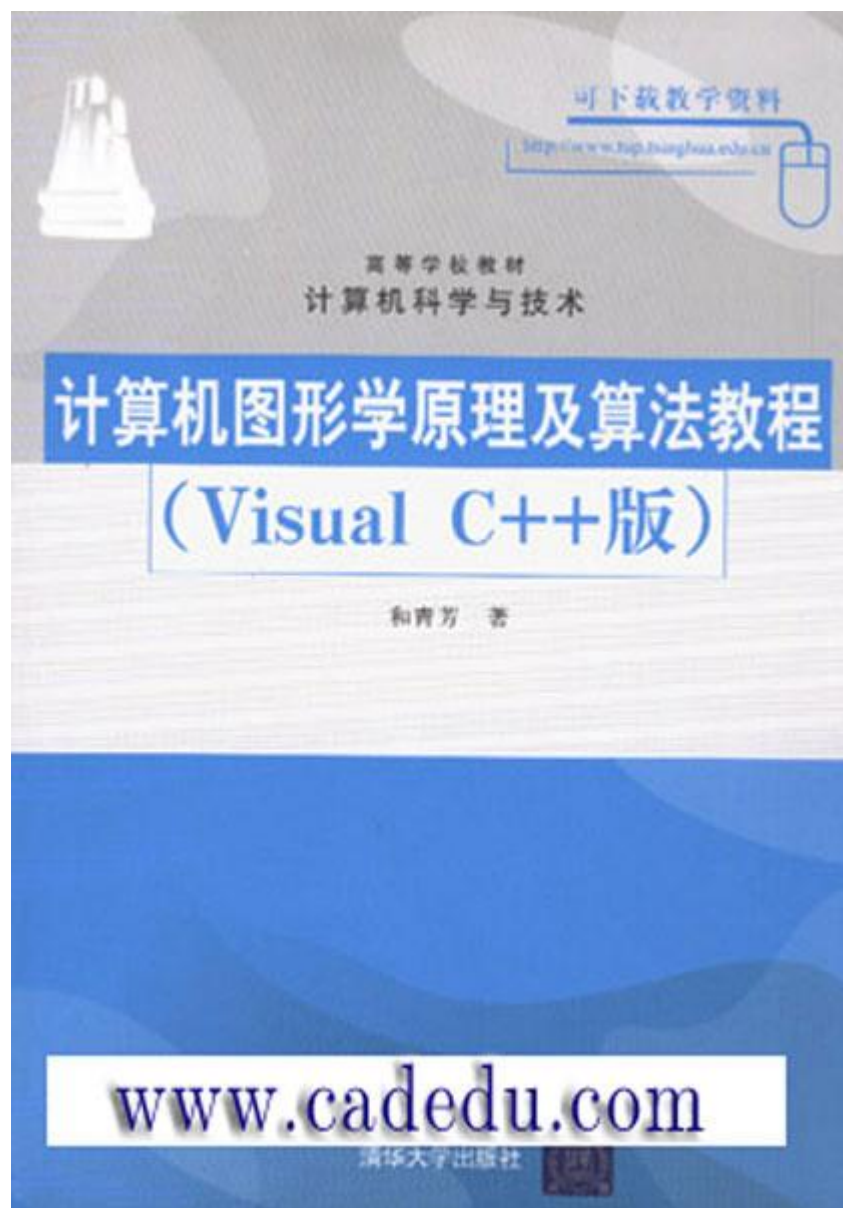


《计算机图形学原理及算法教程》(Visual C++版) 和青芳 清华大学出版社出版



内容提要

本书在系统介绍计算机图形学算法及原理基础上,利用 Visual C++开发环境,编制相应的应用程序,较全面具体地把计算机图形学理论与计算机绘图的实践结合了起来,在计算机理论与交互式图形软件设计之间架起一座桥梁,使学习者在掌握理论和实用知识两方面均感到应用自如。随书光盘提供案例的全部源程序代码,供读者选用,读者可直接在 Visual C++中打开各程序包进行学习或在此基础上修改开发自己的绘图程序,特别适合学习者上机仿效练习,其中的简单 CAD 系统开发实例为读者开发实际图形程序搭建了基本平台。

本书可作为各大专院校计算机图形学教材、上机教材或工程技术人员自学计算机图形学和 VC++的参考书,亦可作为计算机图形学教师理论教学参考和课程设计的素材。CAD 教育网(www.CADedu.com)提供学习支持,为教师免费提供电子教案。

《计算机图形学原理及算法教程》(Visual C++版) 和青芳 清华大学出版社出版

目录

第 1 章 基本图形的生成

1.1 直线

1.1.1 生成直线的 DDA 算法

1.1.2 生成直线的中点算法

1.1.2 生成直线的 Bresenham 算法

1.1.3 程序设计案例

1.2 圆

1.2.1 DDA 算法

1.2.2 Bresenham 算法

1.2.3 程序设计案例

1.3 椭圆

1.4 区域填充

1.4.1 扫描线填充

1.4.2 种子填充

1.4.3 程序设计案例

1.5 字符的生成

1.6 图形裁剪

1.6.1 线裁剪

1.6.2 多边形裁剪

1.6.3 字符裁剪

1.6.4 裁剪程序设计案例

1.7 Visual C++中基本绘图函数

1.8 课后练习

第 2 章 二维图形

2.1 用户坐标到屏幕坐标的变换

2.1.1 窗口到视口的变换内容

2.1.2 窗口区到视图区的坐标变换

2.2 几何变换

2.2.1 基本变换

2.2.2 复合变换

2.2.3 几何变换程序设计案例

2.4 平面曲线图

2.4.1 正叶线

2.4.2 正叶线蝴蝶结

2.5 平面曲线程序设计案例

2.6 课后练习

第 3 章 图形交互技术

3.1 用户接口设计

3.2 逻辑输入设备与输入处理

3.2.1 逻辑输入设备

3.2.2 输入模式

3.3 交互式绘图技术

3.4 交互技术程序设计案例

3.5 课后练习

第4章 简单 CAD 绘图系统开发实例

4.1 计算机图形学绘图基础

4.1.1 Visual C++开发系统基本绘图知识

4.1.2 计算机图形学会图系统设计基本原则

4.1.3 图形程序设计步骤

4.1.4 在 Visual C++集成开发环境下程序的调试

4.1.5 计算机程序结构设计基础

4.1.6 绘图程序设计基本方法

4.1.6.1 图形层次结构和程序模块结构

4.1.6.2 面向对象程序设计

4.1.6.3 绘图子程序和主程序

4.1.6.4 编程绘图方法

4.2 图形的数据结构

4.2.1 图形信息的分类

4.2.2 图形数据结构

4.2.3 计算机对数据的管理—数据文件

4.2.4 图形数据的存储状态

4.2.5 动态文件数据结构的组织原则

4.2.6 简单 CAD 绘图系统编程实例中的数据结构

4.2.6.1 图形元素基类的组织

4.2.6.2 组织图形类系统文档

4.2.6.3 增加图形元素

4.2.6.4 实现各类图形的绘制

4.2.6.5 保存图形数据到文档

4.3 简单 CAD 绘图系统功能简介

4.3.1 简单 CAD 绘图系统运行界面

4.3.2 简单 CAD 绘图系统功能

第5章 三维图形

5.1 三维图形几何变换矩阵

5.2 三维图形基本变换

5.2.1 平移变换矩阵

5.2.2 比例变换矩阵

5.2.3 旋转变换矩阵

5.2.4 对称变换

5.2.5 错切变换

5.3 图形的投影变换

5.3.1 投影变换分类

5.3.2 平行投影

5.3.2.1 正平行投影（三视图）

5.3.2.2 斜平行投影

5.3.2.3 透视投影

5.4 三维变换程序设计案例

5.5 课后练习

第6章 曲线与曲面

6.1 曲线曲面参数表示的基础知识

6.1.1 非参数表示和参数表示

6.1.2 参数表示的基本特征

6.1.3 曲线段之间的连续性

6.1.4 曲线曲面设计中的几个概念

6.2 常用参数曲线

6.2.1 一般规则空间曲线

6.2.2 Bezier 曲线

6.2.3 B 样条曲线

6.3 参数曲面

6.3.1 函数式曲面

6.3.2 旋转曲面

6.4 常用曲面

6.4.1 双曲线曲面

6.4.2 Bezier 曲面

6.4.3 B 样条曲面

6.5 曲面与曲线程序设计案例

6.6 课后练习

第7章 几何造型

7.1 实体的表示模型

7.1.1 形体的边界表示模型

7.1.2 构造表示

7.1.3. 边界表示

7.1.3.1 欧拉操作

7.1.3.2 集合运算

7.2 求交分类

7.2.1 求交分类

7.2.2 基本的求交算法

7.2.2.1 线与线的求交计算

7.2.2.2 线与面的求交计算

7.2.2.3 曲面与曲面的求交

7.3 图形相交-相切程序设计案例

7.4 非传统造型技术

7.4.1 基本概念

7.4.2 分形造型对模型的基本要求

7.4.3 分形造型的常用模型

7.4.4 分数维图形应用

7.5 分形造型程序设计案例

7.6 课后练习

第 8 章 消隐技术

8.1 线消隐

8.2 面消隐

8.2.1 区域排序算法

8.2.2 深度缓存(Z-buffer)算法

8.2.3 扫描线算法

8.3 消隐

8.4 消隐技术程序设计案例

8.5 课后练习

第 9 章 真实感图形绘制

9.1 颜色模型

9.1.1 CIE 色度图

9.1.2 常用的颜色模型

9.2 简单光照模型

9.2.1 Phong 光照模型

9.3 局部光照模型

9.3.1 局部光照模型

9.4 光透射模型

9.4.1 透明效果的简单模型

9.4.2 Whitted 光透射模型

9.4.3 Hall 光透射模型

9.4.4 简单光反射透射模型

9.5 纹理及纹理映射

9.5.1 纹理的概述

9.6 整体光照模型

9.6.1 光线跟踪算法

9.6.2 辐射度方法

9.7 真实感图形学程序设计案例

9.8 课后练习

第 10 章 计算机动画

10.1 计算机动画概述

10.2 计算机动画的应用领域

10.3 计算机动画的分类和原理

10.4 目前计算机动画面临的问题

10.5 计算机动画程序设计案例

10.5.1 帧动画

10.5.1 实时动画

10.6 练习题

参考文献:

第一章 基本图形的生成

计算机图形学已成为计算机领域发展最快的领域，同时计算机图形软件也相应得到快速发展。计算机绘图显示有屏幕显示、打印机打印屏幕上的图样和绘图机输出图样等方式，其中用屏幕显示图样是计算机绘图的重要内容。

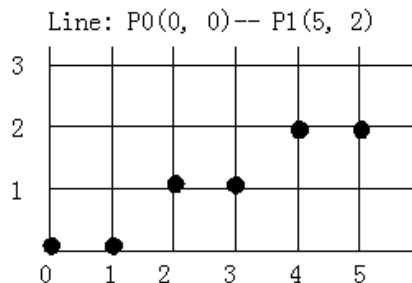
计算机上常见的显示器为光栅图形显示器，光栅图形显示器可以看作像

素的矩阵。像素是组成图形的基本元素，一般称为“点”。通过点亮一些像素，灭掉另一些像素，即在屏幕上产生图形。在光栅显示器上显示任何一种图形，在显示器的相应像素点上画上所需颜色的像素点，即具有一种或多种颜色的像素集合构成图形。确定最佳逼近图形的像素集合，并用指定属性写像素的过程称为图形的扫描转换或光栅化。对于一维图形，在不考虑线宽时，用一个像素宽的直、曲线来显示图形。二维图形的光栅化必须确定区域对应的像素集，并用指定的属性或图案进行显示，即区域填充。

复杂的图形系统，都是由一些最基本的图形元素组成的。利用计算机编制图形软件时，编制基本图形元素是相当重要的，也是必需的。点是基本图

形，本章主要讲述如何在指定的输出设备（如光栅图形显示器）上利用点构造其它基本二维几何图形（点、直线、圆、椭圆、多边形域、字符串等）的算法与原理，并利用 Visual C++ 编程实现这些算法。**1.1 直线**

数学上，理想的直线是由无数个点构成的集合，没有宽度。计算机绘制直线是在显示器所给定的有限个像素组成的矩阵中，确定最佳逼近该直线的一组像素，并且按扫描线顺序，对这些像素进行写操作，实现



显示器绘制直线，即通常所说直线的扫描转换，或称直线光栅化。

由于一图形中可能包含成千上万条直线，所以要求绘制直线的算法应尽可能的快。本节我们介绍一个像素宽直线的常用算法：数值微分法（**DDA**）、中点画线法、**Bresenham** 算法。

1.1.1 DDA (数值微分)算法

一. DDA 算法原理

如图 **1-1** 所示，已知过端点 $p_0(x_0, y_0), p_1(x_1, y_1)$ 的直线段 p_0p_1 ；直线斜率为

$k = \frac{y_1 - y_0}{x_1 - x_0}$ ，从 x 的左端点 x_0 开始，向 x 右端点步进画线，步长=**1**(个像素)，

计算相应的 y 坐标 $y = kx + B$ ；取像素点 $(x, \text{round}(y))$ 作为当前点的坐标。计算 $y_{i+1} = kx_{i+1} + B = kx_1 + B + k\Delta x = y_1 + k\Delta x$ 当 $\Delta x = 1$, $y_{i+1} = y_i + k$ 即：当 x 每递增 1, y 递增 k (即直线斜率)；

注意：上述分析的算法仅适用于 $0 < k \leq 1$ 的情形。在这种情况下， x 每增加 1, y 最多增加 1。当 $k \geq 1$ 时，必须把 x, y 地位互换， y 每增加 1, x 相应增加 $1/k$ (请参阅后面的 Visual C++ 程序)。

1.1.2 生成直线的中点画线

法

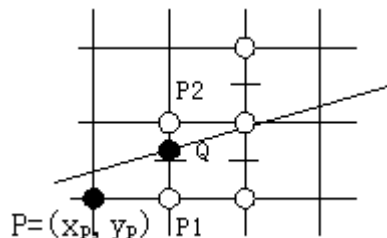


图 1-2 中点画线法每步迭代涉及的像素和中点示意图

中点画线法的基本原理，如图 1-2 所示，在画直线段的过程中，当前像素点为 p ，下一个像素点有两种选择，点 p_1 或 p_2 。M 为 p_1 与 p_2 中点，Q 为理想直线与 $x=x_p+1$ 垂线的交点。当 M 在 Q 的下方，则 P_2 应为下一个像素点；M 在 Q 的上方，应取 P_1 为下一点。

中点画线法的实现。令直线段 $L(p_0(x_0, y_0), p_1(x_1, y_1))$ ，其方程式 $F(x, y) = ax + by + c = 0$ 。

其中， $a = y_0 - y_1$ ， $b = x_1 - x_0$ ， $c = x_0 y_1 - x_1 y_0$ ；点与 L 的关系：

在直线上： $F(x, y) = 0$ ；

在直线上方： $F(x, y) > 0$;

在直线下方： $F(x, y) < 0$;

把 M 代入 $F(x, y)$ 判断 F 的符号，可知 Q 点在中点 M 的上方还是下方。为此构造判别式： $d = F(M) = F(x_p + 1, y_p + 0.5) = a(x_p + 1) + b(y_p + 0.5) + c$

当 $d < 0$ ， $L(Q \text{ 点})$ 在 M 上方，取 P_2 为下一个像素；

当 $d > 0$ ， $L(Q \text{ 点})$ 在 M 下方，取 P_1 为下一个像素；

当 $d = 0$ ，选 P_1 或 P_2 均可，取 P_1 为下一个像素；

其中 d 是 x_p, y_p 的线性函数。

1.1.3 Bresenham 算法

一. Bresenham 算法原理

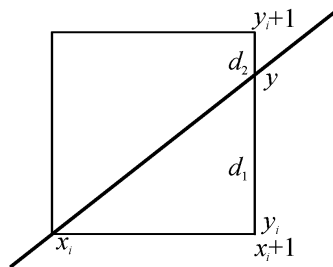


图 1-3 第一象限直线光栅化 Bresenham 算法

Bresenham 算法是计算机图形

学领域使用最广泛的直线扫描转换算法。由误差项符号决定下一个像素取右边点还是右上方点。

设直线从起点 (x_1, y_1) 到终点 (x_2, y_2) 。直线可表示为方程 $y = mx + b$,

其中 $b = y_1 - m \cdot x_1$, $m = (y_2 - y_1) / (x_2 - x_1) = dy/dx$; 此处的讨论直线方向限

于 **1a** 象限(图 1-3)，当直线光栅化时，**x** 每次都增加 **1** 个单元，设 **x** 像素为 (**x_i**, **y_i**)。下一个像素的列坐标为 **x_i+1**，行坐标为 **y_i**，或者递增 **1** 为 **y_i+1**，由 **y** 与 **y_i** 及 **y_i+1** 的距离 **d₁** 及 **d₂** 的大小而定。计算公式为

$$y=m(x_{i+1})+ b \quad (1.1)$$

$$d_1 = y - y_i \quad (1.2)$$

$$d_2 = y_{i+1} - y \quad (1.3)$$

如果 $d_1 - d_2 > 0$, 则 $y_{i+1} = y_i + 1$, 否则 $y_{i+1} = y_i$ 。

式(1.1)、(1.2)、(1.3)代入 $d_1 - d_2$, 再用 dx 乘等式两边, 并以 $P_i = (d_1 - d_2) dx$ 代入上述等式, 得

$$P_i = 2x_i dy - 2y_i dx + 2dy + (2b - 1) dx \quad (1.4)$$

$d_1 - d_2$ 是用以判断符号的误差。由于在 $1a$ 象限, dx 总大于 0, 所以 P_i 仍旧可以用作判断符号的误差。 P_{i+1} 为

$$P_{i+1} = P_i + 2dy - 2(y_{i+1} - y_i) dx \quad (1.5)$$

求误差的初值 P_1 ，可将 x_1 、 y_1 和 b 代入式(2.4)中的 x_i 、 y_i 而得到

$$P_1 = 2dy - dx$$

综述上面的推导，第 1a 象限内的直线 Bresenham 算法思想如下：

- I 1. 画点(x_1 , y_1), $dx=x_2-x_1$, $dy=y_2-y_1$, 计算误差初值 $P_1=2dy-dx$,
 $i=1$;
- I 2. 求直线的下一点位置 $x_{i+1} = x_i + 1$ 如果 $P_i \geq 0$, 则
 $y_{i+1}=y_i+1$, 否则 $y_{i+1}=y_i$;
- I 3. 画点(x_{i+1} , y_{i+1});
- I 4. 求下一个误差 P_{i+1} , 如果 $P_i \geq 0$, 则 $P_{i+1}=P_i+2dy-2dx$, 否则

$P_{j+1} = P_j + 2dy;$

l 5. $i = i + 1$; 如果 $i < dx + 1$ 则转步骤 2; 否则结束操作。

1.1.4 程序设计

一、程序设计功能说明

为编程实现上述算法, 本程序利用最基本的绘制元素(如点、直线等), 绘制图形。如图 1-4 所示, 为程序运行主界面, 通过单击菜单及下拉菜单的各功能项分别完各种对应算法的图形绘制。



图 1-4

二、创建“工程名称为基本图形的生成”单文档应用程序框架

(1) 启动 VC，选择【文件】|【新建】菜单命令，并在弹出的新建对话框中单击【工程】标签。

(2) 选择【MFC AppWizard(exe)】，在【工程名称】编辑框中输入“基

本图形的生成”作为工程名称，单击【确定】按钮，出现 Step 1 对话框。

(3) 选择【单个文档】选项，单击【下一个】出现 Step 2 对话框。

(4) 接受缺省选项，单击【下一个】，在一下出现的 Step 3- Step 5 对话框，接受缺省选项，单击【下一个】按钮。

(5) 在 Step 6 对话框中单击【完成】按钮，即完成“基本图形的生成”应用程序的所有选项，随后出现工程信息对话框（记录以上步骤各选项选择情况），如图 1-5 所示，单击【确定】完成应用程序框架的创建。



图 1-5

2. 编辑菜单资源

设计如图 1-4 所示的菜单项。在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 1.1 表中的定义编辑菜单资源。此时 VC 已自动建好的程序框架如图 1-5 所示。

表 1.1 菜单资源表

菜单标题	菜单项标题	标示符 ID
直线	DDA 算法生成直线	ID_DDALINE
	Bresenham 算法生成 直线	ID_BRESENHAMLINE
	中点算法生成直线	ID_MIDPOINTLINE

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 1.2 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 1.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_DDLINE	CONMMAN	OnDdaline
ID_MIDPOINTLINE	CONMMAN	OnMidpointline
ID_BRESENHAMLINE	CONMMAN	OnBresenhamline

4. 程序结构代码，在 CMyView.cpp 文件中相应位置添加如下代码：

代码见纸书

// DDA 算法生成直线

说明：1 以上代码理论上通过定义直线的两端点，可得到任意端点之间的一直线，但由于一般屏幕坐标采用右手系坐标，屏幕上只有正的 x, y 值，屏幕坐标与窗口坐标之间转换知识请参考第三章。

2. 注意上述程序考虑到当 $k \leq 1$ 的情形 x 每增加 1, y 最多增加 1；当 $k > 1$ 时 y 每增加 1, x 相应增加 $1/k$ 。在这个算法中， y 与 k 用浮点数表示，而且每一步都要对 y 进行四舍五入后取整。

//中点算法生成直线[代码见纸书](#)

说明:

1. 其中 d 是 x_p, y_p 的线性函数, 为了提高运算效率程序中采用增量计算。具体算法如下: 若当前像素处于 $d > 0$ 情况, 则取正右方像素 $P1$ (x_p+1, y_p), 判断下一个像素点的位置, 应计算 $d_1 = F(x_p+2, y_p+0.5) = a(x_p+2) + b(y_p+0.5) = d + a$; 其中增量为 a ; 若 $d < 0$ 时, 则取右上方像素 $P2$ (x_p+1, y_p+1)。判断再下一像素, 则要计算 $d_2 = F(x_p+2, y_p+1.5) = a(x_p+2) + b(y_p+1.5) + c = d + a + b$, 增量为 $a + b$ 。

2. 画线从 (x_0, y_0) 开始, d 的初值 $d_0=F(x_0+1, y_0+0.5)=F(x_0, y_0)+a+0.5b$ 因 $F(x_0, y_0)=0$, 则 $d_0=a+0.5b$ 。
3. 程序中只利用 d 的符号, d 的增量都是整数, 只是初始值包含小数, 用 $2d$ 代替 d , 使程序中仅包含整数的运算。

//Bresenham 算法生成直线

[代码见纸书](#)

说明:

1. 以上程序已经考虑到所有象限直线的生成。

2. Bresenham 算法的优点如下：（1.）不必计算直线的斜率，因此不做除法。（2.）不用浮点数，只用整数。（3.）只做整数加减运算和乘 2 运算，而乘 2 运算可以用移位操作实现。（4.）Bresenham 算法的运算速度很快。

1.2 圆

给出圆心坐标 (x_c, y_c) 和半径 r ，逐点画出一个圆周的公式有下列两种：**1.2.1 直角坐标法** $(x-x_c)^2 + (y-y_c)^2 = r^2$ 由上式导出：

$y = y_c \pm \sqrt{r^2 - (x-x_c)^2}$ 当 $x-x_c$ 从 $-r$ 到 r 作加 1 递增时，就可以求出对应的圆

周点的 y 坐标。但是这样求出的圆周上的点是不均匀的, $|x-x_c|$ 越大, 对应生成圆周点之间的圆周距离也就越长。因此, 所生成的圆不美观。**1.2.1 中点画圆法**

如图 1-7 所示, 函数为 $F(x,y) = x^2 + y^2 - R^2$ 的构造圆, 圆上的点为 $F(x,y) = 0$, 圆外的点 $F(x,y) > 0$, 圆内的点 $F(x,y) < 0$, 构造判别式

$$d = F(M) = F(x_p + 1, y_p - 0.5) = (x_p + 1)^2 + (y_p - 0.5)^2 - R^2$$

若 $d < 0$ 则应取 **P1** 为下一像素, 而且再下一像素的判别式为

$$d = F(x_p + 2, y_p - 0.5) = (x_p + 2)^2 + (y_p - 0.5)^2 - R^2 = d + 4x_p + 4$$

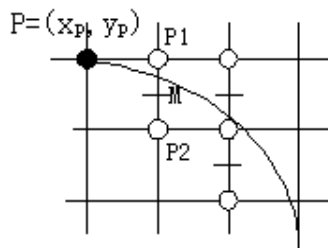


图 1-7 中点画圆法

$d \geq 0$ 若 则应取 **P2** 为下一像素，而且下一像素的判别式为

$$d = F(x_p + 2, y_p - 1.5) = (x_p + 2)^2 + (y_p - 1.5)^2 - R^2 = d + 2(x_p - y_p) + 5$$

我们讨论按顺时针方向生成第二个八分圆，则第一个像素是 **(0,R)**，判别式 **d** 的初始值为 $d_0 = F(1, R - 0.5) = 1.25 - R$

1.2.3 圆的 Bresenham 算法

设圆的半径为 **r**。先考虑圆心在 **(0, 0)**，并从 **x=0、y=r** 开始的顺时针方向的 **1/8** 圆周的生成过程。在这种情况下，**x** 每步增加 **1**，从 **x=0** 开始，到 **x=y** 结束。即有 **x_{i+1} = x_i + 1** 相应的 **y_{i+1}** 则在两种可能中选择：**y_{i+1} = y_i** 或者 **y_{i+1} = y_i - 1**。选择的原则是考察精确值 **y** 是靠近 **y_i** 还是靠近 **y_i - 1**(图 1-8)，计算式为

$$y^2 = r^2 - (x_{i+1})^2 \quad d_1 = y_i^2 - y^2 = y_i^2 - r^2 + (x_{i+1})^2 \quad d_2 = y^2 - (y_i - 1)^2 =$$

$$r^2 - (x_{j+1})^2 - (y_{j-1})^2$$

令 $p_j = d_1 - d_2$ ，并代入 d_1 、 d_2 ，则有

$$p_j = 2(x_{j+1})^2 + y_j^2 + (y_{j-1})^2 - 2r^2 \quad (1.6)$$

p_j 称为误差。如果 $p_j < 0$ 则 $y_{j+1} = y_j$ ，否则 $y_{j+1} = y_{j-1}$ 。

p_j 的递归式为

$$p_{j+1} = p_j + 4x_{j+1} + 6 + 2(y_{j+1}^2 - y_j^2) - 2(y_{j+1} - y_j) \quad (1.7)$$

p_j 的初值由式(1.6)代入 $x_j = 0$ ， $y_j = r$ 而得

$$p_1 = 3 - 2r \quad (1.8)$$

根据上面的推导，圆周生成算法思想如下：

1. 求误差初值， $p_1 = 3 - 2r$ ， $i = 1$ ，画点 $(0, r)$ ；
2. 求下一个光栅位置，其中 $x_{i+1} = x_i + 1$ ，如果 $p_i < 0$ 则 $y_{i+1} = y_i$ ，否则

$y_{i+1}=y_i-1;$

3. 画点(x_{i+1}, y_{i+1}); 4. 计算下一个误差, 如果 $p_i < 0$ 则
 $p_{i+1}=p_i+4x_i+6$, 否则 $p_{i+1}=p_i+4(x_i-y_i)+10$;
5. $i=i+1$, 如果 $x=y$ 则结束, 否则返回步骤 2。

二. 程序设计

1. 创建应用程序框架, 以上面建立的单文档程序框架为基础。
2. 编辑菜单资源

在工作区的【ResourceView】标签中, 单击 Menu 项左边 “+”, 然后双击其子项 IDR_MAINFRAME, 并根据 1.3 表中的添加编辑菜单资源。

此时建好的菜单如图 1-9 所示。

表 1.3 菜单资源表

菜单标题	菜单项标题	标示符 ID
圆	中点画圆	ID_MIDPOINTCIRCLE
	Bresenham 画圆	ID_BRESENHAMCIRCLE

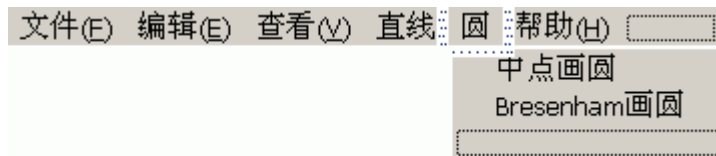


图 1-9

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 1.4 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 1.4 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_MIDPOINTCIRCLE	CONMMAN	OnMidpointcircle
ID_BRESENHAMCIRCLE	CONMMAN	OnBresenhamcircle

4. 程序结构代码，在 CMyView.cpp 文件中相应位置添加如下代码：

1.3 椭圆的扫描转换 1.3.1 椭圆的

特征

$$F(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$$

- 对于椭圆上的点, 有 $F(x, y) = 0$;
- 对于椭圆外的点, $F(x, y) > 0$;
- 对于椭圆内的点, $F(x, y) < 0$ 。以弧上斜率为 -1 的点作为分界将第一象限椭圆弧分为上下两部分 (如图 1-9 所示)。

法向量

$$N(x, y) = \frac{\partial F}{\partial x} i + \frac{\partial F}{\partial y} j = 2b^2 x i + 2a^2 y j$$

引理: 若在

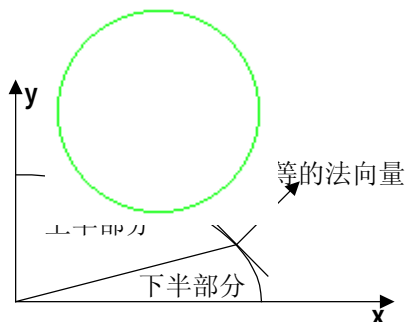


图 1-9 第一象限的椭圆弧

当前点，法向量的 y 分量比 x 分量，即

$$b^2(x_i + 1) < a^2(y_i - 0.5)$$

而在下一个点，不等号改变方向，则说明椭圆弧从上部分转入下部分。

3.3.2 椭圆的中点

Bresenham 算法 算法原

理推导上半部分的椭圆

绘 制 公 式

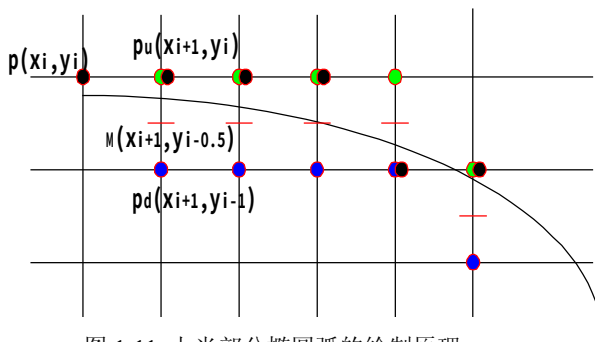


图 1-11 上半部分椭圆弧的绘制原理

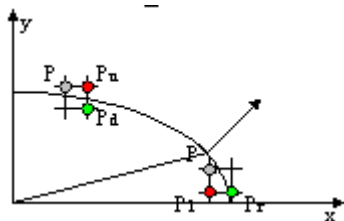
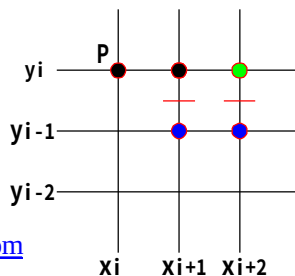
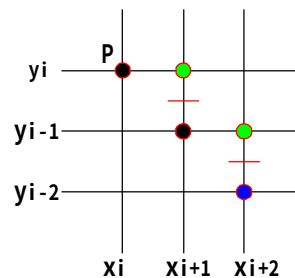


图 1-10 Bresenham 椭圆绘制算法原理

判别式· 若 $d_1 \leq 0$, 取

$P_u(x_i+1, y_i)$

• 若 $d_1 > 0$, 取

(a) $d \leq 0$ 的情况(b) $d > 0$ 的情况

$P_d(x_i+1, y_i-1)$

误差项的递推

$$\begin{aligned} d_1 &= F(x_i+2, y_i-0.5) = b^2(x_i+2)^2 + a^2(y_i-0.5)^2 - a^2b^2 \\ &= b^2(x_i+1)^2 + a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3) \\ &= d_1 + b^2(2x_i+3) \end{aligned}$$

$d_1 \leq 0$: $d_1 > 0$:

$$\begin{aligned} d_1 &= F(x_i+2, y_i-1.5) = b^2(x_i+2)^2 + a^2(y_i-1.5)^2 - a^2b^2 \\ &= b^2(x_i+1)^2 + a^2(y_i-0.5)^2 - a^2b^2 + b^2(2x_i+3) + a^2(-2y_i+2) \\ &= d_1 + b^2(2x_i+3) + a^2(-2y_i+2) \end{aligned}$$

推导椭圆弧下半部分的绘制公式

$$d_2 = F(x_i + 0.5, y_i - 1) = b^2(x_i + 0.5)^2 + a^2(y_i - 1)^2 - a^2b^2$$

原理判别式

判别式的初始值

$$\begin{aligned} d_{10} &= F(1, b - 0.5) = b^2 + a^2(b - 0.5)^2 - a^2b^2 \\ &= b^2 + a^2(-b + 0.25) \end{aligned}$$

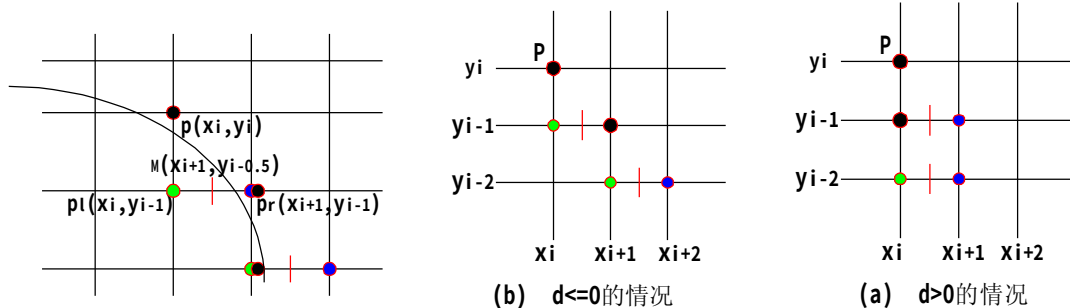


图 1-13 下半部分椭圆弧的绘制原理

- 若 $d_2 > 0$, 取 $P_l(x_i, y_i - 1)$
- 若 $d_2 \leq 0$, 取 $P_r(x_i + 1, y_i - 1)$

误差项的递推

$$\begin{aligned}
 d_2 \leq 0: \quad d_2 &= F(x_i + 0.5, y_i - 2) = b^2(x_i + 0.5)^2 + a^2(y_i - 2)^2 - a^2b^2 \\
 &= b^2(x_i + 0.5)^2 + a^2(y_i - 1)^2 - a^2b^2 + a^2(-2y_i + 3) \\
 &= d_2 + a^2(-2y_i + 3)
 \end{aligned}$$

$$\begin{aligned}
 d_2 > 0: \quad d_2 &= F(x_i + 1.5, y_i - 2) = b^2(x_i + 1.5)^2 + a^2(y_i - 2)^2 - a^2b^2 \\
 &= b^2(x_i + 0.5)^2 + a^2(y_i - 1)^2 - a^2b^2 + b^2(2x_i + 2) + a^2(-2y_i + 3) \\
 &= d_2 + b^2(2x_i + 2) + a^2(-2y_i + 3)
 \end{aligned}$$

算法步骤:

1.输入椭圆的长半轴 a 和短半轴 b 。

2.计算初始值 $d=b^2+a^2(-b+0.25)$ 、 $x=0$ 、 $y=b$ 。

3.绘制点 (x,y) 及其在四分象限上的另外三个对称点。4.判断 d 的符号。若 $d \leq 0$ ，则先将 d 更新为 $d+b^2(2x+3)$ ，再将 (x,y) 更新为 $(x+1,y)$ ；否则先将 d 更新为 $d+b^2(2x+3)+a^2(-2y+2)$ ，再将 (x,y) 更新为 $(x+1,y-1)$ 。

5.当 $b^2(x+1) < a^2(y-0.5)$ 时，重复步骤 3 和 4。否则转到步骤 6。

$$d = b^2(x+0.5)^2 + a^2(y-1)^2 - a^2b^2$$

6.用上半部分计算的最后点(x,y)来计算下半部分中 d 的初值：7.绘制点(x,y)及其在四分象限上的另外三个对称点。

8.判断 d 的符号。

若 $d \leq 0$ ，则先将 d 更新为 $b^2(2x_i+2)+a^2(-2y_i+3)$ ，再将(x,y)更新为(x+1,y-1)；否则先将 d 更新为 $d+a^2(-2y_i+3)$ ，再将(x,y)更新为(x,y-1)。

9.当 $y > 0$ 时，重复步骤 7 和 8。否则结束。

二. 程序设计

1. 创建应用程序框架，以上面建立的单文档程序框架为基础。

2. 编辑菜单资源

在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 1.5 表中的添加编辑菜单资源。此时建好的菜单如图 1-14 所示。

表 1.5 菜单资源表

菜单标题	菜单项标题	标示符 ID
文件(F) 编辑(E) 查看(V) 直线 L 圆 O 椭圆 E 帮助(H) A	中点画椭圆	ID_MIDPOINTELLISPE
	中点画椭圆	

图 1-14

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 1.6 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 1.6 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_MIDPOINTELLISPE	CONMMAN	OnMidpointellispe

4. 程序结构代码

1.4 多边形的扫描转换与区域填充

在计算机图形学中，多边形有两种重要的表示方法：**顶点表示**和**点阵表示**。顶点表示是用多边形的顶点序列来表示多边形，特点直观、几何意义强、占内存少，易于进行几何变换，但由于它没有明确指出哪些像素在多边形内故不能直接用于面着色。点阵表示是用位于多边形内的像素集合来刻画多边形。这种表示丢失了许多几何信息，但便于帧缓冲器表示图形，是面着

色所需要的图形表示形式。光栅图形的一个基本问题是把多边形的顶点表示转换为点阵表示。这种转换称为**多边形的扫描转换**。

1.4.1 多边形的扫描转换

多边形可分为凸多边形、凹多边形、含内环的多边形。

- ① 凸多边形：任意两顶点间的连线均在多边形内
- ② 凹多边形 任意两顶点间的连线均在连线有不在多边形内

③ 含内环的多边形 形内的部分

2.4.1.1 扫描线算法

扫描线多边形区域填充算法是按扫描线顺序，计算扫描线与多边形的相交区间，再用要求的颜色显示这些区间的像素。区间的端点可以通过计算扫描线与多边形边界线的交点获得。对于一条扫描线，多边形的填充过程可以分为四个步骤：

- (1) 求交：计算扫描线与多边形各边的交点；
- (2) 排序：把所有交点按 x 值递增顺序排序；
- (3) 配对：第一个与第二个，第三个与第四个等等；每对交点代表扫描线与多边形的一个相交区间，
- (4) 填色：把相交区间内的像素置成多边形颜色，把相交区间外的像素置成背景色。

具体实现方法：为多边形的每一条边建立一边表；为了提高效率，在处理一条扫描线时，仅对与它相交的多边形的边进行求交运算。我们把与当前

扫描线相交的边称为活性边，并把它按与扫描线交点递增的顺序存放在一个链表中，称此链表为活性边表。另外使用增量法计算时，我们需要知道一条边何时不再与下一条扫描线相交，以便及时把它从扫描线循环删除出去。

为了方便活性边表的建立与更新，我们为每一条扫描线建立一个新边表

(NET)，存放在该扫描线第一次出现的边。为使程序简单、易读，这里新边表的结点应保存其对应边如下信息：当前边的边号、边的较低端点 (x_{min} , y_{min}) 与边的较高端点 (x_{max} , y_{max}) 和从当前扫描线到下一条扫描线间 x 的增量 Δx 。

相邻扫描线间 x 的增量 Δx 的计算，假定当前扫描线与多边形某一条边的交点的 x 坐标为 x_i ，则下一条扫描线与该边的交点不要重计算，只要加一个增量 Δx 。设该边的直线方程为： $ax+by+c=0$ ；若 $y=y_i$ ， $x=x_i$ ；则当 $y = y_{i+1}$ 时，

$$x_{i+1} = \frac{1}{a} (-b \cdot y_{i+1} - c) = x_i - \frac{b}{a}; \quad \text{其中 } \Delta x = -\frac{b}{a} \text{ 为常数,}$$

扫描线与多边形顶点相交的处理方法如图所示。

1. 扫描线与多边形相交

的边分处扫描线的两侧, 则
记为一个交点, 如点 P_5 , P_6 。

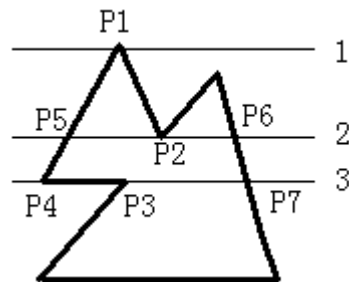


图 1-15 扫描线与多边形相交, 特殊情况的处理

2. 扫描线与多边形相交

的边分处扫描线同侧, 且 $y_i < y_{i-1}$, $y_i < y_{i+1}$, 则计 **2** 个交点(填色), 如 P_2 , 若 $y_i > y_{i-1}$, $y_i > y_{i+1}$, 则计 **0** 个交点(不填色), 如 P_1 。

3. 扫描线与多边形边界重合 (当要区分边界和边界内区域时需特殊处理), 则计 **1** 个交点。

具体实现时，只需检查顶点的两条边的另外两个端点的 y 值。按这两个 y 值中大于交点 y 值的个数是 **0,1,2** 来决定。

算法步骤：

- (1)初始化：构造边表；
- (2)对边表进行排序，构造活性边表；
- (3)对每条扫描线对应的活性边表中求交点；
- (4)判断交点类型，并两两配对。
- (5)对符合条件的交点之间用画线方式填充；
- (6)下一条扫描线，直至满足扫描结束条件。

1.4.2 区域填充算法

这里的**区域**指已表示成点阵形式的填充图形，是像素的集合。区域有两种表示形式：内点表示和边界表示。内点表示，即区域内的所有像素有相同颜色；边界表示，即区域的边界点有相同颜色。**区域填充**指先将区域的一点赋予指定的颜色，然后将该颜色扩展到整个区域的过程。

区域填充算法要求区域是连通的。区域可分为**4 向连通区域**和**8 向连通区域**。**4 向连通区域**指的是从区域上一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意像素；

8 向连通区域指的是从区域内每一像素出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合来到达。

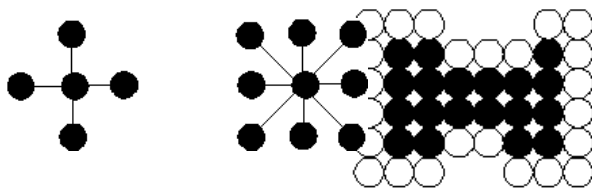


图 1-16 四连通区域和八连通 示内点 ○ 表示边界点

图 1-17 区域的内点表示和边界

1.1.2.1 区域填充的递归算法

上面讨论的多边形填充算法是按扫描线顺序进行的。种子填充算法则是假设在多边形内有一像素已知，由此出发利用连通性填充区域内的所有像素。一般采用多次递归方式。

1.4.2.2 区域填充的扫描线算法

算法的基本过程如下：给定种子点 (x,y) ，首先填充种子点所在扫描线上给定区域的一个区段，然后确定与这一区段相连通的上、下两条扫描线上位于给定区域内的区段，并依次保存下来。反复这个过程，直到填充结束。

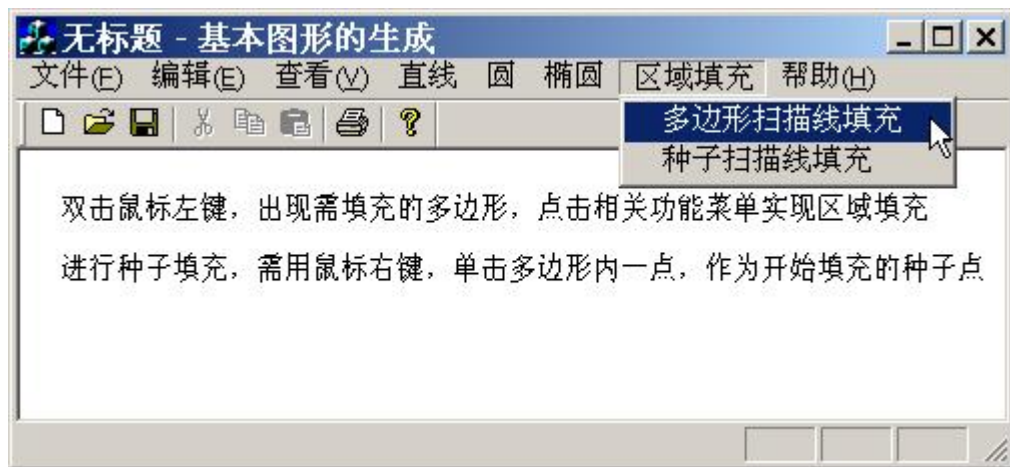
区域填充的扫描线算法可由下列三个步骤实现：

- (1)**初始化：确定种子点元素 (x, y) 。
- (2)**判断种子点 (x, y) 是否满足非边界、非填充色的条件，满足条件，以 y 作为当前扫描线沿当前扫描线向左、右两个方向填充，直到边界。
- (3)** 确定新的种子点：检查与当前扫描线 y 上、下相邻的两条扫描线上的像素。若存在非边界、未填充的像素，则返回第 **(2)** 步进行扫描填充。直至区域所有元素均为填充色，程序结束。

扫描线填充算法提高了区域填充的效率。

二. 程序设计

1. 创建应用程序框架，以上述单文档程序框架为基础。



2. 编辑菜单资源

在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 1.7 表中的添加编辑菜单资源。此时建好的菜单如图 1-18 所示。

表 1.7 菜单资源表

文件(F)	编辑(E)	查看(V)	直线	圆	椭圆	区域填充	帮助(H)	标示符 ID
菜单标题			菜单项标题			ID_MIDPOINTELLISPE		
区域填充			多边形扫描线填充			ID_MIDPOINTELLISPE		

图 1-18

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处

理函数，ClassName 栏中选择 CMyView，根据表 1.8 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 1.8 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_MIDPOINTELLISPE	CONMMAN	OnMidpointellispe

4. 添加程序结构代码

(1) 在“基本图形的生成 View.h”适当位置添加以下黑体字部分代码

代码见纸书

说明：1. 双击鼠标左键，出现需填充的多边形，点击相关功能菜单实现区

域填充。

2. 进行种子填充，需用鼠标右键，单击多边形内一点，作为开始填充的种子点。

[代码见纸书](#)

1.5 字符的生成

字符指数字、字母、汉字等符号。计算机中字符由一个数字编码唯一标识。国际上最流行的字符集是“美国信息交换用标准代码集”简称 **ASCII** 码。它是用 **7** 位二进制数进行编码表示 **128** 个字符，包括字母、标点、运算

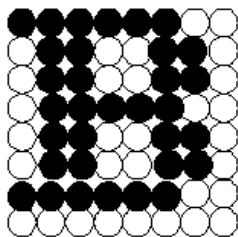
符以及一些特殊符号。我国除采用 **ASCII** 码外，还另外制定了汉字编码的国家标准字符集 **GB2312—80**。该字符集分为 **94** 个区，**94** 个位，每个符号由一个区码和一个位码共同标识。区码和位码各用一个字节表示。为了能够区分 **ASCII** 码与汉字编码，采用字节的最高位来标识：最高位为 **0** 表示 **ASCII** 码；最高位为 **1** 表示表示汉字编码。为了在显示器等输出设备上输出字符，系统中必须装备有相应的字库。字库中存储了每个字符的形状信息，字库分为矢量和点阵型两种。

1.5.1 点阵字符

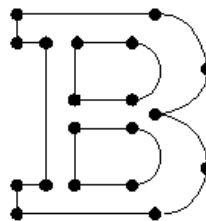
在点阵字符库中，每个字符由一个位图表示。该位为 **1** 表示字符的笔画经过此位，对应于此位的像素应置为字符颜色。该位为 **0** 表示字符的笔画不经过此位，对应于此位的像素应置为背景颜色。在实际应用中，有多种字体（如宋体、楷体等），每种字体又有多种大小型号，因此字库的存储空间是很庞大的。解决这个问题一般采用压缩技术。如：黑白段压缩；部件压缩；轮廓字形压缩等。其中，轮廓字形法压缩比大，且能保证字符质量，是当今国际上最流行的一种方法。轮廓字形法采用直线或二/三次 **bezier** 曲线的集合来描述一个字符的轮廓线。轮廓线构成一个或若干个封闭的平面区

域。轮廓线定义加上一些指示横宽、竖宽、基点、基线等等控制信息就构成了字符的压缩数据。

点阵字符的显示分为两步。首先从字库中将它的位图检索出来。然后将检索到的位图写到帧缓冲器中。



1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



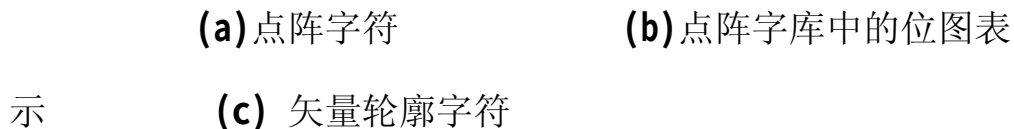


图 1-19 字符的种类

1.5.2 矢量字符

矢量字符记录字符的笔画信息而不是整个位图，具有存储空间小，美观、变换方便等优点。对于字符的旋转、缩放等变换，点阵字符的变换需要对表示字符位图中的每一像素进行；而矢量字符的变换只要对其笔画端点进行变换就可以了。矢量字符的显示也分为两步。首先从字库中将它的字符信

息。然后取出端点坐标，对其进行适当的几何变换，再根据各端点的标志显示出字符。

1.5.3 字符属性

字符属性一般包括：字体、字高、字宽因子(扩展/压缩)、字倾斜角、对齐方式、字色和写方式等，其中：

宋体：如仿宋体 楷体 黑体 隶书；

字倾斜角：如倾斜 ；

对齐：如左对齐、中心对齐、右对齐；

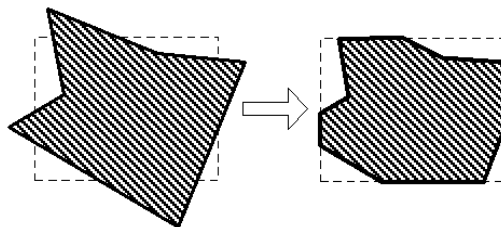
字色：如红、绿、蓝色；

写方式：**替换方式**时，对应字符掩模中空白区被置成背景色。与方式时，这部分区域颜色不受影响。

1.6 图形裁剪

在使用计算机处理图形信息时，计算机内部存储的图形往往比较大，而屏幕显示的只是图的一部分。因此需要确定图形中哪些部分落在显示区之内，哪些落在显示区之外，以便只显示落在显示区内的那部分图形。这个选择过程称为**裁剪**。最简单的裁剪方法是把各种图形扫描转换为点之后，再判断各点是否在窗内。但那样太费时，一般不可取。这是因为有些图形组成部分全部在窗口外，可以完全排除，不必进行扫描转换。所以一般采用先裁剪

再扫描转换的方法。



(a) 裁剪前

(b) 裁剪后

图 1-20 多边形裁剪

1.6.1 线裁剪

1. 直线和窗口的关系可以分为如下 3 类(图 1-21): (1) 整条直线在窗口内。此时, 不需剪裁, 显示整条直线。 (2) 整条直线在窗口外, 此时, 不需剪裁, 不显示

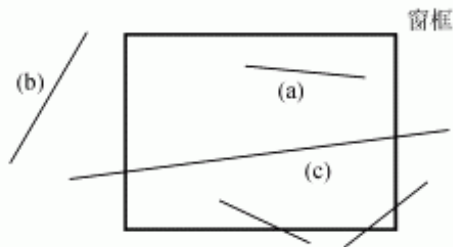


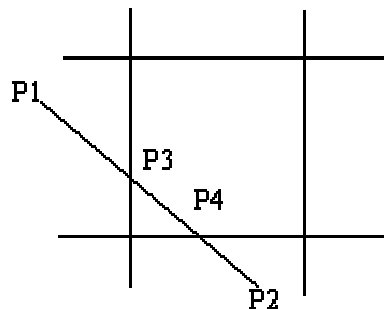
图 1-21 直线与窗口的关系

整条直线。 (3) 部分直线在窗口内, 部分直线在窗口外。此时, 需要求出直线与窗框的交点, 并将窗口外的直线部分剪裁掉, 显示窗口内的直线部分。 直线剪裁算法有两个主要步骤。首先将不需剪裁的直线挑出, 即删去

在窗外的直线。然后，对其余直线，逐条与窗框求交点，并将窗口外的部分删去。

2. Cohen-Sutherland 直线剪裁算法

1001	1000	1010
0001	0000	0010
0101	0100	0110



以区

图 1-22 窗口及其邻域的 5 个区域及与直线的关系

域编 码

为基础，将窗口及其周围的 8 个方向以 4 bit 的二进制数进行编码。如图 2-22 所示的编码方法将窗口及其邻域分为 5 个区域：(1) 内域：区域

(0000)。(2) 上域：区域(1001, 1000, 1010)。(3) 下域：区域(0101, 0100, 0110)。(4) 左域：区域(1001, 0001, 0101)。(5) 右域：区域(1010, 0010, 0110)。当线段的两个端点的编码的逻辑“与”非零时，线段为显然不可见的。对某线段的两各端点的区号进行位与运算，可知这两个端点是否同在视区的上、下、左、右。算法的主要思想是，对每条直线，如 P_1P_2 利用以下步骤进行判断：

(1) 对直线两端点 P_1 、 P_2 编码分别记为 $C_1(P_1)=\{a_1, b_1, c_1, d_1\}$, $C_2(P_2)=\{a_2, b_2, c_2, d_2\}$ 其中， a_i 、 b_i 、 c_i 、 d_i 取值范围为 $\{1, 0\}$, $i \in \{1, 2\}$ 。

(2) 如果 $a_i=b_i=c_i=d_i=0$ ，则显示整条直线，取出下一条直线，返步骤(1)；否则，进入步骤(3)。

(3)如果 $|a_1 - a_2| = 1$, 则求直线与窗上边($y = y_{w-max}$)的交点, 并删去交点以上部分。如果 $|b_1 - b_2| = 1$, $|c_1 - c_2| = 1$, $|d_1 - d_2| = 1$, 作类似处理。(4) 返步骤(1)判断下一条直线。

1.6.2 多边形裁剪

多边形剪裁算法的关键在于, 通过剪裁, 要保持窗口内多边形的边界部分, 而且要将窗框的有关部分按一定次序插入多边形的保留边界之间, 从而使剪裁后的多边形的边仍然保持封闭状态, 以便填色算法得以正确实现

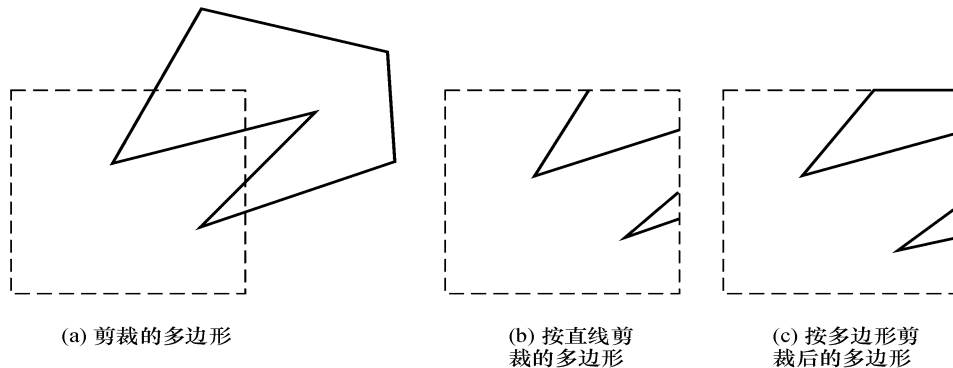


图 1-23 多边形裁剪原理示意图

Sutherland-Hodgman 算法：思路：将多边形的各边先相对于窗口的某一条边界进行裁剪，然后将裁剪结果再与另一条边界进行裁剪，如此重复多次，便可得到最终结果。**实现方法：**

①设置二个表

输入顶点表(向量)—用于存放被裁剪多边形的顶点 **p1-pm**。输出顶点表(线性链表)—用于存放裁剪过程中及结果的顶点 **q1-qn**。

②输入顶点表中各顶点要求按一定顺序排列，一般可采用顺时针或逆时针方向。

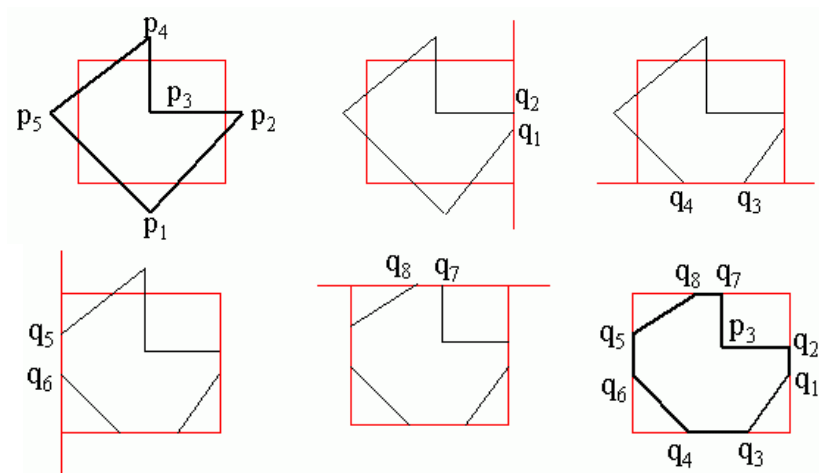


图 1-24 多边形裁剪操作示意图

③相对于裁剪窗口的各条边界，按顶点表中的顺序，逐边进行裁剪。

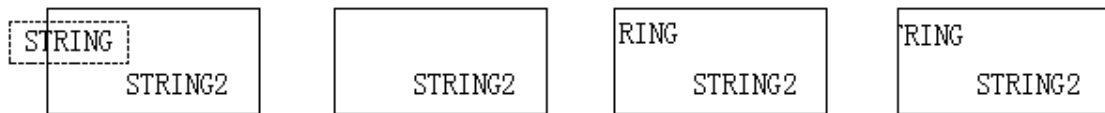
具体操作:

- (1) P_i 若位于边界线的可见一侧, 则 P_i 送输出顶点表
- (2) P_i 若位于边界线的不可见一侧, 则将其舍弃。
- (3) 除第一个顶点外, 还要检查每一个 P_i 和前一顶点 P_{i-1} 是否位于窗口边界的同一侧, 若不在同一侧, 则需计算出交点送输出顶点表。
- (4) 最后一个顶点 P_n 则还要与 P_1 一起进行同样的检查。

1. 6. 3 字符裁剪

前面我们介绍了字符和文本的输出。当字符和文本部分在窗口内, 部分在窗口外时, 就提出了字符裁剪问题。字符串裁剪可按三个精度来进行: 串

精度、字符精度、以及笔画\象素精度。采用串精度进行裁剪时，将包围字符串的外接矩形对窗口作裁剪。当字符串方框整个在窗口内时予以显示，否则不显示。采用字符精度进行裁剪时，将包围字的外接矩形对窗口作裁剪，某个字符方框整个落在窗口内予以显示，否则不显示。采用笔画\象素精度进行裁剪时，将笔划分解成直线段对窗口作裁剪，处理方法同上。



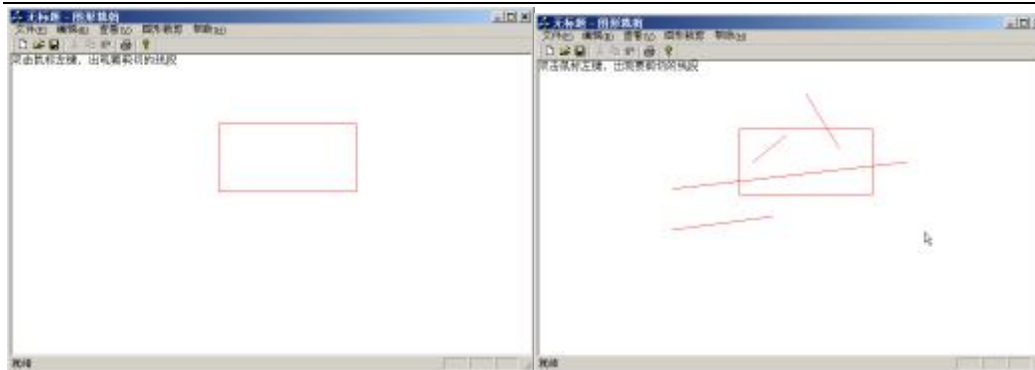
(a)待裁剪字符串 (b)串精度裁剪 (c)字符精度裁剪 (d)象素精度

裁剪图 1-25 字符裁剪

1.6.4 图形裁剪编程

一、程序设计功能说明

如图 1-26 所示为图形裁剪的实用程序界面。为本例程序运行时的主界面，首先根据界面提示，在用户区双击鼠标左键，出现所需要裁剪的各种线段，再单击菜单中【图形裁剪】可选择其下拉菜单的各图形裁剪选项完成各种图裁剪（在窗口中红矩形框外的线段或多边形被裁减掉）。



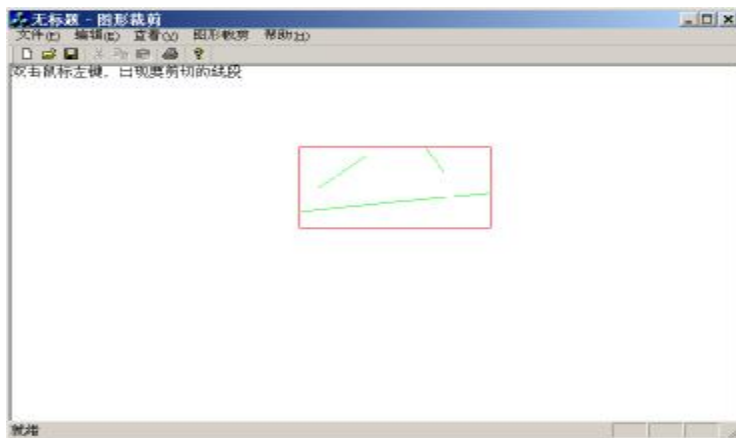


图 1-26

二、程序设计步骤

1. 建工程名称为“图形裁剪”单文档应用程序框架（参看上面单文档

应用程序框架的建立）

2. 编辑菜单资源

设计如图 1-26 所示的菜单项。在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 1.9 表中的定义编辑菜单资源。

表 1.9 菜单资源表

菜单标题	菜单项标题	标示符 ID
图形裁剪	线段裁剪	ID_CLIPLINE
	多边形裁剪	ID_CLIPPOLYGON

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处

理函数，ClassName 栏中选择 CMyView，根据表 1.10 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 1.10 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_CLIPLINE	CONMMAN	OnIDTRANSLATION
ID_CLIPPOLYGON	CONMMAN	OnIDROTATION

4. 添加代码，在图形裁剪应用程序的相应文件中添加如下黑体字部分代码

1. 在“图形裁剪 View.h”文档中的适当位置添加定义存储线段端点的数组：

2. 在“图形裁剪 **View.cpp**”文档中的适当位置手工添加以下黑体部分代码：

[代码见纸书](#)

说明：为避免上述“图形裁剪”程序执行中，在没有双击左键就直接单击【线段裁剪】或没有双击左键就直接单击【多边形裁剪】的情况发生，设置了消息对话框进行警告。

1.7 Visual C++中基本绘图函数

实际利用 Visual C++中编制图形程序，我们可以利用上述算法自己动手编制基本图形程序，作为图形程序的基类，当然我们还可利用系统中已提供的图形基类。下面简单介绍 Visual C++提供的常用绘制图形函数。

1. 点

画点是最基本的绘图操作，在绘图中，画点是通过调用 CDC::SetPixel() 或 CDC::SetPixel()函数来实现的，原型如：

(1) COLORREF SetPixel(int x, int y, COLORREF crColor);

(2) COLORREF SetPixel(POINT point, COLORREF crColor);

(3) BOOL SetPixelV(int x, int y, COLORREF crColor);

(2)BOOL SetPixelV(POINT point, COLORREF crColor);

2. 画笔

一般格式：(1)Cpen():: Cpen(int nPenStyle, int nWidth, CORLORREF crColor);

各属性意义： nPenStyle 设置画笔的式样，式样有：PS_SOLID（实线），PS_DASH（虚线）、PS_DASHDOT（点划线）、PS_DASHDOTDOT（双点划线）、PS_DOT（点线）、PS_NULL（空笔不画线）；nWidth 设置线的宽度，默认值为 1（1 个像素宽）；crColor 表示颜色，可用 DWOR 表示，

也可用 RGB(r, g, b)表示。

3. 画刷，用于指定填充的特征，画刷创建有格式有：

- (1) CBrush : : CBrush (创建一个空的画刷对象)，可用
GreateSolidBrush()，GreatehatchBrush()，GreatehatchBrushIndirect()，
GreatePatternBrush()，GreateDIBPatternBrush()建立画刷；
- (2) CBrush::CBrush()建立单一颜色的画刷，用次画刷画出的图形内部将会
填充指定颜色；
- (3) CBrush::CBrush (int nIndex, COLORREF crColor)；构建名为 hatch 的
画刷，特点为画出的多边形内部将填充 nrColor 指定的线条格式，

nrColor 有：HS_BDIAGONAL（45 度左下→右上的斜线）、HS_CROSS（垂直线和水平线）、HS_DIAGCROSS（45 度左上→右下、左上→右下的相交斜线）、HS_HDLAGNAL（45 度左上→右下的斜线）、HS_HORIZONTAL（水平线）、HS_VERTICAL（垂直线）

- (4) CBrush::CBrush(Cbitmap *pBitmap) 中 pBitmap 指向 Cbitmap 对象的指针，这一位图对象包含用作画刷图按的位图，此位图必须为 8×8 大小，否则将对原位图进行裁剪。

创建画刷和画笔后，还要用 CDC 类选中画笔和画刷，用 CPaintDC，CClientDC 或 CWindowDC 来选中、绘图及撤销对象。

CClientDC 对象代表客户程序区域的绘图画面只能在窗口的客户区域中画图。若需处理整个画面（包括客户程序区域和非客户程序区）设备上下文的调用和释放可用 CWindowDC。

4. 绘制直线函数

(1) MoveTo()函数用来设置当前的 x,y 的位置，创建有格式有：

CPoint MoveTo(int x, int y)

CPoint MoveTo(POINT point)

其中 x, y 用于定义新位置的坐标，point 指定新位置，可为其传递一个 Point 对象。

功能：是将线的起点从当前位置移到新位置 (x,y)，并且只移动点不画线。

(2) LineTo()用于绘制起点坐标到始点直线，创建有格式有：

BOOL LineTo(int x, int y)

BOOL LineTo(POINT point)

其中 x, y 用于定义线的终点坐标，point 指定线段端点位置，可为其传递一个 Point 结构或 Point 对象。

功能：是从当前的位置到新位置 (x,y) 画线（不包括此端点）。

5. 椭圆函数

创建格式有:

```
BOOL Ellipse(int x1, int y1 , int x2, int y2)
```

```
BOOL Ellipse(LPCRECT lpRect)
```

说明: x1, y1, 限定椭圆范围的矩形左上角坐标; x2, y2 限定椭圆范围的矩形右下角坐标。

LpRect 指定椭圆的限定矩形, 可为其传递一个 CRect 对象。

6. 函数绘制一段椭圆弧 Arc()

创建格式有:

```
BOOL Arc(int x1, int y1 , int x2, int y2, int x3, int y3 , int x4, int y4)
```

BOOL Ellipse(LPCRECT lpRect)

x1, y1, 限定椭圆弧范围的矩形左上角坐标； x2, y2 限定椭圆弧范围的矩形右下角坐标。 X3, y3 起点坐标； x2, y2 终点坐标。

7. 矩形函数

创建格式有：

BOOL Rectangle(int x1, int y1 , int x2, int y2)

x1, y1 矩形左上角坐标； x2, y2 矩形右下角坐标。

功能使用当前画笔画一矩形。

8. 连续画线函数

创建格式有:

(1) `BOOL PolyLine(LPPOINT lpPoints , int nCount);`

说明: `lpPoints` 指向 `POINT` 结构数组, 数组中每一个结构标识一个点的坐标;

`nCount`:: 定义数组中的点数, 使用当前画笔右第一个点开始经后续点连续画线直到最后一个点。

(2) `BOOL PolyLineTo(LPPOINT *lpPoints , int nCount);`

说明: lpPoints 指向 POINT 结构数组指针, 画一条或多条直线的指针, 数组中村方直线顶点的坐标;

nCount:: 定义数组中的点数。

(3) BOOL PolyBezier(LPPOINT *lpPoints);

说明: lpPoints 指向 POINT 结构数组指针, 画一条或多条直线的指针, 数组中包括曲线的重点和控制点;

nCount:: 定义数组中的点数。

绘制三次贝塞尔曲线需要 2 个控制点和一个终点和一个起点, 共 4 个点决定一条贝塞尔曲线。

(4) `BOOL PolyBezierTo(LPPOINT *lpPoints);`画一条或多条贝塞尔曲线。

(5) `BOOL PolyBezierTo(LPPOINT *lpPoints);`画一条或多条贝塞尔曲线

`lpPoints` 指向 `POINT` 结构数组指针, 画一条或多条直线的指针, 数组中包括曲线的重点和控制点;

`nCount::` 定义数组中的点数。

9. 多边形绘制函数

创建格式有:

(1) `BOOL Polygon(counst POINT lpPoints, int nCount);`

说明：lpPoint 指定多边形顶点数组中每一点是一个 POINT 结构或一个 CPoint 对象，nCount 指定数组中顶点数。

(2) BOOL PolyPolygon(LPPOINT lpPoints,lpint lpPolyCounts, int
lpPoints);

说明：lpPoint 指向一个 POINT 结构或 CPoint 对象数组，每个数组定义一个多边形的顶点；lpPolyCounts 指向一个整数数组，每个整数说明 lpPoints 数组中一个多边形的顶点数，nCount:: 为 LpPolyCount 数组中的项数，即指定要画的多边形数，最多为 2。

10. 填充函数

创建格式有：

(1) `BOOL FillSolidRect(LPCRECT lpRect, COLORREF crColor);`

(2) `BOOL FillSolidRect(int x, int y, int cx, int cy, COLORREF Clr);`

说明：LpRect 指定矩形可传递一个指向 RECT 结构的指针或 CRect 对象。

Clr 为填充颜色；x, y 矩形左下角，cx 为矩形宽，cy 为矩形高。

(3) `BOOL ExtFloodFill(int x, int y, COLORREF crColor, UINT nFillType);`

说明：X, y 为开始填充处坐标；crColor 为填充颜色；FloodFillBorder

指填充区域由 `crColor` 参数所指定颜色包围部分；`FloodFillSurface` 表示填充区域是由 `nColor` 指定颜色来定义矩形填充

(5) `BOOL FloodFill(int x, int y, COLORREF Clr);`

说明：`x, y` 为填充处逻辑坐标或边界颜色，`crColor` 指定填充颜色。

以上我们简单介绍了系统中提供的基本绘图函数，更多内容请参见其它参考书。

1. 8 课后练习

1. 为什么说直线生成算法是二维图形生成技术的基础？

2. 根据 DDA 算法编制绘制从(10, 10)到(300, 400)直线的程序
3. 将中点画线算法推广以便能画出任意斜率的直线。
4. 使用中点分割算法实现对直线段进行裁剪的程序。
5. 利用本章所建程序框架，在菜单项【圆】的子菜单中添加【直角坐标画圆】子菜单，并添加相应命令和代码实现直角坐标画圆程序，加深对画圆算法的理解并与其它算法程序相比较。
6. 利用 **Visual C++**以定义函数实现上述直线、圆、椭圆等图形的绘制，并进行区域填充。

第二章 二维图形

利用计算机绘制的图形与我们日常见到的图片、照片是有相似之处。除图片、照片等图形外，自然界中还存在丰富多彩的有形物体。一般，根据图形所在空间的不同，可将图形分为：三维图形和二维图形。图片、照片属二维图形，自然界中形形色色的物体属于三维图形。在计算机绘图的过程中，二维图形的绘制是绘制三维图形的基础，研究计算机图形的生成必须从研究

二维图形开始。计算机绘制图形时，无论图形多么复杂，都是利用一些相应图形基元经过图形变换组成的。在计算机绘图中，经常用到图形变换，图形变换是指图形信息经过几何变换后产生新的图形。基本的几何变换研究物体坐标在直角坐标系内的平移、旋转和变比等规则。下面介绍二维图形的这些基本变换规则。

2.1 用户坐标到屏幕坐标的变换

实际图纸上坐标系是实数域中的直角坐标系或极坐标系，统称为用户坐标系；计算机设备（如屏幕）上采用的坐标系为整数域（如屏幕一般为直角左手系），称为设备坐标系。因此用户坐标系中图形需经过变换才能绘制在

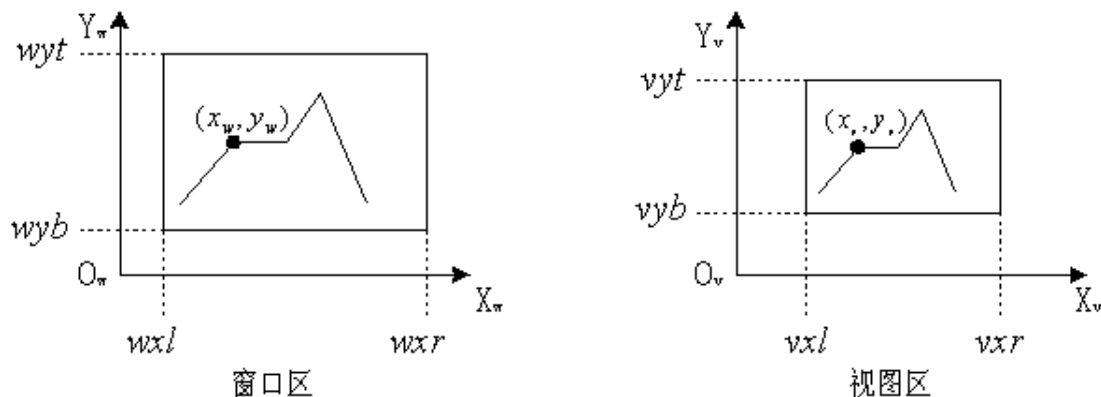


图 2-1 窗口区和视图区示意图

设备（如屏幕）上。用户坐标系中图形一般只有部分图形需要在设备上显示（或绘制），用户坐标中需要显示的图形（矩形区域）称为窗口，在设备（屏幕）上，显示（或绘制）图形的区域（矩形区域）称为视口。在计算机上绘制图形时，实际的窗口区与视图区往往不一样大小，要在视图区正确地显示形体的，必须将其从窗口区变换到视图区

2.1.1 窗口到视口的变换内容

图形从窗口到视口的变换亦称为数据规格化。窗口到视口变换包括以下内容：

- （1）窗口逻辑坐标与设备坐标的转换，当把用户坐标系（逻辑单位）中

的图形变换到视口中，视口中的坐标单位不再为逻辑单位，而是设备坐标（以像素为单位），根据设备的无关性，图形映射在视口上的图形大小应是不变的，这要求有像素与逻辑单位的转换比例（这一比例的大小随屏幕的大小和分辨率的高低有关）。

(2) 用户坐标系所选区域内图形的坐标转换到屏幕上坐标不一定为整数，对转换后坐标值取整。可通过四舍五入的方法将实型值的绝对值圆整化，最简单的方法是用赋值的类型转化规则来实现实型到整型的变换。

(3) 用户坐标系到设备（屏幕）坐标系，坐标轴方向变换。

(4) 屏幕坐标系水平方向与垂直方向刻度若不等（即像素间距不等）时，为保证图形不走样，还要进行比例变换。

2.1.2 窗口区到视图区的坐标变换

如图 2-1 所示，根据图中比例关系，窗口区到视图区的坐标变换公

式可写为：

$$x_v - vxl = \frac{vxr - vxl}{wxr - wxl} (x_w - wxl)$$
$$y_v - vyb = \frac{vyt - vyb}{wyt - wyb} (y_w - wyb)$$

其中：

$$x_v = a \cdot x_w + b$$

$$y_v = c \cdot y_w + d$$

$$a = \frac{vxr - vxl}{wxr - wxl}$$

$$b = vxl - \frac{vxr - vxl}{wxr - wxl} \cdot wxl$$

$$c = \frac{vyt - vyb}{wyt - wyb}$$

$$d = vyb - \frac{vyt - vyb}{wyt - wyb} \cdot wyb$$

总之，用矩阵表示为：

$$\begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & b \\ 0 & c & d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

2.2 二维几何变换

图形基本变换是指图形的比例变换、对称变换、旋转变换、错切变换、平移变换等。通过对原图形上二维向量引进第三个坐标即三维点向量（又称齐次坐标点），简称齐次坐标，在三维齐次坐标下，二维几何变换都可统一用矩阵表示。

所谓齐次坐标就是将一个原本是 n 维的向量用一个 $n+1$ 维向量来表示。

如向量 $(x_1, x_2, \dots, x_n)$ 的齐次坐标表示为 $(hx_1, hx_2, \dots, hx_n, h)$ ，其中 h 是一个实数。显然一个向量的齐次表示是不唯一的，齐次坐标的 h 取不同的值都表示的是同一个点，比如齐次坐标 $[8, 4, 2]$ 、 $[4, 2, 1]$ 表示的都是二维点 $[2, 1]$ 。

引进齐次坐标的优点：

- A. 提供了用矩阵运算把二维、三维甚至高维空间中的一个点集从一个坐标系变换到另一个坐标系的有效方法。

B. 可以表示无穷远的点。 $n+1$ 维的齐次坐标中如果 $h=0$, 实际上就表示了 n 维空间的一个无穷远点。对于齐次坐标 $[a,b,h]$, 保持 a,b 不变, $|V_1| = (x_1 * x_1, y_1 * y_1, z_1 * z_1)^{1/2}$ 的过程就表示了二维坐标系中的一个点沿直线 $ax+by=0$ 逐渐走向无穷远处的过程。

2.2.1 基本变换 1. 平移变换

若图形上任意一点的坐标为 (x,y) , 通过沿 x 和 y 轴分别平移 T_x 和 T_y 后成为新图形上的一点 (x',y') , 坐标变换: $x' = x + T_x$ $y' = y + T_y$

用齐次坐标表示的平移变换为:

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

其中平移变换矩阵为 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$

2. 比例变换

若图形上任意一点的坐标为 (x, y) ，通过沿 x 和 y 轴分别变比变换 S_x 和 S_y 后成为新图形上的一点 (x', y') ，坐标变换：

$$\begin{array}{ll} x' = S_x \cdot x & S_x > 1 \text{ 放大} \\ y' = S_y \cdot y & 0 < S_y < 1 \text{ 缩小} \end{array}$$

用齐次坐标表示的比例变换为：

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中比例变换矩阵为 $\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

3. 旋转变换

若图形上任意一点的坐标为 (x, y) ，通过将对象上的各点 (x, y) 围绕原点逆时针转动一个角度 θ ，后成为新图形上的一点 (x', y') ，

$$\begin{aligned} \text{坐标变换: } x' &= x \cdot \cos\theta - y \cdot \sin\theta & \theta > 0 \text{ 逆时针} \\ y' &= x \cdot \sin\theta + y \cdot \cos\theta & \theta < 0 \text{ 顺时针} \end{aligned}$$

用齐次坐标表示的旋转变换为：

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中旋转变换矩阵为 $\begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix}$

4. 对称变换

若图形上任意一点的坐标为 (x, y) ，关于 x 、 y 和原点分别作对称变换后成为新图形上的一点 (x', y') ，对称变换可分别表示为：

$$\text{关于 } x \text{ 坐标对称变换 } x' = x \quad y' = -y$$

用齐次矩阵表示的对称变换： $[x', y', 1] = [x, y, 1]$

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

其中关于 x 作对称变换矩阵为

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

关于 y 坐标对称变换 $x' = -x$ $y' = y$

用齐次矩阵表示的对称变换：

$$[x', y', 1] = [x, y, 1]$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中关于 y 作对称变换矩阵为

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

关于原点坐标对称变换: $x' = -x$ $y' = -y$

用齐次矩阵表示的对称变换: $[x', y', 1] = [x, y, 1] \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

其中关于原点作对称变换矩阵为 $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

5. 错切

1. 沿 x 方向错切:

沿 $+x$ 方向错切, 坐标变换 $\begin{matrix} x' = cy + x \\ y' = y \end{matrix} \quad c > 0$

沿 $-x$ 方向错切 , 坐标变换
$$\begin{matrix} x' = -cy + x \\ y' = y \end{matrix} \quad c > 0$$

用齐次矩阵表示的对称变换:
$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \pm c & 1 \end{bmatrix}$$

2. 沿 x 方向错切:

向 $+y$ 方向错切 , 坐标变换
$$\begin{matrix} x' = x \\ y' = bx + y \end{matrix} \quad b > 0$$

沿 $-y$ 方向错切 , 坐标变换
$$\begin{matrix} x' = x \\ y' = -bx + y \end{matrix} \quad b > 0$$
 用齐次矩阵表示的对称

变换:
$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & \pm b \\ 0 & 1 \end{bmatrix}$$

2.2.2 二维几何变换的级联

(1) 实际中的几何变换一次有若干个； (2) 基本形式有局限性。图形的级联变换是指图形作一次以上的基本变换，变换结果为每次基本变换矩阵乘积。

设图形经过 n 次基本几何变换，其变换矩阵分别为 $T_1, T_2, \dots, T_n$

则： 经 T_1 后： $[x' \ y' \ 1] = [x \ y \ 1] \cdot T_1$

经 T_2 后： $[x'' \ y'' \ 1] = [x' \ y' \ 1] \cdot T_2 = [x \ y \ 1] \cdot T_1 \cdot T_2$

经 T_n 后： $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot T_1 \cdot T_2 \cdot \dots \cdot T_n = T$ 称 $T = T_1 \cdot T_2 \cdot \dots \cdot T_n$ 为级联变

换的变换矩阵。下面介绍几种常见级联变换。

1. 复合平移

若对图形首先做平移变换 T_1 ，然后再做平移变换 T_2 ，相应的平移变换矩阵分别为：

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & m_1 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_2 & m_2 & 1 \end{bmatrix}$$

变换结果为复合平移变换 T ，其复合平移变换矩阵为：

$$\begin{aligned} T &= T_1 \cdot T_2 \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & m_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_2 & m_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 + l_2 & m_1 + m_2 & 1 \end{bmatrix} \end{aligned}$$

2. 复合比例

设比例变换 T_1 矩阵为:

$$T_1 = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & d_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

设比例变换 T_2 矩阵为:

$$T_2 = \begin{bmatrix} a_2 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

则复合比例变换 T 矩阵为:

$$\begin{aligned} T &= T_1 \cdot T_2 \\ &= \begin{bmatrix} a_1 & 0 & 0 \\ 0 & d_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 & 0 & 0 \\ 0 & d_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_1 \cdot a_2 & 0 & 0 \\ 0 & d_1 \cdot d_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

3. 复合旋转

设比例变换 T_1 矩阵为: $T_1 = \begin{bmatrix} \cos q_1 & \sin q_1 & 0 \\ -\sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

设比例变换 T_2 矩阵为: $T_2 = \begin{bmatrix} \cos q_2 & \sin q_2 & 0 \\ -\sin q_2 & \cos q_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

则复合比例变换 T 矩阵为:

$$T = T_1 \cdot T_2$$

$$= \begin{bmatrix} \cos q_1 & \sin q_1 & 0 \\ -\sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos q_2 & \sin q_2 & 0 \\ -\sin q_2 & \cos q_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(q_1 + q_2) & \sin(q_1 + q_2) & 0 \\ -\sin(q_1 + q_2) & \cos(q_1 + q_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{旋转}$$

变换合比例变换都与参考点有关，上面的旋转变换和比例变换均是相对与原点的。如果相对于某一参考点 (x_0, y_0) 作比例和旋转变换，则其变换过程是先将坐标原点平移到 (x_0, y_0) ，在再新的坐标系下作比例、旋转变换，然后将坐标原点平移回去，即复合变换。

4. 相对于点 (x_0, y_0) 的比例变换

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 - y_0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_0 & y_0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ (1-a)x_0 & (1-d)y_0 & 1 \end{bmatrix}$$

5. 相对于点 (x_0, y_0) 的旋转变换

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 - y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos q & \sin q & 0 \\ -\sin q & \cos q & 0 \\ (1 - \cos q) \cdot x_0 - y_0 \cdot \sin q & (1 - \cos q) \cdot y_0 + x_0 \cdot \sin q & 1 \end{bmatrix}$$

对于复合变换问题，关键是将其分解为一定顺序的基本变换，然后逐一进行这些基本变换，最终得到复合变换结果；或者求出这些基本变换矩阵连乘积，亦可得到复合变换。

综上所述，可以证明利用齐次坐标表示方法，二维图形几何变换矩阵的一般变换过程为：

$$[x' \ y' \ 1] = [x \ y \ 1] \cdot T_{2D} = [x \ y \ 1] \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix}, \text{ 其中 } T_{2D} \text{ 称为二维几何变换得一般}$$

表达式时，进一步可分为以下四个矩阵：

$T_1 = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ 对图形进行比例、旋转、对称等变换；

$T_2 = [l \ m]$ 对图形进行平移变换；

$T_3 = \begin{bmatrix} p \\ q \end{bmatrix}$ 对图形做投影变换。

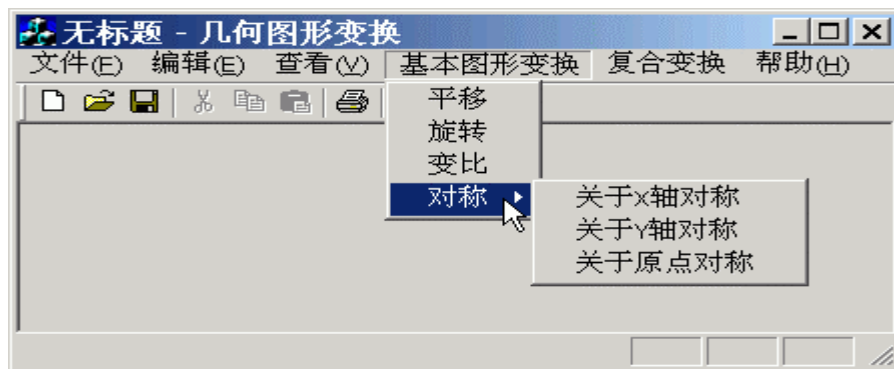
$T_4 = [s]$ 产生整体比例变换。

2.2.3 几何变换程序设计案例

1、程序设计功能说明

[业搜](http://www.yeaso.com)---www.yeaso.com

程序为二维几何变换的应用程序，实现以上介绍的各种二维几何图形的变换。如图 2-2 所示为程序运行时的主界面，通过单击菜单中【几何变换】与【复合变换】及下拉菜单的各功能项完成分别各种几何变换。



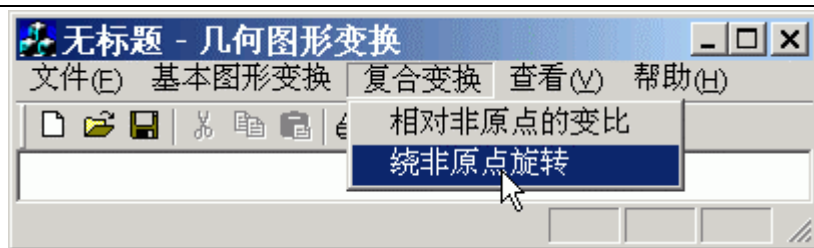


图 2-2

2. 程序设计步骤

(1) 创建工程名称为“几何图形变换”的单文档应用程序框架（详细过程请参见第二章有关内容）

(2) 编辑菜单资源

设计如图 3-1 所示的菜单项。在工作区的【ResourceView】标签中，

单击 Menu 项左边 “+”，然后双击其子项 IDR_MAINFRAME，并根据 2.1 表中的定义编辑菜单资源。创建如图 2-3 所示的程序框架。

表 2.1 菜单资源表

菜单标题	菜单项标题	本图形变换	复合变换	标识	对称ID	帮助(H)
几何图形变换 resource	平移				ID_TRANSLATION	
Accelerator	旋转				ID_ROTATION	
Dialog	对称				ID_SCALING	
Icon	变比					
Menu	关于 X 轴				ID_MIRROR_X	
IDR_MAINFRAME	关于 Y 轴				ID_MIRROR_Y	
String Table	关于 原点				ID_MIRROR_O	
基本图形变换	对称					
Version	关于 原点					
几何图形变换 resource	相对非原点的变比				ID_SCALINGXY	
Accelerator	绕非原点旋转				ID_ROTATIONXY	
Dialog						
Icon						
Menu						
IDR_MAINFRAME						
String Table						
Toolbar						
Version						

图 2-3

(3) 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 2.2 建立如下的消息映射函数，完成有关的函数声明。

表 2.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_TRANSLATION	CONMMAN	OnIDTRANSLATION
ID_ROTATION	CONMMAN	OnIDROTATION
ID_SCALING	CONMMAN	OnIDROTATION
D_MIRROR_X	CONMMAN	OnIDMIRRORX
ID_MIRROR_Y	CONMMAN	OnIDMIRRORY

ID_MIRROR_O	CONMMAN	OnIDMIRRORO
ID_SCALINGXY	CONMMAN	OnIDSCALINGXY
ID_ROTATIONXY	CONMMAN	OnIDROTATIONXY

（4）手工添加几何变换绘图基类

为程序模块化，我们对绘制几何图形变换编制了基类 `MyClass` 类。在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 C/C++ Header File，在【文件】名称输入栏中输入“`MyClass`”；同样，在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 C++ Source File，在【文件】名称输入栏中输入“`MyClass`”。在工作区中系统自动创建的相应的空文件

中，分别添加以下此基类的头文件（.h 文件）和应用文件(.cpp 文件）。

[程序代码见纸书](#)

提示：

在以上 CMyClass 基类中，绘制了字符 M，本例的几何图形]变换应用案例，即对此字符进行各种几何变换，读者可在此基类基础上，对一些函数如 ReadWorkpiece()、DrawViewV()、Calculate()中的部分代码稍加修改，即可绘制各种二维图形，并进一步对所绘制图形进行各种几何变换。

(5) 在几何图形变换 View.h、几何图形变换 View.cpp 添加完成各个菜单消息处理函数，实现既定功能，

说明：下面仅列出几何图形变换 View.h、几何图形变换 View.cpp 的全部代码，其它文件代码全部为系统自动建立，无需改正，为节省篇幅此处省略，详见光盘。下面代码中黑体部分为手工输入，其它代码为系统生成。

// 几何图形变换 View.cpp : implementation of the CMyView class

程序代码见纸书}

2.4 平面曲线图

无论在美术中还是在工程设计中，利用基本图形变换方式绘制的二维图形和工程图都被广泛应用。日常生活中的美术图案（如纺织品图案、装饰图案等）大多是将一个图案单元或几个图案单元经过图形变换、组合排列从而形成一幅美丽的艺术图案。图形或称图案设计在现代生产和生活中也越来越受到人们的重视，其应用范围也越来越广。尤其利用计算机进行图案设计前景更广，计算机绘制图案速度快、线条清晰，也可绘制很复杂的图案，如函数曲线图，随机图、各种简单图形经二维几何变换后的美丽图案等。下面我们详细介绍用 Visual C++ 绘制几种图形或称为图案的程序设计实例

2.4.1 正叶线

正叶线是一种类似植物叶子形状的曲线，数学表达式：

$$\begin{aligned} r &= a \sin(n \times \theta) \\ x &= r \cos(\theta) & (a > 0, n = 2, 3, 4, 5, \dots) \\ y &= r \sin(\theta) \end{aligned}$$

程序经过下面通过修改数学表达式，绘制由二重、三重的正叶线组成的美丽图形。

2.4.2 正叶线蝴蝶结

蝴蝶结故名词义即类似蝴蝶结的图案。是经过设置二维坐标 x, y 函数以及利用旋转变换绘制的图案。画线点的 x 坐标、 y 坐标移下列函数规律变化，即：

```
d=80;
```

```
for(a=0;a<=2*pi;a+=pi/360) {
```

```
  e=d*(1+0.25*sin(4*a));
```

```
  e=e*(1+sin(8*a));
```

```
  x2=int(320+e*cos(a+pi/8));
```

```
  x1=int(320+e*cos(a));
```

```
y1=int(200+e*sin(a));  
  
y2=int(200+e*sin(a+pi/8));  
  
}
```

2.5 平面曲线程序设计案例

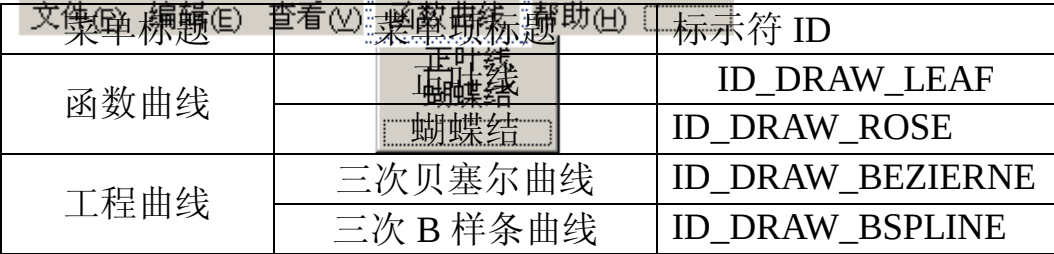
一、程序设计步骤

1. 创建工程名称为“平面曲线”的单文档应用程序框架（详细过程请参见第二章有关内容）。
2. 菜单资源

[业搜](http://www.yeaso.com)---www.yeaso.com

设计如图 2-4 所示的菜单项，根据 2.3 表中的定义编辑菜单资源。

表 2.3 菜单资源表



菜单标题	菜单项标题	标示符 ID
函数曲线	正叶线	ID_DRAW_LEAF
	蝴蝶结	ID_DRAW_ROSE
工程曲线	三次贝塞尔曲线	ID_DRAW_BEZIERNE
	三次 B 样条曲线	ID_DRAW_BSPLINE

图 2-4

3. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 3.2 建立如下的消息映射函

数，ClassWizard 会自动完成有关的函数声明。

表 3.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_DRAW_LEAF	CONMMAN	OnDrawLeaf()
ID_DRAW_ROSE	CONMMAN	OnDrawRose()

4. 在 View.h、View.cpp 添加完成各个菜单消息处理函数，实现既定功能。

说明：下面仅列出 View.h、View.cpp 中需要手工添加代码的部分（黑体部分），其它文件代码全部为系统自动建立，无需改正，为节省篇幅此处省略，详见光盘。

```
// 平面曲线图 View.h : interface of the CMyView class
```

```
////////////////////////////////////
```

```
class CMyView : public CView
```

```
{
```

2.5 课后练习

1. 请写出二维图形几何变换矩阵的一般表达式，并说明其中各个子矩阵的变换功能。
2. 已知四边形各顶点坐标分别为 $(0, 0)$ 、 $(20, 0)$ 、 $(20, 15)$ 、 $(0,$

15) 对此图形编程分别实现如下比例变换:

(1) 使长度方向延长一倍, 高度方向缩小一倍;

(2) 整个图形放大两倍;

3. 已知三角形各顶点坐标分别为 $(10, 10)$ 、 $(10, 30)$ 、 $(30, 15)$ 对此图形分别进行如下比例变换, 写出变换矩阵, 并画出变换后图形。

(1) 沿 x 正向平移 20, 沿 y 正向平移 10, 再绕原点旋转 90° 度;

(2) 绕原点旋转 90° 度, 再沿 x 正向平移 20, 沿 y 正向平移 10;

第三章 交互技术

交互式绘图是指设计者在操作计算机系统绘图时，人与计算机之间可进行信息交换，从而完成复杂的绘图任务。利用一般高级语言绘制图形，为了在屏幕上显示图形，首先需要编制相应的绘图程序，一个绘图程序画出一个图形，需要修改时，必须先修改程序或根据要求再重新编写程序来实现。计算机绘图软件要求通过人机交互式方式进行图形绘制、修改，交互技术计算机绘图软件使用更方便、直观。目前几乎所有绘图应用软件都为人机交互式方法，如 **Autodesk** 公司出品的 **AutoCAD**、**PhotoShop**、**3DMAX** 等等。交互式

绘图已成为编制计算机绘图软件必不可少的特点。**Visual C++**方便的创建界面功能使编制交互式绘图程序变得相当简单。

交互式绘图系统中应具用数据结构的一些数据文件，这些文件中保存着构造图形的几何信息和拓扑信息、属性信息、以及一些非几何数据信息等。图形应用程序是系统中的核心部分，它从应用数据/模型中取得所需要的几何数据及属性信息，按照要求对数据进行变换处理，生成图形，并在图形输出设备上（屏幕、打印机）输出图形，图形的生成、修改、编辑等操作用户可利用人机交互绘图软件、图形输入设备（如键盘、光笔、鼠标等）进行控制输入，绘制需要的图形。

实现软件的交互技术首先应理解以下问题：如何设计一个好的用户接

口、为什么要定义逻辑输入设备、交互式绘图技术有哪些？

3.1 用户接口设计

用户接口决定用户与计算机如何进行信息交换的技术。

用户接口包括用户通过什么途径与图形系统进行联系，通过什么手段来操作系统的功能实现等。用户接口最重要的就是高效率和对用户的友好性。

用户接口一般需要处理用户模型、屏幕的有效利用、用户的反馈、设计的一致性原则、回退和出错处理、联机帮助等。

1. 用户模型

用户模型

(**User Mode**) 是用户接口设计的基础，它提供给用户有关他所处理的对象以及作用于这些对象的处理过程的一个概念性模型。

2. 显示屏幕的有效利用为使屏幕得到有效利用软件设计中应考虑如下几个问题:

(1) 信息显示的布局合理性。

(2) 充分而又正确地使用图符(应用图符(**application icons**)、控制图符(**control icons**))

(3) 恰当地使用各种表示方法进行选择性信息显示。**3. 反馈**反馈就是动态地显示系统运行中所发生的一些变化,以便更有效地进行交互作用 **4. 设计的一致性原则**一致性原则是指在设计系统的各个环节时,应遵从统一

的、简单的规则，保证不出现例外和特殊的情况。还应考虑按照用户认为最正常、最合乎逻辑的方式来设计，减少用户记忆量。 **回退和出错处理**

回退（**undo**）机制包括取消机制、确认机制等。对可能导致错误的一些动作进行预测，设计好的诊断程序，提供出错消息等。**约束机制**：动作与对象相一致

6. 联机帮助

为用户提供**联机帮助(On-Line Help)**措施，能在用户操作过程中的任何时刻提供请求帮助。适应不同的用户，提供多种方法使软件能适应不同熟练程度的用户。

3.2 逻辑输入设备与输入处理

3.2.1 逻辑输入设备

为了使图形软件包独立于具体的硬件设施，图形输入命令常不涉及具体的输入设备，而只涉及该命令所需的数据。根据图形输入信息的不同性质，**PHIGS** 和 **GKS** 将各种图形输入设备从逻辑上分为六种：1. **定位设备**典型方法是定位屏幕光标，把屏幕上的光标移到所需要的位置上，然后按下鼠标，同志计算机取出该处的坐标值，再在此位置构件图形或进行其它操作。定位设备有鼠标器、操纵杆、跟踪球、空间球、数字化仪的触笔或手动光标等。交互时绘图时，一般需要定位，定位就是提供当前绘图的所在位置的

屏幕坐标。在绘制图形时，用户构造图形或对图形进行修改时都需要频繁地定位。

2. **笔划设备** 笔划设备的输入等于多次调用定位设备，产生一系列的坐标值，根据产生的坐标值可产生多边形和曲线等

3. **定值设备** 定值设备常用来输入各种参数和数据。

4. **字符串设备** 字符串设备即进行字符串输入

5. **选择设备** 选择设备用来选择菜单选项、属性选项和用于构图的对象形状等。

6. **拾取设备** 用拾取技术拾取一个图形对象。处理好设备的拾取技术常利用以下方法：在图形对象生成时就对每一个对象确定其拾取优先级、采用依次对拾取图形设立标志的办法、• 找距离最近的对象优先拾取。

3.2.2 输入模式输入模式即如何管理、控制多种输入设备进行工作。常用的输入模式有请求 (**request**)、采样 (**sample**)、事件 (**event**) 及其组合形式等几种。

1. 请求方式 (request mode) 输入设备在应用程序的控制下工作, 程序在输入请求发出后一直被置于等待状态直到数据输入。

2. 取样方式 (sample mode) 此时, 应用程序和输入设备同时工作, 当输入设备工作时, 存储输入数据, 并不断地更新当前数据, 当程序要求输入时, 程序采用当前数据值。

3. 事件方式 (event mode) 每次用户对输入设备的一次操作以及形成的数据叫做一个**事件(Event)**。

思想: 一般一个事件发生时, 往往来不及进行处理, 于是, 就要把事件

按先后次序排成队列，以便先进先出，即先到的事件进入排队，先被取出进行处理。当某设备被置成事件方式，程序和设备同时工作。**4. 输入方式的组合使用**一个应用程序同时可在几种输入模式方式下应用几个不同的输入设备来进行工作。

3.3 交互式绘图技术

交互技术指使用输入设备进行输入的技术。下面介绍一些常用的基本交互绘图技术，这些技术可作为设计应用系统用户接口的基本要素包括：定位技术、橡皮条技术、拖拽技术、菜单技术、定值技术、拾取技术、网格与吸附技术定位技术

定位操作是图形输入和图形操作中常用的输入操作之一。定位有直接定位和间接定位两种方式。直接定位是指使用定位设备直接在屏幕上指定一个点的位置；间接定位是指通过定位设备的运动控制屏幕上的映射光标来进行定位。

2. 约束

约束即在图形绘制过程中对图形的方向、对齐方式等进行规定和校准。

3. 拖曳技术

拖曳技术是将形体在空间移动的过程动态地、连续地表示出来，直到用户满意的结果为止。这种技术常用于部件的装配、模拟现实生活中的实际过程。

4. 橡皮筋技术

橡皮筋技术即针对输入要求，动态地、连续地将输入过程表现出来，直到产生用户满意的输入结果为止。

5. 定值技术

定值技术在交互过程中应用很多，而且是必不可少的。用户经常需要输入一个数值，指定一个数量，完成这种任务需要确定精度（单位），需要的设备是键盘或电位计。定值技术有两种：键入数值、改变电位计阻值产生要求的数量。

6. 菜单技术 菜单是一种很重要的交互技术。它可用于指定命令、确定操作对象或选定属性等多中选一的场合。使用菜单可较好地改善应用系统的用户接口的友善性。包括以下几个方面：菜单的层次结构、菜单的

表示、菜单的显示控制、菜单的选择。**7. 拾取技术** 在图形系统交互作用的许多操作中，常常要在一个分层的对象结构或虽不分层但很复杂的对象结构中拾取一个基本对象（如最底层的对象或一个简单的部分）或一些基本对象的集合（如非最底层的对象），然后对其施加某种操作。拾取一个基本的对象可以通过以下一些方法来实现：指定名称法、特征点法、外接矩形法、分类法、直接法。**8. 网格与吸附** 网格化是帮助绘制整齐、精确图形的一种技术。网格化一般用在用户坐标系统中，按从用户坐标系统的窗口到屏幕视口的变换映射到屏幕上去。网格一般是规则的，且覆盖整个显示区。应用程序将定位器坐标舍入到最近的网格交叉点上去，从而使绘制的图形规

整、精确。有时要从已有的某线段上的点或它的顶点开始绘制另一条线段或其他图形，直接使用定位设备来定位很难保证其重合性。吸附技术则可克服上述困难。

3.4 交互技术程序设计案例

一、程序设计功能说明

演示用 **Visual C++**编写图形程序时基本交互绘图技术的实现：、定位技术，文本输入、拖曳技术等。如图 **3-1** 所示为程序运行时的主界面，用鼠标选择不同的菜单及下拉菜单实现：鼠标画线（橡皮条技术）、利用拖曳技术

绘制矩形、文本输入以及鼠标在客户区移动同时在状态栏动态显示鼠标经过各点的坐标值等功能，如图 3-2 所示。



图 3-1

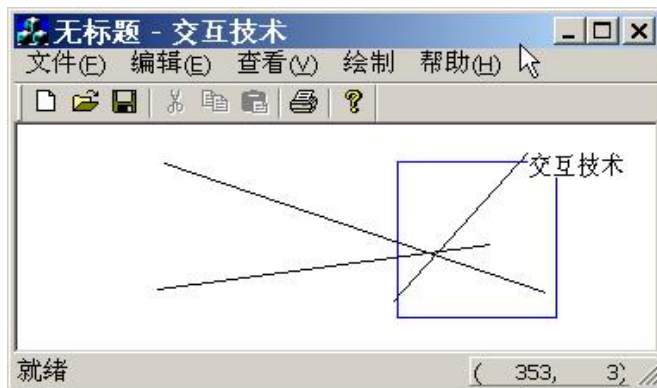


图 3-2

二、程序设计步骤

1. 创建工程名称为“交互技术”单文档应用程序框架（详细创建过程请参见第一章“基本图形的生成”应用程序）

2. 编辑菜单资源

根据 3.1 表中定义编辑菜单资源，设计如图 3-1 所示的菜单项。

表 3.1 菜单资源表

菜单标题	菜单项标题	标示符 ID
基本图形变换	直线	ID_DRAW_LINE
	矩形	ID_DRAW_RECT
	标注文本	ID_DRAW_TEXT

3. 添加消息处理函数

利用 **ClassWizard** (建立类向导) 根据表 3.2 为应用程序添加与菜单项相关的消息处理函数建立如下的消息映射函数。

表 3.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_DRAW_LINE	CONMMAN	OnDRAWLINE
ID_DRAW_RECT	CONMMAN	OnDRAWRECT
ID_DRAW_TEXT	CONMMAN	OnDRAW_TEXT
D_MIRROR_X	CONMMAN	OnIDMIRRORX
ID_MIRROR_Y	CONMMAN	OnIDMIRRORY
ID_MIRROR_O	CONMMAN	OnIDMIRRORO
ID_SCALINGXY	CONMMAN	OnIDSCALINGXY
ID_ROTATIONXY	CONMMAN	OnIDROTATIONXY

4. 在几何图形变换 **View.h**、几何图形变换 **View.cpp** 添加完成各个菜单消息及鼠标响应等处理函数，实现既定功能，

// 几何图形变换 **View.cpp** : implementation of the CMyView class

CMyView::CMyView()

{

程序代码见纸书.

}

说明:

1. 以上仅列出几何图形变换 **View.h**、几何图形变换 **View.cpp** 的全部代

码，其它文件代码全部为系统自动建立，无需改正，为节省篇幅此处省略，详见光盘。代码中黑体部分为手工输入，其它代码为系统生成。

2. 进行文本标注时，在点击功能菜单项之前应先在客户区中选中一点作为文本标注的起始位置。

3. 捕捉所有的鼠标输入介绍

在用鼠标交互绘制一个图形元素时，在已经开始绘制的情况下(如绘制一条直线，在暗中了直线的起点并在屏幕上拖动时)，不希望系统还进行其他操作（如打开菜单执行其它命令）避免造成系统流程和变量初始化等方面的错误。VC++提供两个成员函数来实现以上的功能需要。

SetCapture 函数运行后，时所有后续鼠标输入被送至当前的 CWnd 对象，这样就把所有的鼠标输入采集到，以防系统进行其它程序。ReleaseCapture 与 SetCapture 函数相反，用于释放这种捕捉。这样在开始绘图时，运行 SetCapture 函数，在图形元素交互运行完成后，再运行 ReleaseCapture 函数。

4. 在屏幕上拖动图形

在用鼠标交互绘制图形元素时，为了直观的看到所绘制的图形，一般采用拖动图形的方式。此时一般图形元素只剩最后一个控制点，鼠标拖动改变此控制点形成图形元素的具体形状。产生拖动效果，例如在交互绘制直线

时，在确定了起点后，鼠标移动，拖动一条“橡皮”线移动。

实现拖图形的功能首先要用 CDC 类的函数 `SetRop2` 设置绘制。当把绘制模式设置成 `R2_NOT` 时，因为两次绘制统一图形元素屏幕不便，所以在用鼠标交互绘制图形元素时，在只剩一个控制点时，当鼠标移动时，首先把上一个移动点所决定的图形元素绘制一次（擦除上一个图形元素），再把当前鼠标点决定的图形元素绘制一次，并记录下鼠标点作为下一个移动点时的上一个点，这样当鼠标移动时就实现了拖图形的效果。在编制拖动图形元素的程序时，要注意对第一次移动操作进行处理，因为第一次移动图形元素时，没有所谓“上一个”图形元素要擦除。

3.5 课后练习

1. 一个交互绘图系统的交互任务是什么？
2. 设计交互式绘图系统的主要原则？
3. 编程实现橡皮条绘制直线和圆。
4. 编程实现交互拾取一条直线。

第四章 简单 CAD 绘图系统开发实例

计算机的飞速发展,使计算机已经不再局限于解决数值计算问题,而更多更广泛地使用在非数值计算的各领域。计算机所处理的对象也由纯粹的数值数据发展到诸如字符、图像、声音、信号等各种各样具有一定结构的数据。计算机是一种信息处理装置,而计算机又只能处理数据信息,不能

识别图形。用计算机绘图，就要将图形转化为数据，通过计算机对图形数据进行加工处理，绘制图形。

一个图形系统是有软件和硬件组成的，它对于计算机辅助设计十分重要，它能帮助人们提高工作效率和质量，简化复杂的工作。Visual C++自问世以来一直是 Windows 环境下最主要的应用程序开发系统，由于 Windows 是基于 GUI 的操作系统，而 Visual C++提供了丰富的图形接口（GDI）函数，使得用 Visual C++开发的 Windows 系统下的图形应用程序很方便、简

单。

本章介绍用 Visual C++ 开发图形应用程序的一些知识, 包括: 计算机图形学绘图基础、图形的数据结构, 最后介绍一个简单 CAD 绘图系统编程实例。由于篇幅所限, 本章节在对简单 CAD 绘图系统编程实例的介绍中, 在相应位置, 有针对性地放置相关部分代码, 完整程序代码请参见随本书提供的光盘。

4.1 计算机图形学绘图基础

Windows 应用程序提供一个一般应用程序所需要的全面面向对象软件组建的集成集合。这里应用程序框架是一类库的超级集合，它定义了程序的结构。一个类库是可在应用程序中使用的有关 C++类的集合。例如一个图形类可以支持一般的图形操作。使用类库可通过构建已有类库对象，或派生自己的类来实现。MFC(Microsoft Foundation Class)库是一整套简化 Windows 编程的可重用的类库，MFC 提供如 CString、CFile 等 Windows 编程常用类，封装了通用 Windows API 和数据结构的类。这些数据包括窗口、

控件和设备环境。除此之外，MFC 还提供了应用程序框架，包括组成应用程序继承结构的类，以及对各种 Windows 特性的全面支持。

当我们开发一个大型项目时，同时开发一种结构，但每个人程序的结构往往是不相同的，MFC 库应用程序框架包含自己的应用程序结构，使用 MFC 库编写 Windows 程序，就可以保持值这种一致性，有利于代码的维护和增强。使用 MFC 类库，程序可在任何时候调用 Win32 函数，可以最大程度地利用 Windows。

4.1.1 Visual C++开发系统基本绘图知识

Windows 应用程序框架的核心是文档与视的相互关系, 应用程序框架中的文档和视图结构文档即应用程序处理的数据对象, 如 NotePad 中的文本。视即指文档的应用程序中的表现方式或用户用于改变文档的交互窗口对象。MFC 中与绘图有关系的几个关键类如下:

(1) 文档类 (Document)

文档是用户处理数据的对象。文档即应用程序处理的数据对象, 文档一

般从 MFC 中类 CDocument 中派生, 如果支持 OLE 功能, 可从 ColeDocument 或 ColeServerDoc 类中派生, 由 CDocument 派生的类主要用于存储数据。CDocument 类用于相应数据文件的读取以及存储 Cview 类所需要观察和处理的信息。

(2) 视类 (View)

视相当于文档在应用程序中的观察窗口, 它确定了用户对文档的观察方式和用户编辑文档的方式。如果需要, 用户可在多个视中对文档进行操作,

即多文档结构 (MDI)。对于图形来说视就好比 we 进行绘图工作的画布，对图形的操作都是在视上进行的。因此掌握好视类对我们很重要。

一般情况下，应用程序的视类从 CView 中派生，对于有特殊要求的视，根据情况不同，还可从类 CScrollView、CEditView、CFormView、CTreeView、CListView 或 CRichView 等派生。

另外，视类中有一个重要的成员函数——OnDraw()函数，应用程序中，几乎所有“画”的动作都出现在 OnDraw()中或有它来引发。该函数必须被

重载。重载的 OnDraw()函数要完成两件事，即调用相应的文档的函数获取文档数据和调用 GDI（图形设备接口）的函数在视中画出文档数据。

（3）主窗口类（Main Frame Window）

主窗口是 Windows 应用程序中限定其所有窗口范围的最外边框。应用程序中的所用其它窗口都直接或间接地为主窗口的子窗口，如标准菜单、工具条、状态条等。对于 MDI 程序，视占文档窗口的客户区，而文档窗口又是主窗口的子窗口，一个主窗口可以有多个文档窗口（即含有多个视）；对

于 SDI 应用程序，视是在主窗口中显示的，视占据了主窗口的客户区，主窗口也是文档窗口。

一个 Windows 应用程序一般具有主窗口类。SDI 应用程序的主窗口类应从 CframeWnd 中派生，MDI 程序的主窗口类应从 CMDIFrameWnd 中派生（文档窗口类应当从 CMDIFrameWnd 中派生，一种文档类型应当有一个 CMDIChildWnd 文档窗口派生类）。

（4）文档模板类（Document Template）

文档类模板用于协调文档对象、视对象、和主窗口对象的创建过程。它是从类 `CDocTemplate` 或其派生类中派生的。一个文档模板可以管理同一文档类型的所有文档。对于不同的类型的文档，必须使用不同的文档模板类。

对于 SDI 程序，只包含一个文档模板，它是从 `CsingDocTeplate` 中派生的。对于 MDI 应用程序，所有的文档应从 `CmultiDocTeplate` 中派生的。

(5) 应用类 (Application)

一个应用程序由切只有一个应用类的对象，它控制上述所有的对象，确

定程序的特点，并负责应用程序的初始化和清楚，以便创建和管理程序支持的文档模板对象，一个应用程序对象就代表一个应用程序，当用户启动应用程序，Windows 调用应用程序框架内置的 WinMain 函数，并且 WinMain 寻找一个由 CWinApp 派生的全局构造的应用程序对象，全局对象在应用程序之前构造。

(6) 图形设备接口

PC 相容机种上可以连接许多种不同的视讯设备，所以，GDI 的主要目的

之一是支援与装置无关的图形。**Windows** 程式应该能够毫无困难地在**Windows** 支援的任意一种图形输出设备上执行,**GDI** 通过将您的程式和不同输出设备的特性隔离开来的方法来达到这一目的。图形设备接口 **GDI** (Graphic Device Interface) 管理 Windows 应用程序在窗口中的所有绘图操作和与此有关的诸多方面、。如图形设备的信息,坐标系和映射模式、绘图的当前状态(画笔、画刷、颜色、字体等)、绘图的具体操作(如画线、画圆等)。

一个 Windows 图形设备接口对象类型由一个 MFC 类库表示，这些类有一个共同的抽象基类：CGdiObject。一个 Windows 图形设备接口对象由 CGdiObject 派生类的 C++对象来表示，这些对象有：

CBitmap 位图对象

CBrush 画刷。用于表示区域填充的颜色和样式。

Cpen 画笔用于指定线和边框的性质，如颜色、线宽、线性

等

CRgn 区域是由多个多边形和椭圆组成的组合形状，可以填充、裁剪等操作以及判断鼠标是否位于某一点。

CFont 字体是具有定大小和风格的一套字符集。

CPalette 调色板是一字符映射表，将逻辑颜色和设备的实际颜色相互联系。

(7) 设备环境类

CDC 是 MFC 中最重要的类之一，更是绘图应用程序中最重要的类，CDC

类定义的是设备上下文对象的类。CDC 对象提供处理显示器或打印机等设备上下文的成员函数以及处理与客户区对应的显示上下文的成员，通过 CDC 类的成员函数进行所有的绘图，它封装了 GDI 的大部分内容，如坐标系、绘图方法、各种 GDI 对象的管理等。

设备环境类 CDC 的内容十分丰富，包含了和绘图有关的方方面面。CDC 类提供的成员函数可以用于对设备环境的操作、绘图工具的使用、图形设备接口 (GDI) 对象的选择等，但在使用 CDC 类对象时应注意一个问题：

为使用 CDC 对象, 须先构造一个 CDC 对象, 然后才能调用它的成员函数。使用完成后, 必须在适当的方将其删除, 在 Windows 环境中可获得的设备环境的数量是有限的。如果太多的 CDC 对象没有被删除, 计算机的资源将很快被耗尽, VC++ 也会在调试窗口中报错。

为了特殊应用, MFC 提供了几个 CDC 类的派生类。CpaintDC 类封装了对 BeginPaint 和 EndPaint 的调用, CclientDC 类管理与窗口客户区对应的显示场景。CwindowsDC 类管理与包括主框架和控件在内的设备环境与源文

件。对 CDC 中类的掌握程度直接影响我们绘图程序的质量和效率。

框架大部分的代码是应用程序自动生成的，程序设计的大部分工作是在视类的两个文件（.h 头文件和.cpp 应用文件）中进行的。

4.1.2 计算机图形学会图系统设计基本原则

设计一个图形系统是一个复杂的工作，设计工作一般分为需求分析、设计、编程、测试和运算等几个阶段，每一阶段各有明确任务。

对于一个交互式通用图形系统的设计原则大致如下：

1. 结构层次化、模块化;
2. 要通用性强, 使用方便;
3. 处理速度快;
4. 程序易读、查、改以及移植和扩充。

4.1.3 图形程序设计步骤

编写绘图程序, 其基本要求是可靠、正确和简单清晰。同时再满足结构程序设计模块化要求的前提下, 尽可能做到程序结构合理, 占用内存单元

少,程序运行速度快。另外编出的程序应该满足可读性强、修改调试方便、通用性好,移植性强等要求。编写一个绘图程序,一般需经过以下步骤:

(1) 根据实际需要,确定绘图程序的功能和用途。

设计或编写绘图程序时,首先要明确所编写的绘制程序应具有的功能或说明用途是什么将有利于绘图程序设计具体工作的进行。绘图程序的功能和用途将决定具体编程中下面的一些问题,设置调用函数和参数个数、绘图方式的选择、用户与程序交互信息方式、确定数据存储结构。

(2) 明确所绘制图形的几何形成过程。

在绘制一个图形时，有时需要分析清楚它的几何关系，即弄明白这个图形是怎样形成的，然后才能在此基础上得到绘图的算法，研究图形的几何关系，对于绘制复杂图形是大有帮助的，进而自己也可构造图形。

(3) 根据图形绘制要求对所绘制图形进行系统的模块化分。

算法，即运算方法，它还包括对图形数据（如点的坐标等）的加减乘除四则运算，以及对数据进行的廉洁、变换、循环等操作。绘制一个图形，

一般都要写出绘制算法，最常用的是关于图形上数据的计算公式和各种变量、参数应满足的关系式。计算机通过计算式和关系式等对图形数据进行处理，最后才可绘制出图形。图形算法中体现的严格的数学表示，正确合理的算法式绘图程序能够得到正确结果的前提。

(4) 针对各个模块的功能确定相应绘图算法。

程序总体上为模块化结构，即程序开始为数据说明，然后为各功能函数，最后主程序基本上是由调用函数语句组成，而这些函数语句实际就是一

些功能模块，因此整个程序是模块化结构。

(5) 编写绘图程序。

写程序时，语句要对齐。语句套语句，内套语句应该往后移 3 或 4 个字符，复合语句应与相应的列对齐。同时对变量或参数，要加注释，对函数功能，要做功能简述，做到程序的条理清晰，可读性好。编写的程序还要便于调试。程序每次调试一句，用分号分开的各程序语句，虽然 C++ 允许写在同一行，但不利于调试，若一个程序句一行，可对错误语句准确定位。

将程序分解成合理的模块，即分割成几个甚至很多简单函数，每一个简单函数实现一个单独功能，把各功能模块分别调试好，再组装进程序。同时要更多使用参数，少使用全局变量，

(6) 上机调试、运行和绘图。

程序编号后上机调试。调试程序一般要注意下面几点：

对写好的程序，首先要在纸上进行认真仔细的检查，不要匆匆忙忙上机，程序通过认真检查后再上机调试、编译，初学者容易忽略这一点，在计算

机上占用大量时间。

编译时,可根据提示的信息,找出程序中错误的程序段的位置及错误类型并加以改更。注意有时提示的错误信息位置有误,提示的出错类型也并非绝对准确,也有可能因为错误较多而相互关联,所以要善于分析,真正找到错误所在。当遇到提示的错误信息很多(一大片),往往使人感到问题很严重,事实上也许只是因为所使用的变量未定义,编译时会对所有使用这一变量的语句发出出错信息,这要添加相应变量的定义,所有错误可能都

会消除，另一方面也不要轻易认为程序运行输出一个正确结果就没有问题了。例如，**if** 语句有多个分支，可能在流程经过一个分支时是正确的，而经过另一个分支时可能会出错，因此要考虑全面。

4.1.4 在 Visual C++集成开发环境下程序的调试

- (1) 运行要调试的程序；
- (2) 利用功能键 F9 在需要行设置断点(同样在已设置断点的行上按住 F9 可取消断点)；

(3) 可用功能键 F11 一次执行一条语句，一步步跟踪；可以跟踪进某行中调用的函数，如果不想跟踪进函数，则用功能键 F10。

打开 Watch 窗口（按 Alt+3 组合键）或 QuickWatch 窗口（按 Shift+F9 组合键），也可打开运算窗口或通过主菜单中的 Debug 菜单的选项来打开，观察已经执行过的一段程序中的变量值。在 Watch 窗口中有几个 Watch 子窗口，在每个子窗口的 Name 列任意一行键入变量名后回车后在相应的 Value 列中就得到程序中此变量的值。这样可观察程序运行中变量值的计算

与传递是否正确，常常能找到不容易发现的问题，还可利用窗口值来修改变量的值测试程序对各种情况的

4.1.5 计算机程序结构设计基础

应用程序结构设计，一般包括以下步骤：

1. 编译预处理指令

Visual C++绘图源程序开始部分是编译预处理指令，这是 C 语言的一个特点。编译预处理指令告诉编译系统对源程序进行编译之前应做些什么，

编译预处理指令以#号开头，并与程序语句分开，占用一单独书写行，#号前不能留有空格，指令结尾不用分号。编译预处理指令有三种，即包含文件预处理指令、宏定义预处理指令和条件编译预处理指令。编译预处理指令为程序员提供有效工具，方便编制程序，扩展编程环境，使开发的程序易于阅读、修改和移植到不同的计算机系统上去，编译预处理指令。

(1) 包含文件的预处理指令

包含文件预处理指令的一般格式：

#include <文件名> 指示编译系统按系统设定的目录搜索包含文件;

或 #include “文件名” 表示按指定的路径搜索, 默认值为当前目录。

文件名为磁盘中文本文件的名称。包含文件预处理指令的功能是, 在编译源程序之前, 取代该预处理指令, 也就是从磁盘中读取包含文件, 然后把它写入源程序中预处理指令的位置上, 使之成为源程序的一部分。从而避免程序员的重复性劳动, 提高工作效率。例如:

(2) 宏定义预处理指令

宏定义预处理指令的一般格式：

#define 替换名 字符串 应用程序为便于常量或变量容易记易或修改，常用替代名称代替实际常量或变量。

(3) 条件编译预处理指令：说明语句、主函数、子函数、一般语句、图形初始化语句、绘图语句、程序注释

4.1.6 绘图程序设计基本方法

任何一个图形都是由若干个简单的几何图形构成，而简单图形又是由

点、线、弧等基本的几何元素构成的。用计算机绘图就是要对根据最基本图形信息（点、线、弧、坐标值等）在图形中的几何位置以及相互之间关系进行数学描述，即构造出图形的数学关系式，设计出绘图程序，绘制出图形。图形的数学关系要准确、合理、简洁，绘图程序要求通用、方便、可移植。一般采用模块化设计方法。

4.1.6.1 图形层次结构和程序模块结构

(1) 图形层次结构

一般物体均可分解成许多不同形体元素的集合，即物体可分层来表示。例如，一张床分成床头和床屉不同形体组成，这就是一个一层的结构，在工程设计中，高层与低层之间的关系实际上是物体—子物体、部件-子部件、图形—子图形的关系，这种关系也反映了它们之间的装配、调用关系，那些最底层的形体一般用基本形体来表示。而表示形体的图形是由简单的图素组成较复杂的图形，再由较复杂图形构成更复杂的图形，一层一层地构造下去，得到整个图形。

几何图形或其它形式数据构成的图形一般都有层次结构,利用图形层次结构可以方便地实现模块化的绘图程序设计。

(2) 程序模块结构

程序模块指一个单独的函数,或是逻辑组合在一起的有关函数的集合,或是有关语句的集合。模块化程序设计是指在编写程序中把一个程序分成几个部分,分别编写成函数形式。由于函数的编写、测试可独立进行,从而是的程序的编写和维护更方便,同时又便于编写出优良的程序。这些部

分也就是程序中的模块。

Visual C++绘图程序中主要模块可分为两种：主函数模块和函数模块。一个简单的程序中没有函数，就只有模块，这一模块就是主函数模块。程序中的函数就叫函数模块，它们都是独立的代码，可通过参数接收数据。程序中的每条语句都属于一个特定的模块，C++语言能很好地支持模块化程序。

模块化编程设计有助于计算机绘图程序生成各种各样的图形。图形中一

层子图形可用程序中的一个模块生成，整个图形由程序全部模块生成。例如，现有一画花瓣的模块，将方向、位置各不相同的花瓣组合起来画出一朵花，它是通过调用花瓣模型同时加上方向，位置参数的变化来生成，这朵花又称为另一模块，将方向、位置各不相同的花组合成为花花丛，进一步组成花园。这里按层次构成图形（图像）即花园，其程序模块结构描述为：一个花园图的程序可分为多次调用画花丛的模块；画花丛又可多次调用画花朵的模块；画花朵的程序由可多次调用画花瓣的模块，一幅画花园的图形

就这样通过调用模块化出来了。

4.1.6.2 面向对象程序设计

面向对象程序设计与传统的结构程序设计方法不同，代表了新的程序设计方式，使计算机问题更符合人类自身的思维方式和方法，提高了软件的重用性和扩充性，并能有效控制软件的复杂性和软件维护的开销。

图形软件的设计中，将一些基本的图素，如点、线、面、弧等定义成基本的图素类，图素的操作则可分别通过定义这些相应的数据成员和成员函

数 来 提 供 。

MFC类库中也提供了许多基本图素类，面向对象程序设计更好地支持图形的层次结构，即问题在不同层次上的抽象。在面向对象的程序设计中，是通过类的继承机制来实现。例如将图素类的直线定义为一个基类，即可从这一基类导出折线类、多边形类等第一级导出类，由第一级导出类又可产生圆类、椭圆类、圆弧类等第二级导出类。再往下还可导出 B 样条曲线类、NURBS 曲线类等等。这样就可产生一个非常清楚的类的体系结构，有利于

用户的掌握和使用，同时也可很方便地从其中的某一个类中导出特殊需要的类。所以在进行程序设计，搞清楚所涉及的问题以及体系结构的逻辑关系和层次关系，便于将程序中多次出现的、具有共性的部分提出来成为类，通过提取类的方法来解决复杂问题。

4.1.6.3 绘图子程序和主程序

1. 绘图子程序

编写绘图程序时，首先需要编制一定数量的绘图子程序，根据其功能不

同，绘图子程序由以下几类：

(1) 基本子程序

在编写绘图程序时，把绘制点、直线、圆弧及各种线型、字符等绘图程序成为子程序，称为基本子程序。必要时可利用 MFC 类库提供的基本子程序。

(2) 功能子程序

把绘图过程中经常要用到的绘制三角形、四边形、圆、椭圆等预先编写

成子程序，在编制绘图程序时调用这些子程序进行绘制，这些子程序又是通过调用基本子程序编写的，称为功能子程序，又称二级子程序。必要时也可利用 MFC 类库提供的功能子程序。

(3) 应用子程序

利用上面两种子程序开发编写一些比较复杂的程序，能满足绘图需要、通用性相对较小的程序，称为应用子程序，又称三级子程序，可为某一类程序设计所引用。

因此，在编写大型绘图程序中，一般需按照模块化程序设计方法的要求，把复杂程序分成几个较易调试的子程序即模块来编写。对于更大更复杂的程序，可画出绘图程序的模块结构图，明确程序由那些模块组成然后编写每个模块的程序，一个模块就是一个子程序，程序设计中尽可能采用模块子程序，以至最后的程序由调用各个模块子程序构成。

2. 绘图主程序

绘图主程序即程序执行体。在模块子程序（函数）编写好的基础上，编

写主程序（主函数）较容易，有时只需调用各个模块子程序即可构成主程序，这是典型的模块结构程序。主程序中还可采用人机对话形式，即根据运行时提示图形的几何参数或结构参数由用户一次输入，这是编写较为通用的绘图程序常用的方法编写绘图程序要采用模块子程序和主程序。如果一个几百条语句以上的程序，逻辑关系复杂，整个程序又没有用模块子程序（即函数），这样的程序可能会运行很长时间也不出结果，千万不要以为系统出现问题。若将同样程序划分为一定功能模块的几个子模块然后通过

主程序调用来运行，这样将很快得到结果，这是采用主程序和子程序带来的效率。

4.1.6.4 绘图方法

在编程绘图时，根据所绘图形的复杂程度不同可采用边计算边画图的方法或采用先计算后画图的方法，一般绘制平面图采用前者，绘制复杂曲面图形采用后者。计算机绘制图形又可分为以下几种：解析法、样条法、变形法、拼合法、和创造法。

1. 解析法

根据图形的解析表达式或参数表达式，编写绘图程序时，先计算出图形中各点的坐标值等，然后用绘制函数绘出其图形。这种方法的关键是要将图形用解析式表示，同时如何用解析式取图形上的点、区多少个点都很重

要。这将影响绘图质量，主要是图形的光滑程度。

解析式是常用方法，特别是绘制一般几何图形时。

2. 样条法

当一个物体或图形不是用解析式表示，或者不能用解析精确表示时，往往是用物体或图形的一些实际数据值亦称型值点，构造曲面或曲面拟合它，或者用样条曲线来逼近它，同时通过不断调整、修正样条曲线或曲面，绘制出这些图形。用样条法绘制图形，是工程中绘制自由曲线和曲面的实用方法。

3. 变形法

变形法是对基本图形或称单元图形施行各种几何变换（比如平移、对称、旋转等）从而形成新的或更复杂的图形，如对基本图形矩形进行多次比例、平移变换即可绘制一座小屋。变形法是计算机绘制图形的重要方法之一，

它可使绘制者不必逐笔、逐线、逐个形体进行绘制，而只需找出图形之间的内在关系，对基本图形施行各种几何变换，重新组合排列便可获得所需图形。

4. 拼合法

即将图形分解成若干个基本图形元素（简称图素），把相同部分的图素编写成通用的子程序，绘制图形是可根据实际需要有所不同，但一般要考虑独立性、常用性和可组合性。

5. 创造法

一般绘制图形，是按照一个物体（或景物）绘制出它的图形或是根据原图形绘制出图形。创造法绘图与此不同，实现没有图形或物体，而是设计出绘图算法创造性地绘制图形。例如分型图，预先不能想象出算法所绘制出的图形样子，只有程序运行完后才能知道图形的全貌。

创造法绘制图形为图形领域开辟了另一个美妙的新世界。

一般说图形处理程序要比一般非图形程序复杂，但只要在开始时在头脑中就建立起完好的构思结构及清晰的有关概念，便可克服绘图程序设计中可能出现的混乱现象。

(1) 坚持到底

4.2 图形的数据结构

图形是由点、线、面、体各种几何元素组成的，如何用数据描述图形，图形数据又如何存入计算机，使计算机处理时更准确、方便、完整、迅速地描述图形，这就需要研究图形数据结构。有以上讨论已知一般图形都有

层次结构，任何复杂的图形均可用简单图素来组织描述。而 C++语言具有指针、结构等丰富的数据类型，同时它的面向对象的程序设计方法使图素模块（或绘图模块）之间的关系变得更清晰便于对图形进行修改、删除、插入等操作。

用计算机绘图，就要将图形转化为数据，通过计算机对图形数据进行加工处理，绘制图形。这样用计算机绘图首先要考虑如何在计算机中建立恰

当的模型表示不同图形对象？如何组织图形对象的描述数据以使存储这些数据所要的空间最省，检索、处理这些数据的速度更快。

4.2.1 图形信息的分类

在组织绘制图形过程中会遇到大量信息，这些信息通常被分为：图形信息和非图形信息。

图形信息是图形对象及构成它的点、线、面的位置、相互间关系和几何尺寸等；非图形信息是表示图形对象的线型、颜色、亮度以及供模拟、分析用的质量、比重、体积等数据。

对大量信息的管理，常用概念还有：基本图形元素与段

基本图形元素：图素或图元、体素。用数据来描述，其中图素指可以用一定的几何参数和属性参数描述的最基本的图形输出元素。体素是三维空间中可以用有限个尺寸参数定位和定形的体，常有三种定义形式：

(1)从实际形体中选择出来,可用一些确定的尺寸参数控制其最终位置和形状的一组单元实体

(2)由参数定义的一条(或一组)轮廓线沿一条(或一组)空间参数曲线作扫描运动而产生的形体。

(3)用代数半空间定义的形体。段(也称图段):图形学软件在输出基元和画面之间设置一个中间数据结构,称为图段。图段用规则来描述。图段是由一组输出基元和一组性质构成的集合,图段可以嵌套,图段的常见性质:

可见性、优先度、突出性等。信息中的各个数据元素之间有着一定的结构关系，数据结构就是研究数据的特征以及数据之间存在的关系。

4.2.2 图形数据结构

1. 常用图形数据结构有：线性表、数据和树等逻辑结构以及顺序存储和链表存储等存储结构。

线性表是一种最简单、最基本的线性数据结构，线性表具有相同类型的 n 个 ($n \geq 0$) 数据元素的有限序列，即在线性表中，数据元素之间的相对

位置是线性的，除最后一个数据外，表中其余元素都有且只用一个后继；除第一个外，都有且只有一个前趋。

线性表存储方式有：顺序存储和链表存储。顺序存储是使用一组连续的存储单元一次存储线性表中的各元素。链表是也是一种常见的。链表一般有一个头指针，存放一个地址，该地址指向一个元素，称为节点。一个链表节点包括：数据域和链域，分别用来存放数据和下一个节点的地址指针。链表的最后一个节点不再指向其它元素，称为“表尾”。

数组是线性表的推广，如线性表中的元素是一个单个数据时的数据结构为数组，称为一维数组。当线性表中包含有多个相同描述的数据记录的线性表为多维数组，线性表的长度是可变的，但大小固定。此外还有字符数组等，详细的内容请参考数据结构相关书籍。

2. 数据结构研究主要内容：

- (1) 研究数据元素之间的关系，即逻辑结构。
- (2) 研究数据在计算机内部的存储方式。

(3) 研究如何在数据的各种结构上进行处理, 即算法。

3. 设计图形程序时, 同一图形可以设计不同的数据结构, 但不论采取什么数据结构, 图形数据结构应满足以下要求:

- (1) 能够记录图形的几何和拓扑信息;
- (2) 该数据结构能完全、唯一地描述某一图形;
- (3) 适应对图形进行运算和处理, 边与管理、查找、检索和修改。
- (4) 节省内存。

4.2.3 计算机对数据的管理—数据文件

文件为程序设计中的重要内容，计算机中的数据是以文件形式存在的，文件一般是指存储在外部介质（如磁盘）上数据的集合。一批数据以文件的形式存放在外部介质上，操作系统以文件为单位对数据进行管理，若想要找到存储在外部磁盘介质上的数据，必须先按文件名指定文件，然后再从文件中读取数据。要想向外部磁盘介质上存储数据，也必须先建立一个文

件，才能向外部介质输出数据。

在程序运行时，常常需要将一些数据输出到磁盘上存放起来，需要时再从磁盘输入到计算机内存，即对数据文件进行操作。

Visual C++中把文件看作是一个字符的序列，由一个一个字符的数据按顺序组织起来。根据数据形式，文件可分为 ASCII 代码和二进制文件，ASCII 文件又称为文本（text）文件，其中每一个字节放一个 ASCII 代码，代表一个字符。二进制文件是把内存中的数据阿呢日至数据的各是存储的文件。

用 ASCII 输出的数据与字符一一对应，一个字节代表一个字符，便于对字符进行处理，但占用内存较多，速度慢。用二进制形式输出数值，可节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。

Visual C++对文件的存取是一字符为单位，输入或输出的数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制。在处理文件过程中，系统自动在内存区为一个正使用的文件开辟一个缓冲区，从内存

向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才可送到磁盘，从磁盘向内存读入数据为一次性将一批数据输入到内存缓冲区，再从缓冲区逐个将数据送到程序数据区。

4.2.4 图形数据的存储状态

在编制一个图形系统时，其中最基本也是最重要的是如何组织数据结构和存储方式。数据的存储方式有两类：静态和动态。

静态：即数据不被系统处理时的状态，此时全部数据资料都以外部文件

的形式存储。任何一个成熟的系统都有其自身的文件结构。不同应用程序中的静态文件结构有所不同，一般来说，静态的文件结构有以下两部分：

- (1) 头文件：包含文件中数据的数量、存放格式等所有主控信息；
- (2) 数据部分：数据主体。

一般来说，静态时的数据存储组织相对比较简单，按照确定的组织顺序把各种数据信息放到一个二进制文件中即可。

动态：即数据资料被系统处理时的状态。系统可对数据资料进行增加、

删除、修改、查询等操作。一般静态的文件结构不能满足动态的操作需要。此时系统要求对数据可进行任意使用和重新组织，同时在速度上，要求快速的对数据进行操作。

4.2.5 动态文件数据结构的组织原则

动态文件数据结构的组织原则为足够和可扩容的容量、良好的适应性、较快的速度等，根据系统要求，这些因素共同协调，形成一个有机整体，实现既定目标，而不能厚此薄彼。

下面我们将利用 AppWizard 生成的 Draw 画图应用程序的程序编写过程，讲述在图形系统的设计中，如何组织一般图形元素的数据结构和存储方式。

4.2.6 简单 CAD 绘图系统编程实例中的数据结构

利用 Visual C++ 中面向对象的程序设计和 C++ 的组织方法，组织建立一个基本图形系统的图形元素类。

4.2.6.1 图形元素基类的组织

复杂的图形系统，都是由一些最基本的图形元素组成的。对各种图形元素进行分析，把各类图形元素具有一些相同的属性和操作功能组织存放在一个图形元素基类中，程序中其它一些图形元素再由此基类派生。简单CAD绘图系统编程实例中，在头文件 Draw.Doc.h 中，定义一个图形元素基类 CDraw。代码如下：

```
class CDraw:public CObject //基本图形类，用来存储图形的颜色、线型、  
层等信息
```

```
{
```

```
protected:
```

```
    CDraw(){}    //构造函数
```

```
    short m_ColorPen;    //笔色
```

```
    short m_ColorBrush; //填充刷颜色
```

```
    short m_LineWide;    //线宽（像素）
```

```
    short m_LineType;    //线型（像素）
```

```
short m_Layer;           //所处层

int m_id_only;           //图形元素唯一的识别号

BOOL b_Delete;           //是否处于删除状态

public:

    CDraw(short ColorPen,short ColorBrush,short LineWide,

        short LineType,short Layer,int id_only,BOOL Delete) //构造函数

    {
```

m_ColorPen=ColorPen;

m_ColorBrush=ColorBrush;

m_LineWide=LineWide;

m_LineType=LineType;

m_Layer=Layer;

b_Delete=Delete;

m_id_only=id_only;


```
}
```

```
}
```

```
;
```

说明：在此图形基类中，有两个构造函数，第一个不带参数，第二个构造函数由七个参数，用来初始化类中的成员变量。

1. 直线类 Cline

有了图形元素的基类 CDraw，在此基类的基础上派生一直线类 Cline，直线类的基本参数（线型、线宽、颜色、删出标志等）由基类 CDraw 类中继承，除此之外直线类中的特殊属性（如直线的起点和终点坐标等）则在直线类中定义。在头文件 Draw.Doc.h 中定义直线类 CLine：

```
class CLine:public CDraw    //直线类
{
    代码见纸书 }
```

```
};
```

自此直线类中也重载了两个构造函数，第一没有参数，用来定义一个不带参数的 CLine 直线类对象，另一个带有十一个参数的构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化，并对自己 CLine 类中的四个参数（直线的起点和中的坐标）进行初始化。

2. 连续直线或封闭多边形类组织

连续直线除了具有基类 CDraw 图形元素所有属性外，从图形的几何特

征上连续直线由许多顶点组成，顶点的数目是不确定的，对一条连续直线来说可能只有两个，也可能有几千个顶点，在头文件 Draw.Doc.h 中定义连续直线类 Cpline 派生数据类结构来存储连续直线的一个顶点坐标：

```
typedef struct
```

```
{
```

```
    float x;
```

```
    float y;
```

```
float z;
```

```
}PointStruct;    //存储每个顶点坐标的结构
```

根据连续直线的顶点数目，对于连续直线的顶点坐标，采用动态存储空间的方法，即在 Cpline 对象中分配连续直线的存储空间。在头文件 Draw.Doc.h 中定义连续直线类或多边形区域类 Cpline 如下：

```
class Cpline:public CDraw    //连续直线或多边形区域类
{
```

protected:

int m_Numble; //连续直线或多边形区域的顶点数

BOOL b_Fill; //是否为连续直线

public:

PointStruct* m_PointList; //存储顶点的数组指针

CPLine() //不带任何参数的构造函数

{ m_Numble=0;}

```
CPLine(short ColorPen,short ColorBrush,  
        short LineWide,short LineType,short Layer,int id_only,  
        BOOL Delete,int Numbel,PointStruct* PointList,BOOL Fill)  
:  
CDraw(ColorPen,ColorBrush,LineWide,LineType,Layer,id_only>Delete)  
{  
    m_Numbel=Numbel;
```

```
b_Fill=Fill;

m_PointList=new PointStruct[Numble+1]; //分配空间

if(Numble>0)

{

    for(int i=0;i<Numble;i++)

        m_PointList[i]=PointList[i];

}
```



```
    }  
  
    ~Cpline() //析构函数  
  
    {  
  
        if(m_Numble>0)  
  
            delete m_PointList;  
  
    }  
  
};
```

从 CPline 类的定义中可看出,连续直线或多边形区域的顶点坐标存储在一个结构数组 m_PointList 中,连续直线类中也重载了两个构造函数,第一个没有参数,用来定义一个不带参数的 CPline 连续直线类对象,另一个带有多个参数的构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化,在第二个构造函数中,对顶点数大于 0 的连续直线或多边形动态分配了存储顶点坐标的结构数组。b_Fill 成员变量用来表示图形元素为区分连续直线还是多边形,b_Fill 为真时,表示图形元素为多边

形区域，相反为假时，这个图形为连续直线。

3. 圆类

圆类的定义方法除了具有基类 CDraw 图形元素所有属性外，从图形的几何特征上看，还可以用圆心和半径作为特征参数表示圆的几何特征，在头文件 Draw.Doc.h 中定义圆类 CCircle 派生数据类结构如下：

```
class CCircle:public CDraw //圆及圆形区域类
{
```

[代码见纸书](#) }

};

在圆类中定义了三个成员变量 `m_CircleX`、`m_CircleY`、`m_CircleR` 来记录圆心和半径，同样，利用 `b_Fill` 成员变量用来表示图形元素为区分圆还是圆形区域，`b_Fill` 为真时，表示图形元素为圆形区域，相反为假时，表示图形元素为圆。

此类中也定义了两个构造函数，第一没有参数，另一个带有多个参数的

构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化。上面定义的圆类中用圆心和半径的方法表示圆的几何特征，实际中的图形设计中常采用边界矩形方法表示一个椭圆，而圆只是被看作椭圆的一个特例。

4. 圆弧类组织

圆是圆弧的弧度为 360 度时的特例。圆弧可从圆类派生，在头文件 Draw.Doc.h 中定义圆弧类 CArc 派生数据类结构如下：

```
class CArc:public CCircle    //圆弧类

{

protected:

    float m_Angle1,m_Angle2;    //圆弧的起点和终点角度

    代码见纸书

}

};
```

说明：在圆弧类中定义了两个成员变量 `m_Angle1`、`m_Angle1` 来分别表示一个圆弧的起始和结束弧度。同样，类中也定义了两个构造函数，第一个没有参数，另一个带有多个参数的构造函数中利用前面 `CCircle` 基类中的构造函数的调用对基类圆类和基本图形元素基类中的成员变量进行初始化。

5. 标注文本类

文本标注在图形系统中也为一常用工具，标注文本类除了具有基类 `CDraw` 图形元素所有属性外，还具有位置、字体及标注内容等信息，在头

文件 Draw.Doc.h 中定义标注文本类 CText 派生数据类结构如下:

```
class CText:public CDraw    //标注文本类  
{  
  
    代码见纸书    }  
  
};
```

说明: CText 类中也定义了两个构造函数, 第二个带有多个参数的构造函数中利用前面 CDraw 基类中的构造函数的调用对基本图形元素 CDraw

类中的成员变量进行初始化。

4.2.6.2 组织图形类系统文档

下面讲述利用 MFC 应用程序的文档类管理体系组织文档，实现基本图形系统的文档管理功能。

1. 组织面向对象的文档存储管理机制

面向对象的系统中，数据的管理，即通过文档组织机制，可利用图形元素类创建很多图形元素对象，其中每一图形元素对象作为一个整体来组织

存储空间的分配、保存和读取等功能。通过建立一种存储机制来管理指向所有元素对象的指针，来管理所有图形元素对象。这种文档管理机制具有组织简单、结构化和移植性好等特点，缺点为将占用很大空间。

2. 利用 MFC 模板管理图形元素对象指针的对象

管理一个图形系统文档的思路为：每一个图形元素是图形元素类创建的一个对象，在创建这个对象时得到指向这个对象的指针，通过建立一个对

象指针数组来管理这些指针，达到管理所有图形元素对象的目的。

在 MFC 类中利用类模板 `CTypePtrArray` 来定义一个管理类指针的对象，，例如，定义一个管理 `CLine` 类指针的对象：

其中对象 `m_LineArray` 由第一个参数指向的类(`CObArray`)创建并管理第二个参数指定的类指针，以上代码含义为：对象 `m_LineArray` 是由 `CObArray` 类创建的用来存放 `CLine` 类指针。

针对我们所创建的图形程序在文档类 `CDrawDoc` 中创建的管理各类图

形元素对象指针的 CObArray 对象为:

private:

CTypedPtrArray<CObArray,CLine*>m_LineArray; //管理直线对象指针的
对象

CTypedPtrArray<CObArray,CPLine*>m_PLineArray; //管理连续直线对
象指针的对象

CTypedPtrArray<CObArray,CCircle*>m_CircleArray;//管理圆对象指针的

对象

```
CTypedPtrArray<CObArray,CArc*>m_ArcArray;    //管理圆弧对象指
```

针的对象

```
CTypedPtrArray<CObArray,CText*>m_TextArray; //管理标注文字对象指
```

针的对象

实现文档的管理功能

4.2.6.3 增加图形元素

在上述应用程序中增加一个图形元素（如一条直线），需要进行以下步骤：

- （1）创建一个图形对象，并用图像元素的实际数据初始化这个图形元素对象，
- （2）把指向新创建的图形元素对象的指针，增加到文档类管理图形元素对象指针的对象中
- （3）为实现上述功能，在文档类 CDrawDoc 中定义几个函数，分别来

完成增加各类图形圆的操作功能:

(4) 最后函数返回指向新增图形元素对象的指针。

```
CLine* AddLine(short ColorPen,short ColorBrush,short  
LineWide,short LineType,short Layer,int id_only,float X1,float  
Y1,  
float X2,float Y2);    //增加一条直线  
CPLine* AddPLine(short ColorPen,short ColorBrush,
```

```
short LineWide,short LineType,short Layer,int id_only,int  
Numble,  
  
PointStruct* PointList,BOOL b_Fill);  
  
CCircle* AddCircle(short ColorPen,short ColorBrush,  
short LineWide,short LineType,short Layer,int id_only,  
float CircleX,float CircleY,float CircleR,BOOL Fill);  
  
CArc* AddArc(short ColorPen,short ColorBrush,short
```



```
LineWide,short LineType,short Layer,int id_only,float CircleX,  
float CircleY,float CircleR,BOOL Fill,float Angle1,float Angle2);
```

```
CText* AddText(short ColorPen,short ColorBrush,short  
LineWide,short LineType,short Layer,int id_only,float StartX,  
float StartY,float Angle1,float Angle2,float TextHeight,float  
TextWide,
```

```
float OffWide,unsigned char TextFont,int TextLong,CString);
```

在实现文件 DrawDoc.cpp 中加入函数的实现代码:

```
//添加一个直线对象
```

```
代码见纸书}
```

4.2.6.4 实现各类图形的绘制

利用虚函数来实现各种图形元素的绘制功能,在图形元素的基类 CDraw

中, 抽象定义一个进行绘制操作的虚函数。因为在应用程序中, 不用 CDraw 类直接定义对象, 所以可以将虚函数定义成纯虚函数:

```
virtual void Draw(CDC *pDC,int m_DrawMode,int  
m_DrawMode1,short BackColor)=0;
```

各参数的含义:

pDC: 指向当前绘图对象的指针;

m_DrawMode: 绘制模式。1—R2_COPYPEN, 2—R2_NOT。根据需要可

加入其它的模式。

m_DrawMode1: 直线绘制模式。1—正常显示，2—特殊显示（如鼠标选中时的现实）。根据需要可加入其它的模式。

BackColor: 当参数 **m_DrawMode1** 取 2 时，此参数指定颜色号；

在各种图形元素类中重载虚函数 **Draw**（对于用来创建对象的类，必须有基类纯函数的重载）

Public:

```
virtual void Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short  
BackColor)
```

以下将根据图形元素的绘制方法，在各个图形元素类中加入 Draw 函数的实现代码：

//完成直线的绘制功能

```
void CLine::Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short
```

BackColor)

```
{
```

[代码见纸书](#)

```
}
```

说明:

1.对于连续直线和封闭区域的绘制,主要用到绘图 CDC 类中的 Polygon

成员函数:

Polygon 函数绘制一个多边形，必要时系统会自动把第一个和最后一个顶点连成封闭多边形，用 SetPolyFillMode 设置的填充模式填充。

2.在绘制函数中，根据 CPline 类中的 b_Fill 成员变量的不同，选择不同的操作。

b_Fill 为 TRUE 当前图形元素为一多边形区域，若正常显示 (m_DrawMode==0 || int m_DrawMode1==2) 调用 Polygon 函数绘制一个多边形区域，若为特殊显示 (m_DrawMode1==1)，用连续直线的顶点坐标初

始化一个区域 rgn，用 CDC 类的成员函数 InvertRgn 反色显示这个区域。

b_Fill 为 FALSE，调用 LineTo 函数分段绘制连续的各个直线段。

//圆和填充圆的绘制

```
void CCircle::Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short  
BackColor)  
{  
    if(b_Delete) //如果已经处于删除状态
```


代码见纸书

```
pDC->SelectObject(cjcbakf); //恢复字模  
}
```

4.2.6.5 保存图形数据到文档

用鼠标在屏幕上交互绘制的图形元素，要创建一个图形元素对象并将指

向这个图形元素对象的指针保存起来。即在用鼠标在视图中交互绘制图形元素时，在屏幕上完成图形元素绘制的同时，还需把这一图形元素保存到文档中，否则，重画视图时交互绘制的图形元素就消失了。

用鼠标交互绘制一个图形元素后，要在屏幕上马上显示着以图形元素的实际形态，必须将图形元素以实际的形态重画，因为 VC++ 的 R2_NOT 绘图模式下，线条是彩色的，拓东的图形都不是图形元素的实际颜色，必须对图形元素以实际形态进行重画。重画这个新绘制的图形元素，可通过发

送一个窗口消息使屏幕全部或部分区域重新来实现，但这以方法效果不好，因为要达到使新绘制的图形元素重画的目的，重画的最小区域也包括这一个图形元素的边界矩形（或许为一任意多边形区域），在有大量图形元素的情况下，重画的速度是很慢的，所以一般不采用这种方式。

4.3 简单 CAD 绘图系统功能简介

4.3.1 简单 CAD 绘图系统运行界面

运行简单 CAD 绘图系统得到如图 1-1 的运行主界面。



4.3.2 简单 CAD 绘图系统功能

1. 直线绘制

选择绘制直线操作后，首先用鼠标左键点中直线的起点，此时鼠标移动时就会拖动一条橡皮筋线，再按下鼠标左键就会选中直线的终点而完成直线的交互绘制；按中鼠标右键就会擦除橡皮线放弃直线的绘制。

Visual C++ 中采用 **R2_NOT** 绘制模式(**SetROP2(R2_NOT);**)作用首先将原来的橡皮线擦除，然后绘制直线起点到鼠标点的连线，并将当前鼠标点记为上一个移动点 **mPointEnd**。当鼠标移动时，这一过程一直进行下去，从

而实现拖动橡皮线的功能。

2. 连续直线绘制

选择绘制连续直线操作后，首先用鼠标左键点中连续直线的起点，此时鼠标移动时就会拖动一条橡皮筋，以后继续按下鼠标左键就会增加连续直线的顶点，按下鼠标右键就会完成连续直线的绘制（当顶点数小于 **2** 时，不能绘制成功）。

利用结构数组 **PointXYZ** 来存储已经按下的连续直线顶点的实际坐标。

3. 多边形区域绘制

多边形区域的绘制方法与连续直线相同,不同的是当顶点数目小于**3**时,不能绘制。

4. 圆和圆形区域绘制

圆的绘制用的是圆心半径法。即开始绘制圆的操作后,第一次按下鼠标左键选中圆的圆心,此时鼠标移动时会拖动一个圆,再按下鼠标左键会选中圆上的一点而形成一个圆,按下鼠标右键时会擦除橡皮圆放弃圆的绘制。

5. 圆弧绘制

圆弧绘制用的是三点法。即用起点、经过点、终点三点形成一个圆弧。首先按下鼠标左键选中圆弧的起点，第二次按下鼠标左键选中圆弧的经过点，此时鼠标移动时就会拖动一个圆弧，第三次按下鼠标左键选中圆弧的终点而完成一个圆弧的交互绘制。在绘制过程中，按下鼠标右键会放弃圆弧的绘制操作。

6. 文本标注

当选择进行文本标注操作后，首先用鼠标左键在屏幕上点中要标注文本的左下角起点，点中后会出现如图 **2-1** 所示的对话框，将要标注的文本输入到对话框中的编辑框中，按“确定”退出后，就完成了文本的标注。按“放弃”就会放弃本次标注。

进行标注时，字体的参数靠按下“字体参数”按钮得到。当按下“参数”按钮时，会得到如图 **2-2** 所示的对话框。在对话框中填入字体的信息，“字体高度”和“字体宽度”指的是标注字体的高度和宽度，“字体间距”是指

字体间的距离，它们都是以图形中的实际单位为单位。“标注角度”指的是标注行与 **X** 轴正方向的夹角，以角度表示，逆时针方向为正。“字体旋转角度”指的是标注字体的旋转角度，以角度表示，正常时为 **0**，以逆时针方向为正方向。用户填入相应的参数值，按“确定”将字体参数设置为现在的参数退出，按“放弃”按钮放弃所做的修改退出。

IsOpen 函数用来检查对话框窗口是否建立，**IsVisual** 函数用来检查当前对话框窗口在屏幕上是否可见，**Init** 函数用来初始化对话框类的成员变

量 **m_Text**。

消息处理函数 **DrawText** 由消息 **ID_TEXT_MESSAGE** 激发运行，用来完成在视图中绘制正在标注的文本，当在文本标柱对话框的编辑框中修改标柱文本内容时，这一消息处理函数被调用，将标柱文本的内容会知道屏幕上。



图 2-1 标注信息窗口



图 2-2 标注字体参数设置窗口

7. 图形放大

从“图形操作”菜单中选择执行“图形放大”菜单项，或在工具条中点中“放”按钮，就进入图形放大操作状态，此时需要用鼠标在屏幕上画出一个窗口，窗口中的内容就会放大到整个屏幕上。第一次按下鼠标左键，选择放大窗口的一个顶点，鼠标移动时就会拖动一个矩形移动；第二次按下鼠标左键点中窗口的另一个顶点，系统就会进行图形放大操作，把窗口内的图形放大到整个屏幕中。在放大操作过程中，按下鼠标右键，会放弃

放大操作。

对于图形元素，本系统提供了点选、窗口选择、圆选择、多边形区域选择等多种选择方式。

8. 选中图形元素的特殊显示；

当用鼠标在屏幕上点选图形元素时，通过计算可以判断是否选中了图形元素，同时告诉用户已选中图形元素，这需要把选中的图形用特殊的方式

显示，就用不同的形式重新绘制这一图形元素，在我们的系统中，用虚线显示选中的图形元素。

9. 保留选中的图形元素

在图形元素操作过程中，要进行图形放大、重画等操作，需要保留选中的图形元素，即保留图形元素的数据和视图屏幕。**GraphSelectStruct** 为保留图形元素建立的数据结构。

DrawGraph 函数为屏幕视图重画时，视图重画选中的图形元素。图形元素

的选中操作方法有多种，可以点选，也可以用区域选中。点选图形元素的是当鼠标在视图屏幕上按下鼠标就可得到按下点的屏幕像素坐标，并通过与其它图形元素比较判断这点所在的图形元素，即为选中的图形元素。区域选中使通过判断各种图形元素是否与一个区域相交或包含在一个区域内来实现。区域选中最常用的时窗口选中（矩形区域选中）。本绘制图形系统采用点选的方式选中图形元素

运行菜单“图形选中”下的“点中图形元素”菜单项，或直接点中工具

条的“选”按钮，系统就进入到了点选图形元素状态，此时，用鼠标左键点中图形元素时，就能够选中图形元素。选中的图形元素会用虚线或反色显示。

判断一个点是否在直线上关键是判断一个点与此直线之间的距离是否在有效距离范围之内。

Cline 类的 **IsPoint** 函数的主要操作步骤：

首先判断按中点(**x,y**)是否在直线的边界矩形内，如果在函数返回 **TURE**;

如果按中点在直线的边界矩形内，调用 **PointLine** 函数计算按中点到直线的距离 **xx**，如果距离 **xx** 小于有效距离，即选中了直线，函数返回 **TURE**，否则返回 **FALSE**。

10. 放弃选中

运行菜单“鼠标选中”菜单下的“放弃选中”菜单项，或直接点中工具条上的“清”按钮，系统会放弃对图形元素所做的选中（选中的图形元素恢复正常演示），使选中的图形元素数为 **0**。

放弃选中与选中图形圆素相似，方法包括：一是修改数据，使选中的图形元素为 0。二要在视图中把选中的图形元素恢复到正常显示状态。

11. 从屏幕上删除图形元素

首先用鼠标选中要删除的图形元素，然后选择运行“编辑”菜单下的“删除图形”菜单项，或直接点中工具条上的“删”按钮，系统会把选中的图形元素从图形中删除。

当把选中的图形元素进行删除时，要求在屏幕上实时地把要删除的图形

元素去掉。最简单的办法就是把图形再进行重新绘制，此时作了删除标记的图形元素不再绘制，即从屏幕上消失。利用 **UpdateAllView** 函数进行全屏幕重画；局部重画是只重新绘制要删除的图形元素的边界矩形内的区域，从而达到从视图屏幕上删除图形元素的目的。利用 **InvalidRect(rr,TURE)** 函数。

12. 图形移动

首先用鼠标选中要移动位置的图形元素，然后选择运行“编辑”菜单下

的“图形移动”菜单项，或直接点中工具条上的“移”按钮，系统就进入到移动图形元素操作状态，第一次按下鼠标左键选择一个基点，此时移动鼠标时，会拖动选中的图形元素移动，第二次按下鼠标左键选中目标点，选中的图形元素就会按着这两个点的相对位置进行移动。

以上简单介绍计算机图形学绘图基础、图形的数据结构，和简单 CAD 绘图系统的功能和设计过程，详细介绍请参见有关书籍和随本书提供的光盘中“简单 CAD 绘图系统”应用程序。

第五章 三维图形变换

三维图形的几何变换是指对三维图形的几何信息经过平移、比例、旋转等变换后产生新的图形。三维基本几何变换都是相对于坐标原点和坐标轴进行的几何变换。正如在二维几何变换中提到的那样，用齐次坐标表示点的变换将非常方便，因此在本节中所有的几何变换都将采用齐次坐标进行运算。

5.1 三维图形几何变换矩阵

由于用齐次坐标表示，三维几何变换的矩阵是一个 **4** 阶方阵，其形式如下：

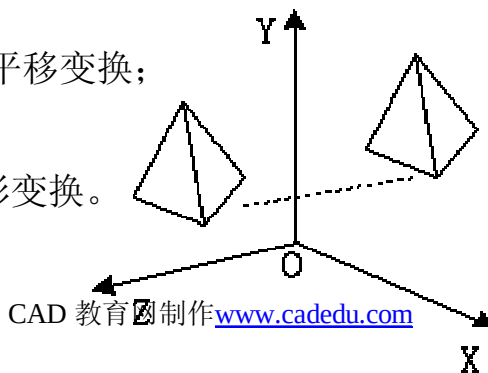
$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \cdot T_{3D} = [x \ y \ z \ 1] \cdot \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ j & h & i & r \\ l & m & n & s \end{bmatrix}$$

T_{3D} 分为四个矩阵,

$$T_1 = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \text{ 对图形进行比例、旋转、对称等变换;}$$

$$T_2 = [l \ m \ n] \text{ 对图形进行平移变换;}$$

$$T_3 = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \text{ 对图形做透视投影变换。}$$



$T_4 = [s]$ 产生整体比例变换。

5.2 三维图形基本变换矩阵

5.2.1 平移变换

三维坐标系中图形的平移如图 5-1 所示。

坐标平移变换: $x' = x + l$ $y' = y + m$ $z' = z + n$

参照二维的平移变换, 我们很容易得到三维平移变换矩阵:

$$[x' \ y' \ z' \ 1] = T \bullet [x \ y \ z \ 1] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} [x \ y \ z \ 1]$$

其中 $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & m & n & 1 \end{bmatrix}$ 为三维平移变换矩阵。

5.2.2 比例变换

$$\begin{aligned} x' &= S_x x \\ \text{坐标比例变换: } y' &= S_y y \\ z' &= S_z z \end{aligned}$$

三维比例变换:

$$\begin{aligned}
 [X' \quad Y' \quad Z' \quad 1] &= [X \quad Y \quad Z \quad 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= [S_x \cdot X \quad S_y \cdot Y \quad S_z \cdot Z \quad 1]
 \end{aligned}$$

相对原点三维图形比例变换矩阵 $T = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

5.2.3 绕坐标轴的旋转变换

考虑右手坐标系下相对坐标原点绕坐标轴旋转 θ 角的变换矩阵。三维基

本旋转变换：绕坐标系 X、Y、Z 轴旋转。

A. 绕 x 轴旋转，

三维变换

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

其中变换矩阵： $T_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\theta > 0$ 右手法则

B. 绕 y 轴旋转

三维变换

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

其中变换矩阵: $T_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\theta > 0$ 右手法则

C. 绕 z 轴旋转

三维变换

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

其中变换矩阵: $T_z(q) = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $q > 0$ 相当于二维逆时针,
是指符合右手法则

D. 绕任意轴的旋转变换

设旋转轴 **AB** 由任意一点 **A** (x_a, y_a, z_a) 及其方向数 (**a, b, c**) 定义,
空间一点 **P** (x_p, y_p, z_p) 绕 **AB** 轴旋转 θ 角到 **P'** (x_p', y_p', z_p'), 如图 5-2
所示, 则:

$$\begin{bmatrix} x'_p & y'_p & z'_p & 1 \end{bmatrix} = \begin{bmatrix} x_p & y_p & z_p & 1 \end{bmatrix} R_{ab}(q)$$

可以通过下列步骤来实现 **P** 点的旋

1. 将 **A** 点移到坐标原点。
2. 使 **AB** 分别绕 **X** 轴、**Y** 轴旋转适当角度与 **Z** 轴重合。
3. 将 **AB** 绕 **Z** 轴旋转 θ 角。
4. 作上述变换的逆操作，使 **AB** 回到原来位置。

所以实现 **P** 点的旋转可写为：

$$R_{ab}(\theta) = T^{-1}(x_a, y_a, z_a) R_x^{-1}(\alpha) R_y^{-1}(\beta) R_z(\theta) R_y(\beta) R_x(\alpha) T(x_a, y_a, z_a) \quad \text{其中各}$$

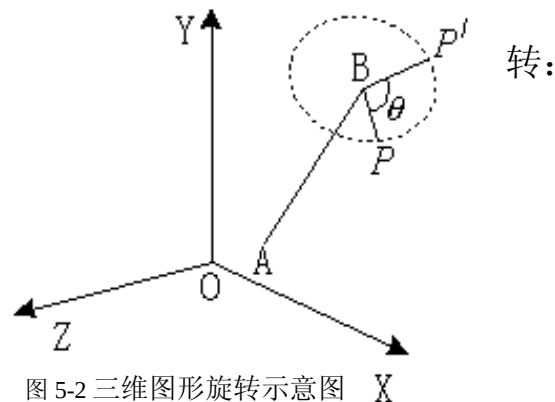


图 5-2 三维图形旋转示意图

个矩阵的形式参照上面所讲的平移、选择矩阵，而 α 、 β 分别是 **AB** 在 **YOZ** 平面与 **XOZ** 平面的投影与 **Z** 轴的夹角。

5.2.4 对称变换

(1) 相对于 **X** 轴对称，坐标变换有

$$\begin{cases} X' = X \\ Y' = -Y \\ Z' = -Z \end{cases},$$

即

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

(2) 相对于 Y 轴对称, 坐标变换有

$$\begin{cases} X' = -X \\ Y' = Y \\ Z' = -Z \end{cases}$$

即

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

(3) 相对于 **Z** 轴对称，坐标变换有

$$\begin{cases} X' = -X \\ Y' = -Y \\ Z' = Z \end{cases}$$

即

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

(4) 相对于坐标原点对称, 坐标变换有

$$\begin{cases} X' = -X \\ Y' = -Y \\ Z' = -Z \end{cases}$$

即

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

(5) 关于三个坐标平面 XY、YZ 及 XZ 的对称变换分别为：

$$T_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2.5 错切变换

图形沿 **X** 轴、**Y** 轴、**Z** 轴方向错切时，其变换矩阵的一般表达式：

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

$$x' = x + dy + gz$$

对应坐标变换表示为： $y' = bx + y + fz$

$$z' = cx + fy + z$$

根据不同的错切方向，三维错切变换矩阵有以下几种。

1. 沿 X 轴方向错切，错切变换矩阵：

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

当 **d=0** 时，错切平面离开 **Z** 轴，沿 **X** 方向沿移动；

当 **g=0** 时，错切平面离开 **Y** 轴，沿 **X** 方向沿移动；

2. 沿 **Y** 轴方向错切，错切变换矩阵：

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

当 **b=0** 时，错切平面离开 **Z** 轴，沿 **Y** 方向沿移动；

当 $h=0$ 时，错切平面离开 X 轴，沿 Y 方向沿移动；

3. 沿 Z 轴方向错切，错切变换矩阵：

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

当 $c=0$ 时，错切平面离开 Y 轴，沿 Z 方向沿移动；

当 $f=0$ 时，错切平面离开 X 轴，沿 Z 方向沿移动；

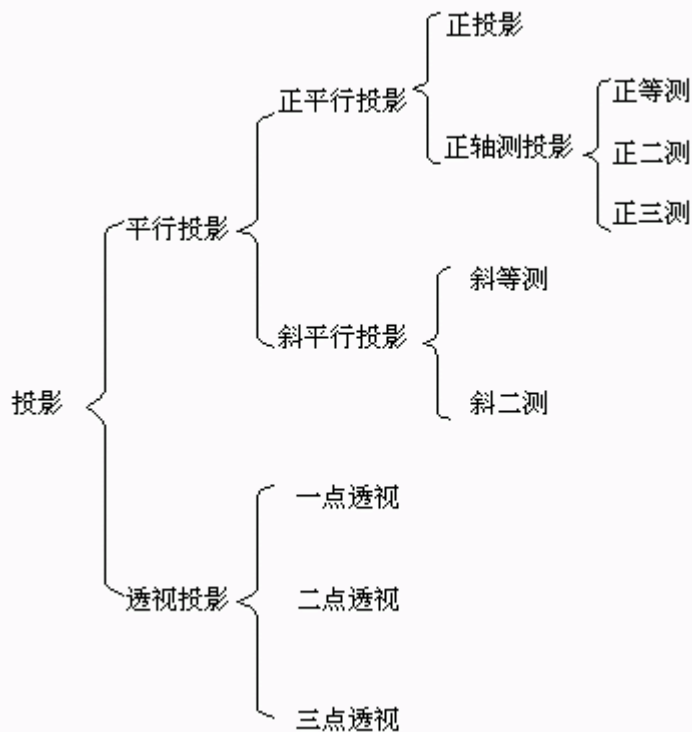
5.3 图形的投影变换

前面我们讨论了二维、三维图形的几何变换，由于显示屏幕是二维的，所以要输出三维形体，需要将三维坐标表示的几何形体变换成二维坐标表示的图形，这就是图形的投影变换。

在三维空间中，选择一个点，记该点为投影中心，不经过这个点再定义一个平面，称该平面为投影面，从投影中心向投影面引出任意条射线，称这些射线为投影线；穿过物体的投影线将与投影面相交，在投影面上形成物体的像，称这个像为三维物体在二维投影面上的投影。这样将三维空间的物体变换到二维平面上的过程称为投影变换。

5.3.1 投影变换分类

投影变换又分为透视投影和平行投影，其主要区别在于透视投影的投影中心到投影面之间的距离是有限的，而平行投影的投影中心到投影面之间的距离是无限的。当投影中心在无穷远时，投影线互相平行，所以，平行投影表示时只给出投影线方向即可，而透视投影要明确指定投影中心的位置。投影变换的分类情况如下表所示：



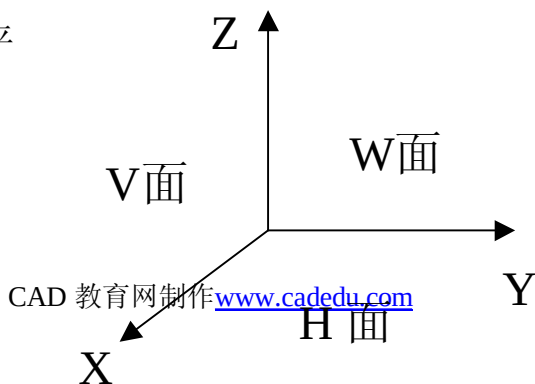
5.3.2 平行投影

平行投影根据投影方向与投影面的夹角分为两类，即正投影与斜投影，当投影方向垂直于投影面时称为正投影，否则为斜投影。

5.3.2.1 正平行投影（三视图）

正平行投影的投影中心到投影面的距离是无限的正投影。正平行投影又包括：三视图和正轴测。

影面之
行投影



1. 三视图

工程中通常将三维坐标系 **OXYZ** 三个坐标平面分为：**H** 面（**XOY** 面）、**V** 面（**XOZ** 面）和 **W** 面（**YOZ** 面），如图 5-3 所示。三维图形在 **V** 面上的投影称为主视图、在 **H** 面上的投影称为俯视图、在 **W** 面上的投影称为侧视图。

(1) 视图 将三维形体向 **xoz** 面（又称 **V** 面）作垂直投影（即正平行投影），得到主视图。主视图变换矩阵：

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

2) 俯视图 三维形体向 **xoy** 面（又称 **H** 面）作垂直投影得到俯视图，

俯视图变换矩阵:

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

3) 侧视图 获得侧视图是将三维形体往 **yoz** 面 (侧面 W) 作垂直投影, 步骤:

侧视图变换矩阵:

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

H 面、V 面、W 面的三个正投影以一定方式摆在某一个平面上——三视图

2. 正轴侧投影变换

若将空间立体绕某个投影面所包含的两个轴向旋转，在向该投影面作正投影，即可得到立体正轴测图。通常选 V 面为轴侧投影面，所以将立体图绕 Z 轴正向（逆时针方向）旋转 θ 角，再绕 X 轴反向（顺时针方向）旋转 ϕ 角，最后向 V 面正投影。因此将绕 Z 轴旋转变换矩阵 T_z ，绕 X 轴旋转变

换矩阵 T_x 和向 V 面正投影变换矩阵 T_v 连乘, 即可得到正轴侧变换矩阵:

$$T_E = T_z \cdot T_x \cdot T_v = \begin{bmatrix} \cos f & 0 & -\sin f \cos q & 0 \\ -\cos f & 0 & -\cos q \sin f & 0 \\ 0 & 0 & \cos f & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(1) 正等侧投影为正轴侧投影中 x 、 y 、 z 三个方向上缩放率相等时的变换, 即 $q = 45^\circ, j = 35^\circ 16'$, 变换矩阵为:

$$T_{\text{正等}} = \begin{bmatrix} 0.707 & 0 & -0.408 & 0 \\ -0.707 & 0 & -0.408 & 0 \\ 0 & 0 & 0.816 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(2) 正二侧投影为正轴侧投影中 x 、 z 三个方向上缩放率相等时的变换，即为： $q = 20^{\circ}42'$ 、 $j = 19^{\circ}28'$ 时，变换矩阵：

$$T_{\text{正二}} = \begin{bmatrix} 0.935 & 0 & -0.118 & 0 \\ -0.354 & 0 & -0.312 & 0 \\ 0 & 0 & 0.943 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.3.2.2 斜平行投影

投影方向不垂直于投影平面的平行投影被称为斜平行投影，如图 5-4 所示，假设 $Z=0$ 的坐标平面为观察平面（H 面），点 (x, y) 为点 $(x, y,$

z) 在观察平面上的正平行投影坐标, 点 (x', y') 为斜投影坐标。 (x, y) 与 (x', y') 的距离为 L 。

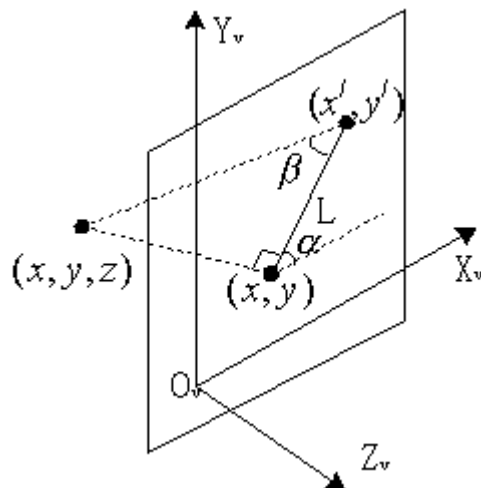


图 5-4 斜平行投影示意图

显然, $x' = x + L \cos \alpha$ $y' = y + L \sin \alpha$

而 L 的长度依赖于 z, β , 即 $\tan \beta = z / L$, $L = z / \tan \beta$

所以
$$x' = x + z \frac{1}{\operatorname{tg} \beta} \cos \alpha \quad y' = y + z \frac{1}{\operatorname{tg} \beta} \sin \alpha$$

令 $l_1 = \frac{1}{\operatorname{tg} \beta}$, 则 $x' = x + z l_1 \cos \alpha \quad y' = y + z l_1 \sin \alpha$,

由此可得斜平行投影的变换矩阵:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ l_1 \cos \alpha & l_1 \sin \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(1)斜等测平行投影, $l_1 = \frac{1}{\text{tg}\beta} = 1, \beta = 45^\circ$

(2)斜二测平行投影, $l_1 = \frac{1}{\text{tg}\beta} = \frac{1}{2}, \beta = \text{tg}^{-1}a$

其中 β 为投影线与投影平面的夹角, α 为投影与 X 轴的夹角。

5.3.2.3 透视投影

透视投影的视线（投影线）是从视点（观察点）出发，视线是不平行的。不平行于投影平面的视线汇聚的一点称为灭点，在坐标轴上的灭点叫做

主灭点。主灭点数和投影平面切割坐标轴的数量相对应。按照主灭点的个数，透视投影可分为一点透视、二点透视和三点透视，如图 5-5 所示。

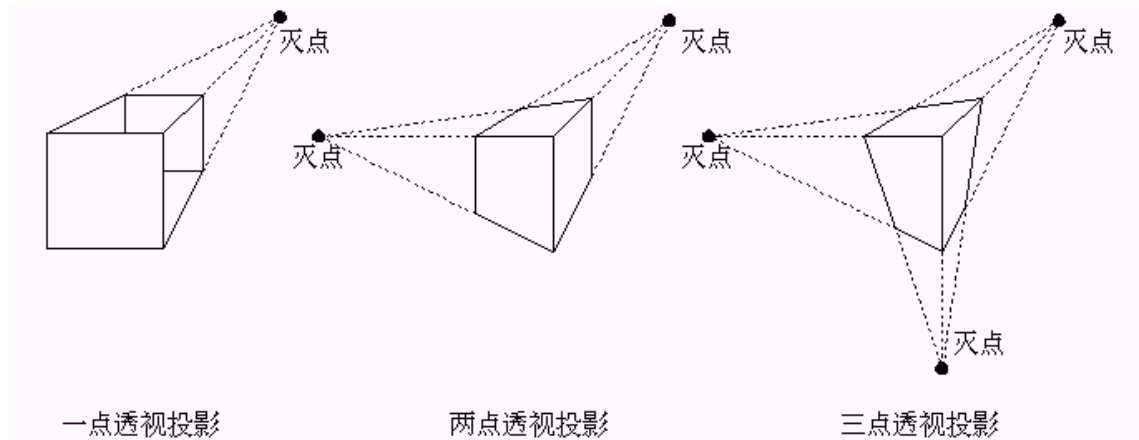


图 5-5 透视投影示意图

1. 一点透视

Z 轴上有一个观察点 **V** (0, 0, h) , 由 **V** 点出发将物体的点 **P** (x, y, z) 投影到 **XOY** 平面上得到(**X**, **Y**, **Z**)。

变换矩阵: 灭点在 **Z** 轴上 (0, 0, -h) ,

$$M_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{h} & 1 \end{bmatrix}$$

灭点在 **X** 轴上 (-h, 0, 0) ,

$$M_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{1}{h} & 0 & 0 & 1 \end{bmatrix}$$

灭点在 **Y** 轴上 (0, -h, 0) ,

$$M_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{h} & 0 & 1 \end{bmatrix}$$

M_x, M_y, M_z 均称为一点变换, 可得到一点透视 (以灭点在 **Z** 轴上为例) 有

$$\begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} = M_s \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \text{ 其中 } H = 1 - \frac{z}{h}$$

2. 二点透视

在变换矩阵中的四行的三个参数起透视变换作用。

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix}, \quad X' = MX$$

若 p , q 二个参数不为零, 则即可得到二点透视,

$$\begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} = M_{pq} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ pX + qY + 1 \end{bmatrix}$$

若固定 Y, Z 令 $X \rightarrow \infty$, 则得灭点为 $(\frac{1}{p}, 0, 0)$;

若固定 X, Z 令 $Y \rightarrow \infty$, 则得另一灭点为 $(0, \frac{1}{q}, 0)$,

即坐标轴上有两个灭点, 因而称为二点透视。

3. 三点透视

同理，可以讨论由三个主灭点的透视变换，其变换矩阵为

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix},$$

三个灭点分别为 $(\frac{1}{p}, 0, 0), (0, \frac{1}{q}, 0), (0, 0, \frac{1}{r})$ 。

透视投影技巧：实际进行透视投影时，为了获得理想的透视投影图，往往先对物体进行旋转和平移，然后再进行透视投影变换。

5.4 三维变换程序设计案例

一、程序设计功能说明

本程序为三维几何变换的实用程序，可实现以上介绍的种三维几何图形变换的绘制。如图 5-7 所示为程序运行时的主界面，通过单击菜单及下拉菜单的各功能项完成分别各种几何变换。

说明：为了使变换结果得更清楚、直观，程序中的几何变换和透视图是对正方体进行的；而投影变换中的平行变换是对类似“椅子”图形进行的变换。





图 5-7

二、程序设计步骤

1. 创建工程名称为“三维变换”单文档应用程序框架
2. 编辑菜单资源

根据 5.1 表中的定义编辑菜单资源，设计如图 5-7 所示的菜单项。

表 5.1 菜单资源表

《计算机图形学原理及算法教程》(Visual C++版) 和青芳 清华大学出版社出版		平移			ID_TRANSLATION
	旋转		绕 X 轴	ID_ROTATION_X	
			绕 Y 轴	ID_ROTATION_Y	
			绕 Z 轴	ID_ROTATION_Z	
	变比		沿 XYZ 变比	ID_SCALING_XYZ	
			整体变比	ID_SCALING_S	
	基本图形 变换	对称	关于 X 轴对称	ID_MIRROR_X	
			关于 Y 轴对称	ID_MIRROR_Y	
			关于 Z 轴对称	ID_MIRROR_Z	
			关于 OXY 平面对称	ID_MIRROR_OXY	
			关于 OYZ 平面对称	ID_MIRROR_OYZ	
			关于 OZX 平面对称	ID_MIRROR_OZX	
			关于原点对称	ID_MIRROR_O	
业搜--- www.yeaso.com		三视图	主视图	ID_V	
			俯视图	ID_H	
			侧视图	ID_W	
			CAD 教育网 www.cadedu.com		
		正轴测图	正等测图	ID_VE	

3. 添加消息处理函数

利用 **ClassWizard**（建立类向导）为应用程序添加与菜单项相关的消息处理函数，**ClassName** 栏中选择 **CMyView**，根据表 5.2 建立如下的消息映射函数。

表 5.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_TRANSLATION	CONMMAN	OnIDTRANSLATION
ID_ROTATION	CONMMAN	OnIDROTATION
ID_SCALING	CONMMAN	OnIDROTATION
D_MIRROR_X	CONMMAN	OnIDMIRRORX
ID_MIRROR_Y	CONMMAN	OnIDMIRRORX
ID_MIRROR_O	CONMMAN	OnIDMIRRORO
ID_SCALINGXY	CONMMAN	OnIDSCALINGXY
ID_ROTATIONXY	CONMMAN	OnIDROTATIONXY

4. 添加三维变换绘图基类

基类 **BaseClass** 类。在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 **C/C++ Header File**，在【文件】名称输入栏中输入“**BaseClass**”；同样，在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 **C++ Source File**，在【文件】名称输入栏中输入“**BaseClass**”。在工作区中系统自动创建的相应的空文件中，分别添加以下此基类的头文件（.h 文件）和应用文件（.cpp 文件）。

```
// BaseClass.h: interface for the CMyClass class.
//
////////////////////////////////////
////////
#endif !defined(AFX_BaseCLASS_H__6250EB80_113B_11D4_81FF_D19FE195501C
__INCLUDED_)
```

```

#define
AFX_BaseCLASS_H__6250EB80_113B_11D4_81FF_D19FE195501C__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define          Scale          1.35

typedef double   array2d[5][5];
typedef double   array[24];

/*****
* 变量说明: Aux1, Aux2, Aux3, Aux4, Aux5, Aux6, Aux7,
* Aux8 是全局变量, 用于存取计算用户坐标系点到观察坐标系点的 *
* 坐标值公式中的正余弦值。X, Y, Z, C, XP, YP, ZP, CP 为一维 *

```

```

* 数组，存放立体顶点齐次坐标，XT, YT, ZT 亦为一维数组，存放 *
* 立体顶点经变换后的坐标值。A, Ah, Aw 二维数组用来接收轴测 *
* 图的变换矩阵与三视图的变换矩阵参数值。 *

```

```

*****/

```

```

class CBaseClass // 定义一个基类
{

```

[程序代码见纸书](#)

```

#endif

```

```

// !defined(AFX_MYCLASS_H__6250EB80_113B_11D4_81FF_D19FE195501C__I
NCLUDED_)

```

说明:

```

* 变量说明: Aux1, Aux2, Aux3, Aux4, Aux5, Aux6, Aux7, *
* Aux8 是全局变量，用于存取计算用户坐标系点到观察坐标系点的 *

```


-
- * 坐标值公式中的正余弦值。X, Y, Z, C, XP, YP, ZP, CP 为一维 *
 - * 数组, 存放立体顶点齐次坐标, XT, YT, ZT 亦为一维数组, 存放 *
 - * 立体顶点经变换后的坐标值。A, Ah, Aw 二维数组用来接收轴测 *
 - * 图的变换矩阵与三视图的变换矩阵参数值。 *

// BaseClass.cpp: implementation of the CMyClass class.

//

////////////////////////////////////

////////

#include "stdafx.h"

#include "三维变换.h"

#include "BaseClass.h"

#include "三维变换 View.h"

#include "math.h"

```
#define PI 3.141592654
```

```
#ifdef _DEBUG
```

```
#undef THIS_FILE
```

```
static char THIS_FILE[]=__FILE__;
```

```
#define new DEBUG_NEW
```

```
#endif
```

```
////////////////////////////////////  
////////  
// Construction/Destruction  
////////////////////////////////////  
////////
```

```
CBaseClass::CBaseClass()  
{  
    ed=2000,eh=100,od=400,h1=1,ps=0;  
}
```

```
CBaseClass::~~CBaseClass()  
{  
  
}
```

```
// 此函数赋轴测图中立体上顶点的齐次坐标值  
void CBaseClass::ReadWorkpiece()  
{
```

```

/*****
* 此函数分别用于三个视图的投影变换，统一用变换后顶
* 点的三个坐标计算公式求其坐标值。这三个公式是由点
* 的齐次坐标乘以变换矩阵得来的。实际上每个视图投影
* 只有二个非零坐标需要计算求得，而另一个坐标是零勿
* 需计算。因此也可以根据三个不同视图的投影，分别采
* 用其投影后的二个坐标计算式来求坐标值。函数中可用
* 条件语句选择不同视图投影采用不同坐标计算式求值。
*****/

```

```

void CBaseClass::Calculate(array2d B)
{
    程序代码见纸书
}

```

代码说明:

以上 **CMyClass** 基类中 **DrawView()** 函数考虑用户坐标到屏幕坐标的变

5. 在几何图形变换 **View.h**、几何图形变换 **View.cpp** 添加完成各个菜单消息处理函数，实现既定功能，

[illegible]

```
#if !defined(AFX_VIEW_H__BFB29846_C242_48D0_8CA7_60B59B46A
728__INCLUDED_)
#define
AFX_VIEW_H__BFB29846_C242_48D0_8CA7_60B59B46A728__INCLUDED_

    程序代码见纸书//      RedrawWindow();

}
```

5.5 课后练习

1. 请写出三维几何变换矩阵,并说明各功能子矩阵作用。
2. 给定一个单位立方体,一个顶点在 $(0, 0, 0)$, 相对另一顶点在 $(1, 1, 1)$, 将单位立方体绕过此两点所连直线旋转 θ 角, 求变换矩阵;

-
3. 编程实现将上述单位矩阵作平移、缩放、和旋转变换。
 4. 设三棱锥各点坐标为 $(0, 0, 20)$, $(20, 0, 20)$, $(20, 0, 0)$, $(10, 20, 10)$, 编程实现三面正投影图;
 5. 编程实现一三棱锥的正等轴测和正二测图形。

第六章 曲线和曲面

6.1 曲线曲面参数表示的基础知识

工程设计中经常绘制各种曲线和曲面，曲线分为规则曲线与不规则曲线，曲面也相应分为规则曲面与不规则曲面。规则曲线有圆锥曲线、圆柱曲线、渐开线等，这些曲线都可用函数或参数方程表示；不规则曲线则是根据给定的离散数据点用曲线拟合逼近得到，常见的有参数样条曲线、

Bezier 曲线、B 样条曲线等，这些曲线采取分段的参数方程来表示。规则曲面常见的有柱面、锥面、球面、环面、双曲面、抛物面等，这些曲面都可用函数或参数方程表示；常见的不规则曲面有 Bezier 曲面、B 样条曲面等，这些曲面采取分片的参数方程来表示。数学上曲线和曲面的表示有多种形式。曲线和曲面的表示方程有参数表示和非参数表示之分。由于参数表示的曲线、曲面具有几何不变性等优点，计算机图形学中通常用参数形式描述曲线、曲面。

6.1.1 非参数表示和参数表示

非参数表示又分为显式和隐式两种表示方式。对于一个平面曲线，显式表示一般形式是： $y=f(x)$ 。在此方程中，一个 x 值与一个 y 值对应，所以显式方程不能表示封闭或多值曲线，例如，不能用显式方程表示一个圆。

如果一个平面曲线方程，表示成 $f(x, y)=0$ 的形式，我们称之为隐式表示。隐式表示的优点是易于判断函数 $f(x, y)$ 是否大于、小于或等于零，也就易于判断点是落在所表示曲线上或在曲线的哪一侧。

对于非参数表示形式方程（无论是显式还是隐式）存在下述问题：

1. 与坐标轴相关；
2. 会出现斜率为无穷大的情形（如垂线）；
3. 对于非平面曲线、曲面，难以用常系数的非参数化函数表示；
4. 不便于计算机编程。

在几何造型系统中，曲线曲面方程通常表示成参数的形式，即曲线上任一点的坐标均表示成给定参数的函数。假定用 **t** 表示参数，平面曲线上任一点 **P** 可表示为： **$P(t)=[x(t), y(t)]$** ；

空间曲线上任一三维点 **P** 可表示为： **$P(t)=[x(t), y(t), z(t)]$** ；最简单的参数曲线是直线段，端点为 **P1**、**P2** 的直线段参数方程可表示为：

$$P(t)=P_1+(P_2-P_1)t \quad t \in [0, 1];$$

圆在计算机图形学中应用十分广泛, 其在第一象限内的单位圆弧的非参数显式表示为:

$$y = \sqrt{1 - x^2} \quad (0 \leq x \leq 1)$$

其参数形式可表示为:

$$P(t) = \left[\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2} \right] \quad t \in [0, 1]$$

在曲线、曲面的表示上, 参数方程比非参数方程有更多的优越性, 主要表现在:

(1) 可以满足几何不变性的要求。

(2) 有更大的自由度来控制曲线、曲面的形状。如一条二维三次曲线的显式表示为:

$$y = ax^3 + bx^2 + cx + d$$

只有四个系数控制曲线的形状。而二维三次曲线的参数表达式为:

$$P(t) = \begin{bmatrix} a_1t^3 + a_2t^2 + a_3t + a_4 \\ b_1t^3 + b_2t^2 + b_3t + b_4 \end{bmatrix} \quad t \in [0,1]$$

有 8 个系数可用来控制此曲线的形状。

(3) 对非参数方程表示的曲线、曲面进行变换，必须对曲线、曲面上的每个型值点进行几何变换；而对参数表示的曲线、曲面可对其参数方程直接进行几何变换。

(4) 便于处理斜率为无穷大的情形，不会因此而中断计算。

(5) 参数方程中, 代数、几何相关和无关的变量是完全分离的, 而且对变量个数不限, 从而便于用户把低维空间中曲线、曲面扩展到高维空间去。

这种变量分离的特点使我们可以用数学公式处理几何分量。

(6) 规格化的参数变量 $t \in [0, 1]$, 使其相应的几何分量是有界的, 而不必用另外的参数去定义边界。

(7) 易于用矢量和矩阵表示几何分量, 简化了计算。

6.1.2 参数表示的基本特征

参数表示的基本形式有：代数形势和几何形式

1. 代数形式

一条曲线段 $Q(t) = [x(t) \ y(t) \ z(t)]^T$, 其中 $0 \leq t \leq 1$, 若表示成 t 的三次多项式,

曲线的代数形式是:

$$\begin{cases} x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x} \\ y(t) = a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_{0y} \\ z(t) = a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z} \end{cases} \quad t \in [0, 1]$$

方程组中 **12** 个系数唯一地确定了一条 **3** 次参数曲线的位置与形状。上述代数式写成矢量式是：

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad t \in [0,1] \quad (6.1)$$

其中 a_0, a_1, a_2, a_3 是代数系数矢量， $P(t)$ 是三次参数曲线上任一点的位置矢量。

2. 几何形式

描述参数曲线的条件有：端点位矢、端点切矢、曲率等。对三次参数曲线，若用其端点位矢 $\mathbf{P}(0)$ 、 $\mathbf{P}(1)$ 和切矢 $P'(0)$ 、 $P'(1)$ 描述，并将 $\mathbf{P}(0)$ 、 $\mathbf{P}(1)$ 、 $P'(0)$ 、 $P'(1)$ 简记为 $\mathbf{P}_0$ 、 $\mathbf{P}_1$ 、 P'_0 、 P'_1 ，代入(6.1)式得（如图 6-1 所示）：

$$\begin{cases} a_0 = P_0 \\ a_1 = P'_0 \\ a_2 = -3P_0 + 3P_1 - 2P'_0 - P'_1 \\ a_3 = 2P_0 - 2P_1 + P'_0 + P'_1 \end{cases} \quad (6.2)$$

将(6.2)代入(6.1)整理后得:

$$P(t) = (2t^3 - 3t^2 + 1)P_0 + (-2t^3 + 3t^2)P_1 + (t^3 - 2t^2 + t)P_0' + (t^3 - t^2)P_1' \quad t \in [0,1] \quad (6.3)$$

$$\text{令: } F_0(t) = 2t^3 - 3t^2 + 1, \quad F_1(t) = -2t^3 + 3t^2, \quad G_0(t) = t^3 - 2t^2 + t, \quad G_1(t) = t^3 - t^2$$

将 F_0, F_1, G_0, G_1 代入(3.1.3)式, 可将其简化为:

$$P(t) = F_0P_0 + F_1P_1 + G_0P_0' + G_1P_1' \quad t \in [0,1] \quad (6.4)$$

(6.4)式是三次

Hermite(Ferguson)曲线的几何

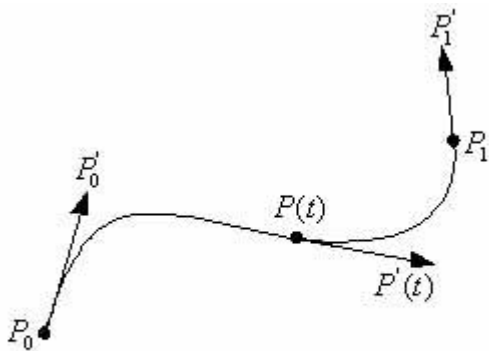


图 6-1 Ferguson 曲线端点位矢和切矢

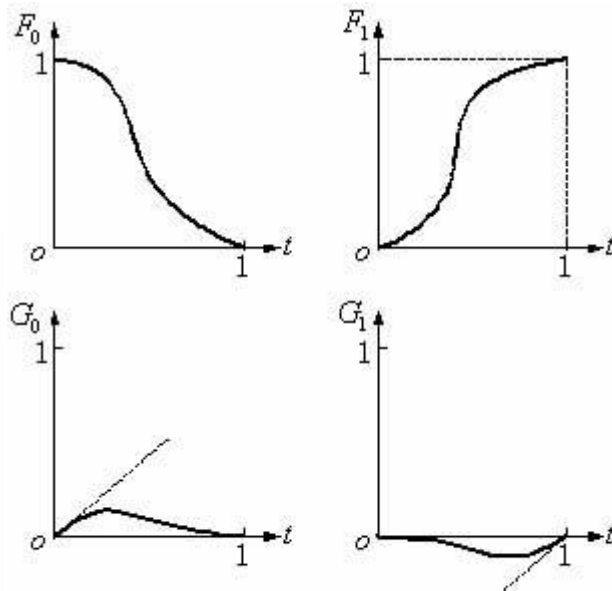


图 6-2 三次调和函数

形式, 几何系数是 $P(0)$ 、 $P(1)$ 、 $P'(0)$ 、 $P'(1)$

F_0, F_1, G_0, G_1 称为调和函数(或混合函数), 即该形式下的三次 **Hermite** 基。

它们具有如下的性质:

$$F_i(j) = G_i(j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

(6.5)

图 7-1-1 Ferguson 曲线端点位矢和切矢

$$F_i(j) = G_i(j) = 0, \quad i, j = 0, 1$$

F_0 和 F_1 控制端点的函数值对曲线的影响, 而同端点的导数值无关; G_0 和 G_1 则控制端点的一阶导数值对曲线形状的影响, 同端点的函数值无关。或者说, F_0 和 G_0 控制左端点的影响, F_1 和 G_1 控制右端点的影响。图 6-2 给出了这四个调和函数的图形。

调和函数不是唯一的, 任何满足(3.1.5)式 C^1 类多项式函数都可以作为调和函数使用, 其中 F_0 和 F_1 必须是单调连续函数。 $F_0(t)=1-t, F_1(t)=t$ 是一次多项式调和函数。

6.1.3 曲线段之间的连续性

设计一条复杂曲线时，常常通过多段曲线组合而成，这需要解决曲线段之间光滑连接的问题。

曲线间连接的光滑度的度量有两种：一种是函数的可微性，把组合参数曲线构造成在连接处具有直到 n 阶连续导矢，即 n 阶连续可微，这类光滑度称之为 C^n 或 n 阶参数连续性。另一种称为几何连续性，组合曲线在连接

处满足不同于 C^n 的某一组约束条件, 称为具有 n 阶几何连续性, 简记为 G^n 。曲线光滑度的两种度量方法并不矛盾, C^n 连续包含在 G^n 连续之中。下面我们来讨论两条曲线的连续问题。如图 6-3 所示, 二条曲线 $P(t)$ 和 $Q(t)$, 参数 $t \in [0,1]$ 的连续性

C^0 连续 (0 阶参数连续) ——前一段曲线的终点与后一段曲线的起点相同。

若要求在结合处达到 G^0 连续或 C^0 连续, 即两曲线在结合处位置连续:

$P(1)=Q(0)$ C^1 连续 (一阶参数连续) —— 两相邻曲线段的连接点处有相同的一阶导数。若要求在结合处达到 G^1 连续, 就是说两条曲线在结合处在满足 G^0 连续的条件下, 并有公共的切矢:

$$Q'(0) = \alpha P'(1) \quad (\alpha > 0) \quad (6.6)$$

当 $\alpha = 1$ 时, G^1 连续就成为 C^1 连续。 C^2 连续 (二阶参数连续) —— 两相邻曲线段的连接点处有相同的一阶导数和二阶导数。

若要求在结合处达到 G^2 连续, 就是说两条曲线在结合处在满足 G^1 连续的前提下, 并有公共的曲率矢:

$$\frac{P'(1) \times P''(1)}{|P'(1)|^3} = \frac{Q'(0) \times Q''(0)}{|Q'(0)|^3} \quad (6.7)$$

代入(6.6)得:

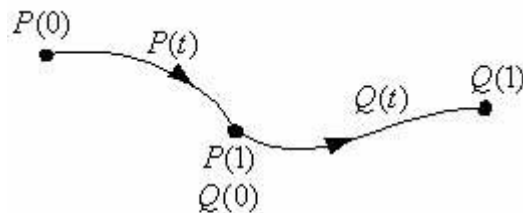
$$P'(1) \times Q''(0) = \alpha^2 P'(1) \times P''(1)$$

这个关系为:

$$Q''(0) = \alpha^2 P''(1) + \beta P'(1) \quad (6.8)$$

β 为任意常数。当 $\alpha = 1$, $\beta = 0$

连续就成为 C^2 连续。



时, G^2

我们看到, C^1 连续保证 G^2 连

连续能保证 G^2 连续, 但反过来

也就是说 C^n 连续的条件比 G^n 连续的条件要苛刻。

图 6-3 两条曲线的连续性

续, C^1

不行。

6.1.4 曲线曲面设计中的几个概念：型值点：是指通过测量或计算得到的曲线或曲面上少量描述其几何形状的数据点。

控制点：是指用来控制或调整曲线曲面形状的特殊点，曲线曲面本身不一定通过控制点。

插值和逼近：这是曲线曲面设计中的两种不同方法。插值设计方法要求建立的曲线曲面数学模型，严格通过已知的每一个型值点。而逼近设计方法建立的曲线曲面数学模型只是近似地接近已知的型值点。

拟合：是指在曲线曲面的设计过程中，用插值或逼近的方法使生成的曲线曲面达到某些设计要求。

6.2 常用参数曲线

6.2.1 一般规则空间曲线

若一条规则空间曲线的函数式以给出，则曲线上一系列点的位置可求出。

1. 球面三叶玫瑰线

极坐标方程为 $r = a \sin(3q)$, 将此已知的平面曲线映射到半球面上, 且半球面的半径为 a , 即可得到空间三叶玫瑰线。

$$X = a \sin(3q) \cos q$$

参数方程为: $Y = a \sin(3q) \sin q$ 令参数 θ 在给定范围内连续取值, 即可求

$$Z = a |\cos(3q)|$$

得空间曲线上的一系列点。

$$X = a \cos q$$

2. 圆柱螺线参数方程为: $Y = a \sin q$

$$Z = kq$$

$$\begin{aligned} X &= q \cos q \\ 3. \quad \text{圆锥螺线参数方程为: } Y &= q \sin q \\ Z &= kq \end{aligned}$$

6.2.2 Bezier 曲线

一. Bezier 曲线定义

贝塞尔曲线是不规则曲线，需要在起点和终点之间构建插值多项式的混合函数，通常由 **n+1** 个顶点定义一个 **n** 次多项式。在给定空间 **n+1** 个点的

位置矢量 $\mathbf{P}_i$ ($i=0, 1, 2, \dots, n$)，则 **Bezier** 参数曲线上各点坐标的插值公式是：

$$P(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0,1]$$

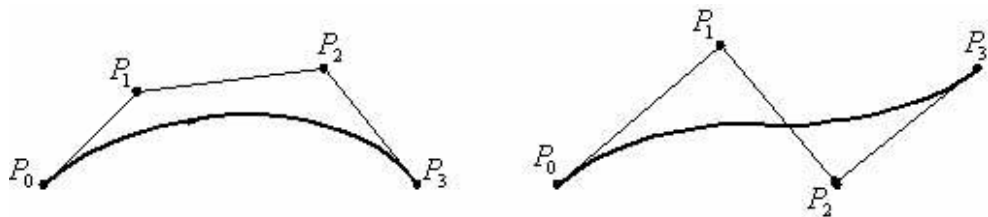


图 6-4 三次 Bezier 曲线

其中， P_i 构成该 **Bezier** 曲线的特征多边形， $B_{i,n}(t)$ 是 n 次 **Bernstein** 基函数：

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i \cdot (1-t)^{n-i} \quad (i = 0, 1, \dots, n)$$

Bezier 曲线实例如图 6-4 所示。

二. **Bezier** 曲线的性质

（1）端点性质

a. 曲线端点位置矢量

当 $t=0$ 时, $P(0)=P_0$; 当 $t=1$ 时, $P(1)=P_n$ 。由此可见, **Bezier** 曲线的起点、终点与相应的特征多边形的起点、终点重合。

b. 端点处的 r 阶导数

$$\text{起点处: } P_{(0)}^r = \frac{n!}{(n-r)!} \sum_{i=0}^r (-1)^{r-i} B_r^i P_i \quad 0 \leq t \leq 1 \quad i = 0, 1, \dots, n$$

$$\text{终点处: } P_{(1)}^r = \frac{n!}{(n-r)!} \sum_{i=0}^r (-1)^i B_r^i P_{n-i} \quad 0 \leq t \leq 1 \quad i = 0, 1, \dots, n$$

一阶导数:

$$\mathbf{t}=0 \text{ 时, } P'_{(0)} = n(P_1 - P_0)$$

$$\text{当 } \mathbf{t}=1 \text{ 时, } P'_{(1)} = n(P_n - P_{n-1}),$$

说明: **Bezier** 曲线的起点和终点处的切线方向和特征多边形的第一条边及最后一条边的走向一致。

二阶导矢:

$$\text{当 } \mathbf{t}=0 \text{ 时, } P''(0) = n(n-1)(P_2 - 2P_1 + P_0)$$

$$\text{当 } \mathbf{t}=1 \text{ 时, } P''(1) = n(n-1)(P_n - 2P_{n-1} + P_{n-2})$$

上式表明：**2**阶导矢只与相邻的**3**个顶点有关，事实上，**r**阶导矢只与(**r+1**)个相邻点有关，与更远点无关。

(2) 对称性。由控制顶点 $P_i^* = P_{n-i}$, ($i = 0, 1, \dots, n$), 构造出的新 **Bezier** 曲线, 与原 **Bezier** 曲线形状相同, 走向相反。因为:

$$C^*(t) = \sum_{i=0}^n P_i^* B_{i,n}(t) = \sum_{i=0}^n P_{n-i} B_{i,n}(t) = \sum_{i=0}^n P_{n-i} B_{n-i,n}(1-t) = \sum_{i=0}^n P_i B_{i,n}(1-t), \quad t \in [0, 1]$$

这个性质说明 **Bezier** 曲线在起点处有什么几何性质, 在终点处也有相同的性质。

(3) 凸包性

由于 $\sum_{i=0}^n B_{i,n}(t) \equiv 1$,

$0 \leq B_{i,n}(t) \leq 1 (0 \leq t \leq 1, i = 0, 1, \dots, n)$, 所以

在 $[0, 1]$ 区间变化时, 对某一个 t

$P(t)$ 是特征多边形各顶点 P_i 的加权

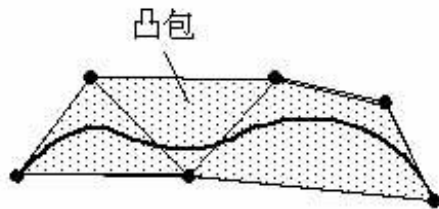


图 6-5 Bezier 曲线的凸包性

当 t
值,
平均,

权因子依次是 $B_{i,x}(t)$ 。在几何图形上, 意味着 **Bezier** 曲线 $P(t)$ 在 $t \in [0,1]$ 中各点是控制点 P_i 的凸线性组合, 即曲线落在 P_i 构成的凸包之中, 如图 6-5 所示。

三. 常用 **Bezier** 曲线的矩阵表示由 Bezier 曲线 $C(u)$ 的定义, 可推出常用的一次、二次、三次 Bezier 曲线矩阵表示

1) 一次 Bezier 曲线

$$C(u) = (1-u)P_0 + uP_1 \text{ 矩阵表示为 } C(u) = [u, 1] \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix}$$

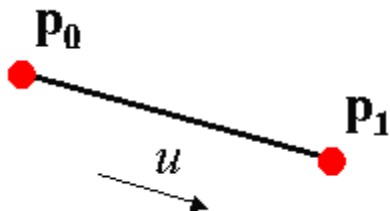


图 6-6 一次 Bezier 曲线

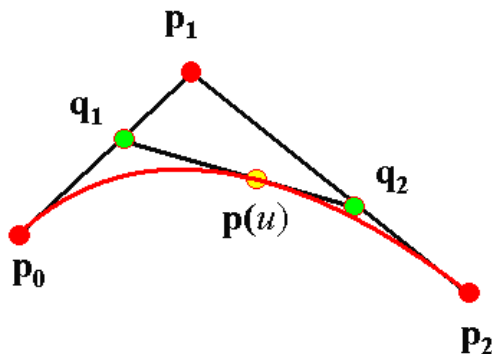


图 6-7 二次 Bezier 曲线

这是一条从 P_0 到 P_1 的直线段，如图 6-6 所示。

2) 二次 Bezier 曲线，如图 6-7 所示。

$C(u) = (1-u)^2 P_0 + 2u(1-u)P_1 + u^2 P_2$ 矩阵表示为

$$C(u) = \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}$$

将 P_0 、 P_1 、 P_2 分解为二维平面上的 X、Y 分量，则：

$$X(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix} = X_0 - 2(X_0 - X_1)t + (X_0 - 2X_1 + X_2)t^2$$
$$Y(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = Y_0 - 2(Y_0 - Y_1)t + (Y_0 - 2Y_1 + Y_2)t^2$$

展开为:

$$X(t) = A_0 - 2A_1 + A_2 t^2$$

$$Y(t) = B_0 - 2B_1 + B_2 t^2$$

式中

$$A_0 = X_0$$

$$A_1 = 2(X_0 - X_1)$$

$$A_3 = X_0 - 2X_1 + X_2$$

3) 三次 Bezier 曲线

$$B_0 = Y_0$$

$$B_1 = 2(Y_0 - Y_1)$$

$$B_3 = Y_0 - 2Y_1 + Y_2$$

$C(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$ 矩阵表示为 :

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

将 P_0 、 P_1 、 P_2 、 P_3 分解为二维平面上的 X 、 Y 分量，则：

$$X(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = X_0 - 2(X_0 - X_1)t + (X_0 - 2X_1 + X_2)t^2$$
$$Y(t) = 1/6[t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = Y_0 - 2(Y_0 - Y_1)t + (Y_0 - 2Y_1 + Y_2)t^2$$

展开为：

$$X(t) = A_0 + A_1t + A_2t^2 + A_3t^3$$

$$Y(t) = B_0 + B_1t + B_2t^2 + B_3t^3$$

式中

$$A_0 = X_0$$

$$A_1 = -3(X_0 - X_1)$$

$$A_2 = -X_0 - 6X_1 + 3X_2$$

$$A_3 = -X_0 + 3X_1 - 3X_2 + X_3$$

$$B_0 = Y_0$$

$$B_1 = -3(Y_0 - Y_1)$$

$$B_2 = -Y_0 - 6Y_1 + 3Y_2$$

$$B_3 = -Y_0 + 3Y_1 - 3Y_2 + Y_3$$

四. **Bezier** 曲线的递推性(de Casteljau 算法)

如图 6-8 所示, 设 P_0 、 P_0^2 、 P_2 是一条抛物线上顺序三个不同的点。过 P_0 和 P_2 点的两切线交于 P_1 点, 在 P_0^2 点的切线交 $P_0 P_1$ 和 $P_2 P_1$ 于 P_0^1 和 P_1^1 , 则如下比例成立:

$$\frac{P_0 P_0^1}{P_0^1 P_1} = \frac{P_1 P_1^1}{P_1^1 P_2} = \frac{P_0^1 P_0^2}{P_0^2 P_1^1}$$

这是所谓抛物线的三切线定理。

当 P_0 、 P_2 固定, 引入参数 t ,

令上

述比值为 $t:(1-t)$, 即有:

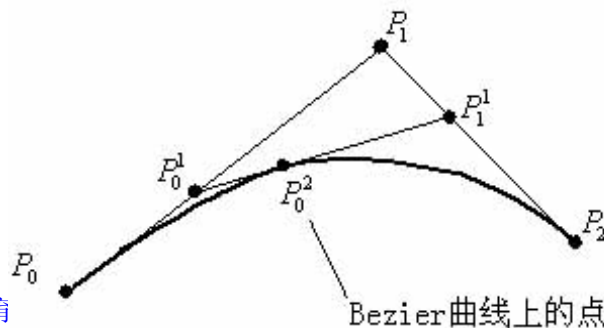


图 6-8 抛物线三切线图示

$$P_0^1 = (1 - t) P_0 + t P_1$$

$$P_1^1 = (1 - t) P_1 + t P_2$$

$$P_0^2 = (1 - t) P_0^1 + t P_1^1$$

t 从 **0** 变到 **1**, 第一、二式就分别表示控制二边形的第一、二条边, 它们是两条一次 **Bezier** 曲线。将一、二式代入第三式得:

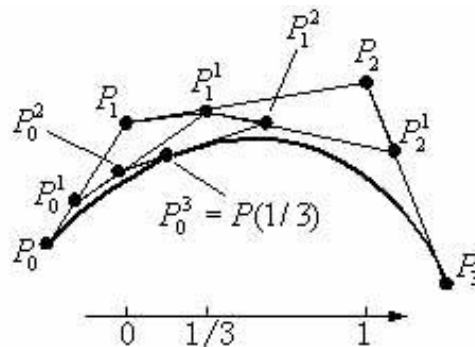
$$P_0^2 = (1 - t)^2 P_0 + 2t(1 - t) P_1 + t^2 P_2$$

当 **t** 从 **0** 变到 **1** 时, 它表示了由三顶点 **P₀**、**P₁**、**P₂** 三点定义的一条二次 **Bezier** 曲线。并且表明: 这二次 **Bezier** 曲线 **P₀²** 可以定义为分别由前两个

顶点 (P_0, P_1) 和后两个顶点 (P_1, P_2) 决定的一次 **Bezier** 曲线的线性组合。依次类推, 由四个控制点定义的三次 **Bezier** 曲线 P^3_0 可被定义为分别由 (P_0, P_1, P_2) 和 (P_1, P_2, P_3) 确定的二条二次 **Bezier** 曲线的线性组合, 由 $(n+1)$ 个控制点 $P_i (i=0, 1, \dots, n)$ 定义的 n 次 **Bezier** 曲线 P^n_0 可被定义为分别由前、后 n 个控制点定义的两条 $(n-1)$ 次 **Bezier** 曲线 P_0^{n-1} 与 P_1^{n-1} 的线性组合:

$$P_0^* = (1-t)P_0^{n-1} + tP_1^{n-1} \quad t \in [0,1]$$

由此得到 **Bezier** 曲线的递推计
公式:



算

图 6-9 几何作图求 **Bezier** 曲线上
一点 ($n=3$, $t=1/3$)

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1, 2, \dots, n, i = 0, 1, \dots, n-k \end{cases} \quad \text{即 de Casteljau 算法。}$$

利用以上递推公式, 在给定参数下, 可求得 **Bezier** 曲线上一点 **P(t)**。上式中的: $P_i^0 = P_i$ 是定义 **Bezier** 曲线的控制点, P_0^n 即为曲线 $P(t)$ 上具有参数 **t** 的点。**de Casteljau** 算法稳定可靠, 直观简便, 是计算 **Bezier** 曲线的基本算法和标准算法。

这一算法可用简单的几何作图来实现。给定参数 $t \in [0,1]$ ，就把定义域成长度为 $t:(1-t)$ 的两段。依次对原始控制多边形每一边执行同样的定比分割，所得分点就是第一级递推生成的中间顶点 $P_i^1 (i = 0,1,\dots,n-1)$ ，对这些中间顶点构成的控制多边形再执行同样的定比分割，得第二级中间顶点 $P_i^2 (i = 0,1,\dots,n-2)$ 。重复进行下去，直到 n 级递推得到一个中间顶点 P_0^n 即为所求曲线上的点 $P(t)$ ，如图 6-9 所示。

五. **Bezier** 曲线的升阶

所谓升阶是指保持 **Bezier** 曲线的形状与定向不变，增加定义它的控制顶点数，也即是提高该 **Bezier** 曲线的次数。增加了控制顶点数，不仅能增加了对曲线进行形状控制的灵活性，还在构造曲面方面有着重要的应用。对于一些由曲线生成曲面的算法，要求那些曲线必须是同次的。应用升阶的方法，我们可以把低于最高次数的曲线提升到最高次数，而获得同一的次数。曲线升阶后，原控制顶点会发生变化。

设给定原始控制顶点 $P_0, P_1, \dots, P_n$ ，定义了一条 n 次 **Bezier** 曲线，曲线提升一阶后的新的控制顶点。

$$P(t) = \sum_{i=0}^n P_i B_{i,n}(t) \quad t \in [0,1]$$

增加一个顶点后，仍定义同一条曲线的新控制顶点为 $P_0^*, P_1^*, \dots, P_{n+1}^*$ ，则有：

$$\sum_{i=0}^n C_n^i P_i t^i (1-t)^{n-i} = \sum_{i=0}^{n+1} C_{n+1}^i P_i^* t^i (1-t)^{n-i}$$

对上式左边乘以 $(t + (1-t))$ ，得到：

$$\sum_{i=0}^n C_n^i P_i t^i (1-t)^{n+1-i} + t^{i+1} (1-t)^{n-i} = \sum C_{n+1}^i P_i^* t^i (1-t)^{n+1-i}$$

比较等式两边 $t^i (1-t)^{n+1-i}$

系数，得到：

化简即得：

项的

$$P_i^* C_{n+1}^i = P_i C_n^i + P_{i-1}$$

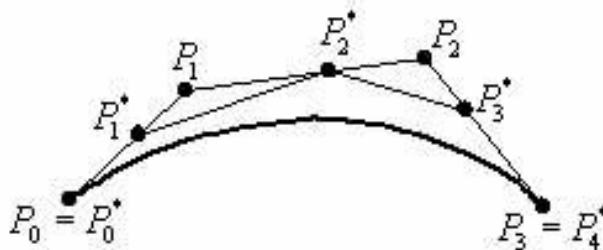


图 6-10 Bezier 曲线升阶

$$P_i^* = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i \quad (i = 0, 1, \dots, n+1) \quad \text{其中 } P_{-1} = P_{n+1} = 0。$$

此式说明：

- 新的控制顶点 P_i^* 是以参数值 $\frac{i}{n+1}$ 按分段线性插值从原始特征多边形得出的。
- 升阶后的新的特征多边形在原始特征多边形的凸包内。

- 特征多边形更靠近曲线。

三次 **Bezier** 曲线的升例，如图 6-10 所示。

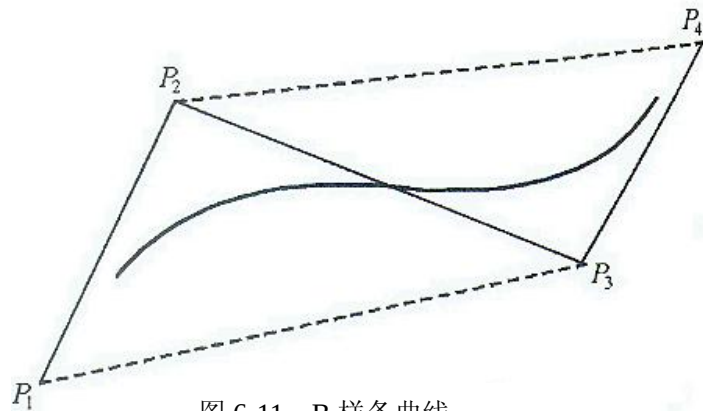


图 6-11 B 样条曲线

6.2.3 B 样条曲线

以 **Bernstein** 基函

阶实

数构

造的 **Bezier** 曲线或曲面有许多优越性，但也有许多 **Bezier** 曲线或曲面不能作局部修改等不足。1972 年，Gordon、Riesenfeld 等人提出了 B 样条方法，在保留 **Bezier** 方法全部优点的同时，克服了 **Bezier** 方法的弱点。

一. B 样条曲线定义

设 $P_0, P_1, \dots, P_n$ 为给定空间的 $n+1$ 个控制顶点， $U = \{u_0, u_1, \dots, u_m\}$ 是

$m+1$ 个节点矢量：称下列参数曲线 $C(u) = \sum_{i=0}^n P_i N_{i,p}(u)$ $a \leq u \leq b$ 为 p 次的 B 样

条曲线，折线 $P_0, P_1, \dots, P_n$ 为 B 样条曲线的控制多边形。次数 p ,

顶点个数 $n+1$, 节点

个数 $m+1$ 具有如下关

系: $m = n + p + 1$ 。二. **B**

样条曲线的性质: **1.**

严格的凸包性: 曲线严

格位于控制多边形的凸

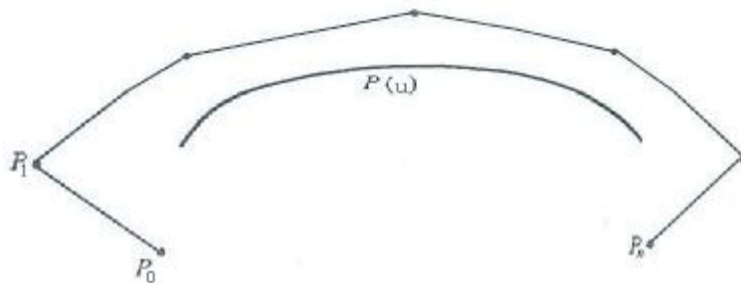


图6-12 **B** 样条曲线凸包性

包内；如果 $u \in [u_i, u_{i+1})$ $p \leq i < m - p - 1$ $C(u)$

位于控制顶点 $P_{i-p}, \dots, P_i$ 所建立的凸包内，如图 6-12 所示。

2.分段参数多项式： $C(u)$ 在每一区间 $u \in [u_i, u_{i+1})$ 上都是次数不高于 p 的多项式；

3.可微性或连续性： $C(u)$ 在每一曲线段内部是无限次可微的，在定义域内重复度为 k 的节点处则使 $p - k$ 次可微或具有 $p - k$ 阶参数连续性；

4.几何不变性: B样条曲线的形状和位置与坐标系的选取无关。

5.局部可调性:

因为 $N_{i,p}(u)$ 只在区间 $[u_i, u_{i+p+1})$ 中为正, 在其它地方均取零值, 所以 p 次的B样条曲线在修改时只被相邻的 $p+1$ 个顶点控制, 而与其它顶点无关。当移动其中的一个顶点 p_i 时, 只影响到定义在区间 $[u_i, u_{i+p+1})$ 上那部分曲线, 并不对整条曲线产生影响。

6.近似性:

控制多边形是 B 样条曲线的线性近似，若进行节点插入或升阶会更加近似；次数越低，B 样条曲线越逼近控制顶点；

7.变差缩减性:

如图 6-13 所示，设 $P_0, P_1, \dots, P_n$ 为 B 样条曲线的控制多边形，某平面与 B 样



条曲线
的交点
个数不

图 6-13 B 样条曲线的变差缩减性

多于该平面与其控制多边形的交点个数。

例子：给定控制顶点 $P_i (i=0, \dots, 8)$ ，定义一条三次B样条曲线。这说明 $n=8$ ， $p=3$ ，各种关系如下确定：

1. 节点矢量 $U = [u_0, u_1, \dots, u_{n+p+1}] = [u_0, u_1, \dots, u_{12}]$ 2. 曲线定义域 $u \in [u_p, u_{n+1}) = [u_3, u_9)$ 3.

当定义域 $[u_3, u_9)$ 内不含重节点时，曲线段数 $= n - p + 1$

$= 6$; 4. 当由 $[P_{i-p}, \dots, P_i] = [P_3, \dots, P_6]$ 四个控制顶点定义，与其他顶点无关

5. 移动 p_3 时将至多影响到定义在 $[u_i, u_{i+p+1}) = [u_3, u_7)$ 区间上那些曲线段的形状

6. 在 $[u_6, u_7)$ 上的三次 B 样条基及计算定义在上那段三次 B 样条曲线将涉及 $u_{i-p+1} = u_4, \dots, u_{i+p} = u_9$ 共 6 个节点。三. 重节点对 B 样条曲线的影响节点的非均匀或非等距分布包含两层含义:

(1) 节点区间长度不等;

(2) 重节点, 即节点区间长度为零。 1) 重节点的重复度每增加 1, 曲线段数就减 1, 同时样条曲线

在该重节点处的可微性或参数连续阶降 1 ;

- 2)当定义域端点节点重复度为 p 时, p 次B样条曲线的端点将与相应的控制多边形的端顶点重合, 并在端点处与控制多边形相切;
- 3)当在曲线定义域内有重复度为 p 的节点时, p 次 B 样条曲线插值于相应的 控制多边顶点;
- 4)当端节点重复度为 $p+1$ 时, p 次B样条曲线就具有和次Bezier曲线相同的端点几何性质; 5) p 次B样条曲线若在定义域内相邻两节点都具有重复度 p , 可以生成定义在该节点区间上那段B样条曲线的Bezier点;

6)当端节点重复度为 $p+1$ 时, p 次B样条曲线的定义域仅有一个非零节点区间, 则所定义的该 p 次B样条曲线就是 p 次Bezier曲线;

四. B样条曲线类型的划分

曲线按其首末端点是否重合, 区分为闭曲线和开曲线。闭曲线又区分为周期和非周期两种情形, 周期闭曲线与非周期闭曲线的区别是: 前者在首末端点是 C^2 连续的, 而后者一般是 C^0 连续的。非周期闭曲线可以认为是开曲线的特例, 按开曲线处理。

B 样条曲线按其节点
节点的分布情况, 可划分
类型。假定控制多边形的
 $P_i (i = 0, 1, \dots, n)$, 阶数为
为 $k-1$), 则节点矢量是
 $T = [t_0, t_1, \dots, t_{n+k}]$ 。

(1) 均匀 B 样条曲线

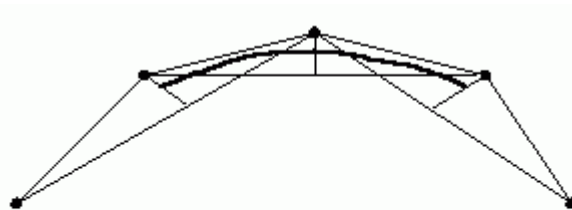


图 6-14 三次均匀 B 样条曲线

矢量中
为四种
顶点为
 k (次数

节点矢量中节点为沿参数轴均匀或等距分布，所有节点区间长度
 $\Delta_i = t_{i+1} - t_i = \text{常数} > 0 (i = 0, 1, \dots, n+k-1)$ ，这样的节点矢量定义了均匀的
B 样条基。图 6-14 为均匀 B 样条曲线实例。

（2）准均匀的 B 样条曲线

与均匀 B 样条曲线的差别
于两端节点具有重复度 k ，这样

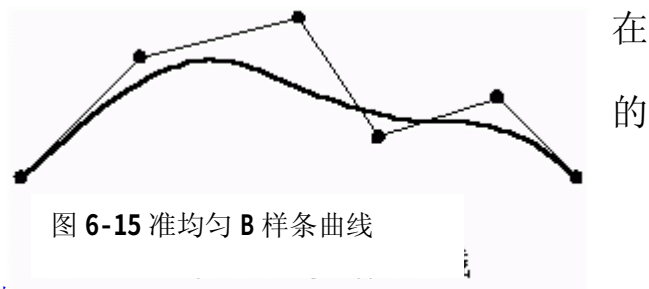


图 6-15 准均匀 B 样条曲线

节点矢量定义了准均匀的 **B** 样条基。

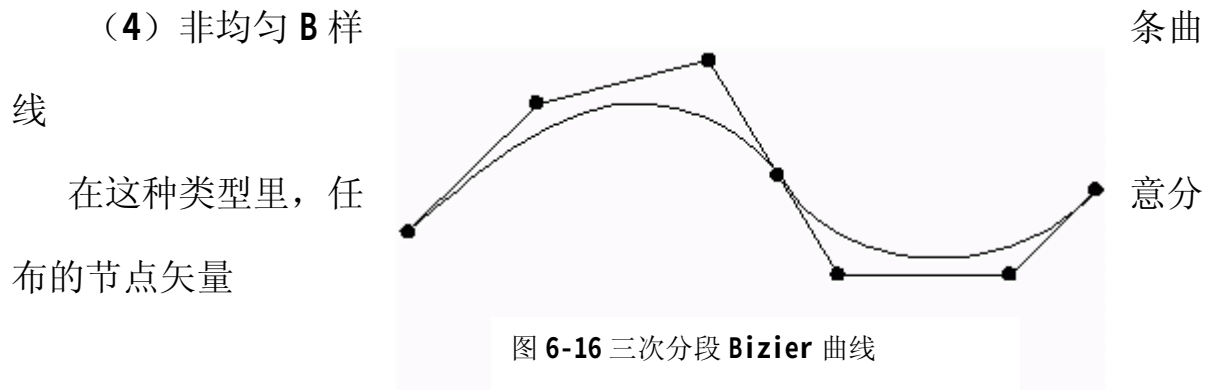
均匀 **B** 样条曲线在曲线定义域内各节点区间上具有用局部参数表示的统一的表达式，使得计算与处理简单方便。但用它定义的均匀 **B** 样条曲线没有保留 **Bezier** 曲线端点的几何性质，即样条曲线的首末端点不再是控制多边形的首末端点。采用准均匀的 **B** 样条曲线就是为了解决这个问题，使曲线在端点的行为有较好的控制，如图 6-15 所示。

（3）分段 **Bezier** 曲线

节点矢量中两端节点具有重复度 k ，所有内节点重复度为 $k-1$ ，这样的节点矢量定义了分段的 **Bernstein** 基。

B 样条曲线用分段 **Bezier** 曲线表示后，各曲线段就具有了相对的独立性，移动曲线段内的一个控制顶点只影响该曲线段的形状，对其它曲线段的形状没有影响。并且 **Bezier** 曲线一整套简单有效的算法都可以原封不动地采用。其它三种类型的 **B** 样条曲线可通过插入节点的方法转换成分段

Bezier 曲线类型，缺点是增加了定义曲线的数据，控制顶点数及节点数都将增加，至多增加将近 **k-1** 倍。分段 **Bezier** 曲线实例如图 6-16 所示。



$T = [t_1, t_2, \dots, t_{n+k}]$, 只要在数学上成立 (节点序列非递减, 两端节点重复度 $\leq k$, 内节点重复度 $\leq k-1$) 都可选取。这样的节点矢量定义了非均匀 **B** 样条基。

五. 均匀 B 样条曲线 这里我们重点讲解均匀 B 样条曲线。在节点矢量中节点为沿参数轴均匀等距分布, 所有节点区间长度 $\Delta_i = u_{i+1} - u_i$ 为大于零的常数; 可将定义在每个节点区间 $[u_i, u_{i+1})$ 上用整体参数 u 表示的 **B** 样条基变

换成用局部参数 $t \in [0,1]$ 表示, 只需做参数变换:

$u = u(t) = (1-t)u_i + tu_{i+1}, \quad t \in [0,1], i = p, p+1, \dots, n$ 则 B 样条曲线可改写为矩阵形

式: $C_i(t) = C(u(t)) = \sum_{j=i-k}^i P_j N_{j,p}(u(t)), \quad t \in [0,1], i = p, p+1, \dots, n$

将上式改写为矩阵形式:

$$C_i(t) = [1 \quad t \quad t^2 \quad \dots \quad t^p] M_p \begin{bmatrix} P_{i-p} \\ P_{i-p+1} \\ \vdots \\ P_i \end{bmatrix}, \quad t \in [0,1], i = p, p+1, \dots, n$$

其中 1-3 次系数矩阵 $M_p(p=1,2,3)$ 分别为:

$$M_1 = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix}, \quad M_3 = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \text{ 则可以很容易写出}$$

$$\text{三次均匀 B 样条曲线的方程: } C_3(t) = \frac{1}{6} \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

六. de Boor 算法

给定控制顶点 $P_i (i = 0, 1, \dots, n)$ 及节点矢量 $T = [t_0, t_1, \dots, t_{n+k}]$ 后, 就定义了 **k** 阶 (**k-1** 次) **B** 样条曲线。计算 **B** 样条曲线上对应一点 $P(t)$, 可利用 **B** 样条曲线方程。采用 **de Boor** 算法, 计算更加快捷。

1. de Boor 算法的导出

先将 **t** 固定在区间 $[t_j, t_{j+1}] (k-1 \leq j \leq n)$, 由 **de Boor-Cox** 公式有:

$$\begin{aligned} P(t) &= \sum_{i=0}^n P_i N_{i,k}(t) = \sum_{i=j-k+1}^j P_i N_{i,k}(t) \\ &= \sum_{i=j-k+1}^j P_i \left[\frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t) \right] \quad (6.9) \\ &= \sum_{i=j-k+1}^j \left[\frac{t-t_i}{t_{i+k-1}-t_i} P_i + \frac{t_{i+k-1}-t}{t_{i+k-1}-t_i} P_{i-1} \right] N_{i,k-1}(t) \quad t \in [t_j, t_{j+1}] \end{aligned}$$

令

$$P_i^{[r]}(t) = \begin{cases} P_i, r = 0; j = j - k + 1, j - k + 2, \cdots, j \\ \frac{t - t_i}{t_{i+k-r} - t_i} P_i^{[r-1]}(t) + \frac{t_{i+k-r} - t}{t_{i+k-r} - t_i} P_{i-1}^{[r-1]}(t), \\ r = 1, 2, \cdots, k - 1; i = j - k + r + 1, j - k + r + 2, \cdots, j \end{cases} \quad (6.10)$$

$P(t)$ 的值可以通过递推关系式 (6.10) 求得。这就是著名的 **de Boor** 算法, 则 (6.9) 式可表示为

$$P(t) = \sum_{i=j-k+1}^j P_i N_{i,k}(t) = \sum_{i=j-k+2}^j P_i^{[1]}(t) N_{i,k-1}(t)$$

上式是同一条曲线 $P(t)$ 从 k 阶 B 样条表示到 可 $k-1$ 阶 B 样条表示的递推公式, 反复应用此公式, 得到: $P(t) = P_j^{[k-1]}(t)$

de Boor 算法的递推关系如图 6-17 所示。

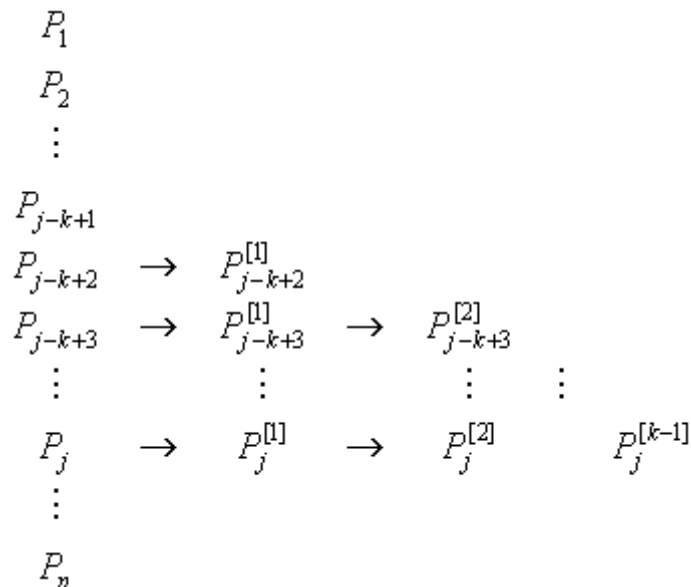


图 6-17 de Boor 算法的递推关系

2. De Boor 算法的几何意义

de Boor 算法有着直观的几何意义 ^{3/4} 割角, 即以线段 $P_i^{[r]} P_{i+1}^{[r]}$ 割去角 $P_i^{[r-1]}$ 。从多边形 $P_{j-k+1} P_{j-k+2} \cdots P_j$ 开始, 经过 **k-1** 层割角, 最后得到 $P^{(t)}$ 上的点 $P_j^{[r-1]}(t)$, 如图 6-18 所示。

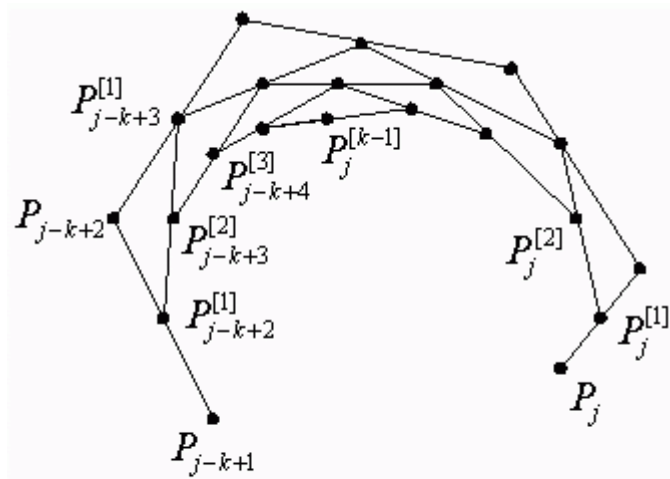


图 6-18 B 样条曲线 de Boor 算法
的几何意义

3. 由 **de Boor** 算法导出三次 **B** 样条的 **Bezier** 表示

在使用时，为了减少计算量，希望曲线次数越低越好，但二次曲线是一条抛物线，不能反应曲线的拐点；所以一般使用三次（四阶）样条曲线。下面来讨论三次(四阶)样条曲线与 **Bezier** 曲线的关系。由 **de Boor** 算法，知下列公式成立：

$$P(t_j) = P_j^{[3]}(t_j) = P_{j-1}^{[2]}(t_j)$$

$$P(t_{j+1}) = P_{j+1}^{[3]}(t_{j+1}) = P_j^{[2]}(t_{j+1})$$

$$P'(t_j) = 3(P_{j-1}^{[1]}(t_j) - P_{j-1}^{[2]}(t_j))$$

$$P'(t_{j+1}) = 3(P_j^{[2]}(t_{j+1}) - P_{j-1}^{[1]}(t_{j+1}))$$

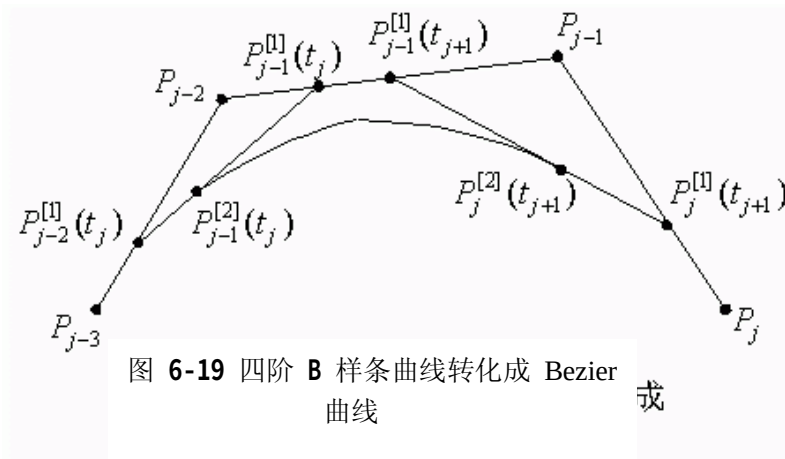
由于 $P(t)$ 在区间 $t_j \leq t \leq t_{j+1}$ 上是三次多项式, 故以上两个性质表明:

这段曲线表示成三次 **Bezier** 曲线, 则其控制顶点为:

$P_{j-1}^{[2]}(t_j), P_{j-1}^{[1]}(t_j), P_{j-1}^{[1]}(t_{j+1}), P_j^{[2]}(t_{j+1})$, 如图 6-16 所示, 即 $P(t)$ 可表示为:

$$\begin{aligned} P(t) &= \sum_{i=j-3}^j P_i N_{i,4}(t) \\ &= P_{j-1}^{[2]}(t_j) B_{0,3} \left[\frac{t-t_j}{\Delta t_j} \right] + P_{j-1}^{[1]}(t_j) B_{1,3} \left[\frac{t-t_j}{\Delta t_j} \right] \\ &\quad + P_{j-1}^{[1]}(t_{j+1}) B_{2,3} \left[\frac{t-t_j}{\Delta t_j} \right] + P_j^{[2]}(t_{j+1}) B_{3,3} \left[\frac{t-t_j}{\Delta t_j} \right] \end{aligned} \quad (6.11)$$

其中, $t_j \leq t \leq t_{j+1}, \Delta t_j = t_{j+1} - t_j$ 。(6.11) 式表明: 对四阶 **B** 样条曲线 $P(t)$ 而言, **de Boor** 算法不仅是求 $P(t)$ 的方法, 也是把 $P(t)$ 转化为一段段 **Bezier** 曲线的工具。



6.3 参数曲面

空间曲面常采用双参数表示,

$$S = S(u, v) = [x(u, v), y(u, v), z(u, v)] \quad (a \leq u \leq b, c \leq v \leq d)$$

其中 a 、 b 、 c 、 d 定义参数 u, v 的变化域, 即曲面的定义域。当 u, v 在定义域中变化时, $S(u, v)$ 在空间坐标系中变化, 即 $O-UW$ 坐标系中的任一点均与 $O-XYZ$ 坐标系中的点成一一映射关系。曲面定义域中的一对参数 u, v 确定曲面上的一个点, 如果令 v 参数不变而变动 u 则可得到 u 线, 反之, 固定 u 而变化 v 求得一条 v 线, 所有的 u 线和 v 线形成一个网, 称为参数曲线网。

当平面域为正方形时, 即 $(0 \leq u \leq 1, 0 \leq v \leq 1)$, 则 u, v 平面上的四条线:

$u = 0, v = 0, u = 1, v = 1$ 对应空间的四条边界线, 此时 $P(0,0)$ 、 $P(1,0)$ 、 $P(1,1)$ 、 $P(0,1)$ 是曲面的四个角点, $P(u,0)$ 、 $P(1,v)$ 、 $P(u,1)$ 、 $P(0,v)$ 称为曲线的四条边界线。

6.3.1 函数式曲面

对于非参数式曲面方程一般先转化成参数式曲面方程再进行绘制。这里给出几个常见曲面的参数方程

1. 球面

球坐标系下, 球心在 (x_0, y_0, z_0) 的球面参数方程为:

$$X = x_0 + R \cos u \cos v$$

$$Y = y_0 + R \cos u \sin v \quad u \in [-p/2, p/2], v \in [-p, p]$$

$$Z = z_0 + R \sin u$$

计算机编程绘制原理, 现任取一个 $u_0 \in [-p/2, p/2]$ 在平面上绘制一个圆, 再使 $u \in [-p/2, p/2]$ 以一定步长变化, 参数 v 对每一个 u 都从 $-p$ 变到 p 得到一系列的圆, 这里由 u 的变化得到的一系列平行的圆和由 v 的变化得到的一组圆相交, 将球面分成一片片小曲面片, 所有小曲面片组成球面。若希望曲

面相对光滑, 则可以将小曲面片分的更细, 即所取步长值更小, 当然曲面的光滑程度还与所使用计算机的分辨率有关, 屏幕绘制时在不考虑消隐的情况下, 一般立体效果不明显。

2. 椭球面

广义坐标系下椭球心位于 (x_0, y_0, z_0) 的椭球面参数方程为:

$$X = x_0 + a \cos u \cos v$$

$$Y = y_0 + b \cos u \sin v \quad u \in [-p/2, p/2], v \in [-p, p]$$

$$Z = z_0 + c \sin u$$

其中 **a**, **b**, **c** 分别为椭球面的长、中和短半轴。

6.3.2 旋转曲面

若空间曲面为一旋转曲面, 则可通过先将旋转母线表示成参数形式, 再将该参数母线绕一轴旋转的悼词旋转参数时表示的旋转参数曲面。平面曲线绕所在平面内的直线旋转生成的曲面称为旋转曲面

1. 圆锥面。圆锥面可看成 **ZOX** 平面上一条过原点的直线绕 **Z** 轴旋转一圈得到, 设此直线与 **Z** 轴夹角为 α , 该直线的参数方程为:

$$\begin{aligned} X &= t \sin \alpha \\ Z &= t \cos \alpha \end{aligned} \quad (0 < t)$$

将此参数圆绕 **Z** 轴旋转, 得到一个旋转参数曲面, 即圆锥面, 圆锥曲面方程为

$$\begin{aligned} X &= t \sin \alpha \cos q \\ Y &= t \sin \alpha \sin q \\ Z &= t \cos \alpha \end{aligned} \quad (0 \leq q \leq 2\pi)$$

3. 环面。圆环面可看成 **ZOX** 平面上圆心在 (z_0, x_0) 半径为 R 的圆绕 **Z** 轴旋转一圈得到, 设在 **ZOX** 平面上, 该圆的参数式为:

$$\begin{aligned} X &= z_0 + R \cos u \\ Z &= x_0 + R \sin u \end{aligned} \quad (0 \leq u \leq 2\pi)$$

将此参数圆绕 **Z** 轴旋转, 得到一个旋转参数曲面, 即圆环, 曲面方程为

$$\begin{aligned} X &= (x_0 + R \sin u) \cos v \\ Y &= (x_0 + R \sin u) \sin v \\ Z &= z_0 + R \cos u \end{aligned}$$

2. 回转曲面。在 **XOY** 平面上有抛物线

$$\begin{aligned} X &= PT^2 \\ Y &= 2PT \end{aligned} \quad (0 < T < Y_{\max} / 2P)$$

Y 的最大值为 $Y_{\max}$ 绕 X 轴旋转生成的回转曲面方程为:

$$r(T, q) = \{PT^2, 2PT \times \cos q, 2PT \times \sin q\}$$

6.4 常用曲面

6.4.1 双曲线曲面

双曲线曲面也是简单曲面之一。设曲面的四个角点在 **UV** 面中由单位正方形的角点 $P(0,0)$ 、 $P(1,0)$ 、 $P(1,1)$ 、 $P(0,1)$ 给定，要构成一个双曲面 $P(u,v)$ 其中 $(0 \leq u \leq 1, 0 \leq v \leq 1)$ 并允许对曲面上任意点作线性插值， $P(u,v)$ 可表示为：

$$P(u,v) = P(0,0)(1-u)(1-v) + P(0,1)(1-u)v + P(1,0)u(1-v) + P(1,1)vu$$

$$\text{或者: } P(u,v) = [(1-u)u] \begin{vmatrix} P(0,0) & P(0,1) \\ P(1,0) & P(1,1) \end{vmatrix} \begin{vmatrix} (1-v) \\ v \end{vmatrix}$$

6.4.2 Bezier 曲面

基于上面 **Bezier** 曲线的讨论, 我们可以方便地给出 **Bezier** 曲面的定义和性质, **Bezier** 曲线的一些算法也可以很容易扩展到 **Bezier** 曲面的情况。

一. 定义

设 $P_{ij}(0,1,\dots,n, j=0,1,\dots,m)$ 为 $(n+1) \times (m+1)$ 个空间点列, 则 $m \times n$ 次张量积形式的 **Bezier** 曲面定义为:

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

其中 $B_{i,m}(u) = C_m^i u^i (1-u)^{m-i}$, $B_{j,n}(v) = C_n^j v^j (1-v)^{n-j}$ 是 **Bernstein** 基函数。依次用线段连接点列 $P_{ij}(0,1,\dots,n, j=0,1,\dots,m)$ 中相邻两点所形成的空间网格, 称之为特征网格。**Bezier** 曲面的矩阵表示式是:

$$P(u, v) = \begin{bmatrix} B_{0,n}(u) & B_{1,n}(u) & \dots & B_{n,n}(u) \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & \dots & P_{0m} \\ P_{10} & P_{11} & \dots & P_{1m} \\ \dots & \dots & \dots & \dots \\ P_{n0} & P_{n1} & \dots & P_{nm} \end{bmatrix} \begin{bmatrix} B_{0,m}(v) \\ B_{1,m}(v) \\ \dots \\ B_{n,m}(v) \end{bmatrix}$$

在一般实际应用中, n, m 不大于 **4**。二. **Bezier** 曲面的几种表达形式

1)双一次 Bezier 曲面: 当 $n=m=1$ 时称为双一次 Bezier 曲面

$P(u, v) = (1-u)(1-v)P_{00} + (1-u)vP_{01} + u(1-v)P_{10} + uvP_{11}$ 这是一双曲抛物面 (马鞍面);

2)双二次 Bezier 曲面: 当 $n=m=2$ 时称为双二次 Bezier 曲面

$P(u, v) = \sum_{i=0}^2 \sum_{j=0}^2 P_{ij} B_{i,2}(u) B_{j,2}(v)$ 该曲面的四条边界是抛物线

3)双三次 Bezier 曲面: 当 $n=m=3$ 时称为双三次 Bezier 曲面, 如图 6-20 所示。

$$P(u, v) = \sum_{i=0}^3 \sum_{j=0}^3 P_{ij} B_{i,3}(u) B_{j,3}(v)$$

该曲面的四条边界都是三次 Bezier 曲线; 可通过控制内部的四个控制顶点 $P_{11}, P_{12}, P_{21}, P_{22}$ 来控制曲面内部的形

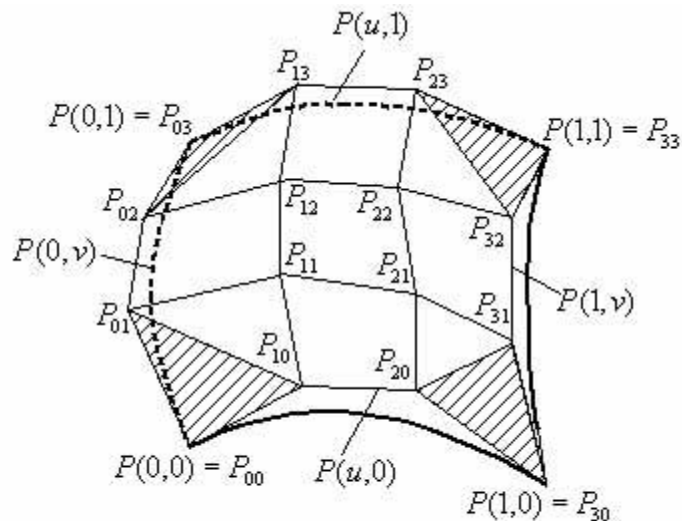


图 6-20 双三次 Bezier 曲面及边界信息

根据图中特征网格的 **16** 个顶点，可导出特征矩阵：

$$[P] = \begin{vmatrix} P(0,0) & P(0,1) & P(0,2) & P(0,3) \\ P(1,0) & P(1,1) & P(1,2) & P(1,3) \\ P(2,0) & P(2,1) & P(2,2) & P(2,3) \\ P(3,0) & P(3,1) & P(3,2) & P(3,3) \end{vmatrix}$$

双三次贝塞尔曲面的数学表达式为：

$$P(u,v) = [(1-u)^3 3u(1-u)^2 3u^2(1-u)u^3] \times [P] \times \begin{vmatrix} (1-v) \\ 3v(1-v)^2 \\ 3v^2(1-v) \\ v^3 \end{vmatrix}$$

$$= [u^3 \ u^2 \ u \ 1] \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} P(0,0) & P(0,1) & P(0,2) & P(0,3) \\ P(1,0) & P(1,1) & P(1,2) & P(1,3) \\ P(2,0) & P(2,1) & P(2,2) & P(2,3) \\ P(3,0) & P(3,1) & P(3,2) & P(3,3) \end{vmatrix} \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix} \begin{vmatrix} v^3 \\ v^2 \\ v \\ 1 \end{vmatrix} \quad \text{其中,}$$

$$\text{令 } [N] = \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}, \quad [V] = [v^3 \ v^2 \ v \ 1], \quad [U] = [u^3 \ u^2 \ u \ 1]$$

则 $P(u,v)=[U][N][P][N]^t[V]^t$ ，其中 **[P]** 为双三次贝赛尔特征网格矩阵，矩阵周围 **12** 个控制点定义了 **4** 条三次贝塞尔曲线，中间 **4** 个控制点控制曲面形状，**4** 个角点位于曲面上。

三. 性质

除变差减小性质外，**Bezier** 曲线的其它性质可推广到 **Bezier** 曲面：

(1) **Bezier** 曲面特征网格的四个角点正好是 **Bezier** 曲面的四个角点，即

$$P(0,0) = P_{00}, \quad P(1,0) = P_{m0}, \quad P(0,1) = P_{0n}, \quad P(1,1) = P_{mn}。$$

- (2) **Bezier** 曲面特征网格最外一圈顶点定义 **Bezier** 曲面的四条边界;
- Bezier** 曲面边界的跨界切矢只与定义该边界的顶点及相邻一排顶点有关, 且 $P_{00}P_{10}P_{20}$ 、 $P_{0n}P_{1n}P_{2n}$ 、 $P_{m0}P_{m-1,0}P_{m-2,0}$ 和 $P_{m0}P_{m-1,0}P_{m-2,0}$ (图 6-20 打上斜线的三角形); 其跨界二阶导矢只与定义该边界的及相邻两排顶点有关。
- (3) 几何不变性。
- (4) 对称性。
- (5) 凸包性。

6.4.3 B 样条曲面

给定参数轴 u 和 v 的节点矢量 $U = [u_0, u_1, \dots, u_{m+p}]$ 和 $V = [v_0, v_1, \dots, v_{n+q}]$, $p \times q$ 阶 B 样条曲面定义如下:

$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} N_{i,p}(u) N_{j,q}(v)$$

$P_{ij} (i = 0, 1, \dots, m; j = 0, 1, \dots, n)$ 是给定的空间 $(m+1)(n+1)$ 个点列, 构成一张控制网格, 称为 **B** 样条曲面的特征网格。 $N_{i,p}(u)$ 和 $N_{j,q}(v)$ 是 **B** 样条基, 分别由节点矢量 **U** 和 **V** 按 **deBoor-Cox** 递推公式决定。

7. B 样条曲面的几种表达形式

1) 双一次 B 样条曲面: 当 $n=m=1$ 时为双一次 B 样条曲面

$$P(u, v) = [(1-u)u] \begin{vmatrix} P(0,0) & P(0,1) \\ P(1,0) & P(1,1) \end{vmatrix} \begin{vmatrix} (1-v) \\ v \end{vmatrix} \quad \text{这是一双曲抛物面 (马鞍面);}$$

2) 双二次 B 样条曲面: 当 $n=m=2$ 时为双二次 B 样条曲面

$$P(u, v) = 1/4 \begin{bmatrix} u^2 & u & 1 \end{bmatrix} \begin{bmatrix} P(0,0) & P(0,1) & P(0,2) \\ P(1,0) & P(1,1) & P(1,2) \\ P(2,0) & P(2,1) & P(2,2) \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^2 \\ v \\ 1 \end{bmatrix}$$

该曲面的四条边界是抛物线

3)双三次 B 样条曲面：当 $n=m=3$ 时为双三次 B 样条曲面

$P(u, v)$

$$= 1/36 \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P(0,0) & P(0,1) & P(0,2) & P(0,3) \\ P(1,0) & P(1,1) & P(1,2) & P(1,3) \\ P(2,0) & P(2,1) & P(2,2) & P(2,3) \\ P(3,0) & P(3,1) & P(3,2) & P(3,3) \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 0 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$\text{其中, 令 } [F] = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}, \quad [V] = [v^3 \ v^2 \ v \ 1], \quad [U] = [u^3 \ u^2 \ u \ 1]$$

则 $P(u, v) = 1/36[U][F][P][F]^t[V]$, 其中 $[P]$ 为 **B** 特征矩阵且为 **16** 个网格点的角点信息。

B 样条曲线的一些几何性质可以推广到 **B** 样条曲面, 图 6-21 是一张双三次 **B** 样条曲面片实例。

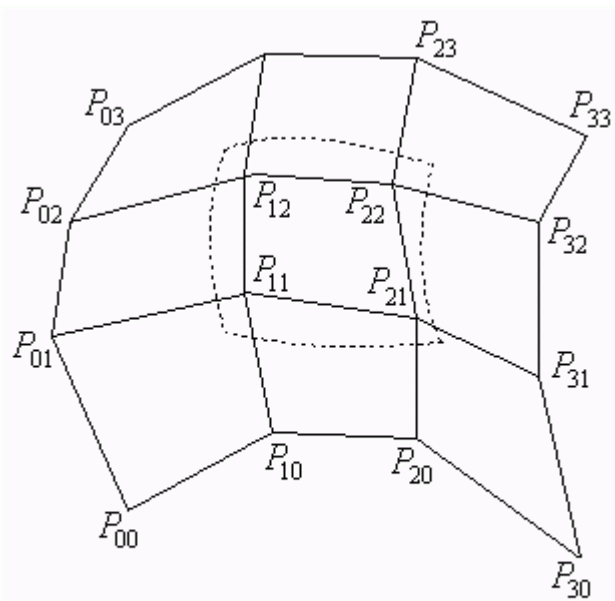


图 6-21 双三次 B 样条曲面片

面片

6.5 曲面与曲线编程案例

一、程序设计功能说明

程序为演示本章讲解的各种曲线曲面算法生成的实用程序。如图 6-22 所示为本例程序运行时的主界面，通过单击菜单中及下拉菜单的各功能项分别完成各种曲线曲面生成。



图 6-22

二、程序设计步骤

1. 创建工程名称为“曲线和曲面”单文档应用程序框架

2. 编辑菜单资源

设计根据 6.1 表中的定义编辑菜单资源如图 6-11 所示的菜单项。

表 6.1 菜单资源表

菜单标 题	菜单项标题	标示符 ID
《计算机图形学原理及算法教程》(Visual C++版) 和青芳 清华大学出版社出版 		
曲线	三叶梅花线	ID_DRAW_ROSE
	圆柱螺线	ID_DRAW_CYLINDER
	圆锥螺线	ID_DRAW_CYLINDER_CONE
	三次贝塞尔曲线	ID_DRAW_3BEZIER
	三次 B 样条曲线	ID_DRAW_3BSPLINE
曲面	环形面	ID_AREA_RING
	锥面	ID_AREA_CONE
	双线性曲面	ID_AREA_LINE
	旋转曲面	ID_AREA_ROTATION
	贝塞尔曲面	ID_AREA_BEZIER
www.yeaso.com CAD 教育网制作 www.cadedu.com		
	B 样条曲面	ID_AREA_BSPLINE

3. 添加函数

在工程中添加应用函数的方法，这里我们先在“View.h”中声明需要使用的函数名称，然后再在“View.cpp”中定义此函数。

// 曲线和曲面[程序代码见纸书](#)

4. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 7.2 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数声明。

表 7.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
--------	----	--------

ID_DRAW_ROSE	CONMMAN	OnDrawRose()
ID_DRAW_CYLINDER		OnDrawCylinder()
ID_DRAW_CYLINDER_CONE		OnDrawCylinder()
ID_DRAW_3BEZIER		OnDraw3bezier()
ID_DRAW_3BSPLINE		OnDraw3bspline()
ID_AREA_RING	CONMMAN	OnAreaRing()
ID_AREA_CONE		OnAreaCone()

ID_AREA_LINE		OnAreaLine()
ID_AREA_ROTATION		OnAreaRotation()
ID_AREA_BEZIER		OnAreaBezier()
ID_AREA_BSPLINE		OnAreaBspline()

5. View.cpp 添加完成各个菜单消息处理函数，实现既定功能，

```
// CMyView message handlers
```

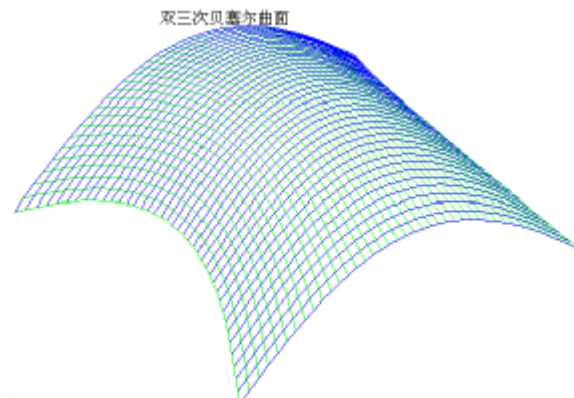
```
void CMyView::OnAreaBezier()
```

```
{//Bezier 曲面
```

```
member=3;
```

```
beier_draw();
```

```
}
```



```
void CMyView::OnAreaBspline()
```

```
{//B 样条曲面
```

[程序代码见纸书](#)

```
}
```

说明:

1. 为了能够让读者清楚曲面的生成过程，我们在程序中用键盘上
“上”、“下”、“左”、“右”箭头改变参数变量的大小，动态观看

参数大小对生成曲面的影响。

2. 以上仅列出 View.h、View.cpp 中的部分代码，其它文件代码全部为系统自动建立，无需改正，为节省篇幅此处省略，详见光盘。
代码中黑体部分为手工输入。

6.6 课后练习

1. 已知 4 个点型直点：(1.0, 2.0)、(2.5, 2.5)、(4.0, 4.5)、(5.0, 4.0)，

求三次样条曲线。

2. 编程绘制一条二次 Bezier 曲线；
3. 编程绘制一条二次 B 样条曲线；

第七章 几何造型技术

一般说, 几何造型技术研究在计算机中如何表达物体模型形状的技术。在几何造型系统中, 描述物体的三维模型有三种, 即线框模型、表面模型和实体模型。线框模型用顶点和棱边来表示物体, 由于没有面的信息, 所以不能表示表面含有曲面的物体; 另外, 它不能明确地定义给定点与物体之间的关系(点在物体内部、外部或表面上), 所以线框模型不能处理许多问题, 如不能生成剖切图、消隐图、明暗色彩图, 不能用于数控加工等, 应用范围受到了很大的限制。

表面模型用面的集合来表示物体, 而用环来定义面的边界。表面模型扩大了线框模型的应用范围, 能够满足面面求交、线面消隐、明暗色彩图、数控加工等需要。但在该模型中, 只有一张面的信息, 物体究竟存在于表面的哪一侧, 并没有给出明确的定义, 无法计算和分析物体的整

体性质，如物体的表面积、体积、重心等，也不能将这个物体作为一个整体去考察它与其它物体相互关联的性质，如是否相交等。

实体模型是最高级的模型，它能完整表示物体的所有形状信息，可以无歧义地确定一个点是在物体外部、内部或表面上，这种模型能够进一步满足物体计算、有限元分析等应用的要求。本章我们主要介绍实体造型技术的有关问题，并简单介绍最近发展起来的分形造型技术。

7.1 实体的表示模型

早期实体造型系统一个共同的特点是用多面体表示形体，不支持精确的曲面表示，优点是数据结构相对简单，集合运算、明暗图的生成和显示速度快；缺点是同一系统表示不唯一，违背了几何定义唯一性原则，而且只是近似表示，存在误差，若要提高表示精度就要增加离散平面片的数量，庞大数据量影响计算速度和计算机的存储管理。早期的几何造型系统还有一个特点，就是

只支持正则的形体造型。正则形体集 (**R-Set**) 的概念为几何造型奠定了初步的理论基础。对于任一形体, 具有 **3** 维欧氏空间 R^3 中非空、有界的封闭子集, 且其边界是二维流形 (即该形体是连通的), 我们称该形体为正则形体, 否则称为非正则形体。图 **7-1** 给出了一些非正则形体的实例。所谓二维流形 (**2-manifold**) 是指这样一些面, 其上任一点都存在一个充分小的邻域, 该邻域与平面上的圆盘是同构的, 即在该邻域与圆盘之间存在连续的一一映射。

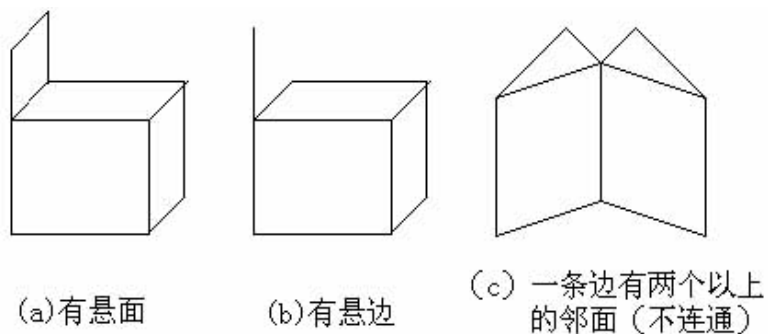


图 7-1 非正则形体

对实体模型的表示基本上可以分为分解表示、构造表示和边界表示三大类。

7.1.1 分解表示

分解表示是将形体按某种规则分解为小的更易于描述的部分。分解表示的一种特殊形式是每一小的部分都是一种固定形状(正方形、立方体等)的单元,形体被分解成这些分布在空间网格位置上的具有邻接关系的固定形状单元的集合,单元的大小决定了单元分解形式的精度。根据基本单元的不同形状,常用四叉树、八叉树和多叉树等表示方法。

分解表示中常见的表示方法是将形体空间细分为小的立方体单元,在计算机内存中对应开辟一个三维数组。形体占有的空间,存储单元中为 **1**; 否则空间为 **0**。这种表示方法简单,容易实现形体的交、并、差计算,但是占用的存储量太大,物体的边界面没有显式的解析表达式,不便于运算,实际应用中一般不采用。

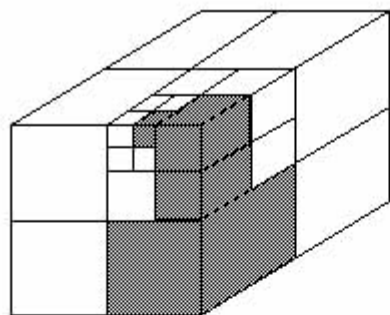
图 7-2 是八叉树表示形体的一个实例。八叉树法表示形体的过程是这样的，首先对形体定义一个外接立方体，再把它分解成八个子立方体，并对立方体依次编号为 **0, 1, 2, ..., 7**。如果子立方体单元已经一致，即为满（该立方体充满形体）或为空（没有形体在其中），

则该子立方体可停止分解；否则，需要对该立方体作进一步分解，再一分为八个子立方体。在八叉树中，非叶结点的每个结点都有八个分支。八叉树表示法有一些优点，近年来受到人们的注意。这些优点主要是：

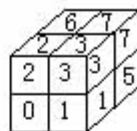
单。

对
需
相
交

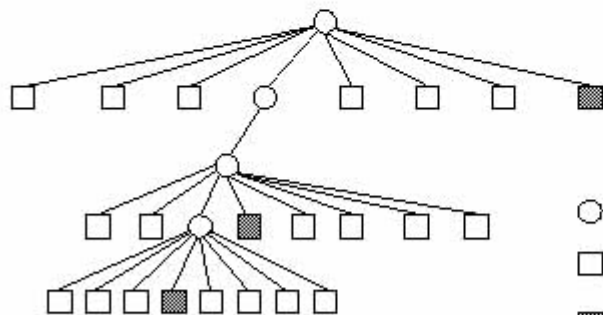
的
体



(a)



(b)



(c)

- 具有子孙的节点
- 空节点
- 实节点

图 7-2 八叉树表示的形体

(1) 形体表示的数据结构简

(2) 简化了形体的集合运算。

形体执行交、并、差运算时，只同时遍历参加集合运算的两形体应的八叉树，无需进行复杂的求运算。

(3) 简化了隐藏线（或面）

消除，因为在八叉树表示中，形上各元素已按空间位置排成了一

定的顺序。

(4) 分析算法适合于并行处理。

八叉树表示的缺点也是明显的，主要是占用的存储多，只能近似表示形体，以及不易获取形体的边界信息等。

7.1.2 构造表示

构造表示是按照生成过程来定义形体的方法，构造表示通常有扫描表示、构造实体几何表示和特征表示三种。

- 扫描表示

扫描表示是基于一个基体(一般是一路径运动而产生形体。可见,扫描表示动的基体,另一个是基体运动的路径;给出截面的变化规律。图 7-3 给出了变体的扫描路径是曲线。

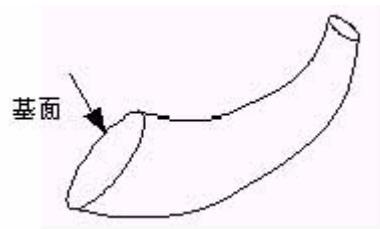


图 7-3 变截面扫描

个封闭的平面轮廓)沿某一需要两个分量,一个是被运如果是变截面的扫描,还要截面扫描表示的例子,扫描

扫描是生成三维形体的有效方法,但是,用扫描变换产生的形体可能出现维数不一致的问题。另外,扫描方法不能直接获取形体的边界信息,表示形体的覆盖域非常有限。

- 结构体元表示

结构体元 (CSG) 表示是通过对体元定义、运算得到新形体的表示方法,体元可以是立方体、圆柱、圆锥等,也可以是半空间,其运算为变换或正则集合运算并、交、差。

CSG 表示可以看成是一棵有序的二叉树，终节点为体元或是形体变换参数。非终结点或为正则的集合运算或是变换（平移和/或旋转）操作，这种运算或变换只对其直接的子结点（子形体）起作用。每棵子树（非变换叶子结点）表示其下两个节点组合及变换的结果，几何变换并不限定为刚体变换，也可以是任意范围的比例变换和对称变换。

CSG 树无二义性，但不是唯一的，它的定义域取决于其所用体元以及所允许的几何变换和正则集合运算算子。若体元是正则集，只要体元叶子是合法的，就保证了任何 **CSG** 树都是合法的正则集。

CSG 表示的优点：

- 1) 数据结构比较简单，数据量比较小，内部数据的管理比较容易；
- 2) **CSG** 表示可方便地转换成边界（**Brep**）表示；

3) CSG 方法表示的形体的形状, 比较容易修改。

CSG 表示的缺点:

1) 对形体的表示受体素的种类和对体素操作的种类的限制, 也就是说, CSG 方法表示形体的覆盖域有较大的局限性。

2) 对形体的局部操作不易实现, 例如, 不能对基本体元的交线倒圆角;

3) 由于形体的边界几何元素(点、边、面)是隐含地表示在 CSG 中, 故显示与绘制 CSG 表示的形体需要较长的时间。

- 特征表示

以 **Pro/Engineering** 为代表的参数化、变量化的特征造型技术，在几何造型领域产生了深远的影响。特征技术产生是在以 **CSG** 和 **Brep** 为代表的几何造型技术已较为成熟，实际应用中，几何建模的效率不仅较低，而且需要用户懂得几何造型的一些基本理论的事实上，同时，实体造型系统需要与应用系统集成在一起，用户对实体模型又提出了更高的要求，推动特征建模的出现。

以机械设计
中设计完成以
析，工艺设计、
工艺设计时，并
面这些几何和拓
械加工特征信

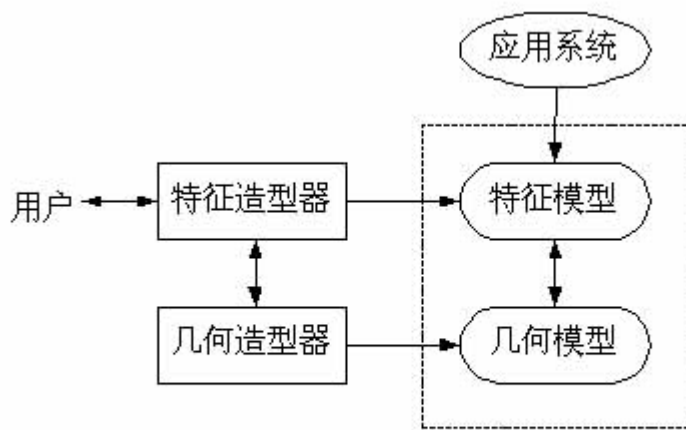


图 7-4 基于特征的造型系统

为例，机械零件在实体系统后，需要进行结构、应力分析、加工和检验等。而用户进行不需要构成形体的点、线、面信息，而是需要高层的信息，诸如光孔、螺孔、环形

槽、键槽、滚花等，并根据零件的材料特性，加工特征的形状、精度要求、表面粗糙度要求等，以确定所需要的机床、刀具、加工方法、加工用量等，传统的几何造型系统远不能提供这些信息，以至 **CAD** 与 **CAPP**（计算机辅助工艺过程设计）成为世界性的难题。

由此可以看出，特征是面向应用、面向用户的。基于特征的造型系统，如图 **7-4** 所示。特征模型的表示仍然要通过传统的几何造型系统来实现。不同的应用领域，具有不同的应用特征。一些著名的特征造型系统（如 **Pro/Engineering**）除提供了一个很大的面向应用的设计特征库外，还允许用户自己定义自己的特征，加入到特征库中，为用户进行产品设计和使 **CAD** 与其它应用系统的集成提供了极大的方便。

所以，在几何造型系统中，不同应用领域的特征都有其特定的含义，例如机械加工中，提到孔，我们会想到，是光孔，还是螺孔，孔径有多大，孔有多深，孔的精度是多少等。特征的形

状常用若干个参数来定义，根据特征参数我们并不能直接得到特征的几何元素信息，而在对特征及在特征之间进行操作时需要这些信息。特征方法表示形体的覆盖域受限于特征的种类。

7.1.3. 边界表示

以上介绍的构造表示通常具有不便于直接获取形体几何元素的信息、覆盖域有限等缺点。边界表示 (**Boundary Representation**) 也称为 **BR** 表示或 **BRep** 表示，它是几何造型中最成熟、无二义性的表示法。实体的边界通常是由面的并集来表示，而每个面又由它所在的曲面的定义加上其边界来表示，面的边界是边的并集，而边又是由点来表示的。边界表示的一个重要特点是在该表示法中，描述形体的信息包括几何信息 (**Geometry**) 和拓扑信息 (**Topology**) 两个方面，拓扑信息描述形体上的顶点、边、面的连接关系，拓扑信息形成物体边界表示的“骨架”，形体的几何信息犹如附着在“骨架”上的肌肉。例如形体的某个表面位于某一个曲面上，定义这一曲面方程

的数据就是几何信息。此外，边的形状、顶点在三维空间中的位置（点的坐标）等都是几何信息，一般说来，几何信息描述形体的大小、尺寸、位置、形状等。

在边界表示法中，边界表示就按照体一面一环一边一点的层次，详细记录了构成形体的所有几何元素的几何信息及其相互连接的拓扑关系。在进行各种运算和操作中，就可以直接取得这些信息。

Brep 表示的优点是：

- （1）表示形体的点、边、面等几何元素是显式表示的，使得绘制 **Brep** 表示的形体的速度较快，而且比较容易确定几何元素间的连接关系；
- （2）容易支持对物体的各种局部操作；
- （3）便于在数据结构上附加各种非几何信息，如精度、表面粗糙度等。

Brep 表示的缺点是:

(1) 数据结构复杂, 需要大量的存储空间, 维护内部数据结构的程序比较复杂;

(2) **Brep** 表示不一定对应一个有效形体, 通常运用欧拉操作来保证 **Brep** 表示形体的有效性、正则性等。

由于 **Brep** 表示覆盖域大, 原则上能表示所有的形体, 而且易于支持形体的特征表示等, **Brep** 表示已成为当前 **CAD/CAM** 系统的主要表示方法。

7.1.2 形体的边界表示模型

用边界表示法建立三维形体时, 经常用到欧拉操作与集合运算, 本小节我们对边界表示的数据结构、欧拉操作及集合运算作一个简单的介绍。

我们已经知道，边界模型由几何信息和拓扑信息两部分组成，表达形体的基本拓扑实体（**Entity**）包括：

1. 顶点

顶点（**Vertex**）的位置用（几何）点（**Point**）来表示。点是几何造型中的最基本的元素，自由曲线、曲面或其它形体均可用有序的点集表示。用计算机存储、管理、输出形体的实质就是对点集及其连接关系的处理。

2. 边

边（**Edge**）是两个邻面（对正则形体而言）、或多个邻面（对非正则形体而言）的交集，边有方向，它由起始顶点和终止顶点来界定。边的形状（**Curve**）由边的几何信息来表示，可以是直线或曲线，曲线边可用一系列控制点或型值点来描述，也可用显式、隐式或参数方程来描述。

3. 环

环 (**Loop**) 是有序、有向边 (**Edge**) 组成的封闭边界。环中的边不能相交, 相邻两条边共享一个端点。环有方向、内外之分, 外环边通常按逆时针方向排序, 内环边通常按顺时针方向排序。

4. 面

面 (**Face**) 由一个外环和若干个内环 (可以没有内环) 来表示, 内环完全在外环之内。根据环的定义, 在面上沿环的方向前进, 左侧总在面内, 右侧总在面外。面有方向性, 一般用其外法矢方向作为该面的正向。若一个面的外法矢向外, 称为正向面; 反之, 称为反向面。面的形状

(**Surface**) 由面的几何信息来表示, 可以是平面或曲面, 平面可用平面方程来描述, 曲面可用控制多边形或型值点来描述, 也可用曲面方程 (隐式、显式或参数形式) 来描述。对于参数曲面,

通常在其二维参数域上定义环，这样就可由一些二维的有向边来表示环，集合运算中对面的分割也可在二维参数域上进行。

5. 体

体 (**Body**) 是面的并集。在正则几何造型系统中，要求体是正则的，非正则形体的造型技术将线框、表面和实体模型统一起来，可以存取维数不一致的几何元素，并可对维数不一致的几何元素进行求交分类，从而扩大了几何造型的形体覆盖域。

7.1.3.1 欧拉操作

欧拉公式表达为对于任意的简单多面体，其面(**f**)、边(**e**)、顶点(**v**)的数目满足公式 $v + e + f = 2$ 。

对于任意的正则形体，引入形体的其它几个参数：形体所有面上的内孔总数(**r**)、穿透形体的孔洞数(**h**)和形体非连通部分总数(**s**)，则形体满足公式： $v + e + f = 2(s - h) + r$ 。

欧拉公式给出了形体的点、边、面、体、孔、洞数目之间的关系，在对形体的结构进行修改时，必须要保证这个公式成立，才能够保证形体的有效性。由此而构造出一套操作，称为欧拉操作，完成对形体部分几何元素的修改，修改过程中保证各几何元素的数目保持这个关系式不变。

7.1.3.2 集合运算

集合运算是实体造型系统中是一种非常有效的构造形体的方法。从一维几何元素到三维几何元素，正则形体和非正则形体，针对不同的情况和应用要求，有许多集合运算算法。

1. 集合运算算法

正则集合运算与非正则形体运算的区别在于增加了正则化处理步骤。集合运算算法包括以下几部分：

(1)求交：参与运算的一个形体的各拓扑元素求交，求交的顺序采用低维元素向高维元素进行。用求交结果产生的新元素（维数低于参与求交的元素）对求交元素进行划分，形成一些子元素。这种经过求交步骤之后，每一形体产生的子拓扑元素的整体相对于另一形体有外部、内部、边界上的分类关系。

(2)成环：由求交得到的交线将原形体的面进行分割，形成一些新的面环。再加上原形体的悬边、悬点经求交后得到的各子拓扑元素，形成一拓扑元素生成集。

(3)分类：对形成的拓扑元素生成集中的每一拓扑元素，取其上的一个代表点，根据点/体分类的原则，决定该点相对于另一形体的位置关系，同时考虑该点代表的拓扑元素的类型（即其维数），来决定该拓扑元素相对于另一形体的分类关系。

(4)取舍：根据拓扑元素的类型及其相对另一形体的分类关系，按照集合运算的运算符要求，要决定拓扑元素是保留还是舍去；保留的拓扑元素形成一个保留集。

(5)合并：对保留集中同类型可合并的拓扑元素进行合并，包括面环的合并和边的合并。

(6)拼接：以拓扑元素的共享边界作为其连接标志，按照从高维到低维的顺序，收集分类后保留的拓扑元素，形成结果形体的边界表示数据结构。

7.2 求交分类

在几何造型中，通常利用集合运算（并、交、差运算）实现复杂形体的构造，而集合运算需要大量的求交运算。如何提高求交的实用性、稳定性、速度、精度等，对几何造型系统至关重要。

当前几何造型系统，大多采用精确的边界表示模型。在这种表示法中，形体的边界元素和某类几何元素相对应，它们可以是直线、圆（圆弧）、二次曲线、**Bezier** 曲线、**B** 样条曲线等，也可以是平面、球面、二次曲面、**Bezier** 曲面、**B** 样条曲面等，求交情况十分复杂。，二次曲面与各种自由曲面并存的混合表示模型，逐渐为人们所接受。

7.2.1 求交分类

在几何造型系统中，用到的几何元素主要有以下 **25** 种，这些几何元素可以按照其维数分为三大类：点、线、面。

点：三维点。

线：三维直线段、二次曲线(包括圆弧和整圆、椭圆弧和椭圆、抛物线段、双曲线段)、**Bezier** 曲线 (有理和非有理)、**B** 样条曲线、**NURBS** 曲线。

面：平面、二次曲面(包括球面、圆柱面、圆锥/台面、双曲面、抛物面、椭球面和椭圆柱面)、**Bezier** 曲面 (有理和非有理)、**B** 样条曲面、**NURBS** 曲面。

求交方法分为点点、点线、点面、线线、线面六种。点只有三维点一种，比较简单。线和面又可分别归为二次曲线、自由曲线和二次曲面、自由曲面两类。

7.2.2 基本的求交算法

计算机内浮点数的误差，求交计算引进容差。假定容差为 ϵ ，则点被看成是半径为 ϵ 的球，线被看成是半径为 ϵ 的圆管，面被看成是厚度为 2ϵ 的薄板。

点与其它几何元素的求交比较简单, 计算两个点是否相交, 实际上是判断两个点是否重合, 判断点和线(或面)是否相交, 实际上是判断点是否在线(或面)上。

我们重点讨论线与线的求交及线与面的求交。根据前面的分类方法, 线与线的求交有二次曲线与二次曲线、二次曲线与自由曲线及自由曲线与自由曲线求交三种。线与面的求交有二次曲线与二次曲面、二次曲线与自由曲面、自由曲线与二次曲面及自由曲线与自由曲面求交四种。

7.2.2.1 线与线的求交计算

1. 二次曲线与二次曲线的求交

二次曲线在非退化的情况下, 也称为圆锥曲线(椭圆、双曲线和抛物线)。由于圆锥曲线在其标准(局部)坐标系下具有标准的隐式方程和参数方程的形式, 因而, 这种求交策略是将坐标系变换到该圆锥曲线的局部坐标系下, 一个圆锥曲线用隐式方程的形式表示, 而另一圆锥曲线采

用参数方程的形式，代入即可获得有关参数的四次方程，四次代数方程具有精确的求根公式，因而可计算出二者的交点。

2. 二次曲线与 NURBS 曲线求交

自由曲线（**Bezier** 曲线、**B** 样条曲线和 **NURBS** 曲线）可用 **NURBS** 方法统一表示。二次将 **NURBS** 曲线的参数方程代入参数的一元高次方程，然后，法解出交点参数，或把圆锥曲线解出交点参数，或把圆锥曲线

3. NURBS 曲线与 NURBS 曲线求交

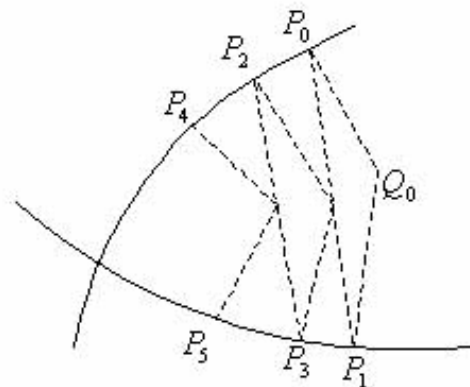


图 7-5 NURBS 曲线与 NURBS 曲线迭代求交

样条曲线和 **NURBS** 曲线）可曲线与 **NURBS** 曲线求交，可圆锥曲线的隐式方程，得到使用一元高次方程的求根方程也表示为参数形式，转化题。线求交

解决这类求交问题，通常采用离散法求初始交点，迭代求精确解的办法，具体求解步骤如下：

(1) 初始化。依据离散精度，将 **NURBS** 曲线形成对应的二叉树表示，叶子结点是对应于该曲线的某一离散子线段及其包围盒，非叶子结点是对应于该段 **NURBS** 曲线的包围盒。

(2) 求初始交点。遍历两曲线的二叉树，若其叶子结点的包围盒相交，则将两者的数据（曲线段中点的参数值，二者坐标的平均值）存入初始交点队列。

(3) 将初始交点迭代求精确交点。迭代方程可形象地用图 7-5 表示。

计算过程为：设初始交点为 (Q_0, s_0, t_0) ，其中 **Q0** 是初始点的空间坐标，**s0.t0** 则分别为两 **NURBS** 曲线的初始交点参数值，将 **Q0** 投影至两曲线上，得两点 $(P_0, s_1), (P_1, t_1)$ ，可得另一更精确初始点 $(\frac{P_0 + P_1}{2}, s_1, t_1)$ ，依次可得 $P_0, \dots, P_{2n}, \dots$ 和 $P_1, P_3, \dots, P_{2n+1}, \dots$ ，直至 **P2n** 与 **P2n+1** 两点间的距离小于 **e** 为止。

7.2.2.2 线与面的求交计算

与自由曲线的表示类似, 自由曲面 (**Bezier** 曲面、**B** 样条曲面和 **NURBS** 曲面) 可用 **NURBS** 方法统一表示。二次曲线与 **NURBS** 曲面的求交计算通常转化为 **NURBS** 曲线与 **NURBS** 曲面的求交计算的问题。

二次曲线与二次曲面的求交的求交计算, 可以把二次曲线的参数形式代入二次曲面的隐式方程, 得到关于参数的四次方程, 然后用四次方程的求根公式, 计算出交点的参数。

NURBS 曲线与二次曲面的求交计算, 可以把 **NURBS** 曲线的参数形式代入二次曲面的隐式方程, 得到关于参数的高次方程, 然后, 用高次方程的求根方法求解。

下面重点介绍 **NURBS** 曲线与 **NURBS** 曲面的求交计算, 计算过程叙述如下:

1. 初始化。

依据离散精度，将 **NURBS** 曲线离散成二叉树的形式，将 **NURBS** 曲面离散成四叉树的形式。四叉树的叶子结点是 **NURBS** 曲面的子曲面片，并存储其包围盒的坐标，非叶子结点记录对应子面片的包围盒。

2. 求初始交点。

遍历该二叉树和四叉树，如果曲线二叉树叶子结点的包围盒与曲面四叉树的叶子结点的包围盒有交点，则将子曲线段中点的参数值、子曲面片的中心点的坐标值与参数值作为初始交点，记录入初始交点点列中去。

3. 对初始交点进行迭代，形成精确交点。

可用牛顿迭代法求解精确交点。设 **NURBS** 曲线为 $\mathbf{C}(t)$ ，**NURBS** 曲面为 $\mathbf{S}(u,v)$ ，则在交点处应满足：

$$C(t) - S(u, v) = 0$$

设 $f(u, v, t) = C(t) - S(u, v)$, 则问题转化为求函数 $f(u, v, t)$ 的根。

$$\because df = \frac{dC(t)}{dt} dt - \frac{\partial S(u, v)}{\partial u} du + \frac{\partial S(u, v)}{\partial v} dv$$

两边同时叉积 $\frac{\partial S}{\partial u}$, 并考虑到 $\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial u} = 0$, 得到:

$$\frac{\partial S}{\partial u} \times df = \left(\frac{\partial S}{\partial u} \times \frac{dC}{dt} \right) dt - \left(\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right) dv$$

两边再点积 $\frac{dC}{dt}$, 并考虑到 $\frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times \frac{dC}{dt} \right) = 0$, 得到:

$$\frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times df \right) = - \frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right) dv$$

类似可得到:

$$\frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial v} \times df \right) = \frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right) du$$

$$\frac{\partial S}{\partial u} \cdot \left(\frac{\partial S}{\partial v} \times df \right) = \frac{\partial S}{\partial u} \cdot \left(\frac{\partial S}{\partial u} \times \frac{dC}{dt} \right) dt$$

令 $D = \frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right)$ ，则可建立迭代方程：

$$\begin{aligned} t_{i+1} &= t_i + \left[\frac{\partial S}{\partial u} \cdot \left(\frac{\partial S}{\partial v} \times dr \right) \right] / D|_{(t_i, u_i, v_i)} \\ u_{i+1} &= u_i + \left[\frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial v} \times dr \right) \right] / D|_{(t_i, u_i, v_i)} \\ v_{i+1} &= v_i - \left[\frac{dC}{dt} \cdot \left(\frac{\partial S}{\partial u} \times dr \right) \right] / D|_{(t_i, u_i, v_i)} \end{aligned}$$

设初值为 (t_0, u_0, v_0) ，一般迭代 **3** 至 **5** 次，便可达到要求的精度。

7.2.2.3 曲面与曲面的求交

在几何元素之间的求交算法中，曲面与曲面之间的求交是最为复杂的一种。曲面与曲面求交的基本方法主要有代数方法、几何方法、离散方法和跟踪方法四种，下面分别作一个简单的介绍。

1. 代数方法

代数方法是利用求解代数方程的方法求出曲面的交线。对于一些简单的曲面求交，如平面和平面，平面和二次曲面，可以直接通过曲面方程求解计算交线，对于某些复杂的情况，则需要进行分析 and 化简的运算后求解。

根据表示曲面的方程的形式可以将曲面分为隐式表示和参数表示两种类型。隐式表示的曲面为 $\mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{z})=0$ ，参数表示的曲面为 $\mathbf{r} = \mathbf{r}(\mathbf{u}, \mathbf{v})$ 。所以，根据参与求交的两曲面的表示形式的不同，可以把求交分为三种情况。

(1) 隐式表示和参数表示的情形，把参数方程代入隐式方程的方法，将交线表示为 $\mathbf{g}(\mathbf{u}, \mathbf{v})=0$ 的形式，得到的交线方程是平面代数曲线方程，根据平面代数曲线理论的方法求解交线。求解的

过程是先构造特征初始点(边界点、转折点和奇异点),这可用数值方法求解方程组得到,特征电把交线分成若干单调段,从特征初始点出发可求出每一单调段。

(2) 两个曲面都是参数表示的情形,只需要将其中之一隐式化,然后用前面的方法求解。而参数多项式或有理多项式曲面的隐式化通过消元来实现。

(3) 两个曲面都是隐式曲面的情形,一种方法是将其中一个曲面参数化,也可用第一种情况来求解。但是,一般情况下这种参数化很困难,对于某些情况可以采用另外的方法计算参数化的曲面。

代数法还有一个严重的弱点是对误差很敏感,这是因为代数法经常需要判别某些量是否大于零、等于零或小于零,而在计算机中的浮点数近似表示的误差常常会使这种判别出现错误,而且这种误差会随着运算步骤的增多而不断扩大。

2. 几何方法

几何方法求交是利用几何的方法, 对参与求交曲面的形状大小、相互位置以及方向等进行计算和判断, 识别出交线的形状和类型, 从而可精确求出交线。对于一些交线退化或相切的情形, 交线往往是点、直线或圆锥曲线, 用几何方法求交可以更加迅速和可靠。几何求交适应性不是很广, 一般仅用于平面以及二次曲面等简单曲面的求交。

3. 离散方法

离散方法求交是利用分割的方法, 将曲面不断离散成较小的曲面片, 直到每一子曲面片均可用比较简单的面片, 如四边形或三角形平面片来逼近, 然后用这些简单面片求交得一系列交线段, 连接这些交线段即得到精确交线的近似结果。离散求交一般包括下面的过程: 用包围盒作分离性检查排除无交区域; 根据平坦性检查判断是否终止离散过程; 连接求出的交线段作为求交结果。

由于 **Bezier** 曲面, **B** 样条曲面具有离散性质, 使得它们最适合于离散法求交。离散法求出的交线逼近精度不高。如果要求的精度较高, 需要增加离散层数。

4. 跟踪方法

跟踪方法求交是通过先求出初始交点, 然后从已知的初始交点出发, 相继跟踪计算出下一交点, 从而求出整条交线的方法。

跟踪法的本质是构造交线满足的微分方程组, 先求出满足方程组的某个初值解, 通过数值求解微分方程组的方法来计算整个交线。

跟踪方法在计算相继交点的时候, 利用了曲面的局部微分性质, 一般采用数值迭代的方法求解, 使得计算效率较高。

跟踪法求交中要考虑的主要问题包括：如何求出初始交点并保证每一交线分支都有初始交点被求出；如何计算奇异情况下的跟踪方向以及合理选取跟踪的前进步长；如何处理相切的情况。

以上几种方法是曲面求交中常采用的几种基本方法。在实际应用中，往往根据具体应用的需要，采用这些方法的结合来实现求交，如在跟踪法中的初始交点常采用离散法求得。

7.3 图形相交-相切程序设计案例

一、程序功能说明

求交在实体造型中有着重要的地位，实际设计图形软件系统相切也很重要，这里我们设计几种典型二维、三维图形相交或相切的程序实例。如图 7-6 所示为本例程序运行时的主界面，通过单击菜单中及下拉菜单的各功能项。



图 7-6

二、程序设计步骤

1. 创建工程名称为“图形相交-相切”单文档应用程序框架
2. 编辑菜单资源

设计根据 7.1 表中的定义编辑菜单资源如图 7-3-1 所示的菜单项。

表 7.1 菜单资源表

3. 添加函数
在工程中
方法，这里我
中声明需要
称，然后再在
义此函数。

菜单标题	菜单项标题	标示符 ID
线段相交 直线与圆相 交 直线与圆相 切 圆与圆相交		ID_LINE_LINE ID_LINE_CIERCLE1 ID_LINE_CIRCLE ID_CIRCLE_CIRCLE
组合体交线	平面截圆柱体 平面截圆锥体	ID_PLANE_CYLINDER ID_PLANE_CONE

和变量
添加应用函数的
们先在“View.h”
使用的函数名
“View.cpp”中定

```
// 曲线和曲面 View.h : interface of the CMyView class
```

```
////////////////////////////////////
```

[程序代码见纸书](#)

```
}
```

```
•
```

```
•
```

```
•
```

4. 添加消息映射

利用 ClassWizard（建立类向导）为应用程序添加与菜单项相关的消息处理函数，ClassName 栏中选择 CMyView，根据表 7.2 建立如下的消息映射函数，ClassWizard 会自动完成有关的函数

声明。

表 7.2 菜单项的消息处理函数

菜单项 ID	消息	消息处理函数
ID_LINE_LINE	CONMMAN	OnLineLine()
ID_LINE_CIERCLE1		OnLineCiercle1()
ID_LINE_CIRCLE		OnLineCircle()
ID_CIRCLE_CIRCLE		OnCircleCircle()
ID_PLANE_CYLINDER	CONMMAN	OnPlaneCylinder()
ID_PLANE_CONE		OnPlaneCone()

5. View.cpp 添加完成各个菜单消息处理函数，实现既定功能

```
void CMyView::OnLineLine()
{
    RedrawWindow();

    PushNum=1;

    CDC*pDC=GetDC(); //获得绘图类指针

    pDC->TextOut(0,5,"用鼠标右键在窗口取选取四个点,当鼠标右键按下第 5 次求交点");

    ReleaseDC(pDC);
}
```

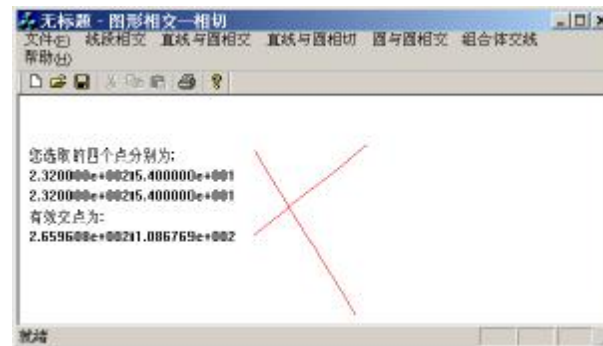


```
void CMyView::OnLineCiercle1()
```

```
{
```

[程序代码见纸书](#)

```
}
```



7.4 非传统造型技术上面介绍的基于欧氏几何的传统实体

造型技术的主要描述工具是直线、平滑的曲线、平面及边界整齐的平滑曲面，这些工具在描述一些抽象图形或人造物体的形态时是非常有力的，但对于一些复杂的自然景象形态就显得无能为力

了, 诸如山、树、草、火、云、波浪等。这是由于从欧氏几何来看, 它们是极端无规则的。为了解决复杂图形生成问题, 分形(Fractal)造型应运而生。

7.4.1 基本概念

1904 年, 由 Helge Von Koch 研究了一种称为雪花的图案, 他将一个等变三角形的三条边三等分, 中间的一段隆起另一个小三角形, 这小三角形依照上面的操作, 如此一直下去, 理论上可证明能够构成一个面积有限, 而边长为无穷长的雪花, 这与传统数学上的观点不相符, 成为一种新的造型技术。

假设用一只 1 000m 的尺子测量不列颠的海岸线, 可得到一个长度值 L_1 , 若要考虑更多细节, 可用更短的尺子, 例如 1m 的尺子测量, 同时得到另一个长度 L_2 , 显然 $L_2 > L_1$, 一般, 尺子长度为 r , 测量结果为 $L(r)$, 若 $r \rightarrow 0$, 则有 $L(r) \rightarrow \infty$, 也就是说, 不列颠的海岸线长度与测量的

尺子有关，为不确定数。Mandelbort 正是通过上述实例向人们展示细节的无穷回归和测量尺度的减小都会得到更多的细节，既复杂现象的自相似性，并提出分形（Fractal）概念。

设每一步细分的数目为 N ，细分放大（缩小）倍数为 S ，则分数维 D 定义为：

$$D = (\log N) / \log(1/S)$$

对于上述 Koch 雪花， $N=4$ ， $S=1/3$ ，雪花边数的分数维 $D=1.2619$ 。

一般二维空间中的一个分数维曲线的维数介于 1 和 2 之间；三维空间的一个分数维曲线维数在 2、3 之间，分数维概念的提出为研究复杂性体提供了新的视角，使人们从无序中发现了有序，为许多科学领域提供新的应用工具。

7.4.2 分形造型对模型的基本要求 分数维图形是由一片陌生而美妙的画面组成的,不是由画家精心创造的,而是由数学家探索绘制出来的,可用计算机停在屏幕上用明暗点绘图来遨游这一世界。生成图形的关键是要有合适的模型来描述图案。分形造型对模型的基本要求:

1. 分形造型技术能逼真地“再现”自然景象。所谓逼真是指从视觉效果上逼真,“再现”即不求完全一致。
2. 模型不依赖于观察距离。即距离远时可给出大致轮廓和一般细节,距离近时能给出更丰富细节。
3. 模型说明应尽量简单,模型应具有数据放大能力。
4. 模型应便于交互地修改。
5. 图形生成的效率要高。

6. 模型适用范围应尽可能地宽。

7.4.3 分形造型的常用模型

为有效模拟海岸线和山等自然景象，引进随机插值模型。

传统的绘制图形模型都与观察距离有关，随机插值模型部实现规定各种图素和尺度大小、数

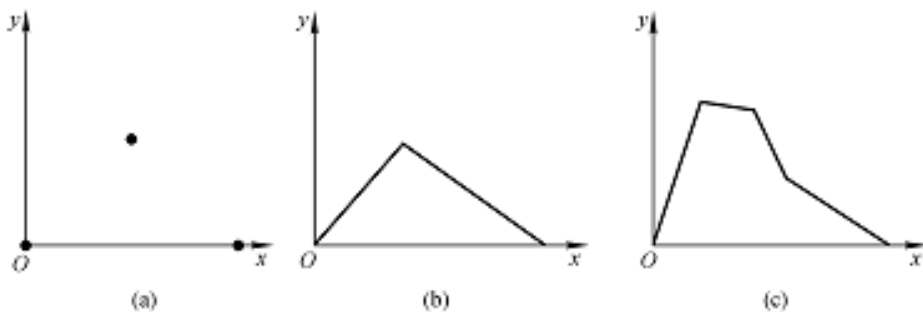


图 7-7 用随机模型构造海岸线的例子

量，用一个随机过程的采样路径作为模型的方法，避免了观察距离的局限性。例如，构建二维海岸线的模型，如图首先选择控制大致形状的若干初始点，然后取相邻两点连线的中垂线上随机偏移一段距离，再将偏移后的点与两端点分别连成两条线段，这样一直画下去，可得到一条曲折的有无穷细节回归的海岸线，其曲折程度由随机偏移量控制，同时决定了分数维的大小。在三维坐标系中，同样道理可构造山的模型，如图 7-8 所示。

2. 粒子系统模型

粒子系统模型是 1983 年为模拟火焰由 W.T.Reeves 提出的。粒子系统模型用大量粒子图元来描述景物，粒子的位置和形状随时间变化，并由随机数决定。火焰可被看成是喷出的许多粒子，众多粒子运动轨迹构造了火焰的模型，用随机数决定每个粒子的属性（如初始位置、初始速度、颜色、大小等）。现在该模型由用来描述草丛、森林等景象。

7. 5 分形造型应用

在日常生活中，花瓶、茶具等器皿以及商品的包装等表面大多都有魅力的花纹图案。分形图的特点就是“奇”、“美”，用分形图作为器皿的花纹图案，即把分形图映射到器皿表面称为一种图案设计，会产生很好的设计效果，称为分形图映射。

纹理图案及映射是计算机绘图的热点，纹理图案是指物体的表面细节。通常考虑两种类型的纹理图案。第一种是在光滑表面上描绘纹理图案，第二种是使表面呈现出凹凸不平的形状。映射中的关键技术是映射后图形不能失真。映射出现的失真简单说包括两个方面：表面展开程度和映射函数，前者属于图形的内在特性，不可改变。这样要减小失真只有选择合适的映射函数。映射函数即被映射的纹理图案和映射物体表面纹理图案间的一一对应关系。一般映射前的纹理图案在

一个二维坐标系中，映射后的图案绘制在三维坐标系中，因此映射前后对应关系应为二维点与三维点之间的对应关系。

例如，对于圆柱面和圆锥面两种映射物体表面，虽然二者使可展开面，但是对于圆柱体表面，在直角坐标系中，通过线性映射或三角函数映射可将一个二维图不失真产生的映射到表面上，但用同样方法映射到圆锥体表面要产生很大失真，我们可把圆锥体切分成许许多多多个近似圆柱体来进行映射，这样产生的失真将大大减小。对于一个不可展开面，一般很难找到不失真的映射函数。

根据上述理论，将一些分型图映射到正方体、圆柱体和球的表面上可产生漂亮的美术图案。分型图映射的过程实质上就是图形（或图案）得分块、旋转、平移等变换形成的。

7.6 分形造型编程设计

一、程序功能说明

分数分图形的许多领域还未开发,而且富于幻想色彩,引人入胜,可以模拟天际的积云,空中的闪电、地上的森林、海里的浪花、燃烧的火焰等,特别是近年来在分形图中的应用,可以设计地板、壁纸等。我们这里根据上述分形造型及分形图映射理论设计如下“分形图”应用程序,如图 7-9 所示,点击【分形图】菜单绘制龙状曲线,又称龙图,它由一条直线段开始,然后隆起变成等腰直角三角形的两腰,其斜边为该直线长,在对两腰分别做和直线段同样的变化,如此动态进行数此,得到一条龙状曲线。在【球体映射】中把上述龙形图案映射到球表面,形成纹理贴图。



图 7-9

二、程序设计步骤

1. 建工程名称为“分形图”单文档应用程序框架。
2. 根据 7.3 表中的定义编辑菜单资源，添加消息映射，建立消息映射函数完成有关的函数声明。

表 7.3 菜单资源表

3.

4.

5.

菜单标题	子菜单项标题	标示符 ID	相应处理函数
分形图	绘制龙图	ID_DRAW_DRAGON	OnDrawDragon()
分形图应用	球体映射	ID_SPHERE_ DRAGON	OnSphereDragon()

3 工添加绘图基类

添加基类方法请参见第一章手工添加绘图基类。这里添加两个基类：**BaseDraw** 和 **Dragon**。并在工作区中系统自动创建的相应的空文件中，分别添加以下基类的头文件（.h 文件）和应用文件（.cpp 文件）

// BaseDraw.h: interface for the CBaseDraw class.// BaseDraw 头文件

#if !defined(AFX_BASEDRAW_H__CB43CA20_175A_11D4_81FF_94DCC6655E1C__INCLUD

ED_)

```
#define AFX_BASEDRAW_H__CB43CA20_175A_11D4_81FF_94DCC6655E1C__INCLUDED_
```

```
#if _MSC_VER > 1000
```

```
#pragma once
```

```
#endif // _MSC_VER > 1000
```

```
#define pi 3.141592654
```

```
class CBaseDraw
```

```
{
```

程序代码见纸书

}

7. 6 课 后 练 习

1. 什么叫几何造型？有那三种模式？各有什么特点？
2. 比较 **CSG** 域 **B-rep** 方法的优缺点？
3. 分形造型对模型的基本要求？分形造型的常用模型？

4. 根据分形造型原理，自己设置或上网查找一些分形图案设计，加深对分形造型的理解。
5. 利用 **BaseDraw** 基类设计圆柱体上的龙的贴图。

第八章 消隐

因为计算机图形处理的过程中，不会自动消去隐藏部分，相反会将所有的线和面都显示出来，所以如果想真实显示三维物体，必须在视点确定之后，将对象表面上不可见的点、线、面消去。执行这种功能的算法，称为消隐算法。根据消隐对象的不同可分为：线消隐 (Hidden-line)和面消隐。

8. 1 线消隐

线消隐处理对象为线框模型，是以场景中的物体为处理单元，将一个物体与其余的 $k-1$ 个物体逐一比较，仅显示它可见的表面以达到消隐的目的。此类算法通常用于消除隐藏线。**1. 凸多面体的隐藏线消隐**

凸多面体是由若干个平面围成的物体。假设这些平面方程为

$$a_i x + b_i y + c_i z + d_i = 0, \quad i=1, 2, \dots, n \quad (8.1)$$

物体内一点 $P_0(x_0, y_0, z_0)$ 满足 $a_i x_0 + b_i y_0 + c_i z_0 + d_i < 0$ ，平面法向量 (a_i, b_i, c_i) 指向物体外部的。此凸多面体在以视点为顶点的视图四棱锥内，视点与第 i 个面上一点连线的方向为 (l_i, m_i, n_i) 。那么第 i 个面为自隐藏面的判断方法是：

$$(a_i, b_i, c_i) \times (l_i, m_i, n_i) > 0$$

对于任意凸多面体，可先求出所有隐藏面，然后检查每条边，若相交于某条边的两个面均为自隐藏面，根据任意两个自隐藏面的交线，为自隐藏线，可知该边为自隐藏边（自隐藏线应该用虚线输出）。

2. 凹多面体的隐藏线消隐

凹多面体的隐藏线消除比较复杂。• 假设凹多面体用它的表面多边形的集合表示，消除隐藏线的问题可归结为：一条空间线段 P_1P_2 和一个多边形 a ，判断线段是否被多边形遮挡。如果被遮挡，求出隐藏部分。以视点为投影中心，把线段与多边形顶点投影到屏幕上，将各对应投影点连线的方程联立求解，即可求得线段与多边形投影的交点。

如果线段与多边形的任何边都不相交，则有两种可能，线段投影与多边形投影分离或线段投影在多边形投影之中，前一种情况，线段完全可见。后一种情况，线段完全隐藏或完全可见。然后通过线段中点向视点引，若此射线与多边形相交，相应线段被多边形隐藏；否则，线段完全可见。

若线段与多边形有交点，那么多边形的边把线段投影的参数区间 $[0, 1]$ 分割成若干子区间，每个子区间对应一条子线段(如图 8-1 所示)，每条子线段上的所有点具有相同的隐藏性，如图所示。为进一步判断各子线段的隐藏性，首先要判断该子线段是否落在该多边形投影内。对于子线段与多边形的隐藏关系的判定方法与上述整条线段与多边形无交点时的判定方法相同。图

8-1 线段投影被分为若干子线段 把上述线段与所有需要比较的多边形依次进行隐藏性判断，记下各条边隐藏子线段的位置，最后对所有这些区域进行求并集运算，即可确定总的隐藏子线段的位置，余下的则是可见子线段。

8.2 面消隐

面消隐(Hidden-surface)处理对象为填色图模型，是以窗口内的每个像素为处理单元，确定在每一个像素处，场景中的物体哪一个距离观察点最近（可见的），从而用它的颜色来显示该像素。此类算法通常用于消除隐藏面。

8.2.1 区域排序算法基本思想：在图像空间中，将待显示的所有多边形按深度值从小到大排序，用前面可见多边形去切割后面的多边形，最终使得每个多边形要么是完全可见，要么是完全不可见。用区域排序算法消隐，需要用到一个多边形裁剪算法。当对两个形体相应表面的多边形进行裁剪时，我们称用来裁剪的多边形为裁剪多边形，另一个多边形为被裁剪多边形。算法要求多边形的边都是有向的，不妨设多边形的外环总是顺时针方向的，并且沿着边的走向，左侧始终是多边形的外部，右侧是多边形的内部。若两多边形相交，新的多边形可以用“遇到交点后向右拐”的规则来生成。于是被裁剪多边形被分为两个乃至多个多边形；我们把其中落在裁剪多边形外的多边形叫

作外部多边形；把落在裁剪多边形之内的多边形叫作内部多边形。 算法的步骤：

- (1) 进行初步深度排序，如可按各多边形 z 向坐标最小值(或最大值、平均值)排序。
- (2) 选择当前深度最小(离视点最近)的多边形为裁剪多边形。
- (3) 用裁剪多边形对那些深度值更大的多边形进行裁剪。
- (4) 比较裁剪多边形与各个内部多边形的深度，检查裁剪多边形是否是离视点最近的多边形。如果裁剪多边形深度大于某个内部多边形的深度，则恢复被裁剪的各个多边形的原形，选择新的裁剪多边形，回到步骤(3)再

做, 否则做步骤(5)。

(5) 选择下一个深度最小的多边形作为裁剪多边形, 从步骤(3)开始做, 直到所有多边形都处理过为止。在得到的多边形中, 所有内部多边形是不可见的, 其余多边形均为可见多边形。**8.2.2 深度缓存(Z-buffer)算法**

深度缓存(Z-buffer)是一种在图像空间下的消隐算法, 包括: 帧缓冲器 -- 保存各像素颜色值(CB); z 缓冲器 -- 保存各像素处物体深度值(ZB)。其中 z 缓冲器中的单元与帧缓冲器中的单元一一对应。

深度缓存(Z-buffer)思路: 先将 z 缓冲器中个单元的初始值置为-1(规范视见体的最小 n 值)。当要改变某个像素的颜色值时, 首先检查当前多边形的

深度值是否大于该像素原来的深度值（保存在该像素所对应的 Z 缓冲器的单元中），如果大于，说明当前多边形更靠近观察点，用它的颜色替换像素原来的颜色；否则说明在当前像素处，当前多边形被前面所绘制的多边形遮挡了，是不可见的，像素的颜色值不改变。Z-buffer 算法的步骤如下：

- (1) 初始化 ZB 和 CB ，使得 $ZB(i, j)=Z_{\max}$ ， $CB(i, j)=$ 背景色。其中， $i=1, 2, \dots, m$ ， $j=1, 2, \dots, n$ 。
- (2) 对多边形 a ，计算它在点 (i, j) 处的深度值 $z_{i, j}$ 。
- (3) 若 $z_{ij} < ZB(i, j)$ ，则 $ZB(i, j)=z_{ij}$ ， $CB(i, j)=$ 多边形 a 的颜色。
- (4) 对每个多边形重复(2)、(3)两步。最后，在 CB 中存放的就是消隐后的图

形。

8.2.3 扫描线算法在多边形填充算法中，活性边表的使用取得了节省运行空间的效果。用同样的思想改造 Z-buffer 算法：将整个绘图区域分割成若干个小区域，然后一个区域一个区域地显示，这样 Z 缓冲器的单元数只要等于一个区域内像素的个数就可以了。如果将小区域取成屏幕上的扫描线，就得到扫描线 Z 缓冲器算法。扫描线算法描述 **for** (各条扫描线)

{将帧缓冲器 $I(x)$ 置为背景色；

将 Z 缓冲器的 $Z(x)$ 置为最大值；

for (每个多边形)


```
{  求出多边形在投影平面上的投影与当前扫描线的相交区间  
  
  for   (该区间内的每个像素)  
  
    if ( 多边形在此处的  $Z$  值小于  $Z(x)$  )  
  
      {  置帧缓冲器  $I(x)$  值为当前多边形颜色;  
  
        置  $Z$  缓冲器  $Z(x)$  值为多边形在此处的  $Z$  值;  }  
  
    }  
  
}
```

用计算机编制绘图程序时,除了利用上述消隐方法外,常常考虑图形的

几何特点（如下面要讲的图形的几何构造等）使图形生成更直观、方便或快捷。

8.3 图形几何构造

从图形几何构造的角度讲，一条边由两个点组成，一个面由几条或若干条边围成，场景中的任意物体都是由点、边、面按一定的拓扑结构组成的。称点、边、面为基本元素，简称基元，其中，点是最小的基元。对图形设计而言，通过几何点并按一定的拓扑结构可以完成场景中任意物体的几何描述，同理，用户可操作的物体对象还有线（边）、三角形小面、四边形小面、**n** 边形、圆、弧、异型面、立方体、球、圆柱和圆环面等线元、面元和体元，点元、线元、面元和体元同称为图形系统的基元。

基元是建立图形系统和应用图形系统的基石。当我们需要通过程序来显示一个特定的场景，点、边和三角形一般为基本的图元,任何一个多边形，无论为单连通还是多连通区域都可以通过系化规则转化成多个三角形来处理。对于二维和三维及图形软件系统，最小的几何图元是几何定点，其次为直线（边）、面和其它面元与体元。对于一个三维实体，无论为凸的、凹的有洞的还是无洞的，计算机所显示的是它的表面。而表面是通过小面来逼近的，这一过程称为表面离散化。在物体表面急过渡区域常采用较多的小面来逼近，在较为平坦的区域，可以用较少的小面来逼近。

定义基元是基于图形描述和图形应用的方便性的而言。基元往往在世界坐标中进行描述并在光栅系统（屏幕坐标或窗口坐标系中）完成显示任务的。可以给基元赋属性，比如，几何定点的属性由几何坐标、顶点颜色、定点法线等，直线的属性有直线的几何拓扑、线宽、线型等，面的属性有面的几何拓扑、填充模式等。在构造几何数据时，往往需要采用链表、队列、树等结构。下面介绍常用的几种基元的几何构造方法：

（1）直线段

一条直线段由两个端点定义，为方便可采用 Windows 编程中经常使用的 MoveTo()和 LineTo()函数绘制直线。首先提供一对顶点，来定义第一条直线

段，以后每增加一条直线段，如第 i 条线段由第 $i-1$ 和第 i 个顶点来定义，其颜色由第 $i-1$ 个顶点的颜色决定。这样构成了一条折线。绘制一条闭合的折线，可将最后的顶点和第一个顶点进行连线，其颜色由最后一个顶点的颜色所决定。

(2) 小面

一个物体，无论其表面形状多么复杂，计算机通常将它视为由若干小面组成的。将一个曲面分割成若干小面，首先要对曲面进行网格的划分，称为有限单元化。其次要获得网格的各种顶点坐标，对顶点进行几何描述形成单个边，并由边构成面，通过面循环构成整个物体的拓扑结构。接着要赋予顶

点信息（如颜色、纹理坐标、法向量等等），这三个重要的步骤通常称为前处理，最后作为输入信息传递给图形处理系统。用什么样的小面逼近曲面是十分重要的，比如，将球用标示地理位置的经纬线来划分，可得到许多四边形小面，而两端要用三角形来逼近，事实上四边形可划分为两个三角形，这样解决了三角形的着色问题也就解决了所有面的着色问题。小面通常为三角形和四边形的小面。

对于四边形也可在一个四边形上延伸出一个四边形片，如图 8-2 所示。四边形的顶点处理的顺序定义如表 8-1。

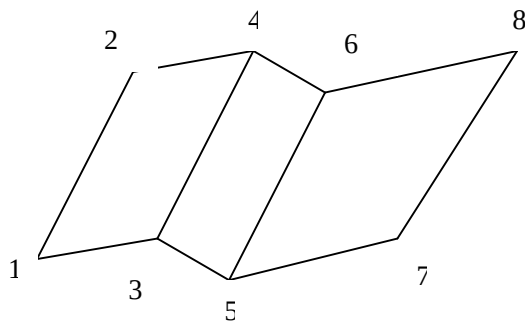


图 8-2 四边形片示意图

表 8-1 四边形片中顶点处理顺序

四边形序号	顶点处理顺序
1	顶点 1, 顶点 2, 顶点 3, 顶点 4
2	顶点 3, 顶点 3, 顶点 4, 顶点 5
3	顶点 5, 顶点 6, 顶点 7, 顶点 8

由于曲面都是由三角形小面和四边形小面来逼近, 而三角形小面是自定义的基本面元, 因而需要对四边形小面进行细化。一个四边形可细化为两个三角形, 如图 8-3 所示。

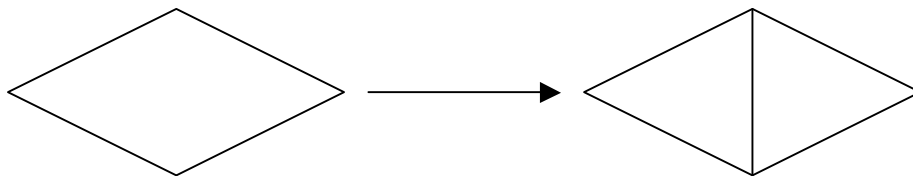


图 8-3 一个四边形细化为两个三角

对于任意一个平面区域或曲面，甚至有离散数据点组成的区域，都应该进行三角剖分，三角剖分法事实上属于几何拓扑分析方法。

三角形由三个顶点来定义，称为三角形基元。多个三角形一般由三角形片或三角形扇来定义。

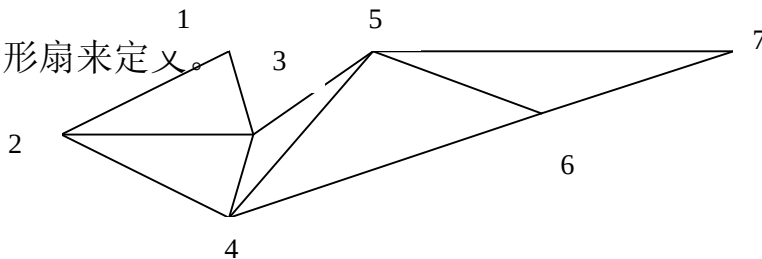


图 8-4 三角形片示意图

如图 8-4 定义了一个三角形片，从图中可知，每增加一个顶点便可增加一个三角形。三角形的顶点处理的顺序可以定义为表 8-2。

表 8-2 三角形的顶点处理的顺序

三角形序号	顶点处理顺序
1	顶点 1, 顶点 2, 顶点 3,
2	顶点 3, 顶点 2, 顶点 4,
3	顶点 3, 顶点 4, 顶点 5,
4	顶点 5, 顶点 4, 顶点 6,
5	顶点 5, 顶点 6, 顶点 7,

图 8-5 表示了一个三角形扇，尽管每增加一个顶点仍增加一个三角形，但每个三角形共用一个顶点，显然这种方法表示的终极部分会很方便，表 8-2 表示三角扇形中三角形的顶点处理顺序。

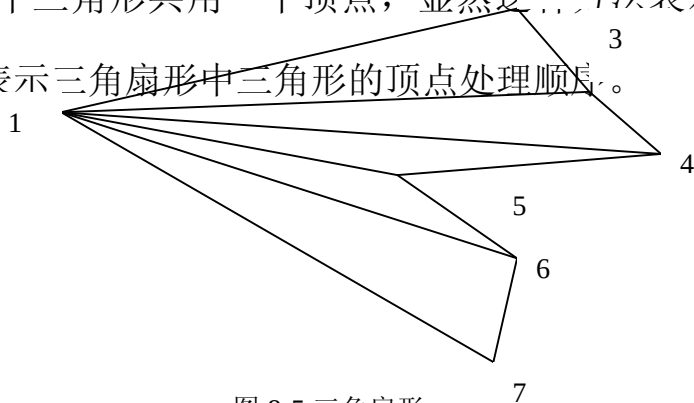


图 8-5 三角扇形

表 8-2 三角扇形中三角形的顶点处理顺序

三角形序号	顶点处理顺序
1	顶点 1, 顶点 2, 顶点 3,
2	顶点 1, 顶点 3, 顶点 4,
3	顶点 1, 顶点 4, 顶点 5,
4	顶点 1, 顶点 5, 顶点 6,

5	顶点 1, 顶点 6, 顶点 7,
---	-------------------

(3) 边的可见性

一个面元（比如三角形）往往为一个凸区域，要绘制图 8-6 左边所示的凹多边形会很费事，可通过定义边的可见性来达到这一目的。将凹区域分成两个凸四边形，使其中增加的连线不可见，从而达到绘制多边形的目的。

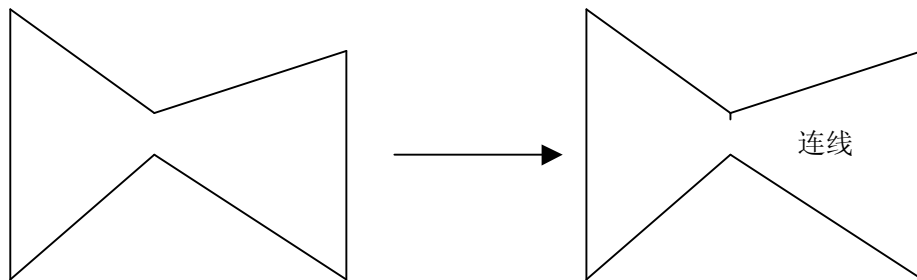


图 8-6 凹区域分成两个凸四边形

下面“消隐”程序设计中，利用远近法绘制圆环。把圆环看成由许许多多四边形基元构成，首先计算各处四边形的顶点坐标，计算各中心点的 Z 坐标并进行排序，根据 Z 值大小沿 Z 轴方向由远及近依次绘制出各四边形，形成整个图形，从而实现隐线处理。

8.4 消隐技术编程案例

一、程序设计功能说明

业搜---www.yeaso.com

CAD 教育网制作www.cadedu.com

利用上述基本原理编制的“消隐”应用程序，包括消隐绘制凸多面体和利用远近法绘制的圆环。如图 8-4-1 所示。

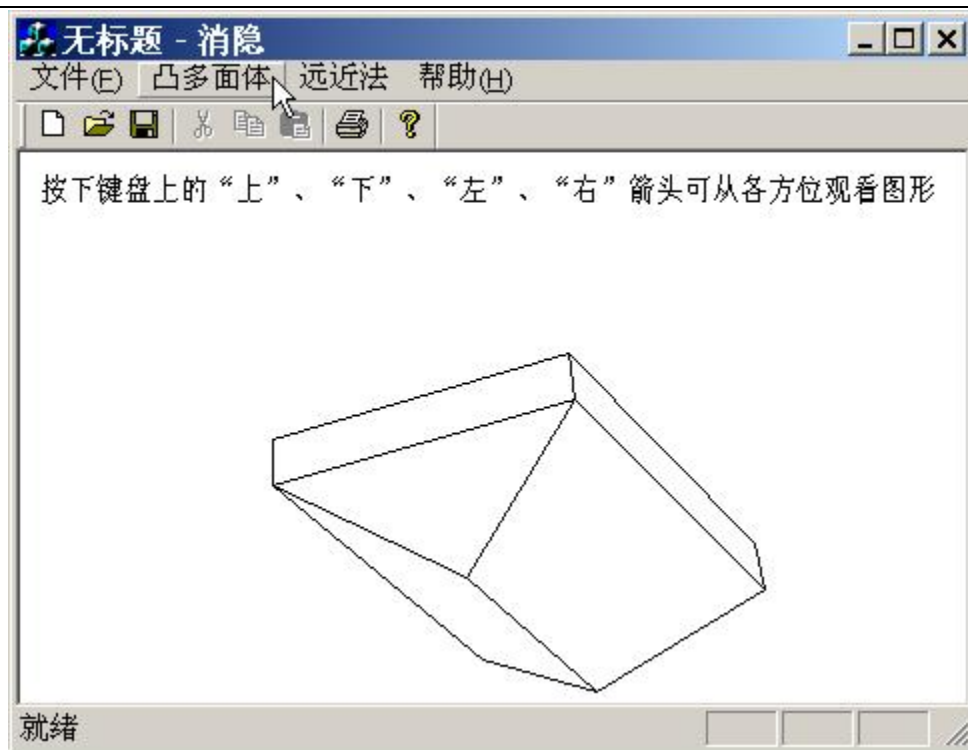


图 8-4-1

二、程序设计步骤

1. 创建单文档应用程序框架
2. 编辑菜单资源，添加消息处理函数

设计如图 1-4 所示的菜单项。在工作区的【ResourceView】标签中，单击 Menu 项左边“+”，然后双击其子项 IDR_MAINFRAME，并根据 8.1 表中的定义编辑菜单资源，建立消息映射函数，完成有关的函数声明。

表 8.1 菜单项的消息处理函数

菜单标题	菜单项 ID	消息	消息处理函数
------	--------	----	--------

凸多面 体	ID_TUMIANTI	CONMMAN	OnTumianti
	ID_	CONMMAN	On

3. 添加程序结构代码，在 CMyView.h、CMyView.cpp 文件中相应位置添加如下代码：

```
// 消隐 View.h : interface of the CMyView class
```

```
//
```

```
////////////////////////////////////
```

[程序代码见纸书](#)

```
}
```

基类代码说明:

为绘制凸多边形与环面，与前面许多绘制方法构建基类不同，本程序通过构建了如下函数：Project(float X, float Y, float Z)、WLineTo(float X, float Y, float Z, CDC* pDC)、WMoveTo(float X, float Y, float Z, CDC* PDC)、ReadVertics()、ReadFaces()、VisionVector(int St1)、NormalVector(int St1, int St2, int St3)、ScaleProduct(float v1, float v2, float v3, float n1, float n2, float n3)、DrawFace(CDC* pDC)、DrawObject()、VisionPoint()来实现。

8.10 练 习 题

1. 消隐的意义
2. 试简述单个凸多面体消隐的基本方法。
3. 编程绘制经过消隐的正方体。

第九章 真实感图形学

真实感图形学是计算机图形学中的一个重要组成部分，在日常工作、学习和生活中已经有了非常广泛的应用。它的基本要求就是在计算机中生成三维场景的真实感图形图像。对于场景中的物体，要得到它的真实感图像，就要对它进行透视投影，并作隐藏面的消隐，然后计算可见面的光照明暗效果，得到场景的真实感图像显示。在本章中，首先介绍颜色视觉几个不同的光照明模型。然后用 **Visual C++** 来实现通过建立“真实感图形学”程序，

包括颜色模型类、光照模型类、材质类、基于颜色缓冲区和深度缓冲区的直线基元和三角形基元类、基于 **z-buffer** 算法的场景深度消隐类、消隐绘制几何球体类等。

9.1 颜色模型

原理上讲任何一种颜色都可以用红、绿、蓝三原色按照不同比例混合来得到。光照明模型中，就是分别计算 **R**、**G**、**B** 三个分量的光强值，得到某个像素点上颜色值，即所谓 **RGB** 颜色模型。

9.1.1 CIE 色度图

根据 **CIE** (国际照明委员会) 选取的标准红、绿、蓝三种光的波长, 分别为: 红光, **R**, $\lambda_1 = 700nm$; 绿光, **G**, $\lambda_2 = 546nm$; 蓝光, **B**, $\lambda_3 = 435.8nm$ 。这样光颜色的匹配可以用式子表示为: $c = rR + gG + bB$ 被称为 **CIE-RGB** 系统。其中权值 **r**、**g**、**b** 为颜色匹配中所需要的 **R**、**G**、**B** 三色光的相对量, 也就是三刺激的值。用等能标准三原色来匹配任意颜色的光谱三刺激值时, 部分三刺激值是负数, 表明不可能靠混合红、绿、蓝三种光来匹配对应的光, 而只能在给定的光上叠加负值对应的原色, 来匹配另两种原色的混合, 但实际上并不存在负的光强。**CIE-XYZ** 系统利用三种假想的标准原色 **X** (红)、**Y**

（绿）、**Z**（蓝），得到的颜色匹配函数的三刺激值都是正值。该系统的光颜色匹配函数定义为如下的一个式子：

$$c = xX + yY + zZ$$

用 **R**、**G**、**B** 三原色（**CIE-XYZ** 标准原色）的单位向量定义一个三维颜色空间，在这三维空间中，一个颜色刺激（**C**）就可以表示为一个以原点为起点的向量，该三维向量空间称为（**R**、**G**、**B**）三刺激空间，该空间落在第一象限，该空间中的向量的方向由三刺激的值确定，因而向量的方向代表颜色。为了在二维空间中表示颜色，我们在三个坐标轴上对称的取一个截面，

该截面通过 (R)、(G)、(B) 三个坐标轴上的单位向量，因而可知截面的方程为 $(R) + (G) + (B) = 1$ 。该截面与三个坐标平面的交线构成一个等边三角形，它被称为色度图。把色度图投影到 **XY** 平面上，所得到的马蹄形区域称为 **CIE** 色度图（图 9-1-1），马蹄形区域的边界和内部代表了所有可见光的色度值，色度图的边界弯曲部分代表了光谱在某种纯度为百分之百的色光。图中央的一点 **C** 表示标准白光，色度图还可用于定义各种图形设备的颜色域，这里不再详细介绍了。虽然色度图和三刺激值精确描述颜色的标准，但是较复杂，在计算机图形学中，通常使用一些基于三维颜色空间的通俗易懂的颜色系统来描述。

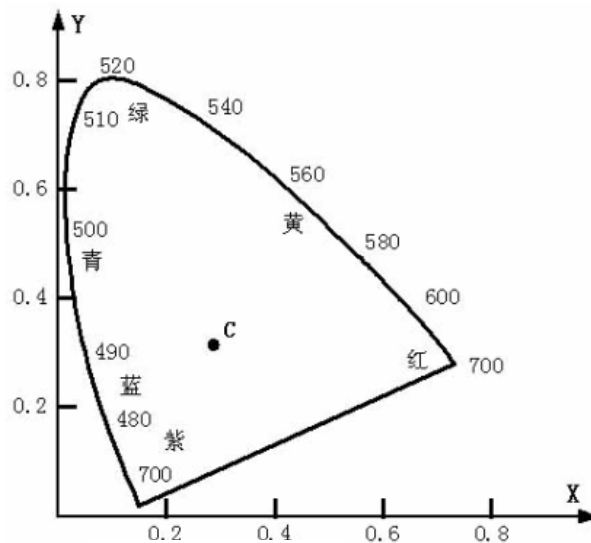


图 9-1 CIE 色度图

9.1.2 常用的颜色模型

颜色模型就是指三维颜色空间中包含某个颜色域的所有颜色的一个可见光子集。**RGB** 颜色模型是三维直角坐标颜色系统的一个单位正方体。真实感图形学中的主要的颜色模型也是 **RGB** 模型，除了 **RGB** 颜色模型，还有常见的 **CMY**，**HSV** 等颜色模型。

彩色阴极射线管等彩色光栅图形显示设备中多使用 **RGB** 颜色模型。如图 9-1-2 所示，红、绿、蓝原色为加性原色，即不同原色混合在一起可以产生复合色。**RGB** 颜色模型所覆盖的颜色域取决于显示设备荧光点的颜色特性，是与硬件相关的。

以红、绿、蓝的补色青 (**Cyan**)、品红 (**Magenta**)、黄 (**Yellow**) 为原色构成的 **CMY** 颜色模型, 常用于从白光中滤去某种颜色, 又被称为减性原色系统, 如图 **9-1-3** 所示。**CMY** 颜色模型对应的直角坐标系的子空间与 **RGB** 颜色模型所对应的子空间几乎完全相同。差别仅仅在于前者的原点为白, 而後者的原点为黑。前者是定义在白色中减去某种颜色来定义一种颜色, 而后者是通过从黑色中加入颜色来定义一种颜色。 **RGB** 和 **CMY** 颜色模型都是面向硬件的。此外还有应于画家的配色的方法的 **HSV(Hue,Saturation,Value)** 颜色模型, 是面向用户。画家用改变色浓和色深的方法来从某种纯色获得不

同色调的颜色。其做法是：在一种纯色中加入白色以改变色浓，加入黑色以改变色深，同时加入不同比例的白色，黑色即可得到不同色调的颜色。

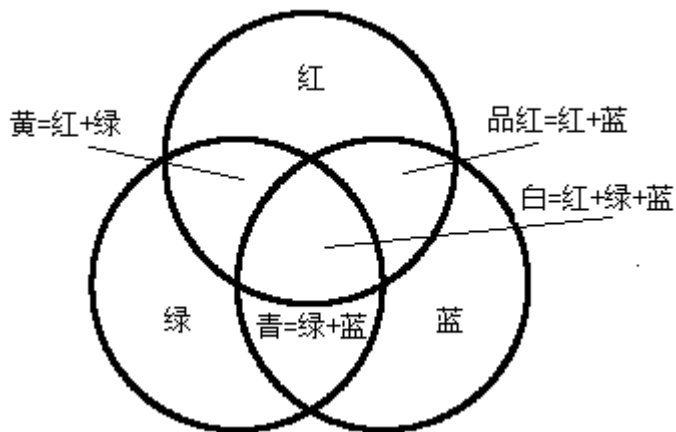


图 9-2 CMY 原色的加色效果

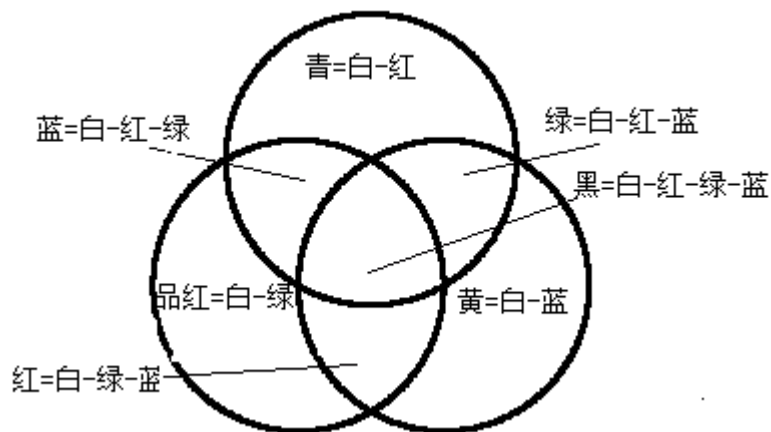


图 9-3 CMY 原色的减色效果

在本章建立的“真实感图形学”程序中的颜色模型 **CFloatColor** 类是在上述 **RGB** 颜色模型基础上，添加 **Alpha** 分量来实现的确定物体上顶点、

边、小面的颜色。当两种颜色进行混合时，**Alpha** 因子决定融合操作中两种颜色成分的比例，程序中处理颜色运算多采用浮点小数形式。

9.2 简单光照明模型

光照射到物体表面时，光线可能被吸收、反射和透射。被物体吸收的部分转化为热。反射、透射的光进入人的视觉系统，使我们能看见物体。为模拟这一现象，建立一些数学模型来替代复杂的物理模型，光照明模型是在已知物体物理形态和光源性质的条件下，计算场景的光照明效果的数学模型。

9. 2. 1 Phong 光照明模型

Phong 模型是一个模拟物体表面对光的反射作用的经验模型, 光源被假定为点光源, 反射作用被细分为镜面反射(**Specular Reflection**)和漫反射(**Diffuse Reflection**)。简单光照明模型只考虑物体对直接光照的反射作用, 而物体间的光反射作用, 只用环境光(**Ambient Light**)来表示。下面介绍 **Phong** 简单光照明模型光反射作用中各个组成部分。

1. 理想漫反射

自一个方向的光, 经漫反射使光均匀向各方向传播。漫反射是由表面的粗糙不平引起的, 与视点无关, 漫反射光的空间分布是均匀的。记入射光强

为 I_p ，物体表面上点 P 的法向为 N ，从点 P 指向光源的向量为 L ，两者间的夹角为 **theta**，当 L 、 N 为单位向量时，则漫反射光强为：

$$I_d = I_p K_d * (L \cdot N)$$

其中， K_d 是与物体有关的漫反射系数， $0 < K_d < 1$ 。在有多光源的情况下，表示为：

$$I_d = K_d \sum_i I_{p,i} * (L_i \cdot N)$$

漫反射光的颜色由入射光的颜色和物体表面的颜色决定，在 **RGB** 颜色模型下，漫反射系数 K_d 的三个分量 K_{dr}, K_{dg}, K_{db} 分别代表 **RGB** 三原色的漫反射系

数，反映物体的颜色的，通过调整它们，可以设定物体的颜色。入射光强 I 也可分为三个分量 I_r, I_g, I_b ，通过这些分量的值来调整光源的颜色。

2. 镜面反射光

对于理想镜面，反射光集中在一个方向，并遵守反射定律。对一般的光滑表面，反射光集中在一个范围内，且由反射定律决定的反射方向光强最大。因此，对于同一点来说，从不同位置所观察到的镜面反射光强是不同的。将 $\mathbf{V}$ 和 $\mathbf{R}$ 都格式化为单位向量，镜面反射光强可表示为：

$$I_s = I_p \cdot K_s (R \cdot V)^n,$$

Ks 是与物体有关的镜面反射系数, α 为视线方向 **V** 与反射方向 **R** 的夹角, **n** 为反射指数, 反映了物体表面的光泽程度, 一般为 **1~2000**, **n** 越大物体表面越光滑。**R** 可由 $R = N2\cos\theta - L = 2N(N \cdot L) - L$ 计算镜面反射光, 在反射方向附近形成得很亮的光斑, 称为高光现象。

对多个光源的情形, 镜面反射光强可表示为:

$$I_s = K_s \cdot \sum_{i=1}^m [I_{p,i} \cdot (R_i \cdot V)^n]$$

镜面反射光产生的高光区域只反映光源的颜色, 如在红光的照射下, 一个物体的高光域是红光, 镜面反射系数 **Ks** 是一个与物体的颜色无关的参

数，在简单光照明模型中，只能通过设置物体的漫反射系数来控制物体的颜色。

3. 环境光

环境光是指光源间接对物体的影响，是在物体和环境之间多次反射，最终达到平衡时的一种光。我们近似地认为同一环境下的环境光，其光强分布是均匀的，它在任何一个方向上的分布都相同。例如，透过厚厚云层的阳光就可以称为环境光。在简单光照明模型中，我们用一个常数来模拟环境光，

用式子表示为: $I_e = I_a \cdot K_a$ 。其中: I_a 为环境光的光强, K_a 为物体对环境光的反射系数。

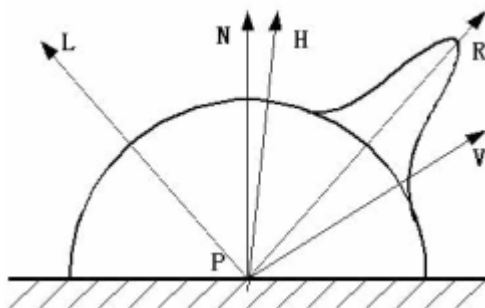


图 9-4 Phong 模型中的几何量

4. Phong 光照明模型

Phong 光照明模型: $I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (R \cdot V)^n$ 8

其中物体表面上一点 **P** 反射到视点的光强 **I** 为环境光的反射光强 **I_e**、理想漫反射光强 **I_d**、和镜面反射光 **I_s** 的总和

Phong 光照明模型是真实感图形学中提出的第一个有影响的光照明模型，生成图像的真实度已经达到可以接受的程度；但是在实际的应用中，由于它是一个经验模型，还具有以下的一些问题：用 **Phong** 模型显示出的物体如塑料，没有质感；环境光是常量，没有考虑物体之间相互的反射光；镜面反射的颜色是光源的颜色，与物体的材料无关；镜面反射的计算在入射角很

大时会产生失真等。在后面的一些光照明模型中，对上述的这些问题都作了一定的改进。在 **Phong** 光照明模型中，由于光源和视点都被假定为无穷远，最后的光强计算公式就变为物体表面法向量的函数，当今流行的显示系统是用多边形表示的物体，每一个多边形法向一致，因而多边形内部的象素的颜色都是相同的，而且在不同法向的多边形邻接处，不仅有光强突变，而且还会产生马赫带效应，即人类视觉系统夸大具有不同常量光强的两个相邻区域之间的光强不连续性。为了保证多边形之间的光滑过渡，使连续的多边形呈现匀称的光强，下面介绍增量式光照明模型。

8. 2. 2 增量式光照明模型

增量式光照明模型是在每一个多边形的顶点处计算出合适的光照明强度或参数,然后在各个多边形内部进行均匀插值,得到多边形的光滑颜色分布。它包含两个主要的算法:双线性光强插值和双线性法向插值,又被分别称为 **Gouraud** 明暗处理和 **Phong** 明暗处理。

1. Gouraud 明暗处理

Gouraud 明暗处理是先计算物体表面多边形各顶点的光强,然后用双线性插值,求出多边形内部区域中各点的光强。基本算法描述如下:

(1) 计算多边形顶点的平均法向,

(2) 用 **Phong** 光照明模型计算顶点的平均光强,

(3) 插值计算离散边上的各点光强

(4) 插值计算多边形内域中各点的光强。

2. Phong 明暗处理

Gouraud 明暗模型具有计算速度快, 相邻多边形之间的颜色突变问题也得到解决, 产生的图像颜色过渡均匀, 图形显得非常光滑的优点, 但是, 由于采用光强插值, 它的镜面反射效果不太理想, 而且相邻多边形的边界处的马赫带效应不能完全消除。**Phong** 提出的双线性法向插值以时间为代价, 可

以部分解决上述的弊病。双线性法向插值将镜面反射引进到明暗处理中，解决了高光问题。与双线性光强插值相比，该方法有如下特点：

- a. 保留双线性插值，对多边形边上的点和内域各点，采用增量法。
- b. 对顶点的法向量进行插值，而顶点的法向量，用相邻的多边形的法向量作平均。
- c. 由插值得到的法向，计算每个象素的光亮度
- d. 假定光源与视点均在无穷远处，光强只是法向量的函数。

双线性光强插值可以有效的显示漫反射曲面，它的计算量小；而双线性法向插值与双线性光强插值相比，可以产生正确的高光区域，但它的计算量要大的多。当然，这两个插值算法的增量式光照明模型本身也都存在着一些缺陷，具体表现为：用这类模型得到的物体边缘轮廓是折线段而非光滑曲线；由于透视的原因，使等间距扫描线产生不均匀的效果；插值结果决定于插值方向，不同的插值方向会得到不同的插值结果等。

在后面“真实感图形学”程序中，应用着色方法即为区域填充的计算模型，包括平面明暗着色模型和光滑明暗着色模型。平面明暗着色处理就是用一种单一颜色填充一多边形，多边形的颜色一般由第一个顶点的颜色来决

定。对于一条直线，也将被着色为第一个顶点的颜色，平面明暗着色处理方法会产生不真实的效果，但这种方法非常适合快速成像和速度重于图片精度的场合。光滑明暗着色处理不再以单一颜色填充多边形，其填充颜色颜色与各顶点的颜色有关，多边形内的颜色是由顶点的颜色（或计算机所需要的其它属性）经过线性插值计算获得的，而每一个顶点的颜色保持不变。这样越靠近顶点，颜色就越是突出。这种方法将多边形的内部填上平滑的渐变色，因而有更强烈的外观。

9. 3 局部光照明模型

在真实感图形学中，局部光照明模型为仅处理光源直接照射物体表面的光照明模型。简单光照明模型，可以计算经点光源照明的物体表面的光强，实际上就是一种局部关照明模型，认为镜面反射项与物体表面的材质无关。可以处理物体之间光照的相互作用的模型称为整体光照明模型。下面介绍后面的整体光照明模型计算局部光强时被经常使用的一个更复杂更普遍的局部光照明模型。

9.3.1 局部光照明模型

考虑电磁波对满反射和镜面反射的影响局部光照明模型表示为：

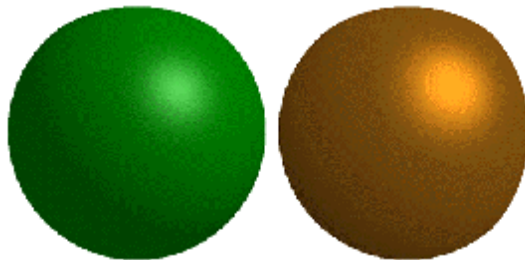
$$I_r = I_a K_a + I_i (N \cdot L) d\omega (K_d R_d + K_s R_s)$$

式中, I_r 为直接光照下物体表面表现出来的反射光强; $I_a K_a$ 的定义与简单光照明模型相同, 表示环境光的影响; 式子中的最后一项是考虑了物体表面性质的反射光强度量。

与上一节介绍的简单光照模型的比较, 本节讨论的局部光反射模型有如下的一些优点:

(1) 局部光照明模型是基于入射光能量导出的光辐射模型, 而简单光反射模型基于经验, 显然前者更具有理论基础。

(2) 局部照明模型的反射项以实际物体表面的微平面理论为基础, 反映表面的粗糙度对反射光强的影响。



(3) 局部照明模型的高光由 **Fresnel** 定律, 根据材料的物理性质决定颜色, 而简单光照模型只以高光颜色与材料无关。

(4) 简单光照模型在入射角接近 **90°** 时会产生失真现象, 而在局部光照模型中可以很好的改进这一点。

9-5 图左边的球用 Phong 光照模型显示, 右边的球用局部光照模型显示两者在高光域有明显的区别

(5) 用简单光照模型生成的物体图像, 看上去象塑料, 显示不出磨亮的金属光泽, 而在局部光照明模型中, 反射光强的计算考虑了物体材质的影响, 就可以模拟金属的光泽。

9. 4 光透射模型

对于透明或半透明的物体, 透射光是在光线与物体表面相交时, 产生折射, 经折射后的光线将穿过物体而在物体的另一个面射出现象。视点在折射光线的方向上时, 就可以看到透射光。本节中将介绍一些典型的光透射模型。

9. 4. 1 透明效果的简单模型

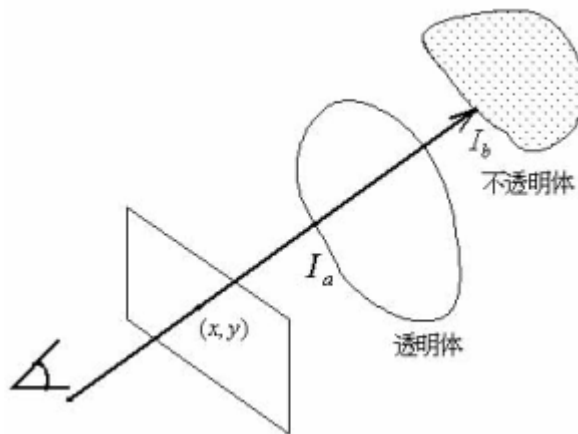
透过透明物体材料可看到其后面的物体。但是由于光的折射，光透过透明物体后通常会改变光的传播方向，要在真实感图形学中模拟折射，需要较大的计算量，在 Whitted 和 Hall 提出光透射模型之前，为了能够看到一个透明物体后面的东西，就有一些透明效果模拟的简单方法。

在这类方法中主要的是颜色调和法，该方法不考虑透明体对光的折射以及透明物体本身的厚度，光通过物体表面是不会改变方向的，故可以模拟平面玻璃，前面介绍的隐藏面消除算法都可以用于实现模拟这种情况。

设 t 是物体的透明度， $t=0$ 表示物体是不透明体； $t=1$ 表示物体是完全透体。可以看到物体后面的背景和其他物体，这些物体的前后位置可以通过隐藏面消除算法计算出来。实际上，我们最终所看到的颜色，是物体表面的颜色和透过物体的背景颜色的叠加。如图 8-4-1 所示，设过象素点 (x,y) 的视线与物体相交处的颜色(或光强)为 I_a ，视线穿过物体与另一物体相交处的颜色(或光强)为 I_b ，则象素点 (x,y) 的颜色(或光强)可由如下颜色调和公式计算

$$I = tI_b + (1-t)I_a$$

其中, **la** 和 **lb** 可由简单光照明模型计算。由于未考虑透射光的折射, 以及透明物体的厚度, 颜色调和法只能模拟玻璃的透明或半透明效果。而在我们后面介绍的两个光透射模型中, 都从光的折射角度来计算透射光强, 可以很好的模拟光的透射。



9-6 颜色调和模拟透明效果

9. 4. 2 Whitted 光透射模型

在简单光照明模型的基础上，加上透射光一项，再加上反射光一项，

Whitted 整体光照模型为：

$$I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (H \cdot N)^n + I_t K_t' + I_s K_s'$$

这里，**Is** 为镜面反射方向的入射光强度；**Ks'** 为镜面反射系数，为 **0~1** 之间的一个常数；其大小同样取决于物体的材料。

4. 4. 3 Hall 光透射模型

Hall 光透射模型是在 **Whitted** 光透射模型的基础上推广而来的，它能够模拟透射高光的效果，同时还可以处理理想的漫透射。

用 **Lambert** 余弦定律描述点 **P** 处的漫透射光的光强为:

$$I_{dt} = I_p \cdot K_{dt} \cdot (-N \cdot L)$$

其中 **I_p** 为入射光的强度, 即点光源的强度, **K_{dt}** 为物体的漫透射系数, 在 **0** 与 **1** 之间。**L** 为光源方向, **N** 为面法向。

Hall 用下面的式子模拟透射高光现象:

$$I_t = I_p \cdot K_t \cdot (T \cdot V)^n$$

其中, I_t 为透射光在视线方向的强度, I_p 为点光源的强度; K_t 为物体的透明系数, n 为反映物体表面光泽的常数。

为减少计算量, 可以和简单光照明模型一样, 作如下假设:

- (1) 假定光源在无穷远处, 光线方向 L 为常量;
- (2) 视点在无穷远处, 视线方向 V 为常量;
- (3) 用 $(H_t \cdot N)$ 代替 $(T \cdot V)$, 这里 H_t 可以视为一个虚拟的理想透射面的法向, 使视线恰好为光线的折射方向。

在使用 **Hall** 光透射模型时, 注意如下的几点:

(1) 只有视点与光源在透明物体的两侧时, 才能透过透明体看到透射高光。

(2) 光线射入和射出透明体, 均会产生折射, 我们通常不考虑第一次折射。

(3) 折射的临界角现象, 当光线从高密度介质射向低密度介质, 而且入射角大于临界角时, 不再发生折射, 而产生内部反射, 这时的临界角为:

$$\theta_c = \arcsin \frac{\eta_2}{\eta_1} = \arcsin \eta$$

9. 4. 4 简单光反射透射模型

综合简单光照明模型, **Whitted** 光透射模型和 **Hall** 光透射模型, 可得简单光反射透射模型:

$$\begin{aligned} I = I_a K_a + \sum_i I_{pi} [K_{di} (L_i \cdot N) + K_{si} (H_{si} \cdot N)^{\eta_i}] \\ + \sum_j I_{pj} [K_{dt} (-N \cdot L_j) + K_{st} (N \cdot H_{tj})^{\eta_t}] + I_t K_t + I_s K_s \end{aligned}$$

上面的式子可以作为本章前面几节的一个很好的总结。

9. 5 纹理及纹理映射

用前面几节介绍的方法生成的物体图像, 由于其表面过于光滑和单调, 看起来反而不真实, 这是因为在现实世界中的物体, 其表面通常有它的表面细节, 即各种纹理, 如刨光的木材表面上有木纹, 建筑物墙壁上有装饰图案, 机器外壳表面有文字说明它的名称、型号等。它们是通过颜色色彩或明暗度变化体现出来的表面细节, 这种纹理称为颜色纹理。另一类纹理则是由于不规则的细小凹凸造成的, 例如桔子皮表面的皱纹。可以用纹理映射的方法给计算机生成的图像加上纹理。在本节中, 我们将介绍纹理的类型、纹理的定义方法以及纹理映射的一些原理。

9. 5. 1 纹理的概述

现实世界中的物体，其表面往往有各种表面细节。从根本上说，纹理是物体表面的细小结构，它可以是光滑表面的花纹、图案，是颜色纹理，这时的纹理一般都是二维图像纹理，还有三维纹理，纹理还可以是粗糙的表面(如桔子表面的皱纹)，它们被称为几何纹理，是基于物体表面的微观几何形状的表面纹理，一种最常用的几何纹理就是对物体表面的法向进行微小的扰动来表现物体表面的细节。纹理映射是把我们得到的纹理映射到三维物体的表面的技术。对于纹理映射，我们需要考虑以下三个问题：

(1) 考察简单光照明模型，我们需要了解，当物体上的什么属性被改变，就可产生纹理的效果。我们先给出简单光照明模型的式子：

$$I = I_a K_a + K_d I_d (N \cdot L) + K_s I_s (N \cdot H)^n$$

通过改变的物体属性，如漫反射系数、物体表面的法向量来改变物体的颜色，得到纹理的效果。

(2) 在真实感图形学中，我们可以用如下的两种方法来定义纹理：

图像纹理：将二维纹理图案映射到三维物体表面，绘制物体表面上一点时，采用相应的纹理图案中相应点的颜色值。

函数纹理：用数学函数定义简单的二维纹理图案，如方格地毯。或用数学函数定义随机高度场，生成表面粗糙纹理即几何纹理。

(3) 在定义了纹理以后，我们还要处理如何对纹理进行映射的问题。对于二维图像纹理，就是如何建立纹理与三维物体之间的对应关系；而对于几何纹理，就是如何扰动法向量。

纹理一般定义在单位正方形区域($0 \leq u \leq 1$, $0 \leq v \leq 1$)之上，称为纹理空间，理论上，定义在此空间上的任何函数可以作为纹理函数，而在实际上，往往采用一些特殊的函数，来模拟生活中常见的纹理。对于纹理空间的定义方法有许多种，下面是常用的几种：

- (1) 用参数曲面的参数域作为纹理空间 (二维)
- (2) 用辅助平面、圆柱、球定义纹理空间(二维)
- (3) 用三维直角坐标作为纹理空间(三维)

9. 6 整体光照明模型

前面介绍的简单光照明模型和局部光照明模型, 虽然可以产生物体的真实感图像, 但它们都只是处理光源直接照射物体表面的光强计算, 不能很好的模拟光的折射、反射等, 也不能用来表示物体间的相互光照明影响; 而基于简单光照明模型的光透射模型, 虽然可以模拟光的折射, 但是这种折射的

计算范围很小，不能很好的模拟多个透明体之间的复杂光照明现象。对于上述的这些问题，就必须要有有一个更精确的光照明模型，即整体光照明模型，它是相对于局部光照明模型而言的。在现有的整体光照明模型中，主要有光线跟踪和辐射度两种方法。

9. 6. 1 光线跟踪算法

光线跟踪算法是真实感图形学中的主要算法之一，该算法具有原理简单、实现方便和能够生成各种逼真的视觉效果等突出的优点。

1. 光线跟踪的基本原理

由光源发出的光到达物体表面后，产生反射和折射，简单光照明模型和光透射模型模拟了这两种现象。在简单光照明模型中，反射被分为理想漫反射和镜面反射光，在简单光透射模型把透射光分为理想漫透射光和规则透射光。由光源发出的光称为直接光，物体对直接光的反射或折射称为直接反射和直接折射，相对的，把物体表面间对光的反射和折射称为间接光，间接反射，间接折射。这些是光线在物体之间的传播方式，是光线跟踪算法的基础。

最基本的光线跟踪算法是跟踪镜面反射和折射。从光源发出的光遇到物体的表面，发生反射和折射，光就改变方向，沿着反射方向和折射方向继续

前进，直到遇到新的物体。但是光源发出光线，经反射与折射，只有很少部分可以进入人的眼睛。因此实际光线跟踪算法的跟踪方向与光传播的方向是相反的，而是视线跟踪。

在光线跟踪算法中，有如下的四种光线：视线是由视点与象素 (x, y) 发出的射线；阴影测试线是物体表面上点与光源的连线；以及反射光线与折射光线。

当光线 V 与物体表面交与点 P 时，光在点 P 对光线 V 方向的贡献分为三部分，把这三部分光强相加，就是该条光线 V 在 P 点处的总的光强：

(1) 由光源产生的直接的光线照射光强，是交点处的局部光强，可以

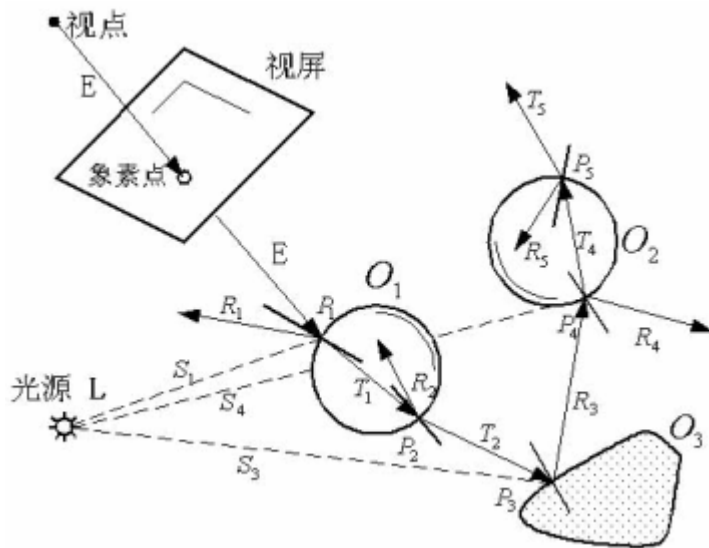
$$I = I_a K_a + \sum_i I_{p,i} [K_{ds} (L_i \cdot N) + K_s (H_{s,i} \cdot N)^{\alpha}] \\ + \sum_j I_{p,j} [K_{dt} (-N \cdot L_j) + K_t (N \cdot H_{t,j})^{\alpha}]$$

由下式计算：

(2) 反射方向上由其他物体引起的间接光照光强，由 $I_s K'_s$ 计算，**Is** 通过对反射光线的递归跟踪得到；

(3) 折射方向上由其他物体引起的间接光照光强，由 $I_t K'_t$ 计算，**It** 通过对折射光线的递归跟踪得到。

如图 9-7 所示, 通过介绍一个由两个透明球和一个非透明物体组成的场景进行光线跟踪的例子, 解释光线跟踪算法的基本过程。



场景中, 一个点光源 **L**, 两个透明的球体 **01** 与 **02**, 一个不透明的物体 **03**。从视点出发经过视屏一个像素点的视线 **E** 传播到达球体 **01**, 与其交点为 **P1**。从 **P1** 向光源 **L** 作一条阴影测试线 **S1**, 其间没有遮挡的物体, 那么用局部光照明模型计算光源对 **P1** 在其视线 **E** 的方向上的光强, 作为该点的局部光强。同时跟踪该点处反射光线 **R1** 和折射光线 **T1**, 它们也对 **P1** 点的光强有贡献。在反射光线 **R1** 方向上, 没有再与其他物体相交, 设该方向的光强为零, 并结束这光线方向的跟踪。然后跟踪 **T1** 方向折射光线, 计算该光线的光强贡献。折射光线 **T1** 在物体 **01** 内部传播, 与 **01** 相交于点 **P2**, 由于

该点在物体内部，假设它的局部光强为零，同时，产生了反射光线 **R2** 和折射光线 **T2**，在反射光线 **R2** 方向，继续递归跟踪计算它的光强，这里不再继续跟踪计算。继续对折射光线 **T2** 进行跟踪。**T2** 与物体 **03** 交于点 **P3**，作 **P3** 与光源 **L** 的阴影测试线 **S3**，没有物体遮挡，计算该处的局部光强，由于该物体是非透明的，可以继续跟踪反射光线 **R3** 方向的光强，结合局部光强，来得到 **P3** 处的光强。反射光线 **R3** 的跟踪与前面的过程类似，算法可以递归的进行下去。重复上面的过程，直到光线满足跟踪终止条件。这样可以得到视屏上的一个像素点的光强，也就是它相应的颜色值。

通过上面光线跟踪算法的基本过程的例子，可以看出，光线跟踪算法实际上是光照明物理过程的近似逆过程，这一过程可以跟踪物体间的镜面反射光线和规则透射，模拟了理想表面的光的传播。

虽然在理想情况下，光线可以在物体之间进行无限的反射和折射，但是在实际的算法进行过程中，我们不可能进行无穷的光线跟踪，因而需要给出一些跟踪的终止条件。在算法应用的意义上，可以有以下几种终止条件：

(1) 该光线未碰到任何物体。

(2) 该光线碰到了背景。

(3) 光线在经过许多次反射和折射以后, 就会产生衰减, 光线对于视点的光强贡献很小(小于某个设定值)。

(4) 光线反射或折射次数即跟踪深度大于一定值。

光线跟踪的方向与光传播的方向相反, 从视点出发, 对于视屏上的每一个象素点, 从视点作一条到该象素点的射线, 调用该算法函数就可以确定这个象素点的颜色。

9.6.2 辐射度方法

辐射度方法是继光线跟踪算法后，真实感图形绘制技术的一个重要进展。尽管光线跟踪算法成功地模拟了景物表面间的镜面反射、规则透射及阴影等整体光照效果，但由于光线跟踪算法的采样特性，和局部光照模型的不完善性，该方法难于模拟景物表面之间的多重漫反射效果，因而不能反映色彩渗透现象。

1984 年，美国 **Cornell** 大学和日本广岛大学的学者分别将热辐射工程中的辐射度方法引入到计算机图形学中，用辐射度方法成功地模拟了理想漫反射表面间的多重漫反射效果。经过十多年的发展，辐射度方法模拟的场景越来越复杂，图形效果越来越真实。与前几章介绍的光照模型与绘制方法有所

不同，辐射度方法基于物理学的能量平衡原理，采用数值求解技术来近似每一个景物表面的辐射度分布。由于场景中，景物表面的辐射度分布与视点选取无关，辐射度方法是一个视点独立(**View independent**)的算法，使之可广泛应用于虚拟环境的漫游(**walkthrough**)系统中。本章的压题彩图就是由辐射度方法绘制出来的。这里限于篇幅和教学的深度，不再详细介绍该方法，有兴趣的读者可以查阅相关的文献。

在我们的“真实感图形学”程序中，我们利用经纬区域划分算法绘制球体，并进行消隐；建立基于颜色缓冲区和深度缓冲区的直线基元和三角形基元，颜色缓冲区的引入将丰富三维物体的绘制，实现物体与物体之间的融合

效果、全局场景抗锯齿、景深模拟等效果。利用 `z_buffer` 的消隐算法实现场景的消隐处理。

9.7 真实感图形学编程案例

一、程序设计功能说明

利用上述基本原理编制的“真实感图形学”应用程序，包括消隐绘制几何球体类、颜色模型类、光照模型类、材质类、基于颜色缓冲区和深度缓冲区的直线基元和三角形基元类、基于 `z-buffer` 算法的场景深度消隐类等。绘制如图 9-8 所示，泛光灯和聚光灯照射下左边为光照线框球体、中间为光照刻球体、最右边为光滑光照球体，。

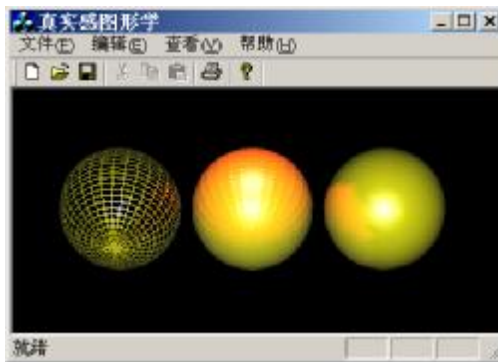


图 9-8

二、程序设计步骤

1. 创建单文档应用程序框架
2. 添加消息处理函数

利用 ClassWizard（建立类向导）为应用程序添加相关的消息处理函数，

ClassName 和 object IDs 栏中选择都选择 CMyView, Messages 栏中选择 WM_CREATE 添加消息映射函数, 接受默认消息处理函数命名。

3. 添加基类

在工程中单击【文件】|【新建】, 在弹出的新建对话框中, 分别选择 C/C++ Header File 和 C++ Source File, 在【文件】名称输入栏中输入依次输入各基类名称 “Bline、Brender、Btriangle、ColorBuffer、Edge、Facet、FloatColor、Lighting、LightObj、Material、Matrix3d、Object3d、Sphere、SubObject3d、TypedStack、Vector3d、Vertex3d、ViewFinder、Zbuffer” 以及 Grphcs、Img 头文件; 在工作区中系统自动创建的相应的空文件中, 分别添

加以下此基类的头文件（.h 文件）和应用文件(.cpp 文件）。

4. 添加相应代码

//CBlind（直线）类

// BLine.h: interface for the CBlind class. //CBlind（直线）类头文件

[程序代码见纸书](#)

}

代码说明:

1. 由于篇幅所限，以上代码只是整个程序中的一部分，详细请参照光盘
“真实感图形学”

2. 只要给定两个端点和对应颜色, 可绘制一条直线。基于颜色深度缓冲器和深度缓冲器的直线基元不再将绘制直线的工作直接与设备描述表和内存位图相链接, 而是通过颜色缓冲区来记录扫描转换后的像素颜色。上面的 CBlind 类和 Ctriangle 类为基于光栅系统的直线和三角形基元。在扫描转换三角形时, 需要单独处理边界和内点。用 Bresenham 算法来扫描转换边界, 并采用使用算法获取扫描点的颜色和深度。对三角形内点, 则通过边界点的颜色和深度利用增量法来进行计算, 这里双线插值法同时应用于颜色和深度的计算。

2. 程序中基于颜色缓冲区和深度缓冲区建立的 CPRender 消隐工具, 具

有以下特征：场景经过光栅化所获得的任何基片都要进行处理，我们来自与规格化坐标系中的 z 值的场景中的深度信息对图形的绘制和显示起决定性作用。利用 z_buffer 算法，场景通过边和三角形的形式经过扫描转换的结果就是最后的图形信息。

3. 三维物体是多种多样的，任何一个三维图形设计系统都需要立方体、球体、圆锥、圆环面等这些最基本的三维形体。

4. 几何拓扑绘制球体，这里采用经纬度的地理划分法将一个球体划分为若干个小区域，这些区域常称为经纬区域，一般两极区域用三角形小面来逼近，其它区域用四边形小面来逼近，地理划分法一般是从北向南递增，北极

点的纬度为 0 度，南极点纬度为 180 度，将球细化来描述求得几何拓扑。

定义顶点：北极点的序号为 0，然后从 Y 轴正向开始，按逆时针防线计算序号递增的顶点，最后的顶点为南极点。

定义边：从北极、Y 轴正方向开始，逆时针方向逐个定义每条经边。

定义小面：从北极、Y 轴正方向开始，逆时针方向逐个定义每条纬度带上的小面，小面顶点排列顺序一小面的发现指向球的外部为基准。以此方法得到球的几何模型，中心位于世界坐标原点。

9.8 课 后 练 习

1. 解释真实感图形学中模拟现实世界场景的基本过程。
2. 在颜色视觉中是如何唯一确定某颜色的三原色混合比例的?
3. 介绍常用的颜色模型。
4. 编写用 **Phong** 光照明模型生成的小球真实感图形的程序。
5. 介绍增量式光照明模型的基本思想。
- 6 编写用双线性光强插值(**Phong**)与双线性法向插值(**Ground**)方法显示一个圆柱体的真实感图形的程序, 并对显示的结果进行比较。

7. 解释局部光照明模型对物体光照现象模拟的合理性与普遍性。

第十章 计算机动画

10.1 计算机动画概述

计算机动画是将图形、图案和画面显示在屏幕上，并按一定的规则或预定的要求在屏幕上移动、变换，从而使计算机显示出动态变化的图形的技术。计算机动画所生成的是一个虚拟的世界，画面中的物体并不需要真正去建造，物体、虚拟摄像机的运动也不会受到什么限制，动画师可以随心

所欲地创造他的虚幻世界。目前，计算机动画已形成一个巨大的产业，随着计算机硬件性能价格比的快速提高，它综合利用计算机科学、艺术、数学、物理学和其它相关学科的知识在计算机上生成绚丽多彩的连续的虚拟现实画面，给人们提供了一个充分展示个人想象力和艺术才能的新天地。与传统媒体中的动态图形载体电影和电视相比，目前众多动画制作软件能使多媒体动画为更多的制作者所掌握，制作者能真正使自己的想法变成看得见的动画。

10.2 计算机动画的应用领域

计算机动画不仅可应用于电影特技、商业广告、电视片头、动画片制作、产品模拟试验、电子游戏，还可应用于计算机辅助教育、军事作战演习、训练模拟，甚至于法院的审理等领域。

10.3 计算机动画的分类和原理

计算机动画是计算机图形学和艺术相结合的产物，它是伴随着计算机硬件和图形算法高速发展起来的一门高新技术。动画是运动中的艺术，运动是动画的要素。计算机动画以其制作方法和表现特征通常可以分为二维动画和三维动画两种形式。

1. 二维动画

传统的卡通动画的实现是连续播放多帧画面，每幅画面表述的是运动物体的若干个瞬间，利用观看者的瞬间视觉残留而得到运动的视觉感受。二

维动画显示的主要是平面图形，制作时就象在纸上作画，通过对象的移动、变形、变色等手法表现其运动的效果；计算机动画原理也是一样，计算机动画的每一帧画面都是一幅数字化的图像。

二维动画不仅具有传统动画功能，还兼有计算机特有功能，例如，计算机生成的图像可拷贝、粘贴、翻转、放大、缩小，任意移位以及自动计算机背景移动等，具有检查方便，保证质量、简化管理、生产效率高、能够有效缩短制作周期。缺点为：在二维动画中，计算机只能起到辅助作用，

并不能取代画家用手工绘制的动画关键帧，并且画面在纸张、照片或计算机屏幕显示，无论画面的立体感多强，终究是二维空间上模拟真实三维空间效果。二维动画中计算机的作用为输入和编辑关键帧，计算和生成中间帧，定义和显示运动路径，交互给画面上色，产生特技效果，实现画面与声音同步，控制运动系列的记录等。

2. 三维动画

三维动画则显示立体图形，其制作就象是在摄影棚中拍电影：首先在三维视图中布置被摄对象的位置、规定其运动、安排好各种灯光，然后在特定位置架设好“摄影机”、也可设定摄影机的推拉摇移，最后计算机计算出在这一立体空间下“摄影机所见的”动态图像效果。尽管在常见的二维动画中也可以模拟三维的立体空间，但其图像的精确度等远不及三维动画。

计算机三维动画数据是在计算机内部生成的，而不是简单的外部输入。制作三维动画首先要创建物体模型，然后让这些物体在空间动起来，

如移动、旋转、变形、变色。再通过打灯光等生成栩栩如生的画面。

三维动画：画中的景物有正面、侧面和反面，调整三维空间的视点，能够看到不同的内容。

二维动画制作方法相对简单，表现的内容也较为简练，能很好地表现动态示意图之类的简单图形，二维动画的模拟三维立体效果在视觉上基本能满足立体的要求；三维动画能较为完美地表现三维立体效果，在对空间感要求较高的动画中，三维动画有其独到的魅力。

按计算机动画实现方法，计算机动画可分为：实时（**Real-Time**）动画、和帧（**Frame**）动画两种，电子游戏机的运动画面属于实时动画；根据运动控制方式将计算机动画分为关键帧动画和算法动画。

一、 关键帧动画

帧动画的基本原理类似幻灯片的制作与播放过程，即把整个动画过程划分为一个个片断，将每一片段作为一幅画像在屏幕上一定区域显示，然后把屏幕上的图像存储在文件中，在动态显示时再按顺序不断读取与播放这

些画面，产生动画效果。熟练的动画师设计卡通片中的关键画面，也即所谓的关键帧，然后由一般的动画师设计中间帧。在三维计算机动画中，中间帧的生成由计算机来完成，插值代替了设计中间帧的动画师。所有影响画面图象的参数都可成为关键帧的参数，如位置、旋转角、纹理的参数等。关键帧技术是计算机动画中最基本并且运用最广泛的方法。从原理上讲，关键帧插值问题可归结为参数插值问题，传统的插值方法都可应用到关键帧方法中。但关键帧插值又与纯数学的插值不同。一个好的关键帧插值方

法必须能够产生逼真的运动效果并能给用户提供方便有效的控制手段。一个特定的运动从空间轨迹来看可能是正确的，但从运动学或动画设计来看可能是错误的或者不合适的。用户必须能够控制运动的运动学特性，即通过调整插值函数来改变运动的速度和加速度。

二、实时动画

实时动画也称为算法动画，它是采用各种算法来实现运动物体的运动控制。在实时动画中，计算机对输入的数据进行快速处理，并在人眼察觉不

到的时间内将结果随时显示出来。实时动画的响应时间与许多因素有关，如计算机的运算速度是慢或快，图形的计算是使用软件或硬件，所描述的景物是复杂或简单，动画图像的尺寸是小或大等等。在实时动画中，一种最简单的运动形式是对象的移动，它是指屏幕上一个局部图像或对象在二维平面上沿着某一固定轨迹作步进运动。运动的对象或物体本身在运动时的大小、形状、色彩等效果是不变的。对象的移动因为相对简单且容易实现，又无需生成动画文件，所以在多媒体应用中经常采用。如果在文字、

图形图像、声音的基础上增加对象的移动，比如跳出文字等，以达到简单动画功能，则能大大丰富视觉效果。但是，对于中间没有停顿的复杂动画效果最好使用二维帧动画预先将数据处理和保存好，然后通过播放软件进行动画播放。这是因为微机，特别是低档微机的处理速度有限，实时处理和显示可能会使处理跟不上显示要求而有损于动画显示效果，甚至影响其它媒体数据如声音的播放。

算法动画是采用算法实现对物体的运动控制或模拟摄像机的运动控制。一般适用于三维情景。算法动画根据不同分为以下几种：

- （1）运动学算法：由运动学方程确定物体的运动轨迹；
- （2）动力学算法：从运动的动因出发，又力学方程确定物体的运动形式；
- （3）反向运动学算法：由已知链接物末端的位置和状态，反求运动方程以确定运动形式。

(4) 反向动力学算法: 由已知链接物末端的位置和状态, 反求动力学方程以确定运动形式。

(5) 随机运动算法: 在某些场合下增加运动控制的随机因素的算法。

用算法控制运动的过程包括: 给定环境描述、环境中的物体造型、运动规律、计算机通过算法生成动画帧。模拟摄影机实际是按照观察系统的变化来控制运动, 从运动学的相对性原理来看是等价的, 但也有其独特的控制方式。

为使计算机显示的动画连续、平稳、美观，就像看电影、电视一样，人们使用各种方法生成图形，这些方法基本上可归为以下几类：

（1）异或运算算法（XOR）

异或运算一个重要特性是具有“还原”作用，对屏幕进行一次异或操作显示一幅图像；再一次异或操作清除前一幅图像，使屏幕“还原”，产生动画效果。

本书第四章中介绍的橡皮条技术绘制直线使用了异或运算算法。

（2）块动画

在屏幕中，改变前景运动的物体再画面中的位置，而背景保持不动既可产生动画效果，这种产生动画方式称为块动画。这种方法是将动画物体的图像保存在储存区中，需要是快速从内存中拷贝到屏幕进行重复显示，并通过对该图像像素与背景像素进行异或算法，可使被前景所遮盖的背景图像部分还原。

（3）帧动画（同上面介绍的概念）

（4）图形变换动画

前面我们已经讲过对图形的二维变换、图形的三维变换，利用这些变换对要运动的图形对象进行上述变换形成动画，并且图形变换后图形的失真率很低。

10.4 目前计算机动画面临的问题

目前计算机动画面临以下问题：真实性和实时性、功能更强、速度更快、效果更好、使用更方便、真实运动生成、物体造型，人体动画、

绘制（渲染）。只考虑了动画算法和局部动画的控制，大型动画片的设计制作局部求精过程。

10.5 计算机动画程序设计案例

10.5.1 帧动画

一、程序设计功能说明

根据以上帧动画原理创建的计算机帧动画程序，演示一般植物生长的基本过程，如图 10-1 所示为程序运行时的主界面及其中的一帧。



图 10-1

二、程序设计步骤

1. 创建名称为“帧动画”单文档应用程序框架（创建单文档详细过程请参见第一章）
2. 编辑菜单资源，添加相应消息处理函数添（如表 10.1 所示）。

表 10.1 菜单资源表

菜单标题	标示符 ID	消息	消息处理函数
3. 播放帧动画	ID_PLAY_FRA ME	CONMM AN	OnIDTRANSLATI ON

用

MFC ClassWizar 对话框，【ClassName】和【Object IDs】中均选择 CMyClass，建立 WM_TIMER 消息处理函数 OnTimer(UINT

nIDEvent)

4. 添加帧动画

单击主菜单中的【插入】—>【资源...】，打开【插入资源】对话框，选择【Bitmap】单击【确定】按钮，利用绘图工具，在当前应用程序中，绘制 Bitmap1 如图 10-5-2 所示。以同样方法添加 Bitmap2、Bitmap3、Bitmap4、Bitmap5 帧图片资源。

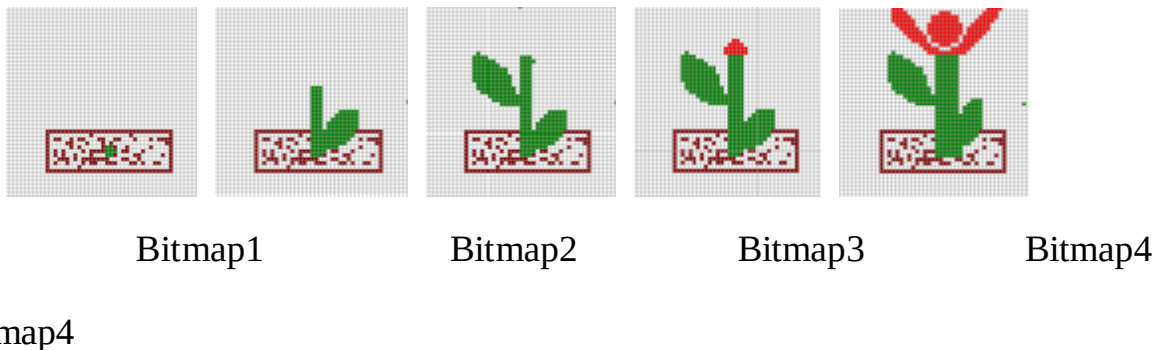


图 10-2

5. 在“帧动画 View.cpp”、“帧动画 View.cpp”、帧动画相应函数
添加如下代码

说明：下面仅列出帧动画 View.h、帧动画 View.cpp 的部分代码，黑体部分为手工输入其它文件代码全部为系统维自动建立，无需改正，为节省篇幅此处省略，详见光盘。

```
// 帧动画 View.h : interface of the CMyView class
```

```
.
```

```
.
```

.

程序代码见纸书

}

10.4.2 实时动画

三、程序设计功能说明

根据以上实时动画原理创建的计算机实时动画程序，演示一弹性小球在

正方体容器中的碰撞过程，如图 10-3 所示为程序运行时的主界面的一个瞬间。

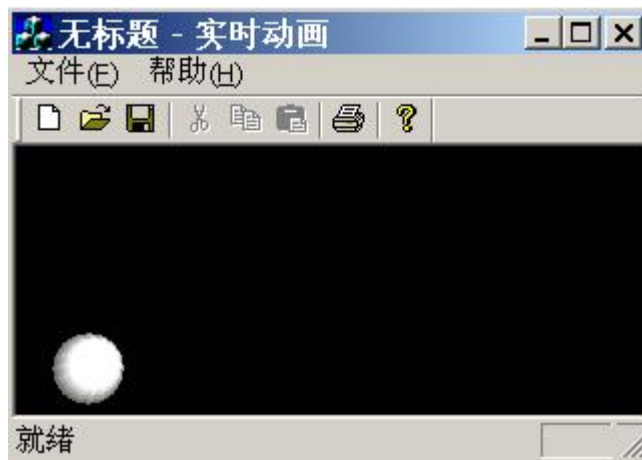


图 10-3

四、程序设计步骤

1. 创建名称为“实时动画”单文档应用程序框架（创建单文档详细过程请参见第一章）
2. 利用 MFC ClassWizar 对话框，【ClassName】和【Object IDs】中均选择 CMyClass，建立 WM_CREATE、WM_DESTROY、WM_TIMER 消息处理函数分别为 OnCreate()、OnDestroy()、

OnTimer()

3. 手工添加绘制球的基类

在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 C/C++ Header File，在【文件】名称输入栏中输入“Sphere”；同样，在工程中单击【文件】|【新建】，在弹出的新建对话框中，选择 C++ Source File，在【文件】名称输入栏中输入“Sphere”。在工作区中系统自动创建的相应的空文件中，分别添加以下此基类的头文件（.h 文件）和应用文件(.cpp 文件)。

// Sphere.h: interface for the CVirtualSphere class.

#ifndef _CVSPHERE_H

#define _CVSPHERE_H

class CSphere

{

[程序代码见纸书](#)

```
}
```

```
}
```

4. 在“实时动画 View.h”、“实时动画 View.cpp”相应函数添加如下代码

说明：下面仅列出实时动画 View.h、实时动画 View.cpp 的全部代码，其它文件代码全部为系统自动建立，无需改正，为节省篇幅此处省略，

详见光盘。下面代码中黑体部分为手工输入，其它代码为系统生成。

```
// 实时动画 View.h : interface of the CMyView class
```

```
.
```

```
.
```

```
.
```

```
class CMyView : public CView
```

```
{
```

程序代码见纸书

}

10.6 练习题

1. 传统动画和计算机动画的区别？
2. 计算机动画研究的内容？目前主要应用领域？
3. 关键帧动画与算法动画？

4. 利用分别应用帧动画和实时动画原理，在 Visual C++中编程绘制在高速公路上奔跑的小汽车。