平台上OpenGL API的结构简图。其中
- “OpenGL”表示OpenGL基本API, 功能包括:
物体描述、平移、旋转、光照、纹理、文字等;
 - “GLU”表示实用API, 功能包括: 绘制二次曲面、NURBS曲面、复杂多边形及纹理、矩阵管理等;
 - “WGL”是Win32为支持OpenGL而设计的编程接口;
 - “GLX”是UNIX系统支持OpenGL的编程接口



(a) Win32 OpenGL API



(b) UNIX OpenGL API

图10.1 OpenGL API结构图

10.1.3 OpenGL的工作流程

- 在屏幕上显示图象的主要步骤是：
 - 构造几何要素, 创建对象的数学描述;
 - 在3D空间中放置对象, 并选择适当的观察点;
 - 直接定义或由光照条件和贴图纹理给出对象的颜色;
 - 将对象的数学描述和颜色信息转换为屏幕上的像素.

10.2 OpenGL程序设计入门

- 基本语法
- OpenGL的工作方式
- 程序的基本结构
- 设置编程环境
- 一个简单的程序

10.2.1 基本语法

Open GL函数有几百个, 其语法规则如下.

(1) 前缀代表OpenGL命令的函数类型

- OpenGL基本库: 函数以gl开头, 共有115个.
如glColor3f();
- OpenGL实用库: 函数以glu开头, 提供高级功能调用.
如gluOrtho2D(), 定义二维图象正投影.
- OpenGL辅助库: 函数以glut开头, 方便简化编程.
如glutInitDisplayMode(), 设置窗口的特性.
- Windows专用函数库: 函数以wgl开头.
- X-Windows专用函数库: 函数以glx开头.
- 常数: 以GL_开头, 均用大写字母, 并用下划线与关键词分开, 如GL_LINE_LOOP, 闭合连线.

(2)后缀表明OpenGL命令的数据类型

后缀	类型	对应C变量
b	8bit int	signed char
s	16bit int	short
i	32bit int	long
f	32bit float	float
d	64bit float	double
ub	8bit unsigned int	unsigned char
us	16bit unsigned int	unsigned short
ui	32bit unsigned int	unsigned long

(3) 掐头去尾, 中间的关键字就是OpenGL函数的功能.

例如, `glVertex2i(100, 200)`表明
是OpenGL的基本函数(`gl-`),
是绘点的函数(`-Vertex-`),
是两个整型参数(`-2i`)。

10.2.2 OpenGL的工作方式

- OpenGL的工作方式是一种状态机, 可以进行各种状态或模式设置, 它们在重新改变之前一直有效。
如颜色状态, 没改变前一直使用该颜色。
- 许多状态变量可以通过glEnable()、glDisable()这两个函数来设置成有效或无效状态。
- 可用glGetBooleanv(), glGetDouble(), glGetFloatv()和glGetIntegerv()等函数来获取某个状态变量的值。
- 使用glPushAttrib()和glPopAttrib()函数, 可以存储和恢复最近的状态变量的值。

10.2.3 程序基本结构

■ 第一部分 初始化部分

主要是设置一些OpenGL的状态开关, 如颜色模式(RGB/ ALPHA)、是否作光照处理、裁剪等。

■ 第二部分 设置观察坐标系下的取景模式和取景框位置及大小。

主要利用3个函数:

- void glViewport(left, top, right, bottom) 设置在屏幕上的窗口大小
- void glOrtho(left, right, bottom, top, near, far) 设置投影方式为正交投影(平行投影), 其取景体积是一个各面均为矩形的六面体

- void gluPerspective(fovy, aspect, zNear, zFar): 设置投影方式为透视投影, 其取景体积是一个截头锥体。

- 第三部分 这是OpenGL的主要部分, 使用OpenGL的库函数构造几何物体对象的数学描述。

包括点线面的位置和拓扑关系, 几何变换, 光照处理等等。

10.2.4 设置编程环境

- 首先, 安装Visual C++ 6.0。
(包含基本库和实用库)
- 从GLUT的Web站点
<http://reality.sgi.com/opengl/glut3/glut3.html>下载最新的GLUT版本源文件。
- 获得的源文件经编译可生成3个文件: glut32.dll、glut32.lib、 glut.h, 且自动将这些文件拷贝到相应目录中。(也可从别的网站下载这些GLUT文件)

- 若得到是上述3个文件, 须手动拷贝到相应目录下:
glut32.dll文件复制到Windows的系统目录下
(Win9X是system目录, 其它是system32目录)。
glut32.lib文件复制到VC的lib目录下, glut.h复制到
VC的include\GL目录下。
- 在Visual C++中创建一个空的Win32的控制台应用程序; 在Project/Settings对话框中的Link选项卡中的Object/library modules一栏中加入opengl32.lib, glu32.lib和glut32.lib。
- 接下来就可加入新的源文件进行编程调试了。

10.2.5 一个简单的程序

```
#include <windows.h>
#include <gl\glut.h>
void RenderScene()
{
    glClear(GL_COLOR_BUFFER_BIT); //清颜色缓冲区
    // --绘图命令函数--
    glFlush(); //绘图结果显示到屏幕上
}
void SetupRC()
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f); //设置背景色
}
```

```
void main()
{
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        //单缓存, RGB颜色模式
    glutCreateWindow(“Simple”); //创建窗口,名为Simple
    glutDisplayFunc(RenderScene);
        //设显示回调函数
    SetupRC();
        //自定义的初始化例程
    glutMainLoop();
        //让GLUT框架开始运行, 处理交互事件
}
```

上面例子显示一个标题为 “Simple” 的标准GUI窗口和一个清晰的蓝色背景。

其中, 主函数前几条语句调用了OpenGL 的 GLUT辅助库, 其作用是一些初始化的设置。

10.3 基本几何元素

- 绘图准备和结束
- 绘制OpenGL的基本几何元素
- 设置几何要素的属性
- 控制多边形绘制
- 法向量

10.3.1 绘图准备和结束

- 在开始绘制新图形前, 必须清除当前已有的图形信息内容, 以免影响绘图的效果, 可以用
 - `void glClearColor(red, green, blue, alpha);`
给定当前屏幕的背景设置颜色
 - `void glClear(mask);`
命令标志要清除的缓冲区(颜色、深度、累加、模板); 如 `GL_COLOR_BUFFER_BIT`
 - `void glColor *();`
设置当前颜色

- 在绘制完图形时, 需要使用两个函数来结束绘图并返回:
 - `void glFlush()` 强制OpenGL命令序列在有限的时间内完成执行;
 - `void glFinish()` 使完成前面发出的全部OpenGL命令的执行, 即等到完全实现全部命令的结束后才返回。

10.3.2 绘制OpenGL的基本几何元素

- OpenGL几何要素有点、线、多边形。光滑曲线、曲面都是由一系列线段、多边形近似得到的。
OpenGL不直接提供绘制曲线、曲面的命令。
- 描述几何要素就是按一定顺序给出几何要素的顶点。 `glVertex()`命令指定一个顶点,并在生成顶点后,把当前颜色、纹理坐标、法线等值赋给这个顶点。该函数只能在`glBegin()`与`glEnd()`之间调用才有意义,它们分别标志几何要素定义的开始和结束。

■ 绘制多边形

```
glBegin(GL_POLYGON);  
    glVertex2f(0.0,0.0);  
    glVertex2f(0.0,3.0);  
    glVertex2f(3.0,3.0);  
    glVertex2f(4.0,1.5);  
    glVertex2f(3.0,0.0);  
glEnd();
```

注：OpenGL点是三维的, 用户设定二维坐标时, Z自动置0。

10.3.3 设置几何要素属性

■ 点属性

- void glPointSize(Glfloat size);

以像素为单位设置点绘制的宽度(size)。缺省1.0

■ 线属性

- void glLineWidth(Glfloat width);

以像素为单位设置线绘制的宽度。

- void glLineStipple(Glint factor, Glushort pattern);

指定点画模式(线型), pattern=0x3F07: 0011 1111 0000 0111, 0-不画, 1-画; factor为线型因子。

在定义线型后, 必须用glEnable(GL_LINE_STIPPLE)命令激活线型。

■ 多边形属性

通常多边形用实模式填充, 但也可以利用下面命令指定某种点画模式(图案)来填充:

```
void glPolygonStipple(const Glubyte * mask);
```

该函数指定多边形点画模式。

mask 指定 32×32 点画模式(位图)的指针, 值为1时绘, 值为0时不绘。

指定多边形点画模式需要利用

`glEnable(GL_POLYGON_STIPPLE)`激活。

10.3.4 控制多边形绘制

■ 选择绘制多边形的方式

利用下面函数可以选择绘制多边形的方式。

```
void glPolygonMode(Glenum face, Glenum mode);
```

face 控制多边形的正面和反面的绘图方式, 有3种绘制方式: GL_FRONT_AND_BACK, GL_FRONT, GL_BACK.

mode 控制绘点、线框或填充多边形, 有GL_POINT, GL_LINE和GL_FILL。

■ 反转多边形面

在OpenGL中, 每个多边形被认为是由两个面组成的: 正面和反面。

`glFrontFace(Glenum mode);`

该函数定义多边形的正面方向,

`mode = GL_CW` (顺时针方向为正面)

`= GL_CCW` (逆时针方向为正面, 缺省值)

■ 法向量

利用下面函数设置当前法向量：

```
void glNormal3{bsidf}(TYPE nx, TYPE ny,  
TYPE nz);
```

nx, *ny*, *nz*指定法向量的 x , y 和 z 坐标。法向量的缺省值为(0, 0, 1)。

利用命令glEnable(GL_NORMALIZE)激活自动规格化法向量, 就会自动规格化glNormal *()所指定的法向量。

■ 指定的法向量, 绘制多边形

```
glBegin(GL_POLYGON); // n0, n1, n2, n3, n4已定义
    glNormal3fv(n0);
    glVertex2f(0.0, 0.0);
    glNormal3fv(n1);
    glVertex2f(0.0, 3.0);
    glNormal3fv(n2);
    glVertex2f(3.0, 3.0);
    glNormal3fv(n3);
    glVertex2f(4.0, 1.5);
    glNormal3fv(n4);
    glVertex2f(3.0, 0.0);
glEnd();
```

10.4 坐标变换

- 概述
- 常用的变换函数
- 视图造型变换
- 投影变换
- 视口变换

10.4.1 概述

- OpenGL的坐标变换的步骤分为如下4步：
 - 第1步 指定视图和造型变换, 两者结合称为视图造型变换。 $WC == \gg VC$
 - 第2步 指定投影变换, 使对象从视觉坐标变换到裁剪坐标。 $VC == \gg CC$
 - 第3步 通过对坐标值除以w, 执行透视除法, 变换到规格化设备坐标 $CC == \gg NDC$
 - 第4步 通过视口变换将坐标变换到窗口坐标。
 $NDC == \gg WinC$

10.4.2 常用的变换函数

- 变换和矩阵是相关的.
- `void glMatrixMode(Glenum mode)`

该函数指定哪一种矩阵为当前矩阵。

*mode*为当前矩阵的类型, 可以有

GL_MODELVIEW --- 后继的操作均在视图造型变换范围内(缺省值)

GL_PROJECTION --- 后继的操作均在投影变换范围内

GL_TEXTURE --- 后继的操作在纹理映射范围内

- `void glLoadIdentity(void)`

该函数设置单位矩阵为当前矩阵。

- `void glMultMatrix{fd}(const TYPE * m)`

该函数用任意 4×4 矩阵替代当前矩阵, m 指定任意矩阵的16个元素。

参数 m 为指定矩阵 M , M 由16个值的向量(m_1, m_2, \dots, m_{16})组成。

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

10.4.3 视图造型变换

- 视图造型变换过程就是一个将顶点坐标从世界坐标变换到观察坐标的过程。
- 变换的顺序
当执行变换A和B时，如果按不同顺序执行时，结果往往会大不相同。

■ 造型变换

– `void glTranslate*(TYPE x,TYPE y,TYPE z);`

该函数以平移矩阵乘当前矩阵,

x, y, z 指定沿世界坐标系 x, y, z 轴的平移量。

– `void glRotate*(TYPE angle,TYPE x,TYPE y,TYPE z);`

该函数以旋转矩阵乘当前矩阵。*Angle*指定旋转的角度; x, y, z 指定旋转轴向量的3个分量。

– `void glScale*(TYPE x, TYPE y, TYPE z);`

该函数以缩放矩阵乘当前矩阵,

x, y, z 指定沿 x, y 和 z 轴的比例因子。

■ 视图变换

视图变换改变视点的位置和方向,也就是改变观察坐标系。作视图变换有以下几种方法:

(1) 利用一个或几个造型变换命令(即 `glTranslate*()`和`glRotate*()`)。

(2)利用实用库函数`gluLookAt()`设置观察坐标系

```
void gluLookAt(Gldouble eyex, Gldouble eyey, Gldouble  
eyez, Gldouble centerx, Gldouble centery, Gldouble  
centerz, Gldouble upx, Gldouble upy, Gldouble upz)
```

eyex, *eyey*, *eyez* --- 指定视点的位置;

centerx, *centery*, *centerz* --- 指定参考点的位置;

upx, *upy*, *upz* --- 指定视点向上方向。

10.4.4 投影变换

■ 透视投影

其取景体是一个截头锥体, 可用`glFrustum()`函数定义这个截头锥体:

```
void glFrustum(Gldouble left, Gldouble right, Gldouble  
bottom, Gldouble top, Gldouble near, Gldouble far);
```

该函数以透视矩阵乘当前矩阵。

left, *right*指定左右垂直裁剪面的坐标;

bottom, *top*指定底和顶水平裁剪面的坐标;

near, *far*指定近和远深度裁剪面的距离。

■ 正交投影

其取景体积是一各面均为矩形的6面体, 可用 `glOrtho()` 函数定义这个6面体:

```
void glOrtho(Gldouble left, Gldouble right,  
Gldouble bottom, Gldouble top, Gldouble near,  
Gldouble far);
```

该函数以正交投影矩阵乘当前矩阵。

10.4.4 视口变换

- 视口就是窗口中矩形绘图区。用窗口管理器在屏幕上打开一个窗口时, 已经自动地把视口区设为整个窗口的大小。
- 可以用glViewport命令设定一个较小的绘图区:

```
void glViewport(Glint x, Glint y, Glsize width, Glsize height);
```

该函数设置视口的大小。

x, y 指定视口矩形的左下角坐标(以像素为单位),

缺省值为(0, 0);

$width, height$ 分别指定视口的宽和高。

10.5 光照明模型处理

- OpenGL光照明模型的基本概念
- 光源定义
- 光照模式
- 材质属性

10.5.1 OpenGL光照模型的基本概念

- 光线由4个部分构成：发射光、泛光、漫反射光、镜面发射光, 这4个成分单独计算, 然后再累加起来。
- 材料颜色取决于反射的红绿蓝光的百分数。与光源的特性相似, 材料也有泛光、扩散、反射颜色。

10.5.2 光源定义

- 定义光源主要就是定义它的各种特性。可用下面函数定义：

```
void glLight{if}[v](GLenum light, GLenum pname, TYPE  
param);
```

*light*用形式为GL_LIGHT*i*的符号常数表示, $0 \leq i \leq 8$;

*pname*设置光源的特性;

*param*设置*pname*特性的值.

光源定义后, 须调用glEnable()打开该光源。

■ 其中,可选的特性参数(*pname*)有

– 颜色

GL_AMBIENT用于指定环境泛光的强度,
(0,0,0,1);

GL_DIFFUSE用于指定漫反射的强度, (1,1,1,1);

GL_SPECULAR用于指定镜面反射光的强度,
(1,1,1,1)。

– 位置

GL_POSITION定义光源位置坐标, 缺省为齐次
坐标(0,0,1,0).

– 衰减

光线的强度随光源距离的增加而减少。

对于位置光源，OpenGL使用下面的衰减因

子来衰减光线强度：

$$\frac{1}{k_c + k_e d + k_q d^2}$$

其中：d为光源位置到物体顶点之间的距离。

k_c 为常数衰减因子

(GL_CONSTANT_ATTENUATION), 1.0

k_e 为线性衰减因子

(GL_LINEAR_ATTENUATION), 0.0

k_q 为二次衰减因子

(GL_QUADRATIC_ATTENUATION), 0.0

10.5.3 光照模式

■ 全局环境光

使用下面函数来指定全局环境光的强度：

```
glLightModelv(GL_LIGHT_MODEL_AMBIENT,glfloat param)
```

■ 观察点性质

定义观察点性质的函数是：

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,glboolean param);
```

当参数 $param$ 取GL_TRUE时, 观察点为本地观察点；取GL_FALSE时,为无穷观察点。

■ 光照面

可以使用下面函数打开或关闭双面光照：

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE,  
Glboolean param);
```

当参数`param`取`GL_TRUE`时, 无论是正面的多边形还是反面的多边形都受光照的影响；

取`GL_FALSE`时, 只为正面的多边形设置光照。

10.5.4 材质属性

- 设置材质属性的函数是：

```
void glMaterial{if}(GLenum face, GLenum  
pname, TYPE param);
```

参数`face`可以是GL_FRONT、GL_BACK或GL_FRONT_AND_BACK, 指定当前材质作用于物体的哪一个方面。

被设置的材质的属性由`pname`标识, 对应于该属性的值由`param`给出。

■ 其中,属性可以是

– 漫反射和泛反射

GL_AMBIENT和GL_DIFFUSE属性定义了物体对漫反射和泛射光的反射率。

– 镜面反射

GL_SPECULAR属性设置镜面反射率。

GL_SHININESS属性可控制亮斑的大小和亮度,这时param的取值范围为0.0至128.0,这个值越大,亮斑的尺寸越小且亮度越高。

– 发射光

用GL_EMISSION作为pname参数的值调用glMaterial*()函数,指定一种RGBA颜色,使得物体看起来像发出某种颜色的光。

10.6 综合举例

见程序。

综合练习

试设计一个室内三维环境, 并利用OPEN GL展示它的三维效果。要求:

- (1) 包含基本的实体元素: 球、多面体、锥体、柱体、曲面等;
- (2) 有全局光照效果和纹理功能;
- (3) 程序具有交互功能.

提交设计报告和源程序.