KH

游戏设计概论

荣钦科技　编著

北京科海电子出版社

内 容 简 介

本书以设计游戏的基础知识为前提，引导读者快速进入游戏设计领域。

通过介绍游戏主机的发展历程，及各种游戏领域里的专有名词，让读者对游戏设计有明确的认识。详细介绍各种程序设计工具，以及DirectX 和 OpenGL 开发函数库，并比较分析了各工具开发游戏的优缺点。通过学习本书让读者建立起良好的游戏基本规则，巧妙地安排故事剧情，华丽效果设计的要诀，理念与现实的互补。以简单的实例带领读者设计游戏的乐趣，让读者在最短时间内，设计出有个性的游戏。

本书理论与实践相结合，适合于软件工程师，游戏爱好者，以及从事游戏设计策划人员。

盘　　名：游戏设计概论
作　　者：荣钦科技
责任编辑：徐建军
排　　版：房利萍
光盘制作：
咨询电话：（010）82896445-8407

出　　品：北京科海电子出版社
印 刷 者：北京科普瑞印刷有限责任公司
发 行 者：新华书店总店北京发行所
开　　本：异 16　　印张：　　字数：　　千字
版　　次：2003 年 7 月第 1 版　　2003 年 7 月第 1 次印刷
印　　数：0001～5000
盘　　号：ISBN 7-900107-00-0
定　　价：00.00元（1CD / 配套手册）

前　言

以近几年游戏发展的速度与规模来看，不论是基于工作上的需要或是出自于本身的兴趣，投入游戏设计领域的人明显地越来越多了。不过如果想要让懵懂无知的初学者变成一个功力非凡的游戏设计高手，其间所要付出的努力及承受的挫折绝对是一般人无法想象的。尤其一般的学生在刚踏入这个领域时，学习的方向往往千头万绪，难为确立，更何况学校中所教授游戏制作相关的知识也十分有限，如何一开始就建立良好的基础与确定未来发展方向，对于一个有心从事游戏设计的新人来说，是非常重要的一项挑战。

目前市面上有许多游戏设计相关的书籍，有些内容的确难懂，仅适合已具有游戏设计经验的老手；有些则是直接由原文翻译过来，内容虽然十分深入专业，不过却让刚入门的读者看得眼花缭乱、一知半解。所以如果想要在众多游戏书籍中，找到一本浅显易懂的入门书，对于一位初学游戏设计的菜鸟而言，绝对是成功的开始。

因此，笔者决心编写一本针对所有欲求游戏设计知识的初学者必读的书籍。在本书中，初学者可以学到游戏设计必需的入门知识，它可以当成初学者跨入进阶游戏设计领域的门坎书，以简单的范例来介绍游戏设计的整体架构，让初学者轻轻松松地学到游戏设计的技巧与知识。在本书中，除了以深入浅出的叙述来介绍一些游戏制作的条件及流程，并且辅以精心设计的程序范例来引道读者在潜移默化中进入游戏细节。我们真心期望大家可以轻松有效地学习游戏设计的基本技术及理论，为我们的游戏产业开创美好的未来。

在本书配套光盘中收录了 3D RPG 游戏中几个重要的编辑器，各编辑器的使用方法请参见附录，希望读者通过对编辑器的实际操作，了解研发团队游戏设计的步骤，引导初学者进入游戏领导。

由于时间仓促，作者水平有限，书中难免有不妥和错误之处，恳请读者批评指正。

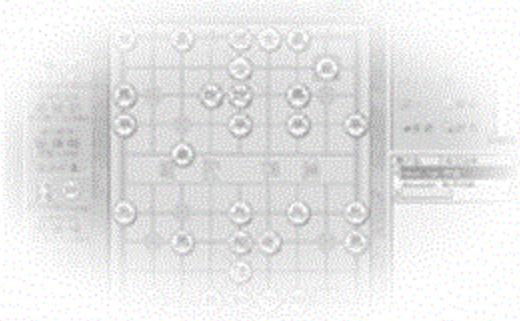
荣钦科技

2003 年 7 月

目 录

第 1 章

游戏设计概论



本章将探讨从骨灰级游戏到现今 3D 游戏的发展与游戏平台延伸的开发过程，以及在制作游戏的同时，我们应该具备哪一些基本概念与基础知识。其实在制作游戏的过程中，有 4 个不可或缺的重要角色：分别是策划、编写程序、美术设计与音乐。而本章也将深入探讨在制作游戏时，我们应该如何来扮演好这 4 个重要的角色的方法。

1.1 游戏的发展历程

从小当我们开始对周围环境感到好奇的时候，“游戏”这个名词就一直存在于我们的生活当中。例如，像小时候我们都玩过的猜拳游戏；长大之后，又接触了许多不同类型的游戏，如球类运动或者益智活动等等。诸如此类，我们都可以把它们称为是游戏的一部分，所以游戏在现代人的成长过程中，绝对是一个不可或缺的角色。

1.1.1 什么是游戏

可以说游戏是一种娱乐我们休闲生活的简单名词。也许您会问：“那什么是游戏呢？”其实简单地说，“游戏”就是本身具有特定行为模式、规则条件、身心娱乐及输赢胜负的一种行为表现。

从上面的解释来看，或许读者对于“游戏”这个名词还是相当的陌生。笔者将从上面定义“游戏”的解释来加以分类，下面将作详细介绍：

行为模式

通常解释“游戏”最简单的因素，就是游戏会有特定的流程模式，这种流程模式是用来贯串整个游戏的行为，我们必须按照它的流程模式来执行。倘若一种游戏没有特定的行为模式，那么我们就会没有执行的行为，在没有执行行为之后，这个游戏就玩不下去了。例如，如果猜拳游戏没有了剪刀、石头、布等行为模式，那么这还叫做“猜拳游戏”吗？所以不管游戏的流程有多复杂？或者多么简单，它一定具备特定的行为模式。

规则条件

当游戏有了一定的行为模式之后，它就必须定出一系列的条件规则。简单来说，这些游戏的条件规则就是大家必须去遵守的游戏行为，只要是大家所一致认同的游戏行为，游戏中的玩家就必须去遵守它。如果我们不能遵守这种游戏行为的话，那么它就失去了公平性。就拿一种简单的球赛来说，球赛的英文解释我们可以用“PLAY GAME”来加以说明：

按照英文字面上的解释，它就是在执行游戏的行为。而球赛必须要有一定的条件规则，并且游戏的参与者都必须去遵守它；如果不能遵守，我们就称为“犯规”。所以不管是什么游戏，它都会具备一定的规则条件，在游戏进行的时候才会有足够的公平性。

身心娱乐

一种游戏所带来的娱乐性，关键就是在于它为玩家所带来的新鲜刺激感，这也是游戏最主要的精华所在。简单地说，不管是很多人玩的在线游戏，还是一个人玩的单机游戏，游戏本身就会存在它的娱乐刺激性，使得玩家们会不断地想要去玩它。

输赢胜负

其实针对游戏的定义来说，输赢胜负是所有游戏的最终目的。一个没有输赢胜负的游戏，仿佛少了它存在的意义。如同我们常常会接触到的猜拳游戏，说穿了最终目的还不是要分出胜负。

游戏又可区分为动态和静态两种形态，动态的游戏必须配合肢体动作，如猜拳游戏；而静态游戏则是较偏向思考的行为，如同纸上游戏。然而不管是动态或是静态的游戏，只要它们不违反上述四项游戏的定义规则，我们都可以将它称为“游戏”。

在读者了解到游戏的定义之后，我们再来谈谈游戏平台与种类。

1.1.2 游戏平台

通常我们认为 “平台”是一种传递的媒介，而“游戏平台”就是让我们可以与游戏沟通的一种媒介。“游戏平台”又可分为许多不同的类型，例如纸上游戏“大富翁”就是与玩家的一种沟通媒介；电视游乐器与计算机也称得上是一种游戏平台，又可称为“电子游戏平台”。由此可知，游戏平台不仅可以执行游戏流程，而且它也是一种与游戏玩家们沟通的管道。接着，我们将从影响游戏发展较为深入的“电子游戏平台”开始谈起。

首先我们以简单的表格来描述电子游戏平台的发展过程，如表 1.1 所示。

表1.1 电子游戏平台的发展过程

年份	处理位	代表游戏机
1983	8 位	红白机任天堂 FC（Family Computer）
1986	16 位	世嘉（SEGA）
1991	16 位	超级任天堂（SFC）
1994.11	32 位	SEGASaturn（SS）
1994.12	32 位	PlayStation（PS）

（续表）

年份	处理位	代表游戏机
1996.6	64 位	任天堂 N64
1998	128 位	DreamCast（DC）
2000.3	128 位	Sony PlayStation 2（PS2）
2001.9	128 位	任天堂 Game Cube
2001.11	128 位	微软 X－BOX

从表 1.1 来看，我们不难发现在各个不同的时期，电子游戏平台是随着硬件技术不断地向上提升。十几年的时间，从最早只能支持单纯的 16 色游戏发展到现在的 3D 高彩游戏。不管是处理器的速度或显像技术的提升，我们看到了游戏平台的品质不断地在进步。

1.1.3 游戏发展

我们从上一节了解到电子游戏平台的变化在快速地发展，游戏的色彩与真实度也越来越高。就以早期的游戏来说，它的玩法技巧较为简单，通常玩家不太需要花费太多的心思与时间就能破关，而游戏模式也较为单纯。这对于胃口逐渐大开的玩家来说，其内容也逐渐不能满足所有玩家们的需求了。

由于电子游戏平台的快速发展，游戏主机可以支持的处理速度也越来越快了，而最直接影响到的就是游戏功能的发展。以前我们只能玩着非常单纯而且变化性不大的游戏内容，到现在玩家们可得花费相当多的心思与时间来玩游戏。绚烂华丽的画面与离奇曲折的剧情，已经成为现代电子游戏不得不必备的呈现内容。

目前电子游戏平台也从电视游戏主机慢慢地走进计算机的世界。以前计算机的最主要功能是用来处理公司资料，而且只有在特定的公司里才会有计算机；但是随着科技的快速成长，现在几乎已经成为家家户户的消费家电用品。此外现代的计算机也可以用来处理相当多的事情；例如数据处理、影像处理、科学运算等等。相信随着电子游戏慢慢地在计算机上的发展，计算机势必成为电子游戏一种不可或缺的游戏平台。

因为游戏平台的快速发展，电子游戏的品质与玩法也越来越华丽与丰富；许多玩家也慢慢跳离只能玩别人设计的游戏的遗憾，而兴致勃勃地投入游戏设计的专业领域中。但是真正进入游戏设计的门坎到底有多困难呢？笔者认为这要因人而异，只要是肯努力学习，相信每个人都有机会成为真正成功的游戏设计大师。

在下一节中，我们将要探讨成为一个入门的游戏设计者，首先必须具备的基本知识与技术。

1.2 如何进入游戏设计的领域

或许您会问：“想要进入游戏设计领域的玩家，是不是要真的才高八斗，具备相当多的知识呢？”以笔者接触游戏设计多年来的经验看，其实不然。要进入游戏设计领域之前，我们需要去学习基本游戏设计的常识，而且在刚踏入这个门坎的过程也确实相当的辛苦，但只要我们有恒心与毅力，相信任何困难都可以迎刃而解、水到渠成的。

在本节中，我们开始学习设计一套游戏所必备的基本原则。

1.2.1 游戏的主题

游戏的主题是设计游戏的开端，今天我们要设计一套游戏就必须先将主题明确地突显出来，这样玩家才不会搞不清楚所设计出来的游戏到底要表现什么。笔者将游戏设计的主题归纳成一些主要的基本要素，作为表示整个游戏主题发挥的重要因素。

时代

“时代”的目的是用来描述整个游戏的时间与空间，代表的是游戏中主角人物所存在的时间与地点。以单纯的时间特性来说，可以包含游戏中人物的服饰、建筑物的构造以及合理的环境对象。所以设置明确的时间才不会让玩家觉得整个游戏的过程中会发生一些不合常理的人、事、物；而空间特性指的是游戏故事的存在定义，例如地上、海边、山上或者是太空。目的就是要让玩家可以很清楚地了解到游戏中存在的方位，所以“时代”的要素主要是描述游戏中主角存活的逻辑意义。

背景

在我们定义出游戏的时代后，接下来就必须去定义游戏中所发生的背景。根据定义的时间与空间，再设计出一连串的合理背景。如果在游戏中常常出现一些不合理的背景；例如将时代定义在远古时代，可是背景却出现了现代的高楼大厦或汽车。除非具有合理的解释，要不然玩家们就会被游戏中的背景搞得晕头转向，不知所措。

人物

通常玩家最直接接触到游戏的部分就是他们所操作的人物与故事中的其他角色，因此在游戏中就必须刻划出故事的正派与反派角色。而且最好每一个设计的人物都拥有自己的个性与特征。如此一来，游戏才能淋漓尽致地突显人物的特色，也让玩家在操作主角人物

时，更能深入其境、浑然忘我。

目的

不管是哪一种类型的游戏，它们都会有独特的玩法与目的，而且游戏中的目的可能不只一种。如同有些玩家为了让自己所操作的人物达到更强的程度，他们就会更加拼命地提升自己主角的等级；有些玩家也会为了故事剧情的发展而去拼命地打击敌人过关，或者为了得到某一种特定的宝物而去收集更多的元素等等。诸如此类，“目的”就是让玩家有了肯继续玩下去的理由，没有了游戏目的，相信玩家可能不到 10 分钟就会觉得索然无味，玩不下去了。

如同上述所谈论的这四项基本原则，一套成功的游戏应该就要从这四项原则中找到更理想、精彩的题材来吸引玩家的目光。在了解到游戏的四项原则后，接下来就请大家跟随着笔者的脚步，一步步地开始游戏设计的学习之旅吧！

1.2.2 游戏设计实例

在您明白了游戏设计主题的基本要素之后，现在我们就来尝试一段简单的游戏主题实际制作过程。从“时代”的要素来说，我们设计了一个未来的时空：在未来时空、战后混乱的城市里，人类太过于依赖计算机，以致让计算机控制了整个世界。从上述简短的描述来看，它就符合了游戏中“时代”与“背景”要素的大纲了。

有了“时代”与“背景”的要素之后，我们可以开始编写出游戏故事的剧情内容。比如说为了打败计算机，人类决定在这个星球的各个角落里挑选出几个英勇的战士，而主角就从这几个战士中产生。主角为了打败计算机，在冒险的路途中开始集结各个地区的骁勇战士，而且在它们之间还会触发一些爱恨情仇的小插曲。上述的故事内容就可以当作是整个游戏的“故事”要素，至于要怎么去设置详细的故事内容呢？在后面的章节还会再为您仔细道来。

有了前三项的要素之后，接下来就可以开始初步地设置出基本的演出角色，例如男主角、女主角、反派角色等等。在这里先来设置男主角的人事背景：男主角是个孤儿，年约 20 多岁，出生在星球上的某个国家，在一次勇士挑选竞赛中脱颖而出。这个帝国的国王告诉男主角人类的遭遇后，男主角决定一肩背负起这个艰巨的任务。

其实主角的设计是要依据最初设计者的想法，不过在“设计”与“设置”的过程中，通常不会由同一个人来执行。所以就必须将“设置者”的想法，以文字表格的叙述方式告诉“设计者”，其目的是为了使设置者的想法与设计者所设计出来的人物差距不要离题太远。笔者在这里拟定了一份初步的人物设置表格供读者参考，如表 1.2 所示。

表1.2 初步的人物设置表格

特征	名称设置
姓名	卡多
年龄	23
身高	181 公分
体重	65 公斤
个性	火爆、少一根筋
衣着	原住民勇士的服饰
人物背景	农村长大，是个原住民勇士，体形高大壮硕

最后我们必须设置几项基本的目的，如打倒怪物之后，主角可以得到某一特定的经验值，而经验值达到一定的程度之后，主角便可以提升某方面的特性与能力，或是为了拿取某把很厉害的武器，主角就必须去收集一些制造武器的元素与工具等等。这些基本的目的都是为了提高游戏的耐玩度和刺激性，而游戏最终的结果还是为了让玩家们可以得到破关之后的满足感与成就感。

说到这里，我们已经将游戏主题中的所有元素都大略地设置出来了，接下来，我们就要进行游戏中主要玩法与系统的设置。

1.2.3 游戏系统基本的设置

游戏系统是定义游戏的基本玩法类型，例如角色扮演、动作、策略、益智等等。简单地说，我们必须定义游戏中几项基本的要素，而这些基本要素就必须从“给谁玩”（WHO）、“玩什么”（WHAT）、“如何玩”（HOW）三项考虑。

给谁玩

“给谁玩”是定义游戏最根本的考虑要素。在游戏设置的初期，我们必须要去了解这套游戏是要给哪一些玩家玩，而这些玩家又比较喜欢玩哪一种类型的游戏（第 7 章中将会谈到）。

玩什么

为了让玩家对游戏产生兴趣，我们就必须让玩家了解游戏到底在玩些什么。是打斗带来的刺激呢？还是解谜所带来的快感？所以我们在设置游戏系统的时候，就必须考虑到这个因素。

如何玩

在设置完游戏的“给谁玩”和“玩什么”要素之后，接下来就要让玩家知道游戏到底要怎么玩。简单地说，“如何玩”就是要告诉玩家要怎么样才能让游戏可以顺利地进行下去。如果将“如何玩”设置的含糊不清或过于复杂，玩家很容易抓不着游戏方向，进而降低玩这款游戏的兴趣。

综合上述要点，我们就来设计一个小型的游戏系统，以便让您有更深刻的认识。例如一开始，我们先将游戏定义在一般较为普通的玩家；以飞弹混乱射击的方式表现出游戏的刺激感，而取得更高的分数与飞机操控的流畅感为主轴。我们再定义出操控一台小飞机在游戏画面中可以四处地飞行，当小飞机遇上敌机的时候，小飞机可以将敌机打落；而被打落的某些敌机当中，它们也会掉落下加强小飞机功能的物品。

从以上的游戏系统来看，已经很轻松地定义出一套类似“雷电”的游戏系统了。因此在设置游戏系统时，基本的论点就不外乎如同上面所提到的三项原则。

完成了游戏的基本系统定义之后，接下来是准备开始定义游戏的基本流程控制。

1.2.4 游戏的流程控制

在定义完游戏主题与游戏系统之后，我们可以尝试画出一个基本的游戏流程图。这个游戏流程图是用来控制整个游戏的运作过程；而流程运作的方式可以从两个基本方向来考虑：那就是游戏要“如何开始”与“如何结束”。笔者就以一个简单地叙述游戏流程图来为大家说明，如图 1.1 所示。

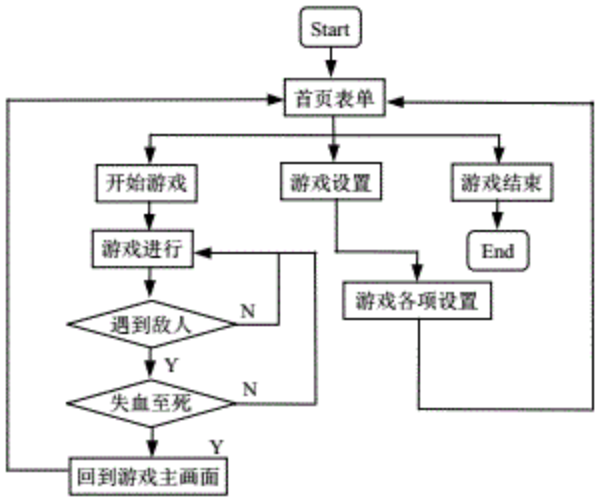


图 1.1　游戏流程图

从图 1.1 可以看出，首先可以看到游戏的首页窗体，玩家可由首页窗体中进入游戏，而在游戏中可能会得到宝物或者是遇到魔王，游戏进行中也可能被敌人打死，最后则是结束游戏。这个流程图的目的是要让设计者更能够掌握整个游戏流程，并且让设计者以外的人了解到游戏的流程运作。在游戏流程图中，我们可以轻易发现游戏如何而生、如何而死，一套有系统的流程观念图更可以呈现出游戏的架构是否好玩与合理。

总而言之，游戏设计是一门很复杂的学科。设计者在凭空想象中，他要将设计思想如何清晰地表达给所有参与游戏设计的人知道，这就成了一个很关键的问题。而且他还要能够控制玩家的心理，让玩家更能接受且了解到游戏的目的与娱乐性，这便是成功设计一套游戏的主要目的了。

1.3 设计游戏的四大要素

在设计一套游戏时，有 4 个极为重要的角色，就是“策划”、“程序”、“美术”、“音乐”，图 1.2 就是彼此关系的简单说明。

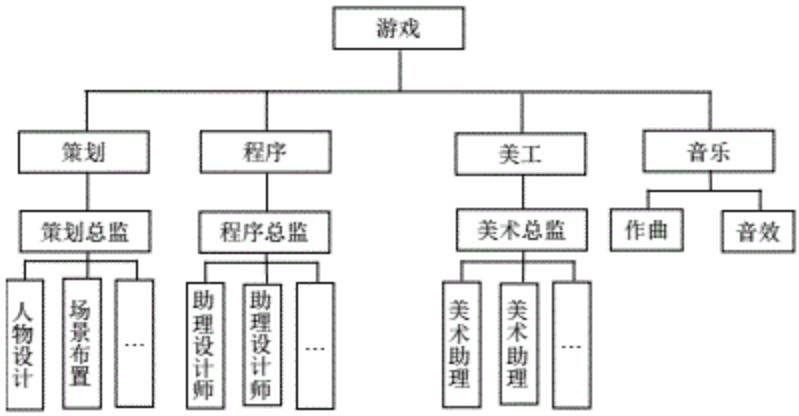


图 1.2　彼此关系的简单说明

接着我们将从游戏品质的角度，来分别说明它们在游戏中的重要性。

1.3.1 游戏的灵魂——策划

“策划”——笔者将它定义成游戏中的灵魂，它的任务就是其他三个角色的核心领导，并控制了整个游戏的规划、流程与系统。策划人员必须编写出一系列的策划书供其他游戏参与人员阅读。

“策划书”是由策划人员将脑海中的想法以文字方式具体落实，目的是让其他人员能

够了解到策划人员对这套游戏真正的理念与意图。

通常策划人员所要做的工作，我们可以将它归为下列几点：

- 游戏规划：游戏制作前的资料收集与环境规划。
- 架构设计：设计游戏的主要架构、系统与主题定义。
- 流程控制：绘制游戏流程与控制进度规划。
- 脚本制作：编写故事脚本。
- 人物设置：设置人物属性及特性。
- 剧情导入：把故事剧情导入引擎中。
- 场景分配：场景规划与分配。

1.3.2 游戏的骨架——程序

程序是用来升华游戏灵魂的一种技术性工具。在策划人员凭空想象的策划书中，我们必须利用程序来加以组合成形。程序设计人员必须了解策划人员的构想计划，根据他们的想法与理念，将设计转化成一种成像的画面或功能。程序设计人员也要具备拆解策划书的能力，将分解出来的游戏功能分配给其他人去编写。而且在其他人将程序编写完毕之后，再将它们整合为一；达到策划人员所要求的画面或功能。

程序设计人员所要做的工作，我们可以将它分成下列几点：

- 编写游戏功能:编写策划书上的各类游戏功能，包括编写各种编辑器工具。
- 游戏引擎制作:制作游戏核心，而核心程序足以应付游戏中发生的所有事 件及图形管理。
- 合并程序代码:将分散编写的程序代码加以结合。
- 程序代码除错：在游戏的制作后期，程序人员可以开始处理错误程序代码，及重复进行侦错的动作。

1.3.3 游戏的皮肤——美术

对于玩家们来说，最直接接触他们的就是游戏中的画面， 在玩家尚未真正操作游戏的时候，他们可能会先被游戏中的绚丽画面所吸引，而动心去玩这款游戏，因此优秀的美术人员是非常重要的。

美术人员所要做的工作，我们可以将它归纳为下列几点：

人物设计

不管是 2D、还是 3D 的游戏，美术人员必须根据策划人员所规划的设置，设计与绘制

游戏中所有需要的登场人物。

场景绘制

在 2D 的游戏中，美术人员必须一张张地刻出游戏所需要的场景图案，而在 3D 游戏中，美术人员就必须绘制出场景中所有必须要使用到的场景对象，以提供地图编辑人员编辑所用。

界面绘制

除了游戏场景与人物之外，还有一种经常在游戏中所看见的画面，那就是使用接口。这种接口就是让玩家可以与游戏引擎做直接沟通的画面。美术人员要依据亲合性与方便性来作为设计使用者接口的原则。

动画制作

游戏中少不了会有几个串场的动画，美术人员会根据策划书的需求而制作出声光十足的动画。

1.3.4 游戏的外衣——音乐

通常在一套游戏中，假如少了音乐的衬托，它的娱乐性可能会大大减半了。譬如玩家在刺杀一个敌人的时候，只能看到在画面中游戏人物一个在砍人，另一个被砍，如此一来，游戏的刺激性便会减少许多。这时如果我们加上了音乐效果的话，玩家便能身历其境般的感受到那份快感。

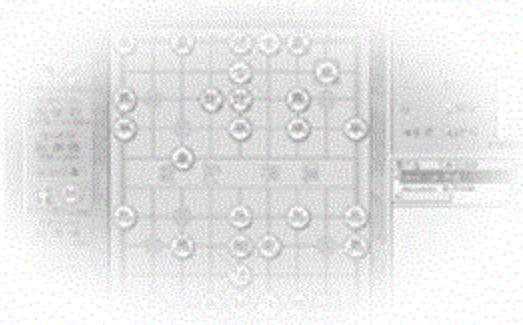
音乐制作人员的工作就比较单纯，他们只要做出游戏中所需要用到的音效与相关的背景音乐即可。不过他们还是必须了解游戏故事的整个剧情发展；哪一段应该是悲伤的情景，就不能播放快乐的音乐，因为这样玩家们会认为文不对题，牛头不对马嘴，反而产生对游戏的不认同与疏离感。

游戏设计四大要素的角色可以简单诠释如下：“音乐”能够震憾玩家们的听觉，“美术”能够吸引玩家们的目光；“程序”能够牵动玩家们的手足；“策划”能够虏获玩家们的内心。对于设计一套游戏来说，这四个要素都必须相辅相成，缺一不可。

通常执行这 4 个要素的工作分别由不同的人来扮演。不过有时候基于成本考虑，也可能一个人身兼一个以上的工作；但是不管如何，身为一个专业的游戏设计人员，如何充分发挥与整合这四项要素，的确是成功完成一套游戏制作的技巧。

第 2 章

游戏设计的专有名词



在学习游戏设计的过程中，有些初学者对于游戏制作的专有名词并不熟悉，以至于妨碍了整个学习进度；本章将为您收录游戏设计时会遇到的一些常用专有名词。

2.1 计算机硬件专有名词

对于计算机内部琳琅满目的硬件设备而言，真的是会让我们晕头转向。而在科技发达的今天，计算机硬件设备的发展更是日新月异，如果我们稍不留意的话，这些新潮的配置倒真的会令我们丈二和尚摸不着头脑。基于这个原因，我们将在本节里为您介绍各式各样目前颇为流行的计算机硬件相关名词。

2.1.1 主机类

当我们看到计算机主机上的电路板时，上面却有许多我们不认识的电子元件。这些新的电子元件可将计算机带入高速传输运算的时代。对于游戏软件而言，它们不仅能够加速游戏画面的运算处理能力，而且还可以帮助我们踏入到网络多媒体的时代，现在我们就来好好地了解这些计算机内部组件的功能。

主板

主板（Mother board 或 Main board），简称 MB，是作为计算机内部各个芯片及组件连接的主要信道。一般而言，主板上会有“中央处理器”（CPU）、“内存”（RAM）、“芯片组”（Chipset）、“其他控制组件”、“接口扩充槽”、“输入/出连接头”（I/O）等等。主板是计算机系统中最主要的一块电路板，通常我们在拆开计算机外壳之后，直接看到的一块电路板，它就是主板。

Active Component

Active Component，中译就是“主动组件”。主动组件是一种可以用来进行数据计算与处理的芯片。而主动组件也是主板上最主要的元件之一。

Passive Component

Passive Component，中译就是“被动组件”。通常被动组件并未对信号进行运算处理，它的作用只是单纯让电流通过，或者是将信号放大而已；例如主板上的电阻或电容，就属于被动组件。

芯片组

芯片组指的是上各个主板重要电路的控制芯片。目前在主板上的芯片组分为两种，一种叫做南桥芯片，主要是负责控制主板的输入/出信号；另一种称为北桥芯片，则负责控制内存的运行等功能。

BIOS

BIOS 即为 Basic Input Output System 的缩写，是常驻在 ROM（Read Only Memory，即为只读存储器）中的一种程序。它不会因为电源的关闭而消失，作用是让使用者可以设置计算机的各种外围装置。

CMOS

CMOS 是 Complementary Metal-Oxide Semiconductor 的缩写，中文称为“互补金属氧化物半导体”。在个人计算机系统中，CMOS 通常扮演着记录硬件相关信息的角色。由于 CMOS 的特殊设计，它所消耗的电力相当少；所以当计算机在关机后，CMOS 仍可以利用主板上充电电池的电力供电，让内存的数据保存一段时间；这就是为什么计算机在关闭之后，其系统时间仍然可以照样运作的缘故。在 CMOS 中，它记录的信息包含一般软式磁盘驱动器的个数和容量、硬盘类型、硬盘的磁道数、磁道的扇区数、扇区数的磁面数、系统日期、显示器类型、内存数量等信息；通常这些设置值也就是我们在 BIOS 中设置修改的存储值。简单地说，CMOS 就是 BIOS 的存储媒介。

DMI

DMI 即是 Desktop Management Interface 的缩写，是一种用系统存储与外围设备等相关信息的接口，而且所有的数据都存储在 MIFD（Management Information Format Database）中。主板上的 BIOS 会自动将 CPU、内存以及扩充槽等数据存储在 BIOS 中的特定位置上，并且利用主板所提供的应用程序（DMI Utility）进行数据库的管理。

System Health Monitor

System Health Monitor 是一种系统状态监控的装置，主要是由一颗晶体管和热感应器所组成，目的是用来监控系统 CPU 的温度、电压及风扇转速是否正常。我们通常可以观察一块主板上的 CPU 装置附近是否装有热感应器来判断主板有没有提供此项功能。

LAN On Motherboard

LAN On Motherboard 即为“主板内建网络芯片”的意思，简称“Lan On MB”。Lan On MB 指的是在主板内部已经装有网卡芯片，使用者不需要再另外购买网卡。随着因特网的普及化，现在的主板都尽量把一些重要芯片组件内建其上，这不仅能够有效降低使用者再采购其他适配卡的成本，并且还能够提升主板的功能。

Bus

Bus 就是总线，它是一种存在于主板上，用来汇集所有数据，并且可将这些数据传输到一个中继站。一般说，系统会通过 Bus 总线安装各种硬件适配卡，并且传输各种数据到适配卡所连接的设备。

USB 2.0

USB 是 Universal Serial Bus 的缩写，中文意思为“万用序列总线”。USB 接口可以算是目前个人计算机中最为广泛使用在外围设备上的传输接口。它可以串接许多的外围设备；特别是不需要设置外围设备上的“I/O 地址”、“IRQ 中断”及“DMA”数据。因为它具有即插即用的特性，所以也不需要经过复杂的安装设置。

USB 最早是在 1995 年问世。经过多年来的改良和扩充，于 1998 年正式推出 USB 1.1 版本，这也就是大家目前耳熟能详的 USB 接口。至于新版本的 USB 2.0 则是发表于 2000 年，它是利用 USB 1.1 版的规格延伸而来，并且 USB 2.0 在英特尔（Intel）的大力推广下，而有逐渐普及趋势。再加上微软所发布的 Windows XP 操作系统，也支持 USB 2.0 的总线接口，因此 USB 未来发展绝对是炙手可热。

ATX

ATX 就是 AT eXpanding 的缩写，它是由 Intel 所制订的主板规格。ATX 可以提供较好的通风性能，使得机壳内部的温度有效降低，同时也提供了软件控制电源的功能。ATX 另一项主要目的在于它可以让硬件及外围的使用更为方便；对于多媒体、通讯及未来的输入/出设备做到最好的支持。

Micro ATX

Micro ATX 规格主板的体积比 ATX 更小，其搭配的机壳也比传统的 ATX 小；Micro ATX 主板提供两个 DIMM 插槽与 3 个可用扩充槽。而 Micro ATX 主板必须与 ATX 的结构兼容，它所使用的电源供应器大约是 90 瓦左右。电源供应器输出电压与 ATX 相同，其电压分别为 12V、5V、3.3V，另外拥有 5V 的备用电压。

ACPI

ACPI 即为 Advanced Configuration and Power Interface 的缩写，中文为“高组态能源接口”。ACPI 是计算机管理电源的设备，主要由英特尔、微软与东芝（Toshiba）三家厂商于 1997 年共同对外宣布此规格接口。ACPI 的主要功能是将电源供应器的开关按钮不接在电源供应器上，而是连接到主板，因此电源供应器无法直接启动计算机的电源，而让主板控制主要的电源。Intel 自 ATX 规格以后，它的所有芯片组就支持 ACPI 的规格了，因此现在的计算机已经没有直接由电源供应器打开电源的主板了。另外，如果再配合硬件设备的话，ACPI 也可以用来侦测主板上的温度、风扇转速及电源供应器的电压等信息。

介绍完主板上的各类专有名词后，接下来，再来看看计算机显示卡相关的专有名词。

2.1.2 显示卡

显示卡（Display Card）的主要功能是用来控制计算机的图形影像输出，凭借显示卡连接显示器，才能够在显示屏幕上看到图形及影像的画面。“显示内存”与“RAMDAC”这些组件会直接影响到显示屏幕上的输出，包括屏幕画面显示的速度、颜色以及分辨率等等。一般而言，绘图模式之下的颜色值为 16 色或 256 色，而 256 色的显示模式即成为后来显示卡的共同标准。我们也通称显示卡为“VGA 卡”。后来由于各家显示芯片的厂商更致力于将 VGA 的显示能力再提升，所以便诞生了“SVGA”（SuperVGA）与“XGA”（eXtended Graphic Array）这两个新名词。近几年来，显示芯片的厂商更将 3D 的功能与 VGA 的功能整合在一起，也就是我们现在常在游戏设备中使用的“3D 加速卡”或“3D 绘图显示卡”。

AGP

AGP 就是 Accelerated Graphics Port 的缩写，中文称为“绘图加速端口”。是为 3D 显示应用所产生的高效能接口规格与设计规范的插槽。在主板上 PCI（Peripheral Component Interconnect）插槽附近，我们就可以找到 AGP 插槽；它通常为棕色，而且比 PCI 插槽稍微短一些。

游戏设计一点通

AGP 的特性

1. 它可以使用 UMA（Upper Memory Area，高层内存区域）。AGP 会将显示功能所需要的记忆空间稍作调整，使其被充分应用。因此可以很轻易地提升数据处理的功能。

2. 它可以排除地址在线的多任务功能，以改进工作效率。由于 AGP 插槽是专为显示功能所设计的，所以它不会提供其他的数据信号进行传输的动作。因此 AGP 可以大大地提升显示传输的工作效率，并且改善数据处理的速度。

3. 它会重新定义数据信号的数据方式。由于显示数据的传输作业环境比较单纯化，所以在 AGP 重新定义这些传输方式之后，就可以利用特别的管道处理这些数据，以提升数据处理的效率。

4. 它还可以利用很巧妙的方式进行显示信息的传输，也就是先保留原来的信息而仅传输被修改过的数据；因此显示资料的传输效率就可大大地提升了。

PCI 总线

PCI 总线在 1992 年才开始建构在计算机主板上，在当时它算是相当快速的汇流接口。然而，在这 10 年来，随着处理器、内存，以及其他外围设置速度的不断提升，PCI 总线每秒 133MB 的数据传输率已经太过于缓慢；这也是现在桌上型计算机最大性能的瓶颈。

虽然 AGP 与 PCI 接口的运算处理方式都是以 32 位来进行的，不过 PCI 的频率只有 33.33MHz，而 AGP 的频率则可达 66.66MHz，并且 AGP 的传输速率是 PCI 的 4 倍。因此 3D 运算及多媒体播放也只有传输速度达到每秒 528MB 的 AGP 接口才足以应付了。

RAMDAC

RAMDAC 是 Random Access Memory Digital-to-Analog Converter 的缩写，中译为“随机存取内存数字模拟转换器”。RAMDAC 的分辨率、颜色数与输出频率也是影响显示卡性能的最重要因素。因为计算机是以数字方式进行运算，因此显示卡的内存也会以数字方式存储显示的数据。对于显示卡来说，这一些 0 与 1 的数字数据可以用来控制每一个像素的颜色值及亮度。不过，显示器并不会以数字方式来进行处理工作，因此在显示卡与显示器之间就必须建立一个“翻译”的角色；而 RAMDAC 的作用便是将数字信号转换为仿真信号以提供显示器能够显示影像。例如，我们常在显示适配卡上看到的“DAC xxx MHz”的字样，其中“xxx”所代表的是数字转换成仿真信号之间的带宽，“MHz”是它的单位，所以 RAMDAC 称得上是一种在绘图显示卡上极为重要的芯片。

显示芯片

显示芯片可以说是显示卡的心脏，在整个计算机输出的过程中占有绝对举足轻重的地位。在计算机的数据处理过程中，CPU 将运算处理后的显示信息通过总线传输到显示卡的显示芯片，而显示芯片再将这些数据运算处理后，通过显示卡将数据传送到显示屏幕上。以目前市场上的 3D 加速卡而言，大部分都是使用 nVIDIA(r) 公司所出产的芯片，如 TNT2、GeForce 256、GeForce2 MX、GeForce2 GTS、GeForce2 Ultra、GeForce3 以及最新的 GeForce4 等等。当然显示芯片运算能力的好坏是影响显示卡性能的主要因素。

显示内存

显示内存的主要功能是将显示芯片处理的数据暂时存储于显示内存上，然后再将显示数据传送到显示屏幕；显示卡上的分辨率越高，屏幕上显示的像素点就会越小越多，并且所需要的显示内存也会跟着增多。每一块显示卡至少要具备 1MB 的显示内存，而显示内存也随着 3 D 加速卡的演进不断地增加。从早期的 1MB、2MB、4MB、8MB、16MB，一直到 TNT2 的 8、32、64MB 的 SDRAM，甚至到最新的 nVIDIA(r) GeForce2、3、4，它们都有了 64MB 显示内存的版本，另外显示内存的种类也从早期的 DRAM 慢慢演变到现在广为流行的 SDRAM 及 DDR。

VGA BIOS

VGA BIOS 里包含了产品标识、控制程序等信息，这些信息一般是由显示卡厂商直接烧录在显示卡上的 ROM 芯片里，让我们可以通过显示卡厂商所提供的 BIOS 更新程序来更新显示卡上的 BIOS 版本。

分辨率

分辨率代表了显示卡在显示器上所能描绘点的最大数量。一般而言，它由横向点数乘以纵向点数来表示，例如标准显示卡的最大分辨率为 1024×768pixel。

颜色数

颜色数指的是显示卡在目前的分辨率下所能显示的颜色数量，一般是以多少色或是多少 bit 来表示。例如标准的 VGA 显示卡在 640×480 的分辨率下，其颜色数为 256 色或 8bit，而 Super VGA 显示卡的基本分辨率为 1024×768pixel，颜色数可达 32bit 之多。

更新频率

更新频率是指影像在显示器上每一秒所更新的速率。简单地说，它就是影像每秒钟在屏幕上出现的画面数。如果显示更新频率越高，屏幕上影像的闪烁感就会越小，影像也就会越趋稳定，表现的视觉效果自然较好。

NTSC

NTSC 即为 National Television System Committee 的缩写，是“美国国家电视系统委员

会”的简称。NTSC 是公元 1941 年由美国联邦通讯委员会正式颁布实施的电视画面播放标准，其电视画面必须具备 525 条扫描线，以及每秒可播放 30 个画面。目前采用 NTSC 标准的有中国台湾省、美国、日本等国家。

PAL

PAL 即为 Phase Alternationby Line 的缩写，它是一种电视画面的播放标准，是于 1949 年由英、德、瑞士等西欧国家所制定，电视显示画面必须具备 625 条扫描线，以及每秒可播放 25 个画面。目前采用 PAL 标准的有中国大陆、西欧及东南亚等国家。

有关显示卡的专有名词大部分都已经介绍完毕，下面我们再来看看计算机上的另一种硬设备——“声卡”。

2.1.3 声卡

您可以试试看将身边的所有音效装置全部关掉，然后再体验一下没有背景音乐、音效及配音的游戏，是不是觉得游戏变得索然无味了呢？其实少了声卡，整个游戏仿佛是一场哑剧，无声无息也很无趣。以现在各位的计算机设备而言，声卡已经是不可缺少的一项基本配置了。不过使用声卡，还是会遇上一些不必要的小问题，那就是经常声卡上会标示着一些令人看不懂的名词或数据，让我们不知道到底哪一块声卡才是符合自己的需求。在本节中我们将为您详细介绍一般市面上声卡有关的术语名词，以解答各位的困惑。

取样频率

利用数字来表示的声音是断断续续的，所以将模拟信号转换成数字信号时，会在模拟声音波形上每隔一个时间里取一个幅度值，这个过程我们称之为“取样”。在取样的过程中，这段间隔的时间我们就称它为“取样频率”。市面上常见的声卡取样频率有8KHz、11.025KHz、22.05KHz、16KHz、37.8KHz、44.1KHz、48KHz等等，而取样频率值越大，声音的失真率就会越小。

波表合成

波表合成会通过乐器的声音进行取样，并将取样的数据保存下来。例如我们在播放音效的时候，就是要靠声卡上的微处理器，或者是 CPU 来处理这些数据的发声。

MIDI

MIDI 又可以称为“电子乐器数字信号”，是由“MIDI 生产商协会”（MIDI Manufacturers Association）所制订。其制订的数据是为了要给所有 MIDI 仪器制造商在音色与打击乐器的一套音效清单标准；MIDI 总共有 128 个标准音和 81 个打击乐器的音色。

DSP

DSP 即为 Digital Signal Processing 的缩写，中译是“数字信号处理”。DSP 是声卡中专门用来处理效果的芯片，又可以称为“效果器”。由于具有这种芯片的声卡价格比较昂贵，所以通常只有较高级的声卡中才会看到。

EAX

EAX 即为 Environmental Audio Extensions 的缩写，中译是“环境音效”。它是由“创新”与“微软”这两家公司联合推出，作为 DirectSound 3D 扩展的一套应用程序技术。EAX 最新的 2.0 版可以表现混音、封闭、阻塞等效果。混音呈现了让虚拟音源随着环境的变化而产生不同的效果；封闭可以模拟听者与音源之间有大面积阻碍的效果；阻塞则是模拟听者与音源之间有小面积阻隔的效果。不过如果您要使用这种技术的话，首先必须具备 4 个以上音箱才能表现出 EAX 的特殊效果。

A3D

A3D 是由 Aureal Semiconductor 所开发完成的一套交互式 3D 定位音效技术，它的主要特色是在于 3D 音效定位上有非常独特的处理方式。最早的 A3D 技术只需要两个音箱作为输出之用，不过在新的版本中，必须使用 4 个音箱才能让 A3D 实现它的 3D 定位效果。

DAC

DAC 即为 Digital-analog Converter 的缩写，中译是“数字模拟转换器”。因为一般音响都只能接受模拟信号的数据，而计算机中所处理的数据通常是数字信号，因此声卡在读出数字信号后，必须通过 DAC 的转换而成为一般音响能够接受的模拟信号，再由音响来带动音箱发出声音。

FR

FR 就是 Frequency Response 的缩写，中译是“频率响应”。FR 是声卡 DAC 转换器频率响应能力的评价指标。由于人类的耳朵对声音的接收范围大致在 20Hz~20KHz 之间，所以声卡就必须在 20Hz~20KHz 频率之间保持“直线式”的响应效果，而功率增益或功率衰减都属失真的现象，所以我们通常都将 FR 控制在±3dB 的范围内（dB 为频率的单位）。

双工

双工是表示当信号同时在一条线，是否能够执行双向的传输。如果信号可由任意方向来传送数据，但是在同一个时间内却只能以一个方向来传输，我们称为“半双工”；而同时可以用来当作收发信号的功能，我们则称为“全双工”

SNR

SNR 即为 Signal-to-Noise Ratio 的缩写，中译为“信噪比”。它是一个诊断声卡控制噪音能力的重要指标。SNR 就是有用信号和噪声信号功率的比值，单位是分贝。SNR 的值越大，则声卡的滤波效果越好，所以优质声卡的 SNR 值至少要大于 80 分贝。

FM 合成

FM 合成技术是早期电子合成乐器所采用的发音方式，后来由 Yamaha 公司将它应用到 PC 声卡上。FM 比 PC 小喇叭所提供的效果还要好；最大的特色就是 FM 的发音方式使声音听起来比较干净、清脆。

声道

5.1 声道已经被广泛地运用在各种电影院及家庭中，一些比较知名的声音录制压缩格式，如“杜比 AC.3”（Dolby Digital）、DTS 等等都是以 5.1 声道系统为技术蓝本。其实 5.1 声道系统的构思是来自于 4.1 环绕系统，这两者不同之处在于 5.1 声道增加了一个中置单元。这个中置单元则传送低于 80Hz 的声音信号，在影片播放的时候更有利于加强人声；原理就是将人的对话集中在整个环境的中央，以提升整体环境效果。

其实好的计算机硬件设备对于游戏来说，具有一定的帮助，至少游戏上华丽的画面、音效及效率都与它们息息相关。您倒不必去死记这些硬件名词解释，而是作为一个好的游戏设计者，在开发游戏的过程中，懂得善于利用它们，肯定会得到最佳的效果。

2.2 游戏类专有名词

在游戏界中，各位可能会经常遇到一些千奇百怪的特殊名词。有些特殊名词我们可以从字义上很轻易地辨识出来，但是有些特殊名词，却很难完全了解它的意思。基于这个理由，我们特别为您在本节里收集整理了游戏界中一些较为常见的特殊专有名词。

2.2.1 TV 游戏主机

在开始要介绍游戏专有名词之前，先来认识一下影响游戏发展最深远的 TV 游戏主机！一般而言，在市面上经常会看到一些游戏主机特有的专有名词，例如 PS、PS2、SFC、XBOX 等等。如果您是一个老练的玩家，那么这些特殊的专有名词，想必对您来说，一点都不陌生！假如您是刚踏进这个领域的新手，那么当您看到这些五花八门的专有名词，一定会被搞得一头雾水！其实，不管您是老手或新手，对于这些特有名词，最好都能更深入地认识它们。

FC

FC 即为 Family Computer 的缩写，它是任天堂（Nintendo）公司所推出的 8 位 TV 游戏主机，中译就是“家用计算机”的意思，产品如图 2.1 所示。

图 2.1　任天堂（Nintendo）公司的 8 位 TV 游戏主机

FC 这个随伴着许多玩家长大的童年玩伴，虽然现在的 TV 游乐主机一直不断推陈出新，不过它们仍然无法取代 FC 在一些老玩家心中的地位！可能有些玩家们曾听过任天堂的红白机吧！为何称为“红白机”呢？那是因为当初 FC 在刚推出发行的时候，它是以红白相间的主机外壳来呈现的，所以 FC 就是“红白机”。

SFC

SFC 即为 Super Family Computer 的缩写，是 FC 的下一代，由任天堂公司所出产发行的 16 位 TV 游戏机，中译是“超级家用计算机”，如图 2.2 所示。

图 2.2　任天堂公司所出产发行的 16 位 TV 游戏机

SFC 就是国内俗称为“超任”游戏主机，在美国市场则称之为“Super NES”，简称“SNES”。这款游戏主机是任天堂公司最成功的 TV 游乐器主机，成功地取代了原本任天堂红白机的地位，并且更为广受欢迎。直到目前为止，这台主机在世界上的销售量还是位居第一。由于它的市场占有率高，所以大部分的电子软件开发商，都曾经加入这款游戏主机的开发行列，使得这款游乐器上拥有许多著名的游戏软件。

MD

MD 即为 MEGA Drive 的缩写，它是世嘉（SEGA）公司所出产的 16 位 TV 游戏机，中译是“兆位驱动”，如图 2.3 所示。

图 2.3　世嘉（SEGA）公司所出产的 16 位 TV 游戏机

MD 是 SEGA 公司的第五代主机，在与超任的竞争之下，虽然 MD 最后失败，不过它在国外市场上却获得了胜利，这让 SEGA 公司奠定了日后继续在 TV 游戏主机市场与其他

公司一较长短的决心。SEGA 公司从这台主机开始，学习了任天堂公司开放第三厂商加盟的做法，因此除了 SEGA 自行研发的游戏外，这台主机上还出现了许多其他游戏公司的作品。甚至于这款主机还推出了 CD-ROM 的接口，不过在市场上的销售寿命却不长，而游戏的表现也都差强人意。在海外上市的时候，这台主机的名称则改为 SS。

SS

SS 即为 SEGA SATURN 的缩写；是世嘉公司所出产的 32 位游戏机，中译为“世嘉土星”，如图 2.4 所示。

图 2.4　世嘉土星

SEGA SATURN 从发行到盛行时，几乎可以与其他游戏主机分庭抗礼，不过自从 DC（Dream Cast）问世后，SS 几乎可以说是顿时失宠、光芒尽失。 其实 SS 游戏主机有非常强的 2D 表现性能，它可以使 CG 画面呈现很棒的效果，同时它在光盘读取的速度上也比 PS（Play Station）快许多，另外它可以额外加装 MPEG 卡，使得主机也可以用来观赏 VCD。

PS

PS 即为 Play Station 的缩写，是 Sony 公司出产的 32 位 TV 游戏机，中译为“玩家游戏站”，如图 2.5 所示。

图 2.5　Sony 公司出产的 32 位 TV 游戏机

对于 PS 游戏主机的历史，足以称为电子史上的一大奇迹。由于这款游戏主机的推出，使得口碑一直不错的 32 位 SS 游戏主机遭受到前所未有的打击，许多游戏软件的大作都移植到这一款游戏主机上。PS 游戏主机最大的特色就是在于 3D 运算，许多游戏都将 PS 游戏主机的 3D 性能发挥到了极致，使得 PS 游戏主机在玩家的心中留下了非常深刻的印象，其中最吸引玩家的地方也是它可以支持许多画面华丽的游戏，使得许多玩家对它真的是趋之若鹜，爱不释手。

N64

N64 即为 Nintendo 64 的缩写，是任天堂公司所出的 64 位 TV 游戏机，中译为“任天堂 64”，如图 2.6 所示。

图 2.6　任天堂公司的 64 位 TV 游戏机

N64 最大特色就是它是第一台以 4 个操作接口为主体的游戏主机，也是众多游戏主机里惟一一台以 64 位运算的主机。N64 游戏主机是以卡匣作为软件媒介，这项优点大大提升了软件读取的速度，不过还是有许多人不愿意尝试 N64，因为这些人嫌 N64 主机的游戏少，不过喜爱 N64 游戏主机的玩家，已有日渐增多的趋势。

DC

DC 即为 Dream Cast 的缩写，是世嘉公司所出产的 128 位游戏机，如图 2.7 所示。

图 2.7　世嘉公司的 128 位游戏机

DC 这一款游戏主机是一台娱乐性十足的平台主机，再加上是由 SEGA 公司所制造发行的游戏主机，所以有许多 SEGA 公司所推出的游戏名作被移植到这台主机上，这也是最吸引玩家之处。不过，它的缺点就是非常容易发热，玩家们在使用 DC 游戏主机时，一定要位于通风良好的地方，不然会很容易当机（CPU 太热了）；如果当玩家当时在游戏进行中且又没有存档的话，那真的是会让玩家哭笑不得、不知所措呢！

PS2

PS2 即为 Play Station 2 的缩写，它是由 Sony 公司出产发行的 128 位 PS 第二代主机，如图 2.8 所示。

图 2.8　Sony 公司出产发行的 128 位 PS 第二代主机

PS2 游戏主机与我们所提过的游戏主机有很大的不同，因为 PS2 游戏主机不但可以玩 PS 游戏主机上的所有游戏作品，它更能够让玩家享受到视听娱乐等附加的功能，而且所有 PS 游戏主机上的所有配备（手把、记忆卡等等）都可以继续延用下去，使得拥有 PS 游戏主机的玩家们也会想要拥有 PS2 游戏主机。虽然 PS2 游戏主机本身具有 DVD 的播放机功能，但是 DVD 画面的品质并不是非常理想，不过，对于一些没有 DVD 播放机的玩家们来说，它还算是一台物超所值的游戏主机。

NGC

NGC 即为 Nintendo Game Cube 的缩写，是任天堂公司所推出的 128 位 TV 游戏机，中译为“任天堂游戏立方体”，如图 2.9 所示。

GameCube 是属于一台纯粹家用的游戏主机，它并没有集合太多影音多媒体功能，另外为了避免和 Sony 的 PS2、微软的 XBOX 正面交锋，任天堂把火力全部集中在 GameCube 的游戏功能，所以 GameCube 的硬件成本自然可以压得较低，售价自然也是最吸引玩家的地方。其实 GameCube 游戏主机并非只是任天堂一家公司所独立完成的，它是得到了许多公司的协助，例如包括了 Matsushita（松下）、IBM、NEC、ATI、Macronix、MoSys、S3、Applied Microsystems、Factor 5、Metrowerks 和 Conexant 等等大厂商的加入，GameCube

游戏主机才得以顺利完成；也因此玩家又可以再次拥抱任天堂游戏的魅力。

图 2.9　天堂公司所推出的 128 位 TV 游戏机

XBO

XBOX 是微软公司所生产发行的 128 位 TV 游戏机，如图 2.10 所示。

图 2.10　微软公司所生产发行的 128 位 TV 游戏机

XBOX 也是微软公司的下一代视频游戏系统，它带给玩家有史以来最具震撼力的游戏体验。XBOX 不但能使游戏设计高手的功力更加强大，带来从未有过的创意想象技术，进而创造出幻觉与现实界线变得模糊的超炫游戏。

NGC、PS2 与 XBOX 这三台都是当今市面上最新、最流行的电子游戏主机，彼此之间互相竞争，都想要在电子领域上争得冠军的宝座，不论是在硬件或者是价格上都拼得您死我活，因而形成了一个游戏主机市场的肉搏战。对于游戏主机未来的形势，目前仍没有办

法下断言；但不管怎么说，我们相信游戏主机争夺战最后的结果绝对是玩家们大获全胜。

2.2.2 掌上型游戏主机

我们已经介绍完 TV 上的游戏主机，接下来，我们再来看看有哪些掌上型游戏主机？

GG

GG 即为 Game Gear 的缩写，是世嘉公司所出产的 8 位掌上型游戏机，如图 2.11 所示。

图 2.11　世嘉公司所出产的 8 位掌上型游戏机

GG 是经典的老牌掌上型游戏机，虽然它比任天堂 GameBoy 掌上型游戏机推出的时间还要晚，但是它是第一款 8 位的彩色掌上型游戏机，其画质要比 GameBoy 好上许多。GG 具有全彩高分辨率 3.2 英寸的背光液晶显示屏幕，因此我们即使在黑夜里也能够玩。不过，这种背光液晶显示屏幕也成了它最大的缺点，其主机的耗电量大约是 6 颗 1.5A 的电池，因此它也只能连续玩 2 个多小时而已，而且重量与体积也比 GameBoy 大许多。尽管如此，GG 主机还是推出了不少好玩的 SEGA 经典游戏。当时 GG 的价格非常昂贵，因此它就无法完全地独占整个掌上型游戏主机的市场，关于这一点就让 GameBoy 独占了先机。

GB

GB 即为 Game Boy 的缩写，是任天堂 8 位掌上型游戏机，中译为“游戏小子”，如图 2.12 所示。

图 2.12　任天堂 8 位掌上型游戏机

GB 是我们最常玩的掌上型游戏机，一直到现在市面上还是非常流行，现在还是持续推出了各式各样的新型 GB 主机。虽然 GB 是以黑白的色调来呈现游戏，不过仍然独占掌上型游戏机的销售榜首。

NGP

NGP 即为 NEO GEO POCKET 的缩写，是 SNK 公司所发行的 16 位掌上型游戏机，中译为“口袋 NEO GEO”，简称“口袋 NG”，如图 2.13 所示。

图 2.13　SNK 公司所发行的 16 位掌上型游戏机

1998 年 10 月由 SNK 公司所发售的这款掌上型游戏机 NGP，可以允许与世嘉的 DC 互连。不过它的游戏大多是移植于大型主机，种类就会太过于单一，所以未能在游戏市场上争得一席之地

GBA

GBA 即为 Game Boy Advance 的缩写，是由任天堂公司推出的新一代掌上型游戏机，具备了许多不同颜色的主机供玩家们选购。而主机的外型也和历代的 GB 颇有分别，最大分别是屏幕的位置改放在十字掣和 A、B 键之间，如图 2.14 所示。

图 2.14　任天堂公司推出的新一代掌上型游戏机

GBA 最大的卖点就是它可以完全支持 GB 的所有游戏，虽然旧游戏放在 GBA 上并不会强化它的画质，不过这并不会减少玩家们购买的意愿；因为在 GBA 专用游戏还未出现之前，GBA 就已经拥有数以万计的 GB 游戏了。

2.2.3 游戏厂商

在众多游戏主机的竞争中，我们相信对于玩家们最具有影响力的应该是主机上所支持的游戏种类与数目吧！其实从以上的游戏主机来看，如果一台游戏主机上的游戏够多且好玩的话，这台游戏主机的存活率便可以大大提升，而制作出这些华丽好玩的游戏，就是幕后这些游戏软件开发商了。

在我们成长的过程中，出现了许许多多好玩的经典游戏，这些游戏厂商的努力真是功不可没，没有他们，也不能让玩家们有不断推陈出新的新款游戏，现在就借这个机会迅速来认识一些知名的游戏开发厂商吧！

Blizzard

Blizzard 游戏公司最具代表性的作品有 Diablo（暗黑破坏神）系列、Warcraft（魔兽争

霸）系列等。

Electronic Arts

Electronic Arts 游戏公司最具代表性的作品有 Command&Conguer: Red Alert2（红色警戒）系列、Star Craft: Brood War（星海争霸）系列等等。

CAPCOM

CAPCOM 中译名称为“卡普空”，是日本著名游戏制造商。其游戏公司最具代表性的作品有 Rockman（洛克人）系列、Bio Hazard（恶灵古堡）系列、Dino Crisis（恐龙危机）系列、STREET FIGHTER（快打旋风）系列、Onimusha（鬼武者）系列等等。

KOEI

KOEI 中译名称为“光荣”，是日本著名游戏制造商，其游戏制作方向多半为策略性游戏，最具代表性的作品有“信长之野望”系列、“三国志”系列等等。

KONAMI

KONAMI 中译名称为“科拿米”，是日本著名游戏制造商，其游戏公司最具代表性的作品有“恶魔城”系列、“沙罗曼蛇”系列、“魂斗罗”系列、“特攻神谍”系列、“沉默之丘”系列等等。

HUDSON

是日本著名游戏制造商，其最具代表性的作品有“炸弹超人”系列、“桃太郎”系列等等。

Microsoft

微软公司也曾制作了许多经典名作，其中最具代表性的作品有“世纪帝国”系列、“仿真飞行”系列等等。

NAMCO

日本著名游戏制造商，最具代表性的作品有“食鬼”、“铁拳”、”刀魂”、”剑魂”等等。

SNK

日本著名街机制造商，最具代表性的作品有“KOF”（拳皇）系列、“METAL SLUG”（越南大战）、“饿狼传说”、“侍魂”系列等等，都深受广大玩家喜爱。但由于该公司投资失利，最终声明破产。后来所幸被 PLAYMORE 收购，玩家们才得以继续玩 SNK 所出的游戏。

SQUARE

日本著名游戏制造商，最具代表性的作品有“FINAL FANTASY”（太空战士）系列、“路行鸟”系列、“王国之心”等等。

在这里已经介绍完市面上较为当红的游戏开发商了，接下来认识有关经典游戏名称缩写的专有名词。

2.2.4 游戏名称

在游戏世界里，有许多耳熟能详的知名游戏，不过一般而言，这些知名游戏在市面上常常会以全名的缩写来表示。如果在卖场上看到这些游戏名称的缩写时，却对它们浑然不知，这对于一个想要进入游戏行列的玩家来说，倒真的是一件颇为痛苦的事！在本节中收录了市面上最为经典与流行游戏名称的缩写，并且为您详细介绍它们的发展史。

CS

CS 即是 Counter-Strike 的缩写，中译为“绝对武力”。在介绍 CS 游戏之前，我们先来了解一下 CS 的前身——“Half-Life”（战栗时空，简称 HL）。在 1998 年以前，相信可能没有人知道世界上存在着一个名叫“Valve”的游戏开发小组，这个游戏开发小组经历了 1998 年那个令人难忘的 E3 电子展之后，竟然几乎没有一个玩家会不知道他们的大名。由他们所开发的“Half-Life”所带来的震撼感，使得一些挑剔的玩家不得不承认它是一款具有强烈感染力的游戏，同时“Half-Life”游戏更获得了超过 40 家媒体厂商所一致公认“年度最佳游戏”的赞誉。在今日竞争激烈的动作类游戏市场中，它与 Quake 系列、Unreal 系列形成一股三强鼎立之势。

Valve 并不单只是制作了 HL 游戏，他们还另外替“战栗时空”做了一套“软件开发工具”（Software Developer's Kit,SDK），让这套开发工具可以再另外开发出其他的外挂模块，而 CS 就是在这样的背景下诞生的。在 1999 年，以 Gooseman（程序设计师）和 Cliffe（美工、音效师）为首的“CS 小组”制作了一套前所未见的 HL 外挂——“Counter-Strike”（绝

对武力)。这套游戏不走 HL 和其他众多外挂的传统科幻路线，而是以拟真为最高原则，它希望能将真实世界中恐怖分子（Terrorists）与反恐怖特种部队（Counter-Terrorists）之间的对抗，搬进计算机与网络的虚拟世界中。

CS 最大的特点就是在于它的“拟真”方面。由于武器和游戏规则都尽量与现实中的情景相同，随之而来也发展出各式各样的战技：例如“突击”、“偷袭”、“蹲射”、“跳射”、“左右闪”、“AK 狙击”及“沙漠狙击”等等。这些技巧数都数不完。当然，我们还可以在 CS 的游戏中进行小队的战术技巧，例如，“交叉掩护”及“分进合击”等常见的战术。毕竟 CS 是一个拟真的游戏，而在现实生活当中，如果没有一个好的团队，那是不可能造就出今日举世闻名的世界级游戏大作。

SW

SW 即为 Sakura Wars 的缩写，中文译名为“樱花大战”。无论是前面 SS 主机时代或是 DC 主机的时代，许多玩家常为了想要玩到“樱花大战”的游戏而另外再添购其他的主机。为什么“樱花大战”的游戏能够如此吸引玩家呢？关于这一点，我们要先从这款游戏的制作人员说起。其实“樱花大战”并不能算是 Sega 公司所自行开发的产品，反而“樱花大战”的策划与研发团队是制作“天外魔境”而闻名的 Red 小组所负责的，Red 小组先前从未制作过“樱花大战”这一类的游戏作品，特别是偏重于角色扮演与射击方面。为了创作出“樱花大战”的游戏魅力，策划小组还特别编造了一段幕后秘辛。他们说这款游戏的灵感是来自团队中某一个策划人员家中仓库内所发现的一本旧书开始，由于这一本旧书记载着帝国华集团与黑之巢会的精彩对抗过程，于是它们就决定要把这种对抗的过程改编成实际的游戏作品。策划小组为了营造出“樱花大战”里头日本大正时代的背景，规划出种种虚构的历史事件与物品，而这种历史事件与物品正是包含了整个游戏的重心所在，但是光靠这些似乎还不足够吸引玩家们的目光，因为它们考虑到游戏中没有一些出色的人物设置是不可能成功掳获玩家们的芳心。“藤岛康介”是日本知名的漫画家，而藤岛康介就是在这种情势下肩负起“樱花大战”的人物设置工作。其实这也并非藤岛康介首次跨进游戏领域，在 Namco 公司的“时空幻境”中同样也是由他担任起人物设置工作，也因此当时就吸引不少漫画迷为此而特地购买这款游戏作品，使其名声大噪。

SF

SF 即为 Street Fight 的缩写，中译为“街头霸王”。在 1987 年，CAPCOM 公司所推出的格斗游戏 Street Fight 出现，瞬间风靡全球，它更为后来推出的格斗游戏奠定了不可抹灭的基础。这款游戏最吸引玩家的地方就在它是全球第 1 个格斗游戏中拥有必杀技和对战模式机制。

KOF

KOF 即为 The King Of Fight 的缩写，中译为“格斗天王”。SNK 公司在 Street Fight 推出之后，也顺势推出了第一套格斗游戏——“饿狼传说”。而 SNK 公司在“饿狼传说”游戏推出的次年再度推出“龙虎之拳”游戏，在“龙虎之拳”游戏中，新增了“存储能量系统”和“超必杀技”的游戏机制。之后，SNK 公司在 1993 年推出了“侍魂”系列游戏，而“侍魂”游戏中首创了以武器作为格斗的方式。而在 1994 年所推出的“饿狼传说 Special”和“龙虎之拳”游戏中的主角“阪崎獠”和“Geese Howard”分别出现在对方游戏故事中，这为“KOF”游戏奠下了重要的基础，SNK 公司为此在 1994 年的 8 月再度推出了 KOF 的游戏。KOF 游戏除了前面“饿狼传说”和“龙虎之拳”的人物之外，它还出现了经典动作游戏“怒”和“Psycho Soldier”游戏中的人物，而当中的主角“草薙京”就是在这款游戏中诞生的。该游戏最吸引玩家的地方是在于游戏中主角人物的描述与以“队伍”为机制的游戏模式。

FF

FF 即为 Final Fantasy 的缩写，中文译名为“最终幻想”，又称为“太空战士”。FF 是 SQUARE（史克威尔）公司所推出的一系列幻想游戏，目前已经出到第 11 代了，它是以感人的故事情节、壮大的世界观和细致的游戏画面，赢得了玩家们的好评。

MGS

即为 Metal Gear Solid 的缩写，中文译名为“特攻神谍”。3 年前，“科拿米”（Konami）公司在 PlayStation 主机上面推出“特攻神谍”游戏后，便络绎不绝地得到玩家的赞赏声，并且马上成为 PlayStation 主机有史以来最卖座的代表作品之一。经过了短短的一年时间，一款以“间谍任务”为卖点的加强版本又顺势推出，更彻底的满足了这些死忠玩家们的热烈支持。后来 Microsoft 公司将它移植到 PC 上，自然又再度引起玩家们的高度期待。 在“特攻神谍”这款游戏中，玩家是扮演一位间谍，他是相当顶尖的一名秘密特务人员。这款游戏的剧情主轴发展就像是一部杰出的间谍电影一样，我们在玩这款游戏时的情绪会随着各个精彩剧情的变化而不断地起伏。“特攻神谍”的游戏通过如真实影片一般的画面呈现方式，不仅可以让玩家们体验到无比的电影魅力，更能够让故事转折变得更加写实与出色。“特攻神谍”游戏是采用第三人称的俯瞰方式来显示整个地图视野，不过也可以通过功能选项来改变成第一人称的游戏视野。

SH

SH 即为 Silent Hill 的缩写，中译“沉默之丘”。当我们进入到一个阴暗寂静、鬼哭狼嚎的梦境时，自然就会产生一种恐怖的心理，游戏开发者就紧紧抓住这一种心理来制作游戏，而这一类的游戏却往往能够让玩家加倍喜欢。“沉默之丘”就是一款十分恐怖的游戏，当我们亲身玩了几个小时以后，真的能够感受到一种极为恐怖的感觉和沉重窒息的压力。而且那种恐怖的心理非常特别，不过它并非是一种邪恶的恐怖，是一种善良且能够直侵我们内心的恐怖！

AC

AC 即为 ACE Combat 的缩写，中译为“空战奇兵”。“空战奇兵”游戏是 PS 平台上颇有盛名的射击类游戏，一直到 2001 年 9 月，首度在 PS2 平台上发行第四代，而第四代的作品不但缔造出超越前作的真实感，并且在音效部分也实地前往日本空军基地，收录飞机起降产生的现场音，在画面部分更挑战 PS2 主机的效能，创造出真假不分的空间感。在游戏中，玩家可以按照自己的个人习惯来切换“驾驶舱视点”（如按下 PS2 摇杆的 L 与 R 键）。在游戏进行的同时，可以随着进度发展的附加故事，以平民的角度来描述玩家们对战争的感观，并且足以发人深省。“空战奇兵”游戏还有一个很吸引人的设计，那就是玩家可以使用特殊武器进行攻击任务，在熟悉游戏并进行到中期后的任务将重视玩家对于战争的判断力，如果不动动脑筋来思考敌方战术的话，玩家想取得胜利就会是一件非常困难的事情。

在今天这一个日新月异的时代里，我们时常有机会接触到更新、更绚丽的游戏。我们没有办法在这里介绍完所有的热门游戏，而您只要多看、多听、多了解，一定可以登堂入室了解游戏世界中无穷奥妙之处。

2.3 游戏常见术语

当我们与其他玩家在游戏中相互共鸣的时候，他们都会从口中道出一些特殊的用语，如果是一个刚踏进游戏领域的初学者，对于他们所讲的用语，倒真令人丈二和尚摸不着头脑。在本节里为您收录了许多在游戏中经常会听见的行家用语，并且加以解释其用法。

必杀技

通常在格斗游戏中会出现，它是利用特殊的摇捍转法或按键按法而使出来的克敌制胜技巧。

超必杀技

指的是比一般必杀技的威力来得强大的超级必杀技。不过在格斗游戏中，它是有某些条件限制的。

连续技

其功能就是以特定的攻击来连接其他的攻击，使对手受到连续损伤的技巧（超必杀技造成的连续损伤通常不算在内）。

贱招

其用语指的是使用重复的伎俩让对手毫无招架之力，并且将对手打败的伎俩。

金手指

金手指是一种外围设备，用来让游戏中的某些设置数值改变，达到所需的利益。例如将自己的金钱、经验值、道具等利用金手指来增加，而不是通过游戏的进化而提升的。

Bug

Bug 即是“程序漏洞”，俗称“臭虫”，是指那些因游戏设计者与测试者疏漏而遗留在游戏中的错误程序，严重的话将会影响整个游戏作品的品质。

密技

其用语是程序设计师在游戏中故意设置的一些小技巧，在游戏中输入某些指令或做了某一些事就会发生一些意想不到的事情发生等等，目的是为了让玩家享受另外一种游戏中投机取巧的快感。

Boss

Boss 即是“大头目”的意思，是在游戏中出现的较为巨大有力且难缠的敌方对手。一般这类敌人在整个游戏过程中只会出现一次，而且是出现在某一关的最后，不像有些跑龙套小怪物在游戏中可以重复登场。

记忆卡

记忆卡指的是 SS、PS、PS2、XBOX 等主机，因为必须使用外部存储介质来记录游戏进度所使用的装置。

E3

E3 即是 Electronic Entertainment Expo 的缩写，指的是美国电子娱乐大展。目前在全球，它是最为盛大的计算机游戏与电视游戏商贸展示会，基本上每年都会在 5 月举行。

改机

指的是针对 SS 与 PS 主机而言，就是改成“台片”适用的主机。换句话说，就是可以玩盗版游戏的主机。

台片

台片即是“台湾盗版”的通称，这也是近几年来受到国人非常关注的“盗版”议题。

Level

Level 是“关卡”的意思，换句话说，即是游戏中一个连续的完整舞台场景，有时候我们也称它为“Stage”。

HP

HP 即是 Hit Point 的缩写，它指的是“生命力”。在游戏中即是人物或作战单位的生命数值。一般而言，HP 为 0 即是表示死亡，甚至 Game Over。

MP

MP 即是 Magic Point 的缩写，指的是“魔法力”的意思。在游戏中即是人物的魔法数值，一旦使用完即不能再使用任何魔法招式。

Experience Point

Experience Point 即是“经验点数”的意思。它常出现在角色扮演游戏中，以数值来计量人物的成长，如果经验点数达到一定数值之后，人物则可以将自己的能力升级，这时人物就会变得更强大。

NPC

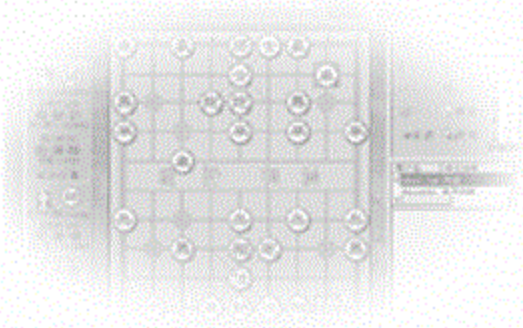
NPC 即是 Non Player Character 的缩写，指的是“非玩家角色”的意思。在角色扮演的游戏中，玩家通常会在游戏的过程中会遇到一些不受控制的人物，而这些人物会提示玩家们重要的情报或线索，使得玩家可以继续游戏。

Storyline

Storyline 就是“剧情”的意思，也就是游戏的故事大纲，它可区分成“直线型”、“多线型”以及“开放型”等 3 种游戏剧情为主轴。

第 3 章

游戏设计架构与规划



一款游戏必须要有它的一套游戏系统及机制，本章将介绍在设计一款游戏时，我们应该如何来规划它的基本架构与整个游戏制作的流程，以致于可以控制游戏开发的进度。

3.1 游戏中戏剧手法的应用

以早期游戏制作的架构而言，虽然在玩法与剧情上的表现，明显地与现在的游戏相比较，是显得粗糙了许多。不过早期的游戏还是具有相当不错的剧情表现。其实我们发现，不论是以前还是现在的游戏，只要有非常出色的剧情并且能融入现代电影的制作手法，这套游戏通常就能大放异彩。以 SQUARE（史克威尔）公司所推出“Final Fantasy”（太空战士）游戏系列来说，它就是一个很好的例子。

其实当我们开始制作一套游戏时，首先必须考虑以下 8 项原则：

- 确立游戏的主题。
- 游戏的过程与发展。
- 故事的讲述方式。
- 设置游戏的主人翁。
- 游戏的描述角度。
- 游戏的情感与悬念。
- 游戏的节奏。
- 游戏的风格。

接下来，我们针对 8 项原则分别为您详加介绍。

3.1.1 确立游戏的主题

当我们在设计一款游戏时，就应该了解到通常具有一般、共同主题的游戏剧本适用于不同文化背景的玩家族群，例如爱情主题、战争主题等等；这些主题可轻易地吸引象玩家们。因此，这类主题对于游戏在不同地区的推广流行是有帮助的。

如果游戏题材比较老旧的话，那么可以尝试从全新的角度来诠释它，或者以奇幻想象的游戏系统来赋予这个老故事前所未有的呈现方式。简单地说，就是做到旧瓶里装新酒，让玩家感觉到我们的游戏有独特的创意，如遨游在异想天开、如梦似幻的游戏新境界中。就好比本公司所制作的“巴冷主公”角色扮演游戏，就是取材于我国台湾省鲁凯传统神话故事，再配合最新 3D 引擎技术来加以改编制作。

3.1.2 游戏的过程与发展

一般而言，在电影戏剧表现手法上可以看到两个极为重要的因素是牵引故事情节的原动力，那就是“障碍”与“冲突”。

如果将这两种重要的因素具体应用到游戏中的话，“障碍”就是游戏过程中玩家必须解决的静态型难题；而“冲突”就是玩家在游戏进行中的动作性阻碍。玩家必须根据主角本身当时的状况，想出一些有效的解决之道。从游戏的角度来描述这两个因素，“障碍”为游戏中的“谜题”，而“冲突”无疑就是游戏中的“战斗”。如同在 RPG 游戏中，这两种因素就经常巧妙地交互穿插使用。

游戏中加入适当的障碍和冲突，让玩家在游戏中有不断克服困难前进的挑战欲，因而逐步带动游戏中的故事情节往后发展。

3.1.3 故事的讲述方式

在游戏中，我们可以将故事的讲述方式分成两种，分别是“倒叙法”与“正叙法”

倒叙法

倒叙法就是将玩家先处于事件发生后的结果之中，然后再让玩家回到过去，去了解事件发生原因，或者让玩家自行去阻止事件发生。如同“MYST”（迷雾之岛）的 AVG 游戏就是最典型的例子。

正叙法

正叙法就是以平铺直叙的表达方式，让游戏故事情节随着玩家们的遭遇而展开。也就是说，玩家对于游戏进行中的一切都是未知的，而未来的发展只等待玩家自己去发现或创造。通常，在游戏中会看到它们多半是以这样的陈述方式来描述游戏故事剧情。

3.1.4 设置游戏的主角

主角是游戏中的灵魂，只有出色的主角才能让玩家在游戏故事的世界中流连忘返，也才能演绎出动人的故事剧情。因此成功设置出一名与众不同的主角时，游戏就有了成功的把握。

其实在游戏中，主角不一定非得是一名善良、优秀的人物不可，它可以大奸大恶，或者是忠奸难辨之间的角色。

通常，邪恶的主角会比善良的主角更容易使游戏成功，更直接一点来说，如果游戏中

的主角能够邪恶到让玩家厌恨他，却又不能放弃他，存在强烈的爱恨交织；如此一来，玩家们会更想看看这个主角到底能够做出什么，在游戏中会有何奇遇，或者是主角在游戏中的下场如何，这种亦正亦邪的人物反而比乖巧善良的主人翁更容易抓住玩家的心。

还有一点要注意的是，当我们在设计主角的时候，不需要将它太偶像化、刻板化，不可流俗，尤其千万不要将主角设置的太“大众化”。主角如果没有自己独特鲜明的个性与形象，那么玩家们很可能因此而感到平淡无奇、兴致索然。

3.1.5 游戏的描述角度

在游戏中，我们可以将最常用的描述角度（视角）分成两种，分别为“第一人称视角”和“第三人称视角”。

第一人称视角

第一人称视角是以游戏主角的亲身经历所描述的角度，通常在游戏屏幕中不出现主角的身影，让玩家感觉到他们就是游戏中的主角，因此玩家较容易投入到游戏的意境中。

第三人称视角

第三人称视角是以一个旁观者的角度来观看游戏的发展。虽然玩家们所扮演的角色是一个“旁观者”，但是就玩家投入感的角度来说，第三人称视角的游戏不会比第一人称视角游戏来的逊色。

从上述两种叙述角度来看，第一人称视角与第三人称视角的游戏都是较容易被玩家们所接受的。不过第一人称视角的游戏在游戏脚本编写上却比第三人称视角的难度还要大。以欧美的国家来说，他们所制作的 RPG 游戏喜欢以第一人称视角来进行游戏的故事剧情，例如“魔法门”系列。

以笔者而言，是比较偏好第三人称视角的游戏（因为笔者在第一人称视角的游戏中常会晕头转向）。在第三人称视角的游戏中，我们也可以利用各种不同的办法来加强玩家对于游戏的投入感，例如主角的名字可由玩家自行命名，或是自行挑选主角的脸谱等等。而从游戏的表现效果来看，其实第一人称视角的游戏是有局限性的。

3.1.6 游戏的情感与悬念

游戏中所要表达的“情感”因素非常重要，因为只有贴近人类最原始的本性，才足以触动人心，使玩家更能够沉醉在游戏中。对于一个游戏开发者而言，应该要先能够打动自己的情感，如此一来才能够算是游戏成功的第一步。

游戏中另外的一个重要因子就是“悬念”。“悬念”可为游戏带来紧张和不确定性的因素，目的是让玩家无法轻易地猜出下一步将要发生什么事情，加入适当的悬念可以使得游戏更加扑朔迷离。“悬念”的因素就好比在一个门后面放着一些玩家所需要的道具或物品，但在门上有几个必须要打开的机关，玩家在错误打开机关的同时会引起爆炸。此时玩家不知道门后面到底放些什么物品，不过可以通过外围的提示，使玩家了解到这些物品会对他有帮助，可是他也知道打开门的同时会发生危险；因此要如何打开门而不会发生危险就成了玩家所要解决的问题。在这样制造悬念的同时，也给玩家制造了一个脑力激荡的难题。

由于玩家在游戏中并不知道游戏的核心是如何来运行，因此玩家对于游戏剧情会有一种忐忑不安的期待。在所有的游戏中，玩家总是通过经验的实现而与不可预测性的事件抗争，如此便提升了游戏对玩家们的那种刺激快感。 从不可预测性的事件来看，我们又可以将游戏分成两种类型，分别为“技能游戏”与“机会游戏”两种。

技能游戏

技能游戏的内部运行机制是确定的，而不可预测性所产生的原因是游戏设计者故意隐藏了运行机制，玩家们可以通过游戏运行的机制与控制（即为某种技能）来解除这种不可预测性的事件。

机会游戏

机会游戏中的运行机制是相当模糊的。它具有随机的因素，玩家不可能完全通过对游戏机制的了解，而消除不可预测性的事件，而游戏动作所产生的结果也是随机的。

由悬念所引起的期待在游戏中至关重要。在游戏中，我们不能让玩家们的期待完全落空，因为这样会让玩家们产生极大的挫折；另外我们也不能让玩家们的期待完全应验，因为这会让游戏失去不可预测性的意义。简单地说，我们应该有时让玩家们的期待变成一种精确的结果，使得玩家们的信心能够增强，并且从其中获得欢乐感，有时候我们可以抑制住玩家们的期待，使得玩家们对其产生疑惑，而疑惑的时间越长，悬念的情绪就越强烈，建立起来的悬念与紧张程度就会越大。

悬念产生的价值不在其本身，而是在于随之而来的解脱。悬念及其解除的过程，实际上与焦虑、释放过程是相对应的。

3.1.7 游戏的节奏

当我们在制作一款游戏的时候，首先就应该明确地指出游戏与现实生活中时间观念的区别。在游戏中，它的时间是由定时器所控制的，而这种定时器可以将它分成两种，分别为“真实时间”（实时）与“基于事件”的定时器。

真实时间的定时器

真实时间的定时器就是类似 C&C（终极总动员）和 DOOM（毁灭战士）的时间表示方式。

基于事件的定时器

基于事件的定时器指的是回合制游戏与一般 RPG 和 AVG 游戏中所计时的表现方式。有些游戏它们会以轮流的方式来定义这两种定时装置，或者是同时采用这两种定时表现方式。例如“红色警戒”中一些任务关卡的设计。

在实时计时类型的游戏中，游戏的节奏直接由时间来控制，但是对于其他游戏来说，真实时间的作用就不是很明显，所以我们就需要利用其他办法来弥补。

归纳一下市面上游戏的时间节奏特性，我们发现在当红的游戏中，都会尽量让玩家来控制整个游戏的节奏，很少由游戏本身的 AI 来控制。其实专业的游戏设计者要尽量控制游戏的节奏表现方式，使得玩家难以察觉。如同在 AVG 游戏中，我们可以调整玩家活动空间的大小（如 ROOM）、玩家活动范围的大小（如游戏里的世界）、游戏谜题的困难度等等，这些调整都可以令游戏改变本身的节奏。在 ACT 游戏中，我们可以调整敌人的数量、生命值等方法来改变游戏本身的节奏。在 RPG 游戏中，除了可以采用与 AVG 游戏中类似的手法外，还可以调整事件的发生频率、游戏中敌人的强度等等。

一般而言，游戏的节奏会因为越接近游戏的结尾部分而越来越快，如此一来，玩家就会越感觉到自己正逐渐加快步伐接近游戏的真正尾声。另外，绝对不要让游戏拖泥带水，这会让游戏显得非常冗长。过于繁琐的进行一个事件的描述，也会使得玩家失去继续进行游戏的兴趣。所以身为游戏设计者要做的就是不断给玩家新的挑战和刺激来满足玩家对于游戏的新鲜感。

3.1.8 游戏的风格要一致

在一款游戏中，应该从头到尾都保持一致的风格，这是非常重要的。游戏风格的一致性包括人物与背景的特性、游戏风格的定位等等。在一般的游戏中，如果不是游戏剧情的特殊需求，就尽量不要让游戏中的人物说出超越当时历史时代的语言，尤其是某些时代特征。

如同上述所说的一样，开始架构一款游戏时，不外乎上述几项基本原则。接着，在利用这些基本原则组织出一套单纯的草图之后，便可以真正来规划游戏的表达方式与呈现的模式了。

3.2 将电影哲学应用到游戏中

从近几年流行的游戏来看，有许多名噪一时的游戏都会将电影的拍摄手法运用到游戏中，使得游戏的品质与节奏感犹如电影一般，让玩家感觉相当过瘾。在本节里就特别介绍这种足以刺激玩家视觉与感觉的电影成像语言。

3.2.1 铁的法则

何谓“铁的法则”呢？其实简单地说，就是在电影中，摄影机的位置与角度在移动的时候，它不能跨越两种物体轴线的法则，如图 3.1 所示。

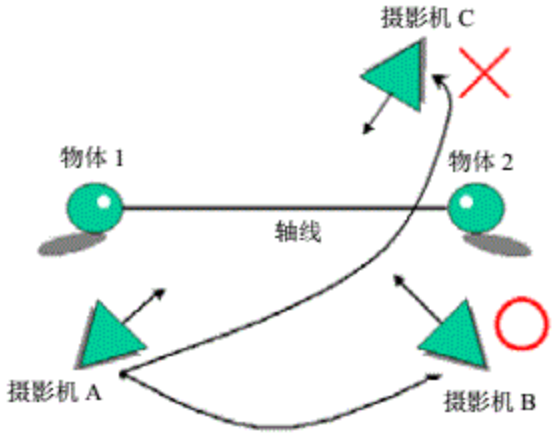


图 3.1　摄影机的位置与角度的移动关系

当摄影机在拍摄两个物体的时候，例如两个面对面说话的人，这两个物体之间的联机就称之为“轴线”。当摄影机在“A”处先拍摄“物体 2”之后，下一个镜头，我们应该要在“B”处拍摄“物体 1”，目的就是要让观众感觉物体在屏幕上的方向是相对的，如此在镜头剪辑后再播放，也不会造成观众对于方向上的混淆。但是如果将摄影机在“A”处拍摄完“物体 2”之后，再于“C”处拍摄“物体 1”的话，那么这就会发生人物好像在屏幕上瞬间移动一样，这会造成观众对于方向上的混乱，并且也是电影拍摄时的大忌。用简单的一句话来说，那就是摄影机在拍摄时，严禁跨越轴线来拍摄物体。

其实严格说起来，倒也不是规定不能跨越这条轴线，只要将摄影机的移动过程让观众能够看见，而且不要将绕行的过程剪辑掉，那么观众便可以自行去调整他们自己的方位视觉。例如以上的这些手法，我们就可以运用在一般游戏的过场动画制作上。

3.2.2 电影中的对话

“对话”在电影艺术中占据了非常重要的地位，为了要突显出游戏中每一个人的性格与特点，那么势必要在游戏中确立每一个人说话的风格。“对话”呈现主角性格与特点的最佳方法。在游戏中，对话不要显得太单调呆板，应该加以夸大，而且必要时增添上一些幽默的成分；毕竟游戏是一项娱乐产品，最终目的是为了让玩家可以在游戏中得到最大的享受和放松。如果游戏的题材不是非常严肃的剧情时，不妨可以适当的放松对话的尺度和口吻，甚至于不必完全拘泥于时代的背景与题材的限制。

“对话”是任何表演艺术中非常关键性的枢纽，无论是在戏剧、电影或是游戏中，生动活泼的人物对话脚本绝对是非常重要的！

3.2.3 游戏中剪辑的应用

对于从事影视创作的人员来说，他们非常喜欢在游戏中利用剪辑的手法来衔接游戏中的各个场景。其实在游戏中，除了特殊的需要之外，剪辑的手法很少应用到实际的游戏制作中。因为游戏总是跟着主角的遭遇来发展的，所以很少会有多线并行的情况发生。不过对于交代剧情和展示全局的情况，剪辑是个不错的选择。例如主角为了与被挡在门外的另一个角色对话时，这时就可以利用剪辑的手法来表现。

3.2.4 视点在游戏中的应用

如同游戏一样，电影手法中也有第一人称视觉和第三人称视觉的观点；惟一不同之处是在同一款游戏中，千万不可做视点之间的切换（一会儿用第一人称视点，一会儿用第三人称视点），因为会导致玩家对于游戏的困惑和视觉的混淆。

如果要使用这种切换视点的方式时，除非是在游戏中的过关演示动画或游戏中交代剧情的动画里，才可适当使用这种不同视点的切换，否则还是尽可能不要去使用它。

3.3 游戏剧本规划与设计

接下来，我们开始进入较为进阶的游戏设计中。在本节里，将会归纳出一些游戏规划的基本细项，以及必须留意的重点。

3.3.1 游戏的类型

首先定义出游戏的玩法类型，游戏的玩法类型如下：

实时战略

游戏是实时计时的，其类型是主角与敌人同时进行运作，不会因为主角的停滞而停止游戏的流程。

DOOM 类游戏

属于第一人称的射击游戏，游戏的娱乐性是取决于游戏给玩家杀敌所带来的真实快感。

RPG

RPG 即为角色扮演游戏，玩家可能扮演一个特定的角色，以这个角色的冒险故事为主轴而进行游戏的流程运作。

AVG

属于冒险类的游戏。此类型游戏是让玩家为了某些原因而进行故事中的剧情发展，目的是带给玩家在游戏中解谜的乐趣。

混合类型

融合若干游戏类型的游戏，如“虚拟人生”系列。在这里，我们先列举这 5 类游戏类型，如果您想看更详细的游戏类型，您可以参阅本书的第 7 章，在这里就不多作介绍了。

在定义出游戏的类型之后，接下来便可以开始设计游戏中的一些小细节了。

3.3.2 游戏设计的一些诀窍

在开始设计一款游戏的细节设置时，应遵循下列几项原则：

- 定时器的作用。
- 游戏中接口的设计。
- 游戏中的真实与虚构。
- 游戏中道具的设计。

- 游戏设计的误区。
- 游戏的交互性与非线性。
- 游戏中的奖励和隐藏。
- 游戏中的死亡。
- 游戏中的对话。

接下来，我们再深入解析上述这几项游戏设计规划的基本原则。

定时器的作用

如同前面所说的一样，在游戏中，定时器的作用是给玩家一个相对的时间概念，使得游戏的后续发展有了一个参考的时间系统。在设计游戏时可以将这两种定时器混合使用，但是关键是不能让玩家对游戏中的时间坐标有所混淆。

游戏中接口的设计

在游戏中，玩家所操作的接口可以设计成简单而明了的文字或图标，尽量采用图像或符号的接口来表达指令的输入，减少采用单调且呆板的文字菜单方式，如果非要使用文字的话，也不一定要使用菜单式，我们可以使用更新潮的图形文字来表达。

游戏中的真实与虚构

当玩家在玩游戏时，主要是可以体验到不同的生活与心路历程，并且从游戏中得到一些心灵上的解放。所以游戏中物质世界的建构可以是虚构的，而人物与感情等因素则不妨是真实的。简单地说，也就是本质核心要与真实生活相当接近，不过游戏题材的选择可以是各式各样的。

游戏中道具的设计

道具的设计必须注意它在游戏中的合理性，如同我们不可能将一辆大卡车装到自己的口袋中一样。另外在设计道具的时候，也要想到道具的完整性，如同在游戏中，玩家需要将一根蜡烛点亮，那么他就需要一个可以点火的工具。所以道具的设计也就成为游戏设计者必须认真思考的问题。例如现实生活中，我们可以利用火把来将蜡烛点亮，那么在游戏中也应该允许玩家能够使用火把来点亮蜡烛；当然也可以不让玩家利用火把来点亮蜡烛，如果玩家利用火把来点亮蜡烛的话，我们可以在游戏中显示“火把的火太大，可能会把整支蜡烛给熔化掉”等信息。从上面所举的例子来看，我们可以让玩家完全遵从游戏设计的方向来进行游戏，也可以让玩家自行去发展道具设计的奥妙之处。 笔者在这里有一个小小

的建议，那就是游戏中的任务是要尽量去帮助玩家，而不是百般地刁难，不然游戏的耐玩度一定会大大受到影响。

游戏设计的误区

在游戏中，可发现两种设计误区，那就是“死路”和“游荡”的误区。

死路

“死路”指的是玩家们在游戏进行到一定的程度后，突然发现没有可以继续进行下去的线索与场景，这种情况也可以将它称为“游戏的当机”。通常会出现这种情况是因为游戏设计者没有做到整体游戏评估的全面性：他没有将所有游戏中可能会发生的流程全部计算出来，这时玩家如果没有按照游戏设计者所规定好的路线前进时，就很容易造成游戏过程中的“死路”。

游荡

“游荡”指的是玩家在广阔地图上任意移动时，一直无法发现游戏下一步发展的线索和途径，这种情况也可以将它称为“卡关”，虽然这种现象在表现上与“死路”很类似，但是它们两者却有不同的本质。要解决“游荡”的方法就是在故事发展到一定程度的时候，我们可以将地图的范围缩小，让玩家可以到达的地方减少，或者是让游戏的线索再明显地增加，让玩家可以得到更多过关的提示，让玩家可以轻松地找到故事发展的目标。

游戏的交互性与非线性

游戏的交互性指的是游戏对于玩家在游戏中所做的动作或选择上有某些特定的反应。举例来说，当主角来到一个村落中，而村落里没有人认识他，因而拒主角于千里之外，但是当主角解决了村落居民所遇到的难题之后，主角便在村落中名声大噪，因此主角可以由村落居民的口中得到下一步任务的进行。再举一个简单的例子，游戏中有一个非常吝啬的有钱人，这个有钱人平常就不太理会主角，但在一个机缘之下，主角救了这个有钱人，尔后有钱人遇到主角时，态度则 180 度的大转变。诸如此类，游戏中主角与其他人物的交互性与彼此间一连串的作为会经常影响到游戏的进行和结局。

游戏的非线性指的是游戏应该是一种开放的结构，而不是单纯的单线或多线制。简单地说，游戏的结构应该是属于网状型的，而不是线状或是树状，所以非线性即是将游戏中的分支交点允许互相跳转，如图 3.2 所示。

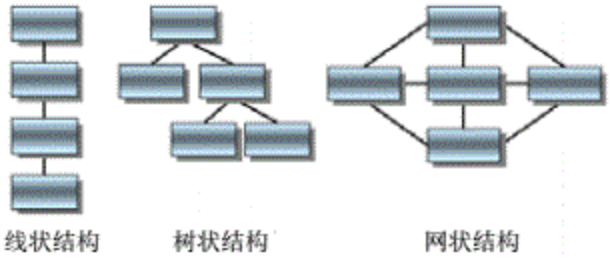


图 3.2　线状结构、树状结构与网状结构

游戏中的奖励和隐藏

游戏进行到一定程度时，可以给玩家一些游戏中的“奖励”，例如精彩的过场动画、华丽的画面呈现，甚至让玩家可以在游戏中得到一些有用的道具等等。在游戏中，与奖励十分类似的因素是“隐藏”的设计，例如隐藏关卡、隐藏人物、过关密码等等。这些无厘头的设计经常别出心裁，这也是现在游戏非常喜欢加入的原因，但是请您注意的是这些设计不可影响到游戏的正常运作，毕竟这些设计只是一个噱头罢了。

游戏中的死亡

通常我们将游戏中主角死亡的原因分成两种，一种是因目的而死亡，另一种是真正的结束。

因目的而死亡

这是一种剧情的需要，例如当主角被敌人打死（其实是受到重伤而已），又被一个世外高人所救，并且从这个高人身上学到一些更厉害的招式之后，再出来到江湖上闯荡。

真正的结束

这种死亡就是所谓的 Game Over，它是让玩家所操作的主角面临真正的死亡。一般而言，玩家必须重新开始或读取进度才能继续进行游戏的冒险。

游戏中的对话

笔者建议游戏中的对话不要过于单调重复。一般而言，一款游戏中至少要出现 50 句以上常用且饶富趣味的对话，而且它们之间又可以互相组合；如此一来，才不会让对话过于单调无聊。还有要尽量避免太过于简单的字句出现，如“您好!”、“今天天气很好!”等等，要尽量做到与目前游戏中的情节相关的话题，增加游戏的真实性。

游戏中的对话可分为下列 3 种类型：

无对话的游戏

如“DOOM”游戏。

特定对话的游戏

如“暗黑破坏神”。

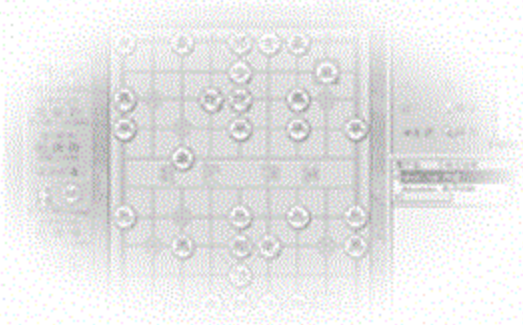
自然对话游戏

对话的设计是带有情绪性与选择性的，而玩家可以做出他想选择的对话内容。如果玩家选择其中一个对话的话，那么其他的话题就会消失，而以后的话题就会依据他第一次所选择的对话而引发。

通常游戏设计者都是从玩家的角色成长起来的，这些人对于游戏制作充满了热情，但是他们却没有丰富的经验，再加上游戏理论的不足，因而导致在游戏设计的领域中似懂非懂的困境。实际上，笔者认为游戏设计是一门非常特殊的艺术，想要掌控它不仅要有创造力、文学能力，甚至对于电影、戏剧、历史、文化等等都要有一定程度的认识，最好是还要对程序设计与美术有相当认识，这样才可能成为一名真正专业的游戏设计人员。

第 4 章

游戏与玩家互动



游戏的趣味性，最主要的基本原理就是游戏与玩家之间的互动程度。例如一套游戏的进行流程非常紧凑，让玩家一直保持紧张的心情，而有身历其境、大呼过瘾的临场感，游戏也就成功了一半。本章将介绍游戏与玩家增加互动性的技巧，让一套游戏可以淋漓尽致地表达其中的意境与趣味性给玩家分享。

4.1 游戏所营造的气氛

制作游戏时，为了让玩家的心情与感觉可以完全融入到游戏中，我们必须在游戏中营造出各式各样的气氛，这些气氛的感染效果，可以利用玩家的 3 种感觉方式来表达，分别为“视觉”、“听觉”及“触觉”，这 3 种感觉的表达方式足以影响玩家对于游戏的整体感观。

4.1.1 游戏的画面与剧情

以早期的游戏来说，很少会以恐怖为题材来制作游戏。因为以早期的成像标准而言，它们只能够在游戏机制上取得优势。随着科技的进步，现在的游戏平台已经可以超越早期游戏的成像标准，所以现在的游戏开发者都将游戏尽量朝向游戏画面营造的气氛与诡异性剧情的方向安排。

4.1.2 游戏画面的气氛

以游戏画面而言，在早期的两人格斗游戏中，我们可以看到两个简单的人和单纯的背景画面。在这种游戏刚出现的时候，玩家被这一种特殊的玩法给深深打动，这种两人“互殴”的游戏带给玩家一份博击的刺激感。不过玩家对于这种游戏的持续热度却保持不久，因为玩家开始厌倦了这种单调的画面，以单纯的打斗来说，这种游戏不能表现出“力”的感觉，因此让玩家对于这种游戏的热度快速下降。

从现在的格斗游戏来看，虽然玩法与机制只做了一点点的改变，不过它们却在游戏画面上增强了华丽度与足以改变玩家心情与情绪的一种特征，这种特征就是在游戏画面中，加入了许多不一定要存在的因子，这些因子就如同在“铁拳”的游戏中，那些站在主角与计算机周围观看格斗的人，虽然这些人对于主角是否可以取胜是完全搭不上关系，但由于它们的出现，使得玩家就好像自己在游戏中打斗一样。简单地说，这种呈现的气氛更能够让玩家有身历其境般的兴奋与真实感。

另外一种能够让玩家容易融入到游戏中的因素是游戏中的“剧情”，而且如果在剧情中加入少许的诡异安排，那就更精彩了。例如，如果是如下一种简单地叙述：

A君面对着B君。
A君说：“听说树林里出现了一些可怕的怪物。”
B君说：“嗯！”
A君说：“这些可怕的怪物好像会吃人。”

上述的对话中，我们很难断定这种对话的情境到底是“悲”的成分还是“忧”的成分比较重。既然不能判断它的情境，那更不用说影响游戏玩家的心情。不过，如果我们将上述的对话修改成：

A君背上背着一把短弓，腰上带着一把生锈的短刀，面色凝重地面向着B君。
A君以微微颤抖的双唇道出：“前几天，我的兄长到村外不远的树林里打猎，可是他这一去就去了好几天，不知道会不会发生什么危险。”
B君道：“村外的树林？您的兄长？唉呀！会不会被怪物抓走了啊！”
A君脸色大变地道：“怪物？村外的树林里有怪物？”

从上面这两个简单的对话中来看，两者的情绪差距相当大；第 2 个对话的例子很容易把玩家带进游戏的情境中，而且会让玩家更加投入游戏中的故事情节。在游戏中，这种文词修饰的诱人气氛，就如同在看电影一样，只要是电影没有到最后尾声，都会有一股迫不急待地想看下去的欲望，所以像这种表现在游戏上的气氛制造可称得上是另类电影剧情的呈现技巧。

4.1.3 视觉感受

从电影学的角度来说，就是一种以视觉的感受来触动人心，使其受到电影中的情节所影响而产生共鸣。例如，当您去看恐怖片的时候，心里会有一种毛骨悚然的感觉！在看温馨感人的文艺片时，心底泛起一股淡淡的暖流！在看无厘头的喜剧片时，心情得到充分减压，遇到搞笑处，不免也手舞足蹈起来。其实从医学的角度来看，眼睛是灵魂之窗，大脑里所接收到外界信息都是来自眼睛的传达。简单地说，可以影响情绪“喜、怒、哀、乐”最直接的方法就是利用视觉的感受来传达。同样道理，在游戏中直接影响最深的就是视觉的感受。

一般而言，如果在游戏中看到以暗沉色系为主的题材时，相信您一定会被这一股莫名的压力给压制住，而游戏所要表达的意境也就是这种阴深、恐怖的情景；如果在游戏中看到以鲜艳色系为主的题材时，游戏所要表达的意境也就是比较活泼、可爱的情景。

4.1.4 听觉感受

除了眼睛之外，第 2 种影响玩家对外界的感观是来自耳朵。耳朵是人类一种接收声波的器官，所以当我们在接收到声音的时候，大脑会去分析并且解释它的定义，然后再通知

身体的每一个部分，并且适时地做出适当的反应。例如，如果一个人将鞭炮声定义成可怕的声音，那么当听到鞭炮声时，他的大脑一定会通知他的手去捂住耳朵，然后再将身体缩成一团，并且等待鞭炮声消失为止。

在游戏的表现上，我们也可以利用声音来强化游戏的品质与玩家们的感受，在现在要求游戏品质相当严谨的时代里，声音已经是不可或缺的角色，虽然它不能成为直接影响玩家情绪气氛的因素，不过它却是加强这种情绪气氛最重要的原因。例如，如果您在玩一种以跳舞为主的游戏时，您只能看到屏幕上那些上下左右的箭头一直在往上跑，但是却不能听到任何的声音，这时，您只能看着那些箭头猛踩踏板，而不能跟着音乐的节拍起舞，那么游戏玩起来不就显得无聊了许多呢？

4.1.5 触觉感受

当我们能够从游戏中接收到视觉与听觉的时候，接下来就是接收触觉了。什么是触觉？其实它不是我们一般所认定身体上的感受，而是一种综合视觉与听觉之后的感受。那什么是视觉与听觉的感受呢？答案很简单，就是一种认知感，当我们从眼睛与耳朵接收到信息的感受之后，大脑就会开始运作，以自己所了解到的知识与理论来评论游戏所带来的感觉，而这种感觉就是一股对于游戏的认知感。

从玩家们对于游戏的认知感来看，一款游戏如果不能表现出华丽的画面、丰富的剧情，或者是“力”与“美”的表现，这会让玩家对游戏开始感到厌恶。如同一款赛车游戏来说，如果游戏不能表现出赛车的速度感，以及物理的真实感（撞车、翻车），纵然游戏画面再怎么华丽、音乐音效再怎么悦耳，玩家还是不能从游戏中感受到赛车游戏所带来的快感与刺激，那么这一款游戏很快便会无疾而终了，所以触觉的感受可以说是视觉与听觉的综合体，它们之间的关系可用图 4.1 表示。

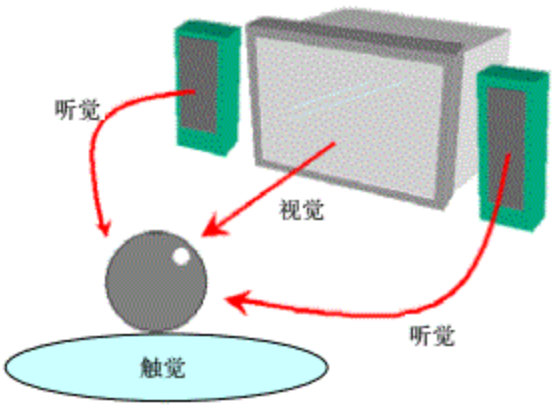


图 4.1　触觉、视觉、听觉三者之间的关系

当您了解到一款游戏要如何来传达信息给玩家之后，接下来便可以开始来表现游戏中所包含的内容。

4.2 游戏剧情的表现

当我们将各项游戏信息传递给玩家们了解后，接着来说明游戏的重心——“剧情”。从现今市场上游戏的评价来看，我们可以将它分成两派，一派是无剧情的刺激性游戏，另一派是有剧情的感观性游戏。

无剧情的游戏着重于游戏带给玩家的临场刺激感，如“战栗时空”。这种游戏的主要目的是让玩家自行去创造故事的发展。在游戏中，它只告诉玩家主角所在的时空与背景，而游戏的流程运作则是要玩家自己去闯荡，在“战栗时空”的游戏中，玩家所扮演的角色是一名拿着枪的人物，并且协同朋友一起去攻打另外一支队伍，而在这种攻打另一支队伍的同时，它也创造出另一个“故事”。

有剧情的游戏着重于游戏带给玩家剧情的触觉感，主要目的是要让玩家随着游戏所编排的故事剧情来运行。在游戏中，它会先让玩家了解到所有的背景、时空、人物、事情等要素，而玩家必须按照游戏剧情的排列顺序来进行发展，如同一般的角色扮演游戏。玩家会扮演故事中的一名主角，而游戏中的剧情发展都是环绕着这名主角所发生，所以有剧情的游戏也就是让“故事”来领导玩家。

4.2.1 剧情的阻碍

所谓的“剧情阻碍”指的是在游戏中，有些事件是玩家必须去解决的问题，而这些问题也就成了玩家继续游戏的阻碍。

通常这些剧情的阻碍是由人、事、物所发生的，例如玩家可能会因为要去救某一个人而去执行任务，或者是为了达到某一件事而去执行任务，或者是为了收集物品而去执行任务。如同在 4.1.1 节里所说的一样，我们可以在阻碍中加入一些诡异的安排。例如玩家为了解救某一个人而去收集某一种物品，而为了得到某一种物品，玩家可能又要实现某一些事情。以反推的方式可以得知，玩家为了要解救某一个人，他就必须要去完成某一件事情，诸如此类，诡异的剧情在游戏中可以增加剧情的丰富性。

4.2.2 预知的剧情

一般而言，预知的剧情通常是被放置在游戏的最初期阶段，是用来交待剧情所用，最主要是要告诉玩家接下来游戏中的目的。以“巴冷公主”来说，游戏画面一开始，玩家会

看到“巴冷”与“阿达里欧”在溪边相遇的情景，正当巴冷要与阿达里欧面对面接触的时候，阿达里欧则化做一团烟雾，并且消失在空气中。顿时之间，巴冷从床上醒来，并且发现刚才的画面原来是一场梦，而这个梦便展开了巴冷与阿达里欧之后的冒险过程。在上述例子中，我们可以看到游戏的预知性，当巴冷在游戏中冒险，巧遇阿达里欧、卡多、依莎莱等伙伴，并且剧情一直让巴冷环绕在阿达里欧的生活当中，最后两个人相爱，这些剧情都已经在游戏画面一开始的时候，我们就已经知道了。不过这种预知性的表现却会让玩家强烈想要更深入了解巴冷与阿达里欧之间是如何相爱、生死与共的故事，因此便可以创造出游戏的延续性，并且让玩家会有继续想看完游戏故事剧情的决心。

4.2.3 情节转移

情节的转移便是将游戏的故事剧情转向，目的是要让玩家可以朝向另外一个全新的方向来进行游戏。如同史克威尔公司所推出的“太空战士 10”游戏中所看到的一样。故事一开始主角会一直环绕在女主角“召唤师”身份的剧情中，让玩家感觉到主角是为了保护女主角而参与故事中的所有任务，直到游戏末期的时候，男主角的角色就渐渐地被突显了出来。原来故事的前因后果都是以男主角为主轴，而女主角的故事只是包含在其中的一小部分而已。这种情节转移的手法，不但让玩家可以感受到除了女主角之外的故事剧情，还可以感受到另一种崭新的故事风貌，并且将前面所发生过的事件内容变得更加合理化。

4.2.4 悬念的安排

我们经常在游戏中看到一种令人觉得很差劲的安排。那就是在游戏中，主角可能会遇上一个打不死的敌人，而在游戏故事里也没有特别交待，使得玩家所操纵的主角很容易就被这种敌人给打死。

其实在游戏中，这种打不死且能力很强的敌人是有存在的必要。因为至少在游戏初期，它可以更强化玩家对于在游戏冒险的决心，不过这种敌人的出现至少要有非常合理的交待，才不会让玩家有一种突如其来的错愕感。举个简单的例子，如果要描述游戏中某种敌人有很强的能力时，我们可以将它的特性给突显出来，由周边的一些物品或人物来衬托出它是一个打不死的怪物。例如在游戏中，主角在一个洞穴里看到满地的骨骸、尸体、或者是在两旁的墙壁上，有许多人被不知名的液体封死在上面，以这种简单的手法让玩家进入到洞穴中的同时，感受到其中的恐惧感与紧张的气氛。

诸如上述的剧情安排，我们可以将它称为“悬念”。这种“悬念”的气氛可以带给玩家一种精神紧绷的刺激感，而游戏中的剧情也就更容易潜移默化地融入玩家的心。

4.2.5 主题的安排

我们都知道，一款游戏的建立是由它的主题延伸拓展而来的，主题也就是贯穿游戏中的整体架构，而且设计出来的游戏主题也可以从玩家的角度演化出许多变化。

以玩家的观点来看，一款游戏的故事剧情被想象成如何是不可预知的，我们只能够以自己的角度来制作编写游戏的故事，而故事发展的内容是否精彩就必须取决于玩家的想象力了。对于一款游戏来说，最差劲的做法就是直接了当地告诉玩家故事的内容。 例如要描述一款游戏中的男女主角之间的情感，玩家所要看的并不是男女主角在游戏中爱得有多轰轰烈烈，而是男女主角在游戏中如何来相爱。如同“太空战士10”中的故事，男主角与女主角在第一次相遇时，虽然他们俩人彼此都有好感，但基于族群的使命安排下，俩个人只能默默地对彼此示爱。故事一直发展到“大召唤师”向女主角示爱之后，男主角才发觉他对女主角有了一股不可抹灭的爱情，而为了阻止女主角与“大召唤师”的连理，遂与“大召唤师”进行一场决斗。在这种故事主题的安排下，我们可以发觉它让玩家有了很大的想象空间，虽然玩家都知道游戏中的男女主角必定是相爱的，不过玩家还是最喜欢以自己的想象力去摸索游戏中故事的发展，而游戏主题的安排只是带领玩家去进行故事的流程而已。

以游戏的剧情描述手法而言，它存在着许多不为人知的暗喻。我们发现美丽且有悬念故事的剧情是最吸引玩家的，美丽的故事让玩家可以感触到游戏中爱恨情仇的情绪变化；悬念的剧情让玩家可以感触到游戏中曲折迷离的紧绷心情。

4.3 游戏的环境界面

对于一款游戏来说，最直接与玩家沟通的画面则是游戏中的环境接口。接口的主要功能就是让玩家可以使用游戏所提供的命令或者游戏信息而已。不过一个游戏接口的好坏是可以直接影响到玩家玩游戏的心情，因此游戏接口的设置上，身为一个专业游戏设计者，似乎也不该掉以轻心。

其实笔者在着手编写这一节的时候，也犹豫了许久，心中疑惑着：“界面？界面有什么好介绍的呢？界面不就是一个游戏中的菜单而已吗？”，其实再仔细地回想一下，从较早期的游戏中，有时候会为了接口（即游戏指令或功能）的功能大思不解，为什么我们需要的功能，游戏却没有提供呢？这时通常我们这些弱势的玩家只能以设计者没有想到这种游戏功能来自己安慰一番，不过以现在刁钻古怪的玩家来说，这种游戏早被他们打入黑名单了；原因就是接口不够人性化、指令不够用，而对游戏的期待度自然大打折扣。

从以上的介绍来看，游戏的界面是可能会直接影响到玩家的心情，就好比我们要与外国人对谈一样，如果没有一个良好的沟通管道（翻译）的话，那么我们就有可能会打退堂

鼓，干脆不要与外国人沟通好了。同理而言，游戏的机制也是如此，如果没有良好的沟通接口，那么玩家可能不会用尽心思去玩这类的游戏，甚至连玩都不想玩。

总归一句话，游戏的好玩与否，是可以由游戏的界面来牵引调整的。在这里，笔者分门别类归纳几项足以影响游戏界面品质好坏的重要因素，请您再继续往下看吧！

4.3.1 容易被干扰的界面

一款游戏最害怕的是它的环境界面会去干扰到玩家所操控的平台。例如，如果一套游戏的环境界面是采用实时框架来呈现时，而这种环境界面的框架又经常会挡住玩家主角的操作（见图 4.2），虽然实时对话框的构思很不错，不过如果我们没有善加利用空间来配置环境接口的位置时，玩家所操作的游戏主角就会时常因为被环境界面挡到而被敌人打得半死，如图 4.3 所示。

图 4.2　人物被对话框挡住了

图 4.3　对话框配置在画面的下方

如同这种做法，一般的游戏就很容易犯下这种错误，它不但对游戏的故事剧情没有帮助，而且它还会导致玩家非常反感。

4.3.2 人性化界面

以界面的功能来说，它是一种介于游戏与玩家之间的沟通管道，所以如果它的人性化考虑越多的话，玩家在使用起来就会越容易与游戏沟通。举例来说，如同现在很热门的实时战略游戏，界面就做得非常地人性化。就以游戏中的人物来说，当我们去选择敌方的部队时，游戏界面上会出现“攻击”的指令图标；而当我们去选择地图上某一个地方时，游戏界面上则会出现移动的“指令”图标。在游戏中不会看到一堆无用的指令，使得整个画面看起来相当的干净与简单。

在此我们要特别强调的就是这种非常贴近玩家的人性化设计。就以笔者的观察来看，玩家通常是不喜欢看游戏说明书的，甚至有的游戏还大力地推广它们游戏说明书多达好几百页，甚至还很骄傲的形容，这是这款游戏的特色之一（通常游戏盒子也是大而无当）。这种言语乍听之下，似乎很动人，不过实际上能将这种说明书看完的人，可就寥寥无几了。就算能看完这么一大本说明书的人，相信原本对于这款游戏的满腔热情已经消失无踪了。

笔者认为一款口碑良好的游戏是不会强调游戏说明书上的功能众多与游戏指令多如牛毛；反而会要求功能简单有力，一个按钮可以做到很多事情。例如在某些游戏中，当我们选择人类的时候，攻击的按钮就会变成“拳头”，意思就是说，人类是以拳头为主要的功击因子；当我们选择魔法师的时候，攻击的按钮则会变成“魔法”，意思就是说，魔法师是以魔法为主要的功击因子，如此聪明简化的设计便可以让玩家很轻易地上手，并且在游戏中也不会占了大部分的画面。

4.3.3 透明化界面

笔者曾经在“善与恶”（Black & White）这一款游戏中，看到了一种令笔者认可的游戏界面，那就是“无界面的界面”。什么是“无界面的界面”呢？简单地说就是透明化的环境界面。在游戏中，我们看不到任何窗体、按钮或菜单，它是利用鼠标的滑动方式来下达“补助指令”。

什么是“补助指令”呢？其实“补助指令”就是除了捡拾物品、丢掉物品或选择人物之外的功能指令。例如在“善与恶”的游戏中，如果要换牵引圣兽的绳子时，只要利用鼠标在空地上画出我们所要的绳子指令就可以更换圣兽上的绳子了，如图 4.4 所示。

我们也可以在游戏中利用这种方式来施展魔法或生产游戏所需要的食物与木材，这让以往纯粹利用单纯按钮来控制的玩家们，感受到游戏设计这份别出心裁的体贴。

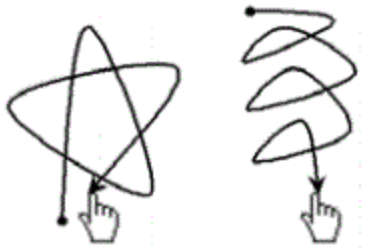


图 4.4　更换圣兽上的绳子

4.3.4 输入装置的搭配

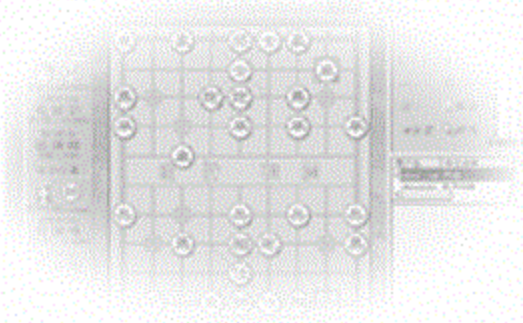
从早期的游戏看来，我们不难发现游戏主要的输入工具，不是键盘就是鼠标，甚至有些游戏会做到鼠标是一种控制模式，再加入键盘为另一种控制模式，而且这两者都互不相关。以一个单纯的玩家而言，这么复杂的输入环境不但令玩家非常的困扰，而且键盘的搭配又不容易记住，导致一套游戏非要这么多按钮才可以玩，这种情况就很容易在某些模拟类的游戏中出现。例如某一种赛车类型的游戏，当我们按“上”键，车子会执行加油前进的动作、按“下”键，车子会执行刹车的动作，而换档则是“1”、“2”、“3”、“4”及“5”键；切换第一人称视角则为“F1”键；切换第三人称视角为“F2”键等等复杂的组合键搞得玩家晕头转向。笔者曾经还玩过一种 3D 第三人称的游戏，其人物的移动控制键分别为“上、下、左、右”、手攻击键为“A 键”、脚攻击键为“S 键”、跳跃为“空格键”，虽然很简单，不过它的左右控制键只是控制人物的左右平移，如果要执行转身的动作，就要使用鼠标来控制。如果没有遇到敌人还好，但一遇到敌人的时候，两只手便开始迅速地来回在鼠标与键盘之间移动，不用说打敌人了，就连主角要逃跑都来不及了，甚至更令笔者心灰意冷的是，竟然在游戏中找不到任何可以改变控制键的选项。就算是一个电子高手，可能也没有办法控制得很流畅（不信您可以试试一方面要按“A 键”打敌人，一方面要移动主角，而且还要使用鼠标来转向）。诸如上述的例子，我们可以发现虽然键盘可以下达许多不同的指令，但是对于一款游戏而言，这种不明智的方式最好是不要使用，因为这会让玩家感到手足无措，完全摸不着游戏的方向。总结一句话，如果没有了良好贴心的输入控制机制，就算游戏有再华丽的画面、故事题材再怎么动人可能都会功亏一篑。

游戏的环境接口算是游戏与玩家执行沟通的惟一工具，纵然有华丽的画面、丰富的故事和剧情，可是不要忘了，这些美丽的红花也要有贴心的绿叶(控制机制)来搭衬，这才算得上是一款好游戏。

游戏的互动性可以直接影响到玩家们对于游戏品质的评判，只有集合丰富的题材、动人的故事、紧凑的剧情、悬念的内容，以及视觉上、听觉上所能够感染玩家的气氛等因素，才能成功地完成一套脍炙人口的好游戏。

第 5 章

游戏设计语言工具



本章将探讨游戏设计过程中可以采用的语言工具，例如 C/C++、Visual Basic、Java 甚至 ActionScript 等。探讨不同程序语言工具所设计出来的游戏的优点与缺点，以及可用的工具。例如使用 C/C++程序语言来开发一套游戏时可以使用 API；使用 Visual Basic，开发上也较简单，而且易学易用；使用 Java 程序则拥有跨平台功能；使用目前流行的 Flash 结合 Action Script 开发，则有文档小的优点等等。

5.1 程序语言与开发环境

在进行游戏设计之前，首先要解决的第 1 个问题就是应该使用何种程序设计语言工具？而第 2 个问题也随之而来，该使用何种开发环境来编写游戏？

当然如果您只是写一些小游戏，自然是使用自己所熟悉的程序设计语言与工具即可。但如果您尚未接触过程序设计语言，而想在较短的时间内学习一种程序设计语言以进行游戏设计的实际制作，您可能就必须考虑以上两个问题，而如果您要从事的是大中型游戏的开发，又考虑获利的可能性，则使用何种程序语言与何种开发工具，更是左右成本与获利的主要关键。

以目前游戏设计常使用的几种程序语言来说，主要有 C/C++、Java、Visual Basic 几种，您可能还听过 Visual C++、Borland C++ Builder、Borland JBuilder 等。但这些并不是程序语言，它是程序语言在建构程序时的“整合开发工具”（Integrated Develop Environment，简称 IDE）。不过它们在游戏设计的过程中也占有相当重要的地位，因为选用正确的整合开发工具，可以加速程序设计的进行、测试与除错；因而对整体进度具有决定性的影响。 在开发游戏时所选择程序语言时有几个必须考虑的问题；执行平台、执行速度、开发难易度、语言本身的功能性、可使用的外部支持功能与可使用的开发环境。

就执行平台而言，您必须考虑的问题之一是玩家可能使用的操作系统。玩家可能使用的是 Windows、Linux 还是 Macintosh？甚至是 OS2 等其他操作系统？当然游戏本身是商业性的娱乐商品，以目前操作系统占有率来说，Windows 操作系统的占有率最高，因此目前市面上可看到的游戏多以 Windows 系统平台为主，而少数的游戏也会特别考虑到 Linux 操作系统平台的玩家来设计。

由于游戏本身也是个程序，程序就必须依赖操作系统才能执行，因此您无法将 Windows 操作系统上的游戏直接拿来在 Linux 上执行，即使一开始您在设计游戏时已考虑了跨平台的可能性，游戏仍然必须适当地修改与重新编译，制作这类游戏自然也有其必须付出的成本。而另一方面，一些程序语言或工具所制作出来的游戏，其本身就已经绑死在某一个系统平台上，例如 Visual Basic 所编写出来的游戏，就绝对只能在 Windows 操作系统上执行。

有些程序本身可能就具备跨平台的功能，例如 Java 程序语言所编写出来的程序，只要

小心不要使用到一些特殊的外部函数，这些程序确实可以在无需重新编译的情况下执行。然而我们在后面介绍程序语言特性时您就会看到，这类程序有个致命的缺点——“慢”，因为它们必须依赖“虚拟机器”（Virtual Machine）才能执行，等于多了几道手续，因此这些程序语言并不适合用来开发大型游戏。

再就是开发难易度的问题，C/C++是所有程序设计人员公认功能较强大的程序语言，也是执行时具有优良速度表现的程序语言，然而 C/C++功能强大性的另一面，就是使用上较为复杂（对于初学程序的人来说可算是相当复杂）。设计时若不小心将可能导致游戏的执行错误，甚至程序终止或当机的情况发生，使用 C/C++所开发出来的程序，在测试除错时所花费的成本有时并不比开发程序少。

您可能会选择使用 Visual Basic，对于初学者来说是最容易上手的。然而所面临的第 1 个问题也是执行速度缓慢的问题，而且简单的程序语言其功能特性通常就有限，对于大型游戏而言，Visual Basic 的速度与功能就显得不足。

在游戏设计时另一个重要的考虑就是程序语言本身的功能特性，以及可使用的外部支持。C/C++本身提供有标准函数库，且可调用操作系统本身所提供的一些组件功能，例如 DirectX。以 Java 为例，其提供有网络联机的功能，使用它来设计网络联机程序会比使用 C/C++更方便。Visual Basic 的事件处理则是最为直观，初学者可以轻易的掌握事件来设计游戏。

以上的简介引出最后一个问题，那就是整合式开发环境的选择对游戏设计人员绝对有举足轻重的地位。无论是程序语言本身的复杂度、功能性与可使用的外部支持，选用正确的整合式开发环境都可以使得一些问题得到适当的解决。过去常用来开发游戏的是 Visual C++，而近期使用 Borland C++ Builder 来设计游戏也有越来越多的趋势，因为 Borland C++ Builder 使用方便，且由于硬件技术的加强，使用 Borland C++ Builder 所开发出来的程序，在执行的速度上已经与 Visual C++差别不大。

以上是对程序语言与开发工具在游戏设计时的一些建议，在接下来的几个小节中我们将深入探讨几种不同的程序语言，它们的基本运作原理以及应用在游戏设计上的一些考虑、优点与缺点。

5.2 C/C++程序语言

C 语言问世至今已有 30 多个年头了，早期的游戏在编写时多半都以 C 语言搭配汇编语言。C 语言是个程序导向的语言，着重程序设计的逻辑、结构化的语法，而 C++是以 C 语言为基础，改进了一些输出 / 输入的方法、并加入了对象导向的观念，如果要编写大中型游戏的话，使用 C/C++来编写程序是一种不错的选择。

执行平台

C/C++程序语言是高阶程序语言，它的使用较贴近于人类语言的语法，让程序设计人员能以人类思考的方式来编写程序，例如 if、else、for、while 等。以下是一小段 C 语言程序，您可以大略了解它的编写方式：

```
#include <stdio.h>
int main( void )
{
    int int_num;
    printf( "请输入一个数字： " );
    scanf( "%d", &int_num );
    if ( int_num%2 )
        puts( "您输入一个奇数。" );
    else
        puts( "您输入一个偶数。" );
    return 0;
}
```

即使您没有学过 C 语言，从这个程序表面的语意来看，也大致可以知道这个程序的作用，然而计算机无法直接了解 C/C++程序语言所编写出来的程序，所以这个程序必须经过“编译器”（complier）的编译，将这些语法解译为计算机所能看得懂的机器语言。

机器语言是由 0 与 1 交互组合而成的语言，在不同的操作系统平台上，对机器语言的定义也不相同，再加上 C/C++本身所提供的标准函数库有限，所以往往必须调用系统所提供的功能，因此使用 C/C++编写出来的程序，无法直接移植到另一个平台上使用，而必须重新编译；并修改一些无法运行的程序代码。因此使用 C/C++编写出来的一些程序只能局限于单一平台上执行。

然而 C/C++所编写出来的程序有利于调用系统所提供的功能，因为早期一些操作系统本身就以 C/C++程序语言来编写，因此在调用系统功能或组件时最为方便，例如调用 Windows API、DirectX 功能等等。

执行速度

由于 C/C++必须编译才能成为计算机可以理解的机器语言，所以在执行时可直接加载内存，而无需经过中间的转换动作。所以利用 C/C++来编写程序，在速度上会有较优良的表现，不过在追求更高的执行速度时，往往还需要配合汇编语言来编写一些基础程序，尤其是在处理一些图像时。

语言特性与功能

C/C++本身的功能强大，其“指针”（pointer）的特性可以让程序设计人员直接处理内存中的数据，也可以利用指针来实现动态规划的功能。例如内存的配置管理、动态函数的执行。在需要规划数据结构时，C 语言的表现最为出色。在早期内存容量不大时，每一个位的使用都必须珍惜，而 C 语言的指针就可提供这方面的功能。

C++是以 C 为基础，改进一些输入与输出时容易发生错误的地方，保留指针功能与既有的语法，并导入对象导向的概念。对象导向在后来的程序设计领域中变得相当重要；它将现实生活中实体的人、事、物，在程序中具体的以对象来加以表达，这使得程序能够处理更复杂的行为模式。而另一方面，对象导向程序在适当的规划下更能够以编写完成的程序为基础，开发出功能性更复杂的组件，这使得 C++在大型程序的开发上极为有利，目前所看到的大型游戏几乎都是以 C++程序语言来进行开发时。

开发环境

C/C++程序语言的整合开发环境相当的多，商业软件方面有微软的 Visual C++、Borland 的 C++ Builder，非商业软件方面有 Dec C++、KDevelop，这些程序中可以编写 C 语言或 C++，通常商业软件提供的功能较多，使用上也方便许多，对于程序完成后的测试与除错功能也更为完备，如图 5.1 所示为 Dev C++开发环境。

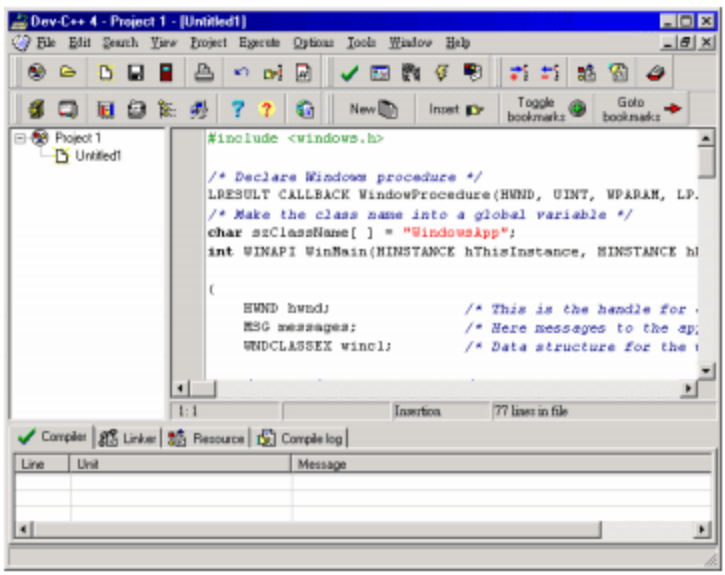


图 5.1　Dev C++开发环境

开发环境与程序语言究竟有什么关系？基本上程序语言在编写时只要有纯文字编辑器就可以进行编写了，而在编译程序时也只需要编译器。然而如果程序发生了错误，您必须自行判断程序错误的所在并加以改正，而所有的组件也必须自行重新打造。整合式开发环境一般会提供上述的辅助功能，让您在开发程序时效率更高。

早期在开发大中型游戏时多使用 Visual C++。使用 Visual C++所提供的组件在早期算是十分方便，至少不用从头编写这些组件。如同 C++ Builder 一样，虽然使用时复杂度较高，但在执行速度上却快了许多。

C++ Builder 在近期逐渐有人拿来开发一些游戏，因为它包装了更多的预设组件，虽然执行速度上略差于 Visual C++，不过这点被日渐增强的计算机执行速度所弥补了。目前常用来作为一些制作游戏的辅助工具，例如设计地图编辑器等，由于本身都是使用 C++语言来编写，因此在组件的功能沟通上并不会发生问题。

5.2.1 Visual Basic 程序语言

Visual Basic 严格来说并不仅是程序语言，它与开发环境紧紧地结合在一起，也就是说您无法只使用纯文字编辑器来编写 Visual Basic 后进行编辑，而必须使用 Visual Basic 开发程序来进行程序的编写，如图 5.2 所示为 Visual Basic 开发程序。

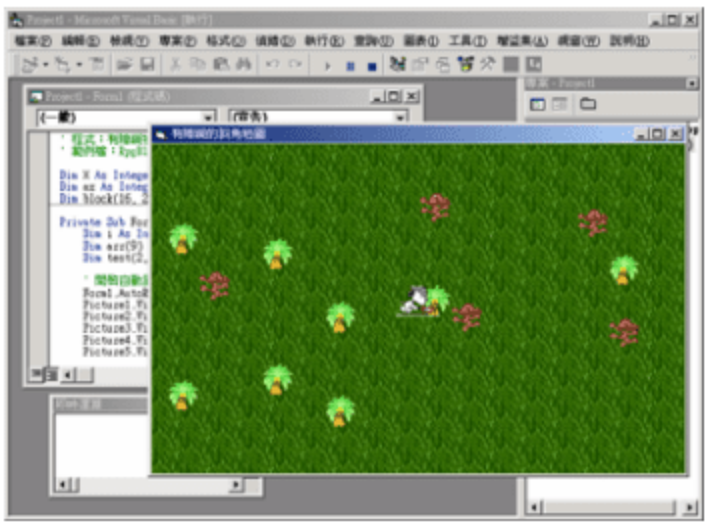


图 5.2　Visual Basic 开发程序

执行平台

Visual Basic 是属于高阶程序语言，它必须经过编译的动作才能被计算机所执行，而且

Visual Basic 与 Windows 操作系统紧紧地结合在一起，它所提供的组件功能都是针对 Windows 操作系统来量身打造。所以 Visual Basic 所开发出来的程序，肯定只能在 Windows 操作系统上执行，另外您必须额外进行安装，将一些 Visual Basic 执行时所需的组件（类似虚拟机器）安装到操作系统中，才能执行 Visual Basic 所编写出来的程序，如果您的计算机没有安装过这类程序时，Visual Basic 程序则无法执行。

Visual Basic 并不是使用 C/C++的语法与关键字，它是微软所提出的程序语言，因为 Visual Basic 与 Windows 是同一家公司的产品，所以 Visual Basic 可以调用 Windows API 与 DirectX 等组件，然而如果您调用了 Windows API，就更要注意平台的相依性问题。因为有些 API 在 Windows 95/98 及 Windows 2000/XP 上会有些不同，也就是说如果您使用了 Windows API，那么您可能就无法跨越 Windows 操作系统以外的平台。

执行速度

Visual Basic 确实是我们所见过最简单易学的程序语言（环境），使用它来设计窗口程序或开发游戏是相当方便的。然而方便的背后就隐藏了更多的执行细节，包装了更多的组件，因此 Visual Basic 所开发的程序有一个致命的缺点——“慢”，尤其是在图像与绘图的处理速度上（例如绘图时的闪烁问题），而这偏偏又是游戏中最重要的环节之一，因此在早期编写游戏时，都不建议使用 Visual Basic 来编写，顶多只能运用在一些小游戏的设计上。

或许是为了鼓励程序设计人员多多使用 Visual Basic 来编写游戏，微软在 DirectX7 之后提供了 Visual Basic 调用的接口机制，使得 Visual Basic 可以跳过操作系统直接存取绘图装置、输入输出装置、音效装置。这使得 Visual Basic 在绘图、装置的撷取速度上都有了明显的提高，而结合 Visual Basic 本身简单易用的功能，使得 Visual Basic 设计游戏的程序设计人员也有明显的增加，尤其是在一些需要绘制图像的程序上，也经常看到使用 Visual Basic 来进行编写。不过由于先天上的限制，在开发大型的游戏时，我们仍然不建议使用 Visual Basic。

语言特性与功能

Visual Basic 之所以简单，是因为以 Basic 语言为基础，一路演进至今。Visual Basic 没有 C/C++中一些隐含易错的语法，例如数据类型转换的问题；如果程序设计人员忽略了数据类型转换的问题，通常程序本身会自动进行转换处理，而且 Visual Basic 本身不使用指标，几乎所有的设置都可以使用默认值。而另一方面 Visual Basic 本身的语法关键字比 C/C++更贴近于自然语言，以下是一小段 Visual Basic 程序代码：

```
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    ' 指定横向地图的区域进行贴图
```

```
    Form1.PaintPicture Picture1, 0, 10, w, h, _
        Xc . w / 2, 0, w, h, Visual BasicSrcCopy

    If KeyCode = 39 Then ' 如果按下向右键
        Xc = Xc + 10
    ElseIf KeyCode = 37 Then ' 如果按下向左键
        Xc = Xc . 10
    End If

    ' 判断是否遇到地图的左右边界
    If Xc < w / 2 Then
        Xc = w / 2
    ElseIf Xc > 1600 . w / 2 Then
        Xc = 1600 . w / 2
    End If
End Sub
```

您可以比较出，Visual Basic 的语法更为清楚易读。而事实上我们后面所要介绍的一些实例，考虑有些读者可能没有学习过程序语言，所以也是使用 Visual Basic 来进行编写；除了语法上的简化性之外，Visual Basic 最让初学者接受的是它的编写的环境：它提供了许多现成的组件，初学者只要使用鼠标就可以利用拖曳选择的方式来轻松完成一个完整接口，而各种预设工具窗口的设计，使得使用者在设置窗口接口时更为直观，如图 5.3 所示为 Visual Basic 提供各种方便的工具窗口。

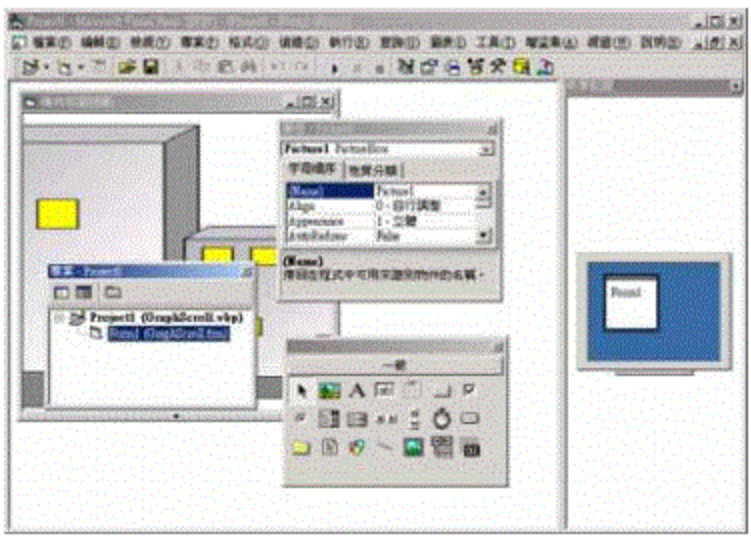


图 5.3　Visual Basic 工具窗口

然而当程序语言越简单方便时，另外一面必定就是功能有限。Visual Basic 在设计大中型程序时，确实会让人觉得束手束脚，虽然 Visual Basic 宣称其具有对象导向功能，然而多数的程序设计人员都知道这只是个口号，Visual Basic 并不具备完整的对象导向功能，在 6.0 之后的 Visual Basic.NET 中才具有较完整的对象导向功能。

开发环境

正如前面所说的 Visual Basic 程序语言与开发环境是结合在一起的，您无法仅使用纯文字编辑器来编写与编译程序，而 Visual Basic 在全世界拥有最多的程序设计人员使用，最重要的原因就在于它所提供各种工具的便利性。

5.3 Java 程序语言

Java 程序语言以 C++的语法关键字为基础，由 Sun 公司首先提出。这个计划一度面临被迫停止的可能性，然而后来却因为因特网的兴起，使得 Java 顿时之间成为当红的程序语言；这说明了 Java 的程序在因特网平台上拥有极高的优势。以在因特网平台上的优势，它具有跨平台的优点，然而 Java 适不适合拿来进行游戏制作呢？过去您在网络上玩过的一些小游戏可能已经给了您答案，而事实上早有一些书籍专门在介绍 Java 于游戏设计上的应用，如图 5.4 所示为运用 Java 程序所编写出来的小游戏。

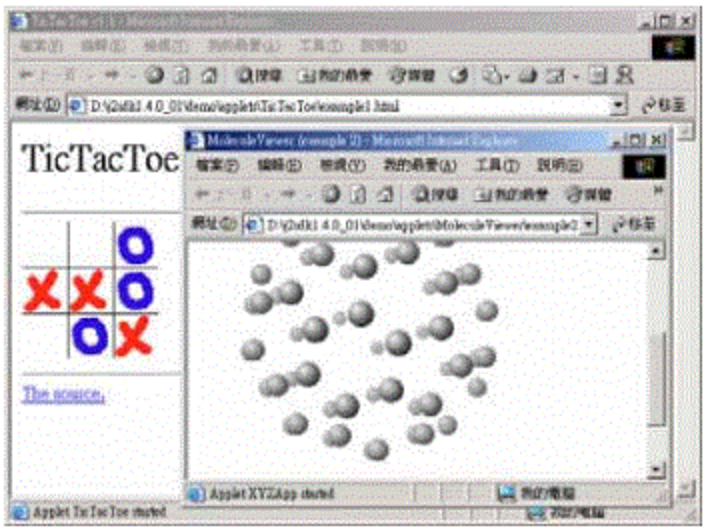


图 5.4　运用 Java 程序所编写出来的小游戏

执行平台

Java 程序具有跨平台的能力，相信这句话对于多数的程序设计人员来说都没有异议。所谓跨平台功能，指的是 Java 程序可以在不重新编译的情况下，直接运行于不同的操作系统上。这个机制之所以可以运行的原因在于“字节码”（byte code）与“Java 执行环境”（Java Runtime Environment）的配合。

Java 程序在编写完成之后，第一次使用编译器编译程序时，会产生一个与平台无关的字节码文件（扩展名*.class），字节码是一种贴近于机器语言的编码，这个文件若要加载入内存中执行，则计算机上必须安装有 Java 执行环境，Java 执行环境与平台相依，会根据该平台对字节码进行第二次编译，而成为该平台上可理解的机器语言，并加载到内存中加以执行，如图 5.5 所示为 Java 程序的执行流程，如图 5.6 所示为程序设计人员与 Java 执行环境之间的关系。

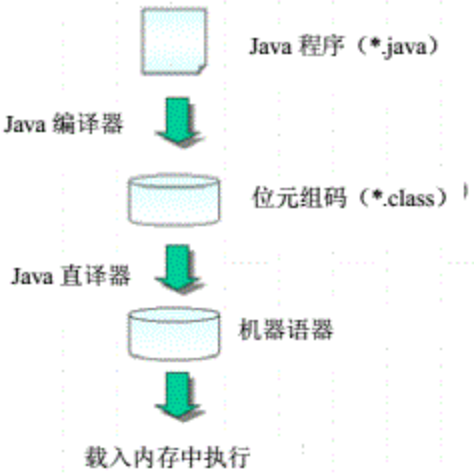


图 5.5　Java 程序的执行流程

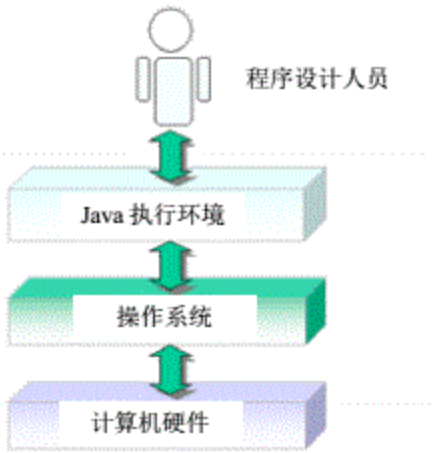


图 5.6　程序设计人员只要针对 Java 执行环境进行设计即可

Java 执行环境是建构于操作系统上的一个虚拟机器，程序设计人员只要针对这个执行环境进行程序设计，至于执行环境如何与操作系统进行沟通则是执行环境自个儿的事，程序设计人员无需理会。程序设计人员只要利用 Java 所提供的类别库与 API，避免使用第三方厂商所提供的组件，或调用操作系统的程序，基本上就可以达到跨平台的目的。

Java 程序若应用于游戏上，则可以有两种显示方式。一种是运用窗口应用程序，另一种是使用 Applet 内嵌于网页之中，但其实它们两者都是相同的，因为 Applet 程序基本上也算是一种窗口程序的呈现方式。我们前面所看到的 Java 程序执行图片，就是使用 Applet 的方式，而我们也可以利用纯窗口的形式来显示，如图 5.7 所示。

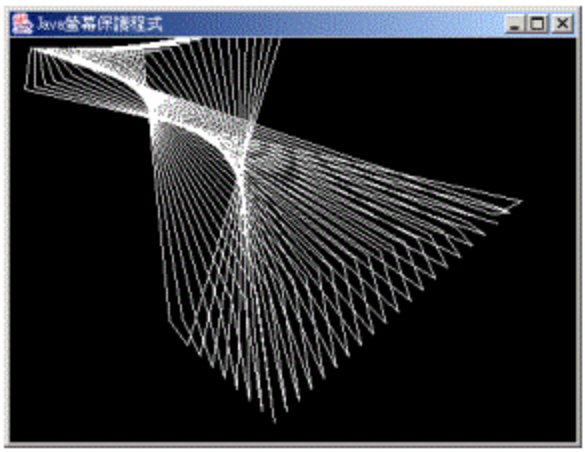


图 5.7　一个 Java 程序窗口

前面提过 Java 程序的计划当初一度面临终止的命运，是因为因特网的蓬勃兴起才得以延续。足见 Java 这种跨平台的特性非常适合于网络上，使得 Java 程序再度受到程序设计人员的重视，由于 Java 程序可以用 Applet 的形式内嵌于网页之中，使用者浏览到 Java Applet 程序的网页时，则会将 Applet 程序文件下载，然后由浏览器启动 Java 虚拟机器以执行 Java 程序，所以在这方面便可以称 Java 程序是以网络作为它的最佳执行平台。

执行速度

执行速度永远是游戏设计时的一个重要考虑，而这也是对 Java 程序最不利的地方。程序设计人员对 Java 程序执行速度的普遍评价就是“慢”，这是因为 Java 程序在执行前必须经过第二次编译，且 Java 程序只有在需要使用到某些类别库功能时才加载相关的类别，虽然是为了资源使用上的考虑，但动态加载多少造成了执行速度上的延迟。

不过执行速度上的问题也逐渐得到改善，在后来的版本中，编译与加载的速度上作了极大的改善，当然计算机速度的提高也弥补了速度慢的缺点；所以运用 Java 来开发一些小游戏时有所见，而一些中型游戏（例如非讲求绘图速度的 RPG 游戏）有时也可在网络上见到。

语言特性与功能

如同上面所讲的一样，Java 程序是以 C++的关键字语法为基础，目的是使得 C/C++的程序设计人员能快速切入 Java 程序语言，而 Java 过滤了 C++中的一些容易犯错或忽略的功能，例如指针的运用，并采用“垃圾收集器”（Garbage Collector）机制来管理无用的对象资源，这些都使得 Java 程序极为容易编写而且较不易发生错误。以下是一小段 Java 程序代码：

```
public static void main(String args[])
{
    ex1103 frm = new ex1103();
}
private void check()
{
    for(int i = 0; i < p.length; i++)
    {
        if(p[i].px < 0 || p[i].px > 400)
            p[i].dx = -p[i].dx;
        if(p[i].py < 10 || p[i].py > 300)
            p[i].dy = -p[i].dy;
    }
}
```

如果没有仔细观察一些小地方，Java 确实与 C/C++语法一模一样，其实 Java 与 C/C++语法上最大的不同点就在于 Java 程序完全以对象导向为中心。编写 Java 程序的第 1 步就是定义类别（class），若不从执行速度上来考虑，其实 Java 程序本身适用于大中型程序的开发。

在历经数个不同版本的改进与功能加强之后，Java 程序无论是在绘图、网络、多媒体等各方面都提供了相当多的 API 链接库，甚至包括 3D 的领域，所以使用 Java 来设计程序可以获得相当多的资源，而 Java 程序可以使用 Applet 来显示的特性，更使其有更大的发挥空间。而另一方面，目前有许多手机程序与游戏，也逐渐以 Java 程序来进行开发。

开发环境

为 Java 设计的整合开发环境相当的多，例如商业软件的 Visual J++、JBuilder，非商业软件的 forte、NetBeans 等，不过从目前来看，Java 应用于大中型游戏的例子不多，所以其整合式开发环境对游戏设计的影响并不大。

5.4 Flash 与 Action Script

近年来 Flash 逐渐在网络动画的领域创下一片天，主要还是因为网络上的动画容量不能太大，而 Flash 则利用每一个可重复利用的图片片段，加上补间动画方式来达到图片使用量的最小化。另外 Flash 可以对图片进行压缩，并针对已下载的图片先加以播放，而无需等到所有的图片都下载完毕。

补间动画的基本原理是利用“时间轴”，也就是在每个时间片段上设置动画的起点与终点，并设置图片的移动轨迹，而中间的移动过程则由播放程序自行计算，因此即使是复杂的轨迹运动，所使用到的图片永远只有一张，图 5.8 是时间轴与移动轨迹的基本示意图。

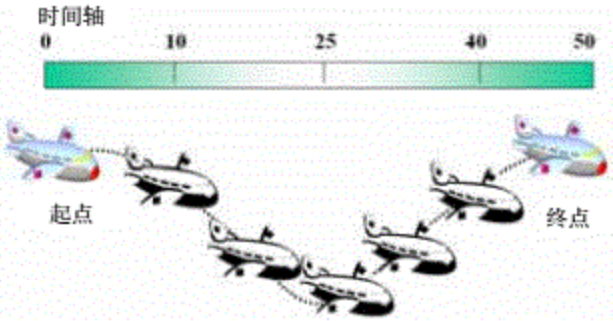


图 5.8　Flash 动画制作的基本原理

Flash 若只有动画制作的功能，相信也无法满足程序设计人员对游戏制作的需求，其实它可以完成游戏设计的主要原因还是在于 Action Script。Action Script 本来只是用来辅助动画的制作，使动画移动更具多样性，然而在后来的改版中，Action Script 的语法内容越来越丰富，使得 Action Script 慢慢具有一个程序语言所需的各种基本要素。在新版本的 Flash MX 中，Action Script 更具备了对象导向的特性，使得 Action Script 逐渐有走向程序设计应用的趋势。

Flash 主要的舞台是在网络，它可以内嵌于网页之中，也可以编译为 Windows 中单独执行的 exe 文件。要执行 Flash 动画的话，使用者的计算机中必须安装 Flash 的播放器。

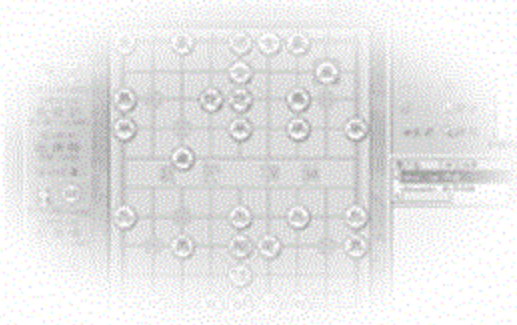
Flash 是在小游戏的设计领域中迅速走红的，由于设计出来的画面精美、容量也小，各式各样的迷您游戏都纷纷以 Flash 来编写出来，甚至于有一些早期 DOS 下的经典游戏，以及 SEGA、任天堂上的游戏，都重新以 Flash 的方式复活在网络上，市面上也有越来越多讲解 Flash 游戏设计的书籍问市。

Flash 在设计游戏的定位上算是相当清楚的，一些 2D 平面游戏都可使用 Flash 编写，也可适当地规划制作出闯关游戏、平面 RPG 游戏。Flash 是由 Macromedia 公司所提出，该公司看好网络游戏与动画的市场，也不断的积极增强 Flash 与 Action Script 的功能，在新版本的 Flash MX 中就拥有原来 Flash 5.0 所没有的功能。

在以上的几节中，您应该大致了解了各种程序语言或工具在游戏制作上的优点与缺点，而游戏设计的一个重要考虑是，您的对象与平台在哪边？因为每种程序语言与工具各有其擅长的领域。您所要决定的是依开发的目的与对象来选择最合适的一种语言与工具，如果您是站在学习游戏设计的初学者角度来看，建议您不妨从 Visual Basic 或 Flash 来入手，这可以带给您较大的成就感，如果您有志于大型游戏的开发的话，您应该学习 C/C++，这是多数大型游戏开发时的最佳选择。

第 6 章

OpenGL 与 DirectX 的简介



在游戏设计领域，我们通常会使用 OpenGL 与 DirectX 这两种开发工具来开发游戏。本章开始介绍为什么要使用这两种开发工具，并且讨论这两种开发工具的好处，以及一些简单的基本用法。

6.1 游戏开发的准备

以早期的游戏开发而言，它是一件既麻烦又辛苦的事情，从以前所使用的 DOS 操作系统来说，要开发一套游戏就必须使用程序代码来控制计算机内部的所有运作，例如显像、音效、键盘等等。而这些控制必须自行另外再开发一套游戏系统工具，如此一来才能完成控制游戏系统的基本蓝图。

6.1.1 游戏开发工具

在计算机设备越来越进步的同时，计算机内部掌管运作操作系统的功能也越来越强大，虽然计算机市场上的操作系统非常繁多，不过现今的计算机市场上，计算机还是会以“微软”(Microsoft)公司所开发的 Windows 操作系统为主要趋势，因为它几乎可以兼容市面上的所有硬件设备以及驱动程序，可以让我们省去很多不必要的麻烦。

虽然我们已经有一套非常好用的操作系统，可是似乎少了一些与计算机操作系统沟通的工具。要与计算机操作系统沟通，就必须要利用到程序代码来告诉计算机要做哪些特定的事情，而游戏画面与机制便是在做这种与计算机沟通的工作。

然而在我们有了与操作系统沟通的工具之后，对于制作游戏这方面似乎还是要自行去控制游戏中的所有事件与方法，对于一个初学者而言，似乎是一件很难办到的事情。针对游戏本身最基础的成像技术而言，如果没有一套完善的开发工具，我们就必须要自己写一套与计算机能够沟通的桥梁，对于一个游戏设计者来说，这是一件既花时间、又费精力的工作。因为与计算机沟通的桥梁是相当的低，而且我们必须要做到与计算机硬件沟通的同时不能发生任何的差错，不然成像出来的画面就不会是我们想要的结果。

因此在计算机硬件与游戏程序代码之间便加入了一个开发工具为桥梁，一来解决自行开发沟通工具的困扰，二来开发工具都是较低层的方式所构成，处理速度也比较快。

6.1.2 函数库

为了解决与计算机之间较为低层的动作，绘图显示卡厂商就共同研发了一套成像标准函数库“OpenGL”与微软公司所自行开发的工具函数库“DirectX”，它们各自都有其优缺

点，至于使用者要如何去选择开发工具，那只有按照自己的需求来决定了。

使用函数库的目的是要让使用者能够更加轻易地开发一套游戏，而我们可以从图 6.1 中看出成像标准函数库在制作游戏时所占的地位。

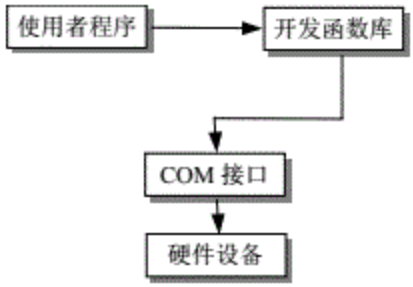


图 6.1　标准函数库在制作游戏时所占的地位

从图 6.1 中我们不难发现，使用者的程序代码与计算机直接沟通的过程似乎是不可能实现的，因为当程序代码与计算机的 CPU 在做沟通前，程序代码必须通过 COM 接口等重重关卡。而 COM 接口是计算机硬件内部与程序之间沟通的桥梁。简单地说，程序必须先经过 COM 接口的解译才能直接对 CPU、显示卡或其他硬设备作出要求或反应，而这种与 COM 接口直接沟通的程序代码却不易编写。不过现在您大可不必这么担心，因为还有两种不同功能的函数库可以使用，而这两种函数库是可以直接对计算机的 COM 接口做低层的联系，它们就是 OpenGL 与 DirectX 开发工具。这两种开发工具可以很轻易地通过 COM 接口与 CPU、显示卡或计算机的硬设备做直接的沟通，而且它们把所有的细节，不管是显示、音效、网络等多媒体的接口都包含进来了；我们只要在这种开发工具上轻轻松松地下达几个参数或命令便就可以轻易地得到我们所要的结果。

Microsoft（微软）与 SGI（视算科技）这两家公司是 OpenGL 的原始支持和培育者，微软公司以前曾经一度想要将 OpenGL 的技术并入到 DirectX 技术中，它将这个计划称作为 Fahrenheit。但是这个短命的 Fahrenheit 计划最终还是声明失败。对于微软公司而言，计算机业界相对的受到了一个很大的打击。而现在他们还是希望能够建立一个可运行于 Microsoft 操作系统之内与外的通用 API（Application Program Interface），并且将绘图系统的开发提升到一个共同的架构上，不过这似乎是一件不太可能完成的任务。Microsoft 公司被指责是一个缺乏意愿的培养者；SGI 公司则是因为支持的问题而导致不良结果的环境而被批评。不管是什么原因，这些都可能还包括一些容易出错的人为因素在内。Fahrenheit 计划虽然已经成为过去，但是 Microsoft 公司现在还是倾全力的开发资源，投入到 DirectX 技术当中，它想要将 DirectX 技术变成一个高阶绘图与游戏开发 API 工作上的基本工具。

在介绍完这两种开发工具函数库之后，接着来看看 OpenGL 与 DirectX 这两者之间到底有什么差别。

6.2 OpenGL

OpenGL 即是 Open Graphics Libraries 的缩写，它是一套“计算器三维图形”处理函数库，由于它是由各家显示卡厂商所共同定义的函数库，所以也是绘图成像的工业标准。

“计算器三维图形”指的是利用数据描述的三维空间经过计算机的计算，再转换成二维图像并显示或打印出来的一种技术，而 OpenGL 就是支持这种转换计算的链接库。

6.2.1 OpenGL 的简介

OpenGL 是由“SGI”公司为了图形工作站所开发的“IRIS GL”，在跨平台移植的过程当中，便发展成为“OpenGL”。SGI 公司在 1992 年 7 月时发布了 OpenGL 1.0 版，OpenGL 则成为显示工业的一项特定标准，接下来 OpenGL 再由成立于 1992 年的独立财团 OpenGL Architecture Review Board (ARB)所控管。SGI 与 ARB 成员以投票方式来产生这项特定标准，并制作成规范文档(Specification)公之于市，从此各家软硬件厂商则依据这种原则标准来开发自己系统上的显示功能。各家软硬件厂商的产品只要是通过了 ARB 的全部规范标准测试之后，产品才能称得上是支持 OpenGL 的成像技术。

在 1995 年 12 月的时候 ARB 又批准了 OpenGL 1.1 版本，而现在最新的规范标准版本是在 1999 年 5 月所通过的 OpenGL 1.2.1 版。 OpenGL 后来被设计成独立于硬件与操作系统的一种显示规范，它可以运行于各种操作系统与计算机上，并且能在网络环境以客户（Client）和服务器(Server)模式之下工作，它是专业图形处理与科学计算等高阶应用领域的标准图形函数库。它在低阶成像应用上的主要竞争对手是 Microsoft 的 Direct3D 图形函数库。Direct3D 图形函数库是利用 COM 接口的形式来提供成像处理，所以其架构较为复杂，稳定性在于 OpenGL 开发函数相比较之下，DirectX 也比较差。另外因为 Microsoft 公司拥有该函数库的版权，所以到目前为止，DirectX 只能在 Windows 平台上才可以使用 Direct3D。而 D3D 的主要优势就是在于处理运算速度上，但是以目前低廉价格的显示卡来说，它们也都能提供很好的 OpenGL 硬件加速了。所以游戏设计者做 3D 影像处理就不必局限于只能使用 Direct3D。

在专业图形的高阶应用方面，目前还没有出现以 Direct3D 技术为基础的例子，而游戏等低阶的应用程序也有开始转向 OpenGL 的趋势，相信 Direct3D 在未来极有可能会被 OpenGL 给取代的。

6.2.2 OpenGL 的要求处理

OpenGL 可分为程序式（Procedural）与非描述式（Descriptive）两种绘图 API 函数，

使用者不须要直接描述一个场景，而只须规范一个外观的特定效果的步骤，而且这个步骤是以 API 的运作方式去调用。优点就是可移植性高，以及具有超过 2000 个以上的指令与函数的绘图功能。

OpneGL 本身并没有提供窗口控制指令、事件及档案的输出及输入。不过上述这些函数都可以使用 Windows 里所包含的 API 函数来做到。其实 OpenGL 的主要用意是在于使用者表现高阶需求的时候，可以利用低阶的 OpenGL 来控制。

其 0penGL 在处理绘图影像要求的时候，我们可以将它归纳成两种方式来表示，一种是软件要求，另一种是硬件要求。接着来看看这两种要求的不同之处。

软件要求

OpenGL 软件要求架构可以从图 6.2 中看出。通常显示卡厂商会提供 GDI（Graphics Device Interface，图形设备接口）的硬件驱动程序来实现画面输出的需要，而 OpenGL 的主要工作就是接受这种绘图需求，并且将这种绘图需求建构成一种影像，然后再将影像交给 GDI 处理，最后再由 GDI 送至显示绘图卡；如此一来，绘图显示卡才能将成果显示于屏幕上。简单地说，OpenGL 的软件要求是必须通过 CPU 的计算，再送至 GDI 去处理影像，再由 GDI 将影像送至显示装置，这样的过程才能算是做一次绘图显像处理的动作。从上述的成像过程中，我们不难看出在处理显像的速度上可能会降低许多，为了提高显像的速度，我们只有将数据直接送至绘图显示卡来处理，让绘图显示卡直接做处理与显像的工作，才能有最快的效果。

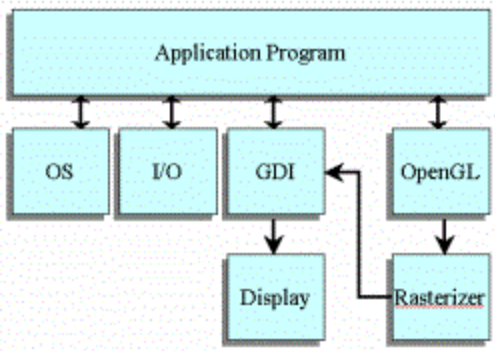


图 6.2　OpenGL 软件要求架构

硬件要求

接着先来看看 OpenGL 的硬件要求基本架构图，如图 6.3 所示。

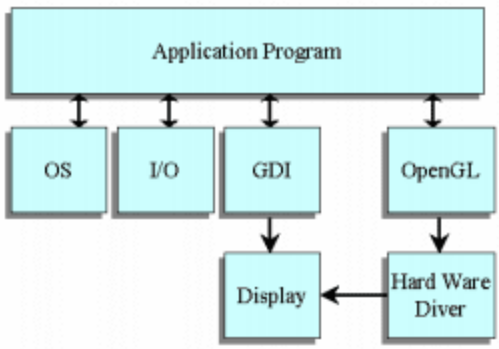


图 6.3　OpenGL 硬件要求基本架构图

OpenGL 的硬件处理要求是将显像资料直接送往绘图显示卡上，而让绘图显示卡去做绘图要求的建构与显像的动作，其过程不必再经过 GDI 的软件装置，如此一来便能省下不少的运算过程，因此显像的速度便可以大大地提升了。

6.2.3 OpenGL 的基本运作

我们在前面就谈过，OpenGL 是利用 API 的表现方式来运作。现在我们来看看 OpenGL 是如何来处理绘图要求的数据，如图 6.4 所示。

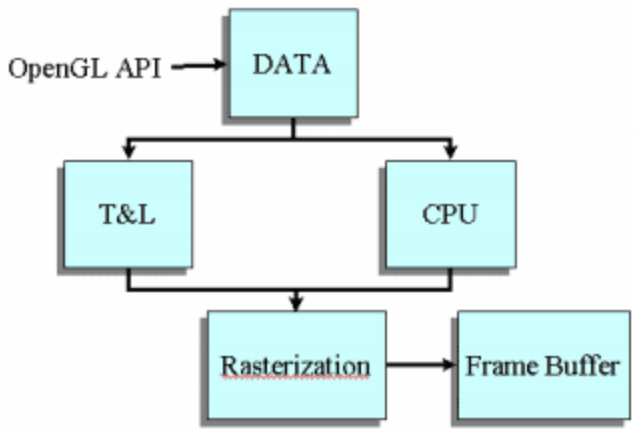


图 6.4　OpenGL 与绘图要求的数据之间的关系

由图 6.4 得知，当 OpenGL 在处理绘图数据的时候，它会将数据填满整个缓冲区，而这个缓冲区内的数据包含指令、坐标点、材质信息等，再由指令控制或缓冲区被清空（Flush）的时候，将数据送往下一个阶段去做处理的动作。在下一个处理阶段中，OpenGL 会做坐

标转换与灯光（Transform & Lighting,T&L）的运算，其目的是在运算物体实际成像的几何坐标点与光影位置。在完成上述处理过程之后，其数据会再被送往下一个阶段。在这个阶段中，其主要的工作是将运算后的坐标数据、颜色与材质数据经过扫描显像（rasterization）的技术来建立一个影像，然后影像再被送至“Frame Buffer”（绘图显示装置）上的内存中，最后才由绘图显示装置将影像呈现于屏幕上。

在早期的 OpenGL 绘图加速显示卡上，它们只不过是在扫描显像的技术步骤上做加速的动作而已，而上述管道中，转换与灯光的过程则由 CPU 来运算，不过现在在绘图显示卡技术提升与价格低落的情况下，几乎每一张绘图显示上都有“T&L”的加速功能。再加上绘图卡上的内存不断地扩充，因此在绘图显像的过程中似乎都不需要再经过 CPU 和主存储器的运算了。

6.3 DirectX

以现今市面上的计算机而言，不管是企业或个人使用，大多数都是采用 Microsoft 的 Windows 操作系统，而在 Windows 操作系统环境下所运行的游戏，大多也都需要支持 DirectX。可是对一个初学者的玩家来说，DirectX 到底是什么东西呢？为什么游戏非需要它不可呢？这些答案将很快为您揭晓，下面我们来详细地介绍一下 DirectX 的基本理论。

6.3.1 DirectX 简介

DirectX 是一种 Windows 系统的应用程序接口（Application Programming Interface，简称 API），它可以让以 Windows 为操作平台的游戏或多媒体程序获得更高的执行效率，而且还可以加强 3D 图形成像和丰富的声音效果，另外提供设计人员一个共同的硬件驱动标准，让游戏开发者不必为每一个厂商的硬件设备编写不同的驱动程序，同时也降低了使用者安装及设置硬件的复杂度。

举例来说，以前我们在玩 DOS 环境下的游戏时，必须设置声卡的品牌，然后再设置声卡的 IRQ、I/O、DMA，如果其中有一项设置不对的话，那么游戏就无法发出声音了。这部分的设置不但让玩家伤透脑筋，而且对于一个游戏设计人员来说，这也是非常头疼的一件事，因为游戏设计者在制作游戏之初，就需要把市面上所有声卡硬件数据都收集过来，然后再根据不同的 API 函数编写不同的声卡驱动程序，这种过程是一个相当烦杂而且吃力不讨好的工作。

> 提示：IRQ：它是 Interrupt ReQuest 的缩写，中文解释为中断请求。因为计算机中的每个组成组件都会拥有一个独立的 IRQ，除了使用 PCI 总线的 PCI 卡之外，每一组件都会单独占用一个 IRQ，而且不能重复使用。

DMA：它是 Direct Memory Access 的缩写，中文翻译成“直接存取内存”。

不过幸运的是现在玩 Windows 环境下的游戏时，我们就不需要做这些硬设备的 IRQ、I/O、DMA 等设置了，因为 DirectX 提供了一个共同的应用程序接口，只要游戏是按照 DirectX 方式来开发的话，不管使用的是哪一家厂商的显示卡或声卡、甚至于网卡等等，它们统统都可以被游戏所接受。而且 DirectX 还能发挥比 DOS 环境下更好的声音与图像效果，当然前提必须是显卡或声卡的驱动程序也要支持 DirectX 才行。

6.3.2 DirectX SDK

Microsoft DirectX 提供了一套非常好用的应用程序接口，其中包含了设计高性能、实时应用的程序代码，它称为“DirectX SDK”（DirectX Software Development Kit，俗称“DirectX 开发包”）。DirectX SDK 技术能够帮助我们轻易地建构计算机游戏和多媒体的应用程序，其中包括了 DirectDraw、DirectSound、DirectPlay、Direct3D 和 DirectInput 等部分的 API 指令及媒体相关的组件（Component）。

其实 DirectX 里面就包括了两个部分，分别是“执行时期”（Runtime）和“SDK”这两个部分。在 DirectX 开发的阶段里，这两个部分都会使用到，但是在 DirectX 应用程序运行的时候，只需要使用到执行时期的部分。在 Windows NT 4.0 以上的版本当中，它就包含有 DirectX 执行时期的部分，而 Windows 98/98 则没有。不过我们可以在网络上很轻易地获取到 DirectX 执行时期的这个部分。由于 Windows NT 的系统架构较为特殊，所以在 NT 4.0 以前的版本是不能执行 DirectX 程序的。

DirectX SDK 目前较为流行的版本有第 7 版与第 8 版，而现在微软也正在紧锣密鼓地开发第 9 版；其目的是将 DirectX SDK 设计成为游戏开发所必备的工具。在不同的 DirectX SDK 版本当中，它们具有不同的执行时期，不过新版本的执行时期还是可以与旧版本的应用程序配合，也就是说 DirectX 版本的执行时期是可以向上兼容的。

6.3.3 DirectX 的架构

如同前面所说的一样，DirectX 只是提供一致性的应用程序接口，其实只要深入地了解到 DirectX 的各项组件便可以明白为什么有许多游戏都要支持 DirectX 了。

Direct3D

Direct3D 简称 D3D，它的大名相信读者们都曾听过。对于现在的游戏来说，D3D 实在是太重要了！由于 3D 游戏的兴起，各大厂商纷纷推出 3D 显像加速卡，而为了让显示卡能够避免如同声卡规格的统一性，微软从 DirectX3.0 以后就加入 D3D 这个 API 的技术，

让 3D 游戏有一个共同的开发标准。如此一来，当游戏在执行的时候，如果需要使用到绘图的部分时，DirectX 就会通过 D3D 向显示卡驱动程序提出成像要求，并且让显示卡完成绘图的动作。

DirectDraw

DirectDraw 是 DirectX 中非常重要的一部分。它担任的工作是 2D 图形的处理。在以前 DOS 环境下设计游戏的时候，为了要考虑到游戏的整体运行速度，我们只要能够让游戏程序直接对映硬件设备的内存区段来加快运行速度。不过在 Windows 操作系统这种保护模式之下，所有图形的接口动作都必须经过 GDI 这个图形处理中心来处理，而不能直接对硬件设备下命令。但是 GDI 接口对连续画面处理的效果就非常的不好，如果一款游戏一旦通过 GDI 接口来处理的话，那么其成像的效果画面将会大打折扣。

提示：GDI：图形设备接口（Grahpics Device Interface），它是负责在屏幕上显示图形所用的接口。GDI 由百余个函数所组成的。GDI 函数必须通过设备描述表（简称 DC）的句柄来控制绘图。

基于这个理由，DirectDraw 主要的工作也就是用来帮助 Windows 的应用程序能够直接进行硬件设备的操作，更进一步它还可以加速显示卡的速度，使得游戏的画面呈现起来更加地流畅。另外 DirectDraw 还支持 CPU 的 MMX、3DNow!及 AGP 等特殊的指令技术，而且它还可以处理多屏幕的显示，让 Windows 环境下运行的游戏更加地多姿多彩。

提示：MMX：MMX 处理器是由 Intel 公司所研发出来的 CPU 指令集，它含有专门处理声音、影像和图片的附加指令。
3DNOW! AMD 公司利用一种更先进的 3D 算法为主流 PC 上的 3D 应用服务，我们称它为“3DNow!”技术（x86 构架上的 3D 技术的创新）。
简单地说，“3DNow!”技术就是在 x86 构架上的一种 3D 创新，使得 x86 构件的瓶颈得以突破。计算机能够产生更加真实、有趣、生动的 3D 画面。
AGP：计算机内部的图形卡专用插槽。它由 Intel 所设计出来的，它的传输速度比一般 PCI 卡快上两倍。AGP 可以让特定的显示卡从使用专用内存来存储或撷取游戏，3D 应用程序的图片类型。

DirectX 从 8.0 开始，DirectDraw 和 Direct3D 的组件就已经合并在一起，Microsoft 称这种技术为“DirectX Graphics”，而且其 API 指令也进行了大幅度的更新，所以现在的“DirectX Graphics”技术就更容易让使用者所使用了。“DirectX Graphics”还支持最新的图形硬件接口，而最引人注目的地方就是它也可以支持着色引擎(Shader)，至于着色引擎的功能与用途请参阅其他 Direct3D 相关书籍。

提示：着色器：着色器是用着色语言编写的一段代码，着色语言是专为在可编程顶点或图形组件中使用而设计的。

DirectMusic

Direct Music 的核心包含了“MIDI 播放系统”及“互动音乐系统”这两个部分，当 Windows 加入了 DirectMusic 的功能之后，它对 MIDI 的支持能力便大大地提高，因为以前 Windows 经常被脉冲及时序标签等问题所困扰，而现在这些问题也都迎刃而解了，并且 DirectMusic 也采用了最近正式被通过的 DLS（Downloadable Sounds）1.0 技术。所以它也解决了 MIDI 音乐播放时，效果不一致的老问题，另外每首 MIDI 音乐还可以使用任何自定义的音源，以及突破性的 128 种 General MIDI 音源限制。

DirectSound

DirectSound 是用来处理声音的 API 指令函数，除了播放声音和处理混音之外，还加强了 3D 音效的部分，并且提供录音的功能。对于前面介绍的声卡兼容的问题，现在我们可以利用 DirectSound 的技术来解决。

DirectX 从 8.0 开始，DirectSound 与 DirectMusic 的组件就合并了，Microsoft 称其为“DirectX Audio”。WAV 音效文件或其他音效资源如今可以由 DirectMusic 加载器来加载，并且还能够通过 DirectMusic 的演奏器来进行播放，也可以与 MIDI 音响进行同步演奏。

DirectInput

DirectInput 是用来处理游戏的一些外围设备装置，例如摇杆、GamePad 接口、方向盘、VR 手套、力回馈等外围装置。以往要在 DOS 环境下使用方向盘来玩赛车游戏的话，我们就必须先调整好方向盘设备的 IRQ、DMA 等设置，才能使用方向盘装置，而现在 DirectInput 则可以提高这些设备与游戏之间的兼容性，而且也不需要对外围装置做任何特别的设置了。

DirectPlay

DirectPlay 是为了满足目前流行的网络游戏所开发的 API 指令，而且它还支持许多通信协议，让玩家可以利用各种连网的方式来进行网络游戏的对战，此外 DirectPlay 也提供网络对谈的功能，以及保密的措施。

通常在制作游戏的时候，不管是 2D 或 3D 的游戏，其显像部分都是整个游戏的重点所在。而在制作游戏之前，我们就必须详加考虑到应该要使用哪一种显像技术或者是直接编写包含这两种显像的技术，并且要考虑到显示卡市场的支持性是否足以应付我们所开发的应用程序或游戏。而其他部分，例如音效、网络联机则可以利用 DirectX 的技术来实现。

第 7 章

游戏类型

在学习游戏设计的过程中，有些初学者对于游戏制作的专有名词并不熟悉，以至于妨碍了整个学习进度；本章将为您收录游戏设计时会遇到的一些常用专有名词。

7.1 角色扮演类

角色扮演游戏又称为 RPG（Role Playing Games）类型游戏。以现今的游戏类型来说，虽然称不上是最多的，不过可以确信这一种类型的游戏玩法是目前最流行的，但有许多读者对于 RPG 游戏的类型只能一知半解，在本节里我们将先来介绍 RPG 游戏。

在开始介绍游戏类型之前，相信许多读者被游戏类型的一些英文缩写搞得焦头烂额，笔者在这里整理了一些目前游戏界中经常会看到的游戏类英文缩写，如表 7.1 所示，供读者参考。

表7.1　中英文游戏缩写对照表

英文缩写	英文全名	中文解释
ACT	ACTION GAME	动作游戏
RPG	ROLE PLAYING GAME	角色扮演
ARPG	ACTION RPG	动作角色扮演
SRPG	SIMULATION RPG	策略角色扮演
SLG	SIMULATION GAME	策略类
FTG	FIGHTING GAME	格斗类
SFTG	SIMULATION FTG	策略格斗类
SPG	SPORT GAME	运动类
RAC	RACE GAME	赛车类
AVG	ADVENTURE GAME	冒险类
PUZ	PUZZLE GAME	益智类
TAB	TABLE GAME	桌面类
STG	SHOTING GAME	射击类
ECT	ETCTERA GAME	其他类

7.1.1 RPG 游戏的发展史

最初的角色扮演游戏是在纸上进行的，而纸上角色扮演的最早起源是来自纸上的战略游戏。事实上很久以前，欧洲和美国的许多人都非常热衷于玩纸上的战略游戏。所谓纸上

战略游戏，其玩法就是参与游戏的玩家在纸上画出一个特定的地图，而在地图上再利用各种抽象的纸片、符号或者是以一般的小塑料块来玩战术与战略的过程。这种游戏通常以第一次世界大战和第二次世界大战为背景，在当时来说这种游戏是相当受欢迎的。

纸上角色扮演游戏（TRPG, Table talk-Role Playing Game）是由纸上战略游戏演变而来，如果要玩这一类游戏时，需要几个玩家来配合，而在这几个玩家中必须选择一个主持人和准备一些纸片道具。在游戏进行的时候，游戏者需要以掷骰子来决定前进的步数，再由主持人来讲述此游戏的故事内容，然后让玩家知道自己遇上什么事件。当时的游戏中，主持人就是游戏的灵魂，也是这个游戏的创作者与故事讲述者，同时也是规则的解释人。所有的玩家就等于是在担任主持人故事中的一个特定角色，而这个故事的精彩与否则是直接取决于主持人的能力。以投掷骰子的方式，体验不可预知的结果和玩家的行动，这就是角色扮演游戏最原始的雏形。

纸上游戏系统在近几年中，以一套“AD&D”的核心成功地虏获了玩家的心，更将“AD&D”核心带进电子游戏中。或许您会问：“那什么是‘AD&D’呢？”这个答案将在下一节内容为您详加介绍。

7.1.2 AD&D

一个非常新的名词，或许会有许多人问：“什么是‘AD&D’？‘AD&D’又代表什么意义呢？”其实“AD&D”的最前身就是采用“龙与地下城”的架构。“龙与地下城”是以剑与魔法的奇幻世界为主要背景的 TRPG 规则系统，通常称为“D&D”（Dungeons & Dragons）。目前已经出版到第三版，而最新的版本因为在进行游戏的时候，主要会使用到 20 面骰子，所以称为“d20”系统。

基本上，我们几乎可以说“龙与地下城”创造了 RPG 这种游戏类型。“龙与地下城”的第一版发行于 1973 年，是由纸上战棋游戏转化而来。而当时这种纸上战棋游戏还没有 RPG 的成分存在，而在“龙与地下城”所推出的初版获得相当大的成功之后，也带起了 RPG 的发展开端。

在改版之后，“龙与地下城”便移植到计算机游戏上，并且掀起了“金盒子系列”的 RPG 游戏风潮。近年来的“柏德之门”、“冰风谷”等系列游戏也是采用了“龙与地下城”的规则系统。另一方面，“龙与地下城”在 20 世纪 80 年代初期就被引进到日本，所以“龙与地下城”的规则系统不论是对日本的电子游戏界或奇幻文学界，甚至以日本自行编写的游戏系统来说，也都造就相当深远的影响。

事实上，在许多今日盛行的网络游戏中，我们还是可以看到“龙与地下城”的影子。例如风靡全球的“无尽的任务”（Ever Quest）游戏中，它可以说是将“龙与地下城”的精神加以“形象化”。而我们国内相当盛行的网络游戏“天堂”更是参照了“龙与地下城”第一版的早期规则。

在进行 TRPG 的时候，“龙与地下城”必须要使用到一些核心规则书，而这些核心规则书包括下列 3 本：

（1）玩者手册：它是在说明游戏进行的规则、人物创造、人物的技能与成长方式，以及装备、法术等细节设置。一般玩者只要拥有这本书，就可以参与其游戏。

（2）地下城主指南：它是游戏主持人（也就是“地下城主”）所需要的书，其内容包括游戏的指导、如何设计冒险和世界的说明，以及“玩者手册”中尚未提到的幕后规则与备用规则。它是游戏设计者与地下城主必备的参考书。

（3）怪物图鉴：它所包含的信息是详尽的怪物设置数据、战斗数据以及生态等信息，并且还附有精美的插图。

“AD&D”创造出 RPG 的游戏基础架构，也影响到尔后的 RPG 发展方向，不管现在的 RPG 游戏变化有多复杂，总还是脱离不了“AD&D”的规则系统。

7.1.3 RPG 游戏架构

以 RPG 的游戏架构来说，它是由许多游戏玩法机制所综合而成的。单纯以一个简单的场景来看，当我们所操作的人物在路上行走时，可能会不预期地遇上敌人的攻击、捡拾到装备宝物或者是触发一些特定的事件等等，这些都必须要经过策划人员事先深思熟虑地设计。关于 RPG 游戏的做法，或许您会问：“那要怎么样才能让玩家在游戏里找到乐趣呢？”其实不管我们所设计出来的 RPG 游戏有多复杂或者有多困难，都不外乎下面几项基本原则：

人物的描写

RPG 游戏最主要的就是在强调人物的特性描写与它的背景故事表现，以达到角色扮演的目的。简单地说，RPG 游戏就是要让玩家能够感觉到游戏中的人物就好像是自己在扮演一样。

宝物的收集

RPG 游戏的另一项较为有利的原则，就是宝物的收集。不管是装备或者是宝物，甚至于如同“太空战士”（FINAL FANTASY）游戏系列中的“召唤兽”机制，这些都足以紧紧吸引住玩家的目光。

剧情的事件

RPG 游戏的主要轴心就是它所呈现的故事剧情内容。这种故事剧情能将角色扮演的成

分提升至最高，并且强调角色在故事里存在的必要。

华丽的画面

为了提高 RPG 的游戏品质水准，华丽的战斗画面是不可忽略的重点之一，因为它常常会使得玩家对游戏爱不释手，就如同“太空战士”系列一般，它的 3D 真实战斗画面深深地吸引着玩家，让玩家们不知不觉地成为它衷心的爱好者。

职业的特色

这是 RPG 游戏类型较为成功的游戏机制，所有的人物都有自己独特的个性，再加上本身所属的职业，让角色的特性更突显出来；如勇士、魔法师、僧侣等等。每一种角色又可以与其他角色的能力做互补，这项原则更加强了 RPG 游戏的品质与内容。

以 RPG 的游戏来说，相信上述几项原则便可以很轻易地了解到 RPG 游戏的重点因素。

7.1.4 代表性游戏

以单纯的 RPG 游戏来说，如同史克威尔公司(SQUARE)出的“太空战士”系列（FINAL FANTASY，简称 FF），就是一个很明显的例子；以“AD&D”系统来说，如同“柏德之门”（Baldurs Gate），它也是一个相当成功的例子。柏德之门以“AD&D”系统作为游戏的核心，成功地将原本是纸上 RPG 游戏系统带进电子游戏系统中。

当 RPG 游戏从纸上慢慢地移植到电子游戏平台的过程中，其过程发展的速度并不是很快，经过不断地修改角色扮演游戏的方式；不管是在纸上的 RPG 游戏、还是电子游戏平台上的 RPG 游戏，它始终占有一席不败之地。无论是刚接触 RPG 游戏的玩家或者是资深的老手，对于 RPG 游戏来说，它始终是玩家的最爱。

7.2 动作类

动作类游戏又称为“ACT”（Action Game），它可说是游戏界上占有最大市场的游戏类型，只要游戏玩家，相信一定不会排斥这种动作类游戏的玩法，因为动作类游戏是所有游戏类别最基本的游戏玩法模式。

从近年来的主机平台功能的升级，动作类游戏在玩法上变得更复杂、内容更加丰富，它带给游戏玩家的那一份刺激感是不可否认的。接下来就来探讨一下动作类游戏成功的地方。

7.2.1 动作类游戏的发展史

动作类游戏可以说是所有游戏类型中最为单纯的，所有的游戏类型都是从动作类游戏的变形演化而来的。在早期的游戏产业里，游戏平台只能支持低位的成像处理，因而平台不能做非常复杂的运算，所以动作游戏就在这个时候诞生了，如同我们小时候玩的“小蜜蜂”游戏，这类游戏不须要花费我们太多的心思与时间，即可让游戏顺利地进行下去，到接下来红白机任天堂上的“超级玛璃欧”游戏，它将动作游戏的指标带至顶峰，当时多少人为了破“超级玛璃欧”的游戏，而不分昼夜、无时无刻去玩它，更夸张的是竟然有人可以练到破除“超级玛璃欧”的所有关卡只要花上 3 分钟的时间，诸如此类，动作游戏所带来的热潮是从以前到现在永远都不会退烧的。

7.2.2 动作游戏的类型

动作游戏是一个很广泛的名词，它代表的游戏方式相当的多，为了区分各种不同类型的动作游戏，所以又将它细分出下列几点：

2D 动作

2D 动作类是以 2D 平面的表现方式来呈现游戏中的画面。在画面中，玩家必须躲过所有的障碍物与攻打敌人为主要轴心，并且在游戏中寻求特定的目的来破除关卡。

3D 动作

3D 动作游戏与 2D 的动作游戏玩法大致相同，不同的是 3D 动作游戏的画面讲究的是 3D 立体真实感而已。

射击类

射击类的动作游戏可以利用 2D 平面或者是 3D 立体画面来呈现，玩法大都是以飞机射击类为主，强调的是画面中枪林弹雨所带来的刺激感。

格斗类

格斗游戏发展时间较一般的动作游戏晚，它是从单纯的动作游戏所演变而来的。从红白机任天堂的“功夫”游戏来看，它就是属于非常单纯的格斗游戏。只要一个场景中、一个由玩家所操作的主角、一个由计算机所控制的敌人，便可形成一种您打我、我打您的游

戏机制。格斗类游戏一直发展到现在，又加入了剧情、连续技与多人格斗的游戏机制，这些机制让格斗游戏的内容更加丰富有趣。

第一人称射击类

第一人称的射击游戏可说是动作游戏类型发展过程中最晚的一种游戏形式，如同以前在计算机 PC 上的“德军总部”。当时它发挥了超高的 3D 显像技术，让许多玩家叹为观止，尔后便陆陆续续地出现了许多更为精致、更有内容的第一人称射击游戏，如同“战栗时空”（Counter Strike，简称 CS）。

其他

除了上述各种类型的动作游戏外，我们又归纳了一些热门的动作游戏机制，如大型机台的跳舞游戏机制：它能带动玩家做更刺激的动作行为。当时将跳舞机制呈现在游戏中，造成一股玩跳舞机的热潮，因为这种热潮的影响，它又渐渐被带到电视主机与计算机 PC 平台上。在了解到动作游戏的类型之后，接下来我们将介绍动作游戏的架构设计。

7.2.3 动作游戏的架构

动作游戏的架构是所有游戏中最为简单的一种，我们先以一个简单的流程图来说明动作游戏的进行方式，如图 7.1 所示。

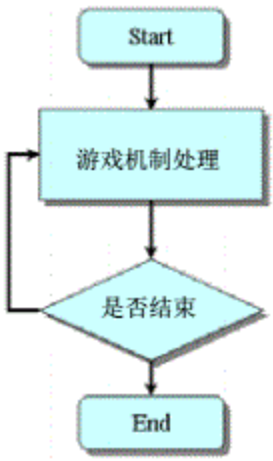


图 7.1　动作游戏进行方式的简单流程图

从图 7.1 中可看出，游戏是由一个循环所组成，在游戏还未结束之前，它会一直环绕

着循环的原则执行。而在循环中，游戏必须处理各种事件因子，这些事件因子是用来处理游戏中所发生的每一件事情，如玩家所操控的飞机、飞机的子弹发射、敌机的飞行控制等等，其过程原则不外乎下列三项：

游戏的开始

游戏要如何开始？玩家在游戏开始时必须要经过哪些程序？如以飞机的射击游戏来说，玩家就必须先选择自己所喜爱的机型、武器，甚至关卡等等，让玩家清楚地了解到开始游戏时所操作的角色能力或装备。

游戏中的处理

它是动作游戏的精华所在。为了让动作游戏更好玩，游戏设计者就必须在游戏中，将所有可能发生的事件设计出一系列的玩法机制；如飞机打下敌机之后，敌机会掉落宝物可供飞机提升攻击能力或防御能力等等，这些吸引人的机制就必须在设计中加入。

游戏的结束

游戏的结束是动作游戏的目的或死亡的行为。在玩家玩到某种程度的时候，就必须让玩家做结束游戏的行为。特别注意的是，不要将游戏拖泥带水，因为动作游戏的玩法比较单纯，玩家往往只会在游戏中寻求它的刺激感，而不是要观看最后的结局是如何。如果游戏一直不断地出现 100 关、200 关，甚至更多，相信没有一个玩家会受得了这种折磨，还真成了歹戏拖棚。

其实拿 RPG 游戏与动作游戏来比较，动作游戏就显得单纯许多，而故事剧情也比 PRG 游戏还要短。所以为了让动作游戏更加好玩，就必须在玩家玩的过程中下很多功夫。一般而言，动作游戏的流程速度是非常快且刺激的，而 PRG 游戏主要的成分是在讲述故事，所以它的流程速度可能没有动作游戏来的快，不过 RPG 游戏的内容却远远多于动作游戏。既然这两种游戏类别有着不同的优点，那么何不将这两种游戏的类型合二为一呢？是的！在游戏世界里，是可以将这两种游戏机制的优点合二为一，那就是“动作角色扮演”游戏。

7.3 动作角色扮演类

“动作角色扮演”（Action Role Playing Games，ARPG），它所发展的时间较 RPG 游戏与动作游戏还要晚。因为它是采取动作游戏紧凑的玩法与 RPG 游戏剧情的流程为主轴，让玩家可以玩到动作游戏所带来的刺激感与 RPG 游戏角色扮演的乐趣，所以它又让游戏产业

再度掀起一股独特的风潮。

7.3.1 动作角色扮演的发展

以动作角色扮演游戏来说，最早带起这股风潮的应该算是计算机 PC 上的“暗黑破坏神”（Diablo）与电视游戏主机上的“萨尔达传说”（Legend of Zelda）。它们打败了当时单纯的 RPG 故事剧情叙述与单纯的动作游戏：它以 RPG 游戏故事为轴心，再以动作游戏的表现方式，让玩家可以在游戏中看到整个角色扮演的故事情节的发展与直觉式的打斗方式。在未来的几年里，动作角色扮演游戏的表现方式，有可能会占掉所有游戏产业的一大半市场。

7.3.2 动作角色扮演游戏的技术

以动作角色扮演的技术来看，是所有游戏类别中最为专业的。在设计一套动作角色扮演游戏时，设计者就必须考虑得非常细致，而程序设计者又必须将动作游戏与 RPG 游戏这两种游戏机制合并成另一种游戏玩法。简单地说，就如同在做两套游戏一样，一种是动作游戏的角色控制，另一种是故事剧情的流程安排，而所花费的精力与时间相当多。笔者建议如果要做一套动作角色扮演游戏的话，就必须很精确地估算开发的时程、流程的控制，以及所花费的人力与金钱，不然很容易让游戏胎死腹中。接下来，我们来介绍一下开发一套动作角色扮演游戏需要了解的基本技术。

故事剧情架构

动作角色扮演的故事剧情其实与 RPG 游戏是相同的，所以也必须编写出一系列可以引导故事中人物背景的交待。

人物特色表现

同前面所讲的一样，人物特色与 RPG 游戏设计的方式是相同的。

场景对象配置

场景与对象的配置手法要比单纯的 RPG 游戏还来的仔细，因为在场景中，不只玩家所操控的主角在移动，它还包含怪物、NPC 人物的移动等等。如果分配的不好，那势必会发生玩家根本打不到怪物或拿不到宝物的奇怪现象。

物体动作设计

动作 RPG 主要的操作模式就是在于玩家所操作的主角去做任何一件可以做的事，如打敌人或拿宝物等行为，游戏设计者必须去编列所有物体的动作行为，如此一来，才能够让游戏更为生动。

7.3.3 动作角色扮演游戏的架构

其实它的主要架构很简单，就是将动作游戏与 RPG 游戏的架构合并成一种而已。从图 7.2 中可以简单地看出动作角色扮演游戏的基本架构。

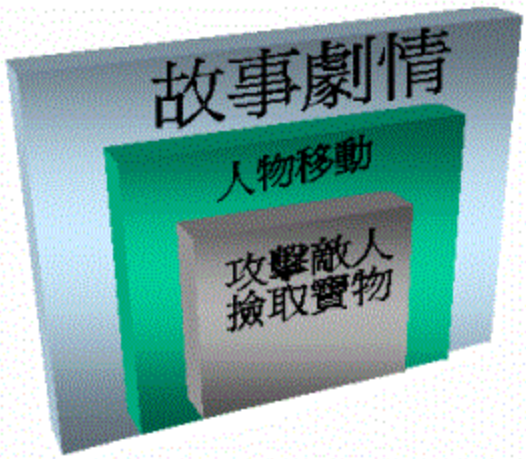


图 7.2　动作角色扮演游戏的基本架构

从图 7.2 中可以看出，由 PRG 游戏的故事剧情包含着动作游戏的人物行为，而人物行为又包含着杀敌人、取宝物等基本事件。如此便可让角色扮演动作游戏同时拥有 RPG 剧情与动作打斗的成份。

从“暗黑破坏神”发售以后，它改变了游戏产业的视觉热潮。随后有许多游戏仿模仿“暗黑破坏神”的游戏玩法模式出现。在近几年来，动作 RPG 游戏仿佛已经席卷了整个游戏市场，特别是它又加入了网络联机的机制，让玩家不仅可以在单机平台上玩，而且更能够邀请朋友在游戏中大肆杀敌。越来越多玩家接受它的游戏机制，也越来越多游戏厂商投入设计动作 RPG 游戏的行列。而动作 RPG 游戏的刺激感与故事剧情的发展，所带来的娱乐性远远超过单纯的动作游戏或 RPG 游戏。

在下一节中，我们会研究一个与动作 PRG 类型非常类似的游戏玩法，那就是“冒险类游戏”。

7.4 冒险类

如果说冒险类游戏（Adventure Game，AVG）是动作类游戏，不如说是一种动作 RPG 类游戏。其实在很多游戏类型当中，它们都有许多共同点，惟一不同的是它们只有一些特殊玩法机制不太相同而已。针对冒险类游戏来说，它有角色扮演游戏类型的人物特色，却没有角色扮演游戏类型的人物升级系统。简单地说，在冒险类游戏中非常强调人物与故事剧情的进行，可是人物本身的等级强弱却不会是游戏中强调的重点。

7.4.1 冒险类游戏的发展

冒险类游戏的内容含有相当多的解谜与冒险成分，主角的属性通常也是固定的，而游戏本身最主要的目的是要让玩家在游戏中不断地思考，来获得解决问题的方法。例如，“卡普空”（Capcom）公司所发行的“恶灵古堡”（Biohazard）系列游戏与“Eidos”公司所发行的“古墓奇兵”（TOMB RAIDER）系列游戏，虽然它们的故事内容不尽相同，可是它们却都有一个共同点，那就是以解谜为游戏的主轴。

从现在游戏的发展中，我们不难发现冒险类游戏通常以悬念紧张的故事为游戏轴心，例如主角可能会来到一个充满机关的城镇或建筑物中，在这些地方则藏着一些不可告人的秘密或宝藏，玩家必须突破各种机关或关卡，来实现游戏中的目的，这些思考与紧凑的剧情让玩家乐在其中、怡然自得。

7.4.2 冒险类游戏的特点

在制作冒险游戏的时候，可以秉持着下列几项特点：

强调人物的刻划

冒险类游戏强调的是说明角色在故事里的存在性，也就是说各个角色的背景需要非常明确地让玩家了解，让所有在故事剧情里出现的人物都必须要有它们存在的合理性与意义。

合理的故事剧情

冒险游戏非常重视故事剧情的发展，这也是吸引着玩家会想继续玩下去的最有利原因。合理又悬念的剧情让玩家能够很容易地融入在游戏当中，使其会想一直不断地玩下去。

丰富的机关结构

冒险游戏最主要的结构就是以各式各样的机关来构成，情节丰富而且合理，再加上不会太难破解的机关，让玩家都能够流连忘返。而游戏中的机关通常是游戏进行的主要干道，所有的故事剧情都可能在机关的前后时段发生，所以机关便成了冒险游戏不可或缺的主要钢梁。

7.4.3 冒险类游戏的架构

其实冒险类游戏不外乎是动作 RPG 游戏类型的变形，它具有动作 RPG 游戏类型所没有的特性，虽然缺少了 RPG 游戏的角色升级系统，但多了解谜冒险的成分，让玩家除了可以玩到角色扮演的机制外，又可以享受另一种不一样游戏的感受。

冒险类的游戏架构其实大致上与 RPG 游戏类型的架构非常相似，只是冒险类游戏还必须加上大量的合理机关与剧情发展，让玩家感觉就好像在看一场电影、一本小说一样。如果配合复杂一点的设计，它还可在游戏中加入分支的剧情，以提升冒险类游戏内容的丰富性。

当初以美式风格为主的冒险游戏在刚进入我国台湾省游戏市场的时候，有许多玩家很难去接受它的游戏玩法，但从近几年的冒险游戏受欢迎的程度来看，它多彩多姿的内容与美式电影风格的制作手法，确实让冒险游戏成功地融入玩家的心。

7.5 策略类

策略类游戏（Strategy Game，STA）也是属于让玩家融思考于娱乐之中的一种游戏类别。早期的策略型游戏是以战棋为主，如象棋、军战棋等等。主要是要让玩家能够在一场特定的地形中，运用自己的思路来布置属于自己的棋子，以打败对方为目的来进行的一种攻防游戏。早期的战棋游戏是对方走一步，我们才能走一步，而现今的策略型游戏又加入了实时的游戏机制；简单地说，在一个特定的地形中，玩家可以做内政、军事的准备，对方一样也可以做相同动作的准备，不会因为玩家没有动作，对方就不动作，相当符合现今的战略游戏。如果玩家没有善用各种命令来整备，到那时候只有被对方攻打的份了。

7.5.1 策略类游戏的发展

其实策略型游戏发展得相当早，以象棋游戏为类，就是一个非常经典的策略型游戏，以本身所属的战棋，再依据自己的思路布置战棋来攻打对方或防守。象棋是属于较为单纯

的策略型游戏，因为它只能以“一次走一卒”的方式来进行游戏，这不但少了游戏的紧凑性，也少了一分战争紧锣密鼓的乐趣。经过多年来不断地改良游戏，现今策略型游戏中加入了“实时”的游戏机制后，它便成功地打造出策略型游戏的另一个天空。如“Blizzard”公司推出的“星海争霸”（StarCraft）就是一个成功的例子，它是以各种不同的种族为基本单位，而这些不同种族中又更细分出不同能力的小单位，让战棋游戏更添加了不少乐趣，后来“微软”公司所推出的“世纪帝国”（Age of Empires）游戏更征服了整个游戏市场，而后不管是国内或国外更以这种游戏机制推出了大大小小不同的游戏，可见策略型游戏的趋势还是持续看好。

7.5.2 策略类游戏的特色

策略型游戏的最大特色就是在于多人联机的机制。以早期的战棋游戏来说，可能只能让两个人进行游戏，而现今的策略型游戏的主要乐趣就是在于多人联机的厮杀过程，在游戏中可以结盟也可以反目成仇，以两个人的力量来灭掉另一个种族；也可以翻脸不认人，在同盟时期又去杀同盟国。在游戏里可以利用以物克物的方式来攻打对方，对方也能够利用同样的方式来攻打我们，所以策略游戏最大的特色就是在于要如何运用自己的思路来配制战棋兵种以及管理内政或开发。

其实策略类游戏除了战术模式之外，它还包括另一种游戏方式，这种游戏方式就是“经营”。所谓“经营”即是让玩家去管理一种操作系统，例如城市、交通、商店等等。玩家需要凭着自己的经验来经营一种操作系统，像较为有名的经营策略型游戏——美商艺电公司所发行的“仿真城市”系列与“仿真市民”系列。

策略型游戏还有一种较为人知的游戏模式，那就是“养成”游戏机制。如同较为经典的“美少女梦工厂”系列，它是以供养一个小女孩为主轴的游戏，目的是要小女孩成为游戏中有名有利的人物。

还有一种策略型游戏，那就是谋略类。最为代表性的游戏就是“KOEI”公司所出的“三国志”系列游戏。其实它也偏向经营管理，不过此种类型的游戏多了一种可以攻打敌人的游戏机制，游戏整体看起来，就像是古时候的谋略战争，所以也将它归属于策略类游戏。

其实策略类的游戏是所有游戏中包含最多类型的一种游戏模式，不过我们倒可以很容易去分辨它的。只要游戏的模式是让玩家花上心思，来实现另外一种目的所做的游戏模式都能够称得上是策略型游戏。

7.5.3 策略类游戏的架构

策略型游戏的架构较为简单，以现今的游戏来说可分为两大类，分别是“单人剧情类”与“多人联机类”两种。

单人剧情类

以单人单机为主，其目的是让玩家可以操作自己的战棋来破解单关的故事剧情，玩家可以进行丰富的故事剧情，也可以随着自己的思路来布置攻守计算机的战棋，以完成单关的任务。

多人联机类

以多人多机的方式来进行游戏，其目的是让游戏中的玩家可以邀请朋友在游戏中进行一场大厮杀。在没有联机的情况下，玩家也可以与计算机进行对战，以自己的思路来打败对方。

随着网络的蓬勃发展，联机游戏也如雨后春笋般的越来越多。以策略型游戏来说，几乎占掉大部分的网络游戏市场，而使得策略型游戏成为近几年来最为热门的游戏玩法。我们已经说了这么多的游戏类别，可是却一直还没讲到在线游戏的机制类型。说真的，我们不将在线游戏机制另外归类成一种游戏类别，原因是在线游戏的类别还是脱离不了上述所提的几种游戏类型，不同的是在线游戏多了一种网络机制，而且游戏中的数据库与玩法也比上述所提的几种类型更加庞大罢了。

7.6 模拟类

模拟类游戏（Simulation Game，SIM），就是在模仿某种行为模式的游戏系统，最主要的目的是实现生活中某种真实行为。通常仿真类游戏模仿的对象有汽车、机车、船、飞机，甚至于宇宙飞船都有，因为我们可能买不起汽车或飞机，不过却能从游戏中感受到汽车奔驰在马路上的快感或飞机遨翔在天空中的舒畅，这就是模拟类游戏希望带给玩家的最主要乐趣。

7.6.1 模拟类游戏的发展

以模拟类游戏来说，它的由来应该是当时用在模拟飞行机器的操作系统。当时一台飞行机器的造价都非常不高，而对于一个不太熟悉的飞行机器的驾驶员来说，冒然地让他驾驶一台飞行机器是既危险又有可能损失一名飞行员与飞行机器，所以科学家就开始研发一套可以模拟飞行机器各种物理现象与突发状况的飞行练习器以供飞行员练习。对于直接驾驶飞行机器来说，驾驶飞行练习器是可以减少许多不必要的麻烦，这就是模拟类游戏的最初原始构想。

之后这种仿真系统慢慢地被引进游戏当中，当时玩家的心态是自己或许没有办法买到昂贵的飞行机器，但是却能够从游戏中寻找到驾驶的乐趣，因而模拟类游戏也在此时渐渐浮出水面。

演变至今，模拟类游戏开始从飞行的机器，慢慢地衍生到其他的硬件机器上，例如汽车、机车等等。直到现在甚至还能看到模拟云霄飞车、火车及未来机器人的仿真系统。

7.6.2 模拟类游戏的特色

仿真类游戏最大的特色就是在模拟机器的真实感与它的刺激感，而模拟游戏着重于机器的物理原则与大自然不变的定律，让玩家可以从玩游戏中感受到真实存在游戏中的虚拟环境。现今模拟类游戏更朝向人类的虚拟生活目标设计，例如“虚拟实境”（Virtual Reality，VR）；它可以模拟出人类生活的周围环境，如建筑物、花园等等。在建筑物还未建造之前，我们可以利用虚拟实境的技术，先造出一系列的建筑物环境，提供使用者去观赏。

模拟类游戏的最大特色就是可以让使用者或玩家在没有接触真实硬件机器设备之前，可以很容易地感受到这些机器设备所带来的乐趣。

提示：VRML（Virtual Reality Modeling Language，虚拟现实建模语言），它是一种 Web 上描述三维空间的标准表示方法，VRML 是让人们可以利用 HTML 的技术来为 3D 空间建模，然后由浏览器的实际计算来表现这些模型。VRML 并不是一种指令语言，而是某种图像文件格式。VRML 的文件扩充名为“.wrl”，即是“world”，其 MIME 类型目前是 x.world/x.vrml，而最后将注册为 model/vrml。

7.6.3 模拟类游戏的架构

仿真类游戏的架构较重视物体的大自然物理反应，就如一颗铅球由半空中落下，它就不会像羽毛那样，还会随着风而飘动。一个物体的移动，例如车子，它必须符合自然规律才可行，如违反物理原则的话，那么会使得玩家感到无所适从、毫无临场真实感，所以在制作模拟类游戏时，就必须包含许多大自然物理的规律，如风阻力、摩擦力等等。越表现真实，模拟游戏越会吸引玩家。

模拟类游戏具有科学的表现，因为游戏的内容有着许多的数学及物理运算。以现今模拟类型的游戏来看，它拥有一些固定玩家的市场，由于许多人都喜欢车子，可是又买不起真实的车子或者是不能在真实的生活中寻求到赛车的乐趣，所以只能在模拟游戏中去寻找欢乐，就如同接下来要讨论的运动类游戏一样。

7.7 运动类

运动类游戏（Sports Game，SPG）与模拟类游戏有异曲同工之处。运动类游戏也必须符合大自然的规律，不过模拟类游戏较注重机器运行，而运动类游戏就比较注重人体活动行为。什么是人体活动行为呢？简单地说，只要是具备重复性动作的活动都可视为人体活动行为，也称为“运动”，而运动类游戏的目的就是用来实现这种行为的工具而已。

7.7.1 运动类游戏的发展

以早期的运动类游戏看来，运动类游戏仿佛只是为了让喜好运动而不能亲自参与所制作出来的游戏。后来运动越来越让人喜欢，而使得运动类游戏种类大幅增加，从篮球、足球和高尔夫球运动等，慢慢地被带入游戏之中。甚至奥运会中五花八门的各种比赛项目也被带进游戏之中，使得运动类游戏在玩家的心中占有一席之地。

其实运动类游戏也让玩家有着另外一种很奇妙的行为，那就是有许多玩家可能会因为支持真实比赛中的某支队伍，而爱上玩运动类游戏，甚至为了让支持的队伍打败其他的队伍，不惜日夜地锻练支持的队伍，来满足心中的不平与遗憾。

7.7.2 运动类游戏的特色

运动类游戏最主要之处就是在表现某一种类型的运动，在与真实运动相同的环境下，和计算机或朋友打一场属于自己的运动竞赛，以满足自己或朋友对运动的热忱。

运动类游戏以球类运动占大多数，不管是篮球、高尔夫球还是足球，只要是许多人热衷的运动，此种类型的运动游戏的占有率则会越高，最主要的特色就是突显出此类运动所带来的刺激性与满足感。

7.7.3 运动类游戏的架构

其实运动类游戏的架构与动作类架构大致相同，不同的是运动类游戏着重于特定的运动项目，主要的架构会依据运动的类型与规则来制作。例如网球运动，人物架构可能是由一个玩家操作一至两个人，而足球运动则可能由一个玩家来操作一至多人，所以要订出运动类游戏架构就必须按照真实运动类型来定义或规划。

不管是哪一种运动游戏，我们都可以依据下列两项原则来规划运动类游戏：

确认运动的类型

每一种运动的规则与场上人物的配置大不相同，在规划某运动游戏之前，我们必须先了解此种运动的完整规则与人物上场时所分配的位置，甚至于赛场上的战术等等，以提供计算机人工智能制作时所用。

突显运动的特性

每一项运动都有它的基本特性。以篮球运动来说，它的特点就是在两支队伍攻防当中的临场刺激与紧张的赛况。而对于撞球运动来说，它的特点即是着重于球的运动方向与撞球的准确度。上述两者都是属于运动类的游戏，不过玩法却是大不相同，所以为了让玩家能够更容易接受，我们必须突显运动类游戏本身的特点才行。

虽然运动类游戏在游戏市场中，只占了不大的地位。不过由于现实生活中的运动热潮，例如从世界足球赛来看：在比赛的前中后时段里，不知道有多少人为之疯狂，更为足球运动带来无限商机，且带动运动类游戏大为流行。

7.8 益智类

益智类游戏（PUZZLE GAME，PUZ）较着重于玩家的思考与逻辑判断，运用玩家的思路来完成游戏中的目的。通常玩益智类游戏的玩家必须要有恒心与耐心来思索着游戏中的所有问题，再依据自己的判断来执行，目的是突破各项不同的关卡。益智类游戏不会让玩家有拼命按键盘的动作，而所要走的步骤都必须经过自己冷静的思考，在一定的时间内做出正确的判断。

7.8.1 益智类游戏的发展

以最早期的益智类游戏而言，黑白棋与五子棋类型应该是益智类游戏最原始的玩法。益智类游戏的发展，应该是从早期的纸上游戏所衍生出来的，从小时候玩的方块游戏来看，它是由各种不同形状的方块所构成的游戏，目的是将各个方块拼凑在一个方框内，而在多种排列组合之下，玩家必须按照自己的思路来完成拼凑。以上述的方块游戏来看，我们不难发现电子游戏中“俄罗斯方块”游戏的玩法是不是有些相似的地方呢！以“四川省”的游戏来看，它的玩法就是如果两个相同图案的方块连接在一起就可以使方块消失，当所有的方块都消失的时候就可以进入下一关，这种游戏玩法是不是有点像小时候玩的“宾果”游戏呢？

到了益智类游戏的中后期时，它开始以博奕为主要的市场，将益智类游戏制作成平日所玩的纸牌、麻将；将原来只有在桌面上的博奕游戏，搬至电子游戏中。发展至今，更多地如赛马、赛船或赛车等赌博益智游戏出现。

7.8.2 益智类游戏的特色

益智类游戏的特色就比较单纯，在游戏中它不需要以快节奏的方式呈现，反而它比较偏向慢条斯理的玩家，最主要的目的是能让玩家可以去运用自己的思考逻辑来做不同的判断。

7.8.3 益智类游戏的架构

介于益智类游戏的机制大多都是来自早期的纸上游戏，所以在规划益智类游戏的同时，就必须去分析了解到纸上游戏的玩法规则，并且将它加以变化。简单地说，取纸上游戏的主要架构，再加上自己所创造的游戏玩法或机制，就可以制作出与原先纸上游戏更多变化的益智游戏。

其架构可以归纳出几点要素，我们按照这几点要素来制作一套益智类游戏。

游戏的规则与玩法

在制作益智游戏之前，必须先去了解游戏的全盘规则以及它的各种玩法。

游戏的衍生与变化

将原来的纸上游戏转变为另外一种形态来表现，例如上述所讲的“俄罗斯方块”游戏一样，原来只有将固定的方块拼凑在特定的方框内，现在也可以将它转变成由上往下且不固定方块的游戏拼凑方式。

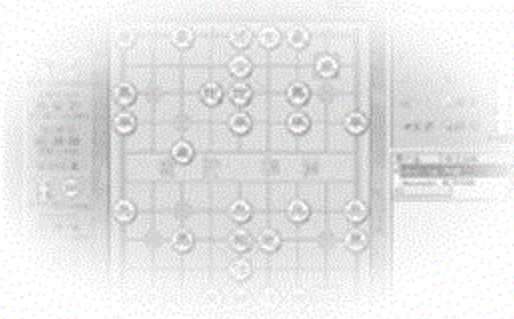
独创性的游戏机制

益智类游戏的变化成分原本就不多，因此必须加入游戏的独创性，让益智类游戏的内容更加丰富有趣。

讲到这里，我们几乎把制作游戏应该要认识的条件都说明了。接着从下一章开始将探讨制作游戏的精华所在——各式各样游戏的基本演算机制。

第 8 章

2D 基本算法



本章将介绍在2D贴图的游戏开发时所有可能会用到的一些基本算法，例如基本贴图、动画贴图、横向滚动条移动、前景背景移动、接触等等，让您不必再为了寻求算法而自寻烦恼。

8.1 2D平面贴图

在图片显像的世界里，影像贴图是一门最基本的课程。在游戏中最吸引玩家的地方就是游戏中的画面。简单地说，只要抓住玩家的味口，那么游戏就会很容易被玩家所接受。

或许您会问道："在2D的游戏中，到底可以做到什么样的程度啊？"其实最笨的方法就是在平面图片上下功夫就可以了。不过，这可能会引发出更多的问题，一来它可能会累垮所有的美术人员，二来游戏画面没有动态的变化，看起来就会显得单纯乏味，就算美术人员针对游戏画面再怎么下工夫，挑剔的玩家还是很难从游戏中得到一点乐趣。

因此，我们就来介绍在2D游戏中经常会使用到的一些基本算法用于提高2D图片的变化规律。

8.1.1 XY坐标系统

在2D游戏的领域里，它的坐标系统较3D游戏简单。只要了解到2D平面中的X与Y坐标的运作原理，那么就可以做出很多种不同的变化。

至于什么是XY坐标系统呢？从数学的角度而言，X坐标代表的是象限中的横向坐标轴；而Y坐标代表的是象限中的纵向坐标轴，如图8.1所示。

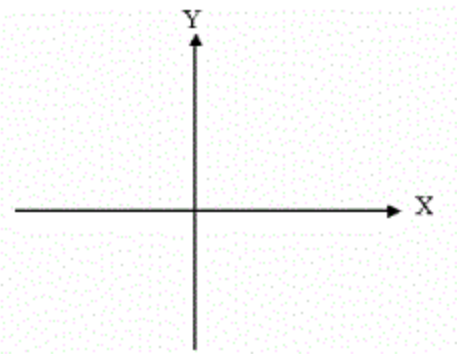


图8.1 XY坐标系统

而屏幕的坐标系统也是如此，只不过在数学上的象限坐标系统中，它所用的Y坐标轴

是向上递增的，而在屏幕坐标系统中，它的 Y 坐标值是向下递增的，请不要混淆了，如图 8.2 所示。

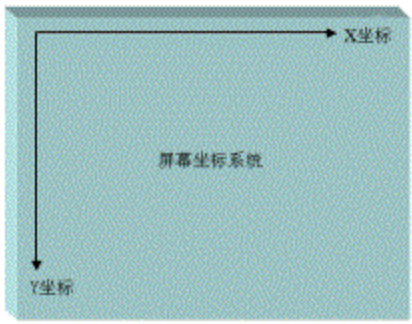


图 8.2　屏幕坐标系统

屏幕中的 XY 坐标系统也可以接受负值（小于 0）的数值，只不过 X 或 Y 坐标如果为负值的话，那么坐标点必然位于屏幕外的坐标系统中。也就是说如果 Y 坐标为负值的话，它是不会被显示在屏幕上的，同理 X 坐标也是如此。屏幕中坐标系统的大小，通常可以利用屏幕的显示分辨率来决定，而屏幕显示分辨率的高或低通常要看显示卡或屏幕设备是否支持来决定。一般而言，我们经常会使用到的屏幕显示分辨率有“320×200”、“640×480”、“800×600”及“1024×768”4 种，它们是屏幕所能对应的坐标点，如“640×480”的解释就是 X 坐标轴上有 640 个像素点、Y 坐标轴上有 480 个像素点的意思。

“320×200”通常被使用在以前的 DOS 环境下，因为以前 DOS 的内存比较少而且又有一定的限制，而“320×200”对内存的用量不会很多（颜色值不多）；所以在 DOS 环境中，常常被游戏定义为最基本显像坐标系统的标准。一直到现在 Windows 时代，“320×200”的显示坐标系统已经不能再满足使用者的需求，甚至这种分辨率已经被淘汰。目前最基本的显示坐标系统的标准为“640×480”，而游戏通常也都使用这个坐标系统。或许您会问：“那为什么不用‘800×600’与‘1024×768’的坐标系统呢？”其实答案是也可以，不过我们必须先考虑到计算机系统是否可以接受这两种坐标系统，而且这些要考虑的问题包括系统内存是否足够、硬件设备是否支持，或者显示卡可不可以使用这两种显像模式等问题。一般在市面上所看到的游戏，虽然可以支持这两种显示模式，不过它们还是会将“640×480”模式作为最基本的显像模式，而另外这两种显示模式只是让拥有高效能计算机设备的玩家所使用，以实现游戏画面的精确程度。

8.1.2 屏幕颜色

对于上述这几种显示模式，它们都有其对应的颜色模式，而可以对应的颜色模式也要看显示卡或屏幕设备是否支持来决定。通常它们所支持的颜色模式可分为 16 色、256 色、

高彩 16 位（2^{16}=65 536）、全彩 24 位（2^{24}=16 777 216）及全彩 32 位（2^{32}=4 294 967 296）等颜色值，因为它们都牵涉到系统与显示卡内存上的限制，所以在设置游戏的颜色显像时，就必须考虑到这个问题。

原本显示卡厂商所定义出来支持高彩颜色值的部分只有 16 位与 32 位，不过在考虑到使用者的设备及高彩颜色值所需的内存前提下，16 位的颜色值似乎太少了，而 32 位的颜色值所占的内存又太多了，所以之后就有某些显示卡厂商才会又定义出在这两者之

间的颜色值 24 位。

8.1.3 基本贴图

从本节，我们将介绍基本 2D 显像技术。在制作游戏的过程中，不管是 2D 或是 3D 的模式，我们必然要先了解最基本的贴图方法。而贴图不外乎就是将图片贴在显示卡的内存上，然后再通过显示卡来呈现在屏幕上的过程，我们可以利用 Windows 提供的最基本显示功能 GDI 或 API 的方式来进行贴图的工作，或者是利用 DirectX 或 OpenGL 的显像技术来实现贴图的目的，这完全是可以自己去决定的。不过可能也要考虑到一些特殊的贴图方式或如何提升贴图的速度等问题，这时如果使用 DirectX 或 OpenGL 开发包来制作贴图的话，效果会更佳。

接下来告诉您图片贴在屏幕上的方法。如同上述所说，屏幕上有 X 与 Y 的坐标点，而图片也有其本身的长与宽，所有的图片都是以矩形的几何图形来表示的。一般说来，要在一张纸上画出一个矩形的几何图形，我们只要知道这个矩形在纸上的左上角坐标及长与宽，就能画出，如图 8.3 所示。

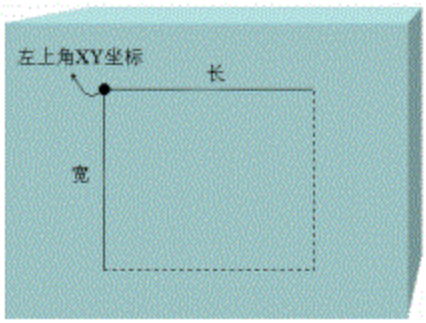


图 8.3　在一张纸上画出一个矩形的几何图形

同样只要知道图片在屏幕上的左上角 XY 坐标及本身的长与宽，就可以将图片贴在屏幕上了。将图片贴在屏幕上之后，我们只要改变其 X 或 Y 点的坐标，那么图片自然就会在屏幕上改变它的显像位置，如图 8.4 所示。

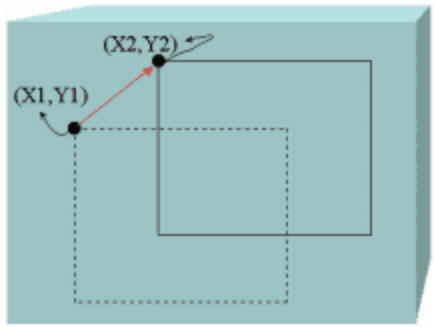


图 8.4　改变 X 或 Y 点的坐标，图片在屏幕上改变它的显像位置

另外如果想要将图片从屏幕的左边慢慢地移动到右边，我们可以编写一个循环程序代码，而这个循环则是用来改变图片在屏幕上的 X 点坐标，让这种效果看起来就像是图片自己在移动一样，如图 8.5 所示。

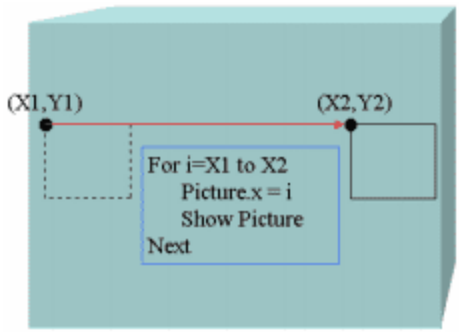


图 8.5　一个循环程序代码，用来改变图片在屏幕上的 X 点坐标

上面介绍的是最基本的图片移动的原理，其实图片移动不只是单纯的横向或纵向移动，它还可以做出很多种不同的运动方式，稍后我们就会一一介绍。

相信您在游戏中可能会看到一连串的人物动作的动画，例如一个武功高强的人发出强大的气功，像这种一连串的动作播放就称为“动画贴图”，在下一节里我们来看看这种基本的动画贴图要如何实现。

8.2 动画贴图

通常在 2D 游戏里所谈到的动画贴图可以将它分成两个部分，一种是图形自己不断改变内部的图片，称为“连续贴图”；另一种是图形的运动过程，称为“粒子运动”。接下来，就来看看这两种动画贴图有何不同。

8.2.1 连续贴图

所谓贴图，它可以包含两个部分，一个是放图片的框框，如同日常生活的相框一样；而另一个是图片，它就如同放在相框里的照片一样，如图 8.6 所示。

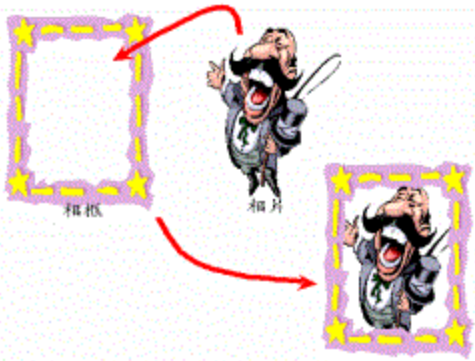


图 8.6　贴图

而连续贴图就是在相框中，一直连续不断地更换里面的相片，这些照片会按照动作的顺序排列，就如同播放卡通一样，卡通的制作就是将一张张图片画一连串的动作，然后再利用播放来实现图片中的物体好像在运动一样的效果，如图 8.7 所示。

图 8.7　连续贴图

如图 8.7 所示，我们将人物的跑步动作分成 6 个，假设一个动作图的长为“W”、宽为“H”，而每一张图的长与宽都是一样的，如果我们要算出某一个图的位置，我们可以利用数学中“等差级数”的公式算出。等级数的公式如下所示：

```
公式：an=a1+（n-1）*d
a1为首项
an为第n项
n为项次
d为等差值
```

由于第一张人物动作图的 X 坐标为“0”，所以我们的“a_1”就会为“0”，不过，我们还是建议读者将这个值也填上去，因为我们以后可能不一定会将物体的连续动作图放在框框内的（0,0）坐标上，所以我们最好是能够自动补上这个值，以识别第一项的值。

举例来说，如果要算出第 3 张 X 坐标上的位置，我们可以将已知的值代入等差级数的公式中，使其得到所需要的值，如下所示：

```
a3 = a1-(3-1)*W
a3 = 0-2W => a3=2W
```

从上面算式中，我们的“a_3”就是第 3 张图的 X 偏移坐标，而第 3 张图的 Y 偏移坐标则是“0”，所以我们便可以很轻易地取出第 3 张的人物动作了，如图 8.8 所示。

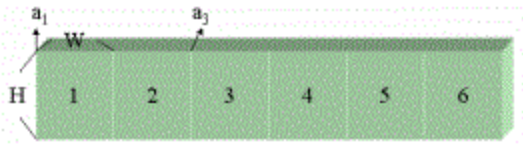


图 8.8　取出第 3 张人物动作

依此类推，如果我们的人物动作要从第 1 张播放到第 6 张的时候，我们可以编写一个循环来取出这几张图片的等差级数 XY 坐标，程序代码如下所示：

```
For i=1 to 6
srcPicture.X = 0 + (i-1)*W
srcPicture.Y = 0
Delay(1)   '暂停1秒
Next
```

如此一来，图形框内便能播放我们的人物连续动画了。

接下来，我们再来看看另外一种排列方式，如图 8.9 所示。

图 8.9　将物体的动作串成一张大图

这一种排列是将物体的动作串成一张大图，而大图中又分成 3 列，分别是“A”、“B”与“C”，如果只计算 A 排中的某一个图素，相信对您来说已经不算什么难事了，我们只要使用上述所提过的等差级数便可以得知“1”、“2”或“3”的图素坐标。如果要读取 B 与 C 排的图素坐标，那么就必须分别算出它的偏移“w”值与“h”值，假设今天要在图档里读

取第 5 号图素的 XY 坐标值，其方法如下所示：

```
W5＝w  '此为第5号图素的左上角X坐标
H5＝h  '此为第5号图素的左上角Y坐标
```

上述算式是利用肉眼的方式从图 8.9 中看出来的，万一如果图素数量一多的话，那么再用肉眼去辨别势必会很难，不过我们可以使用一种公式来解决这一问题。从上面的例子来看，已知的值有横向总张数 3 张、纵向总张数 3 张，而以横向坐标（X 坐标）来说，我们可以取 5 除以 3 的余数来作为第 5 张横向的张数；同样地，我们也可以取 5 除以 3 的余数来当作是第 5 张纵向的张数，其公式如下所示：

```
Wn＝Ax1+[（n MOD 横向总张数）-1]*单张宽度
→W5＝0+[（5 MOD 3）-1]*w
Hn＝Ay1+[（n MOD 纵向总张数）-1]*单张长度
→H5＝0+[（5 MOD 3）-1]*h
```

也可以将公式编写如下所示：

```
Wn＝Ax1+（n /横向总张数）*单张宽度
→W5＝0+（5/3）*w
Hn＝Ay1+（n/纵向总张数）*单张长度
→H5＝0+（5/3）*h
```

因为以除法而言，不管是横向或纵向都能够取得目前张数减 1 的值，所以我们不必要再自行减去 1 了。现在用上面的公式来算出第 5 张图素的位置到底是多少，如下所示：

```
W5＝0+[（5 MOD 3）-1]*w
→W5＝0+[1-1]*w→W5＝w
H5＝0+[（5 MOD 3）-1]*h
→H5＝0+[1-1]*h→H5＝h
```

从上面所得到的结果是不是与肉眼所看到的值是一样呢？

这种做法能够被使用在方格图标的演算上，不管图素的数量有多少，我们都可以利用上面的两个公式来算出我们所要的图素坐标值。

8.2.2 粒子贴图

或许您会这么问：“粒子系统到底是什么东西啊？”所谓粒子系统是将大自然中的物体运动和自然现象，用一系列的粒子来描述，再将这些粒子运动的轨迹映射到显示屏幕上，接着便可以在屏幕上看到物体运动和自然现象的模拟效果了。

关于粒子系统，它可以在屏幕上表现出许许多多的特殊效果，例如火焰、火苗、瀑布、雪花飞舞等等。其实粒子系统不怕做不到，而怕我们想不到而已，只要发挥丰富的想象力

便可以创造出意想不到的效果来。

好了，现在开始正式切入本节的主题。粒子系统的效果与应用，关键就是在于如何描述粒子的运动轨迹，也就是构造这些粒子的运动函数，而函数选择的恰当与否，则决定其粒子效果的逼真程度。

粒子系统中的每一个粒子就好像有生命一样，每一个粒子都有生与死的变化、循着轨迹的运动，所以我们可以将粒子定义出如下 4 个基本特性：

- 生成位置。
- 生命值。
- 速度与方向。
- 加速度。

接下来，我们介绍并解释这些粒子所拥有特性的内容。

生成位置

这个特性是决定粒子开始的初始位置，在粒子系统中，每一个粒子的生成位置可以在同一个地方，例如瀑布特效；也可以在不同的地方，例如雪花特效。

生命值

这个特性是决定粒子在特效系统内的存在时间，每一个粒子的生命值都不固定，有的比较短、有的比较长，如火焰。也就是说有的火苗可以飘的较高较久、而有的火苗可以存活的时间就较为短暂。

速度与方向

每一个粒子都有方向的运动轨迹特性，在粒子生成的时候，它也会有运动方向，且有一个基本的飘移速度值，如图 8.10 所示。

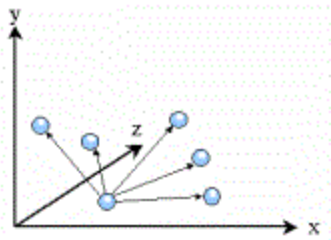


图 8.10　粒子的运动轨迹特性

加速度

其特性是让每一个粒子看起来更加地逼真。正如同大自然中，一个物体从高处往下掉，在它还未到达地面之前，它的速度也就会越来越快，我们就称之为“加速度”。

谈到这里，相信您对粒子系统还是模糊不清吧！其实简单地说，我们只要将粒子系统做到类似于大自然所包括的规律，那么它的效果就可以非常地真实了。

在这里，我们还不会谈到其特效与真正的算法，下一节中再针对某些特殊的分子系统来探讨其粒子运动算法。首先，我们就来谈谈几个较为常见的粒子系统原理。

8.2.3 火焰粒子

火焰算法在 2D 特效中可以说是最简单的一种粒子效果，而构成火焰效果则需要以下几个条件或过程：

热源

热源可利用各种随机函数在发热体表面上随机播撒一些火种就行了，这个过程的关键是如何选择恰当的随机函数。简单地说，我们可以在火焰特效中将每一个粒子的生成位置当成是一个热源。

热传播

热传播的算法有很多，不过每一种算法都大同小异。按照常理，火焰是从下往上传播，在计算热传播的时候，我们先求出每一点热能的平均值之后，再将所有点往上平移一个单位就行了。热能的平均值算法一般是取前一点周围各点的平均值即可，如图 8.11 所示为热的传播方式图。

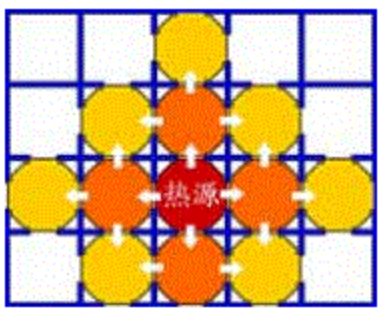


图 8.11　热的传播方式

热损失（冷却）

热损失过程只是为了调节火焰的高度，使得火焰看起来更逼真一些。以上有关热源与热传播两个过程是火焰算法的躯体，而热损失则是火焰算法的精神。前面两个过程产生出来的火焰可以说是死火，毫无生气，因为在现实生活中是有气流的存在，所以火焰在向上传播的过程中要受气流的影响。只有将现实中的种种干扰因素考虑进来，如此一来生成的火焰才会富有真正生命力。简单地说，这个过程则是将粒子的生命值特性归于 0，相对地，在生命值递减的时候，我们也将粒子的颜色值由深变淡，直到完全消失为止。

8.2.4 瀑布粒子

瀑布特效也是一种很简单的粒子系统，它是利用抛物线的方式来进行粒子的运动，如同在上物理课的时候，老师会将一颗球从桌子的一端慢慢地让它滚向另一端，等到球离开桌子之后，球会做一个抛物线的运动，直到落到地上，如图 8.12 所示。

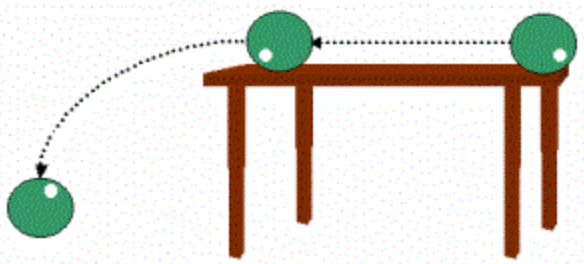


图 8.12　球从桌子滚下的抛物线轨迹

根据物理学的原理，如果推球的力量越大，球的抛物线弧度则越大，如图 8.13 所示。

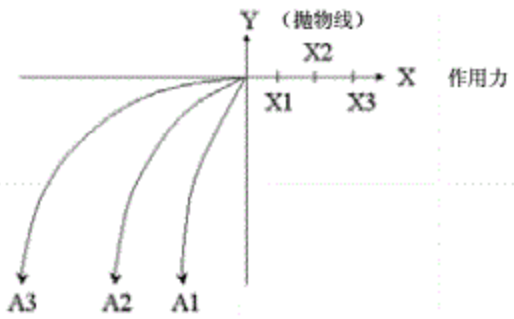


图 8.13　作用力与抛物线示意图

而瀑布的原理也是这样，如果水力较大，其喷射的抛物线也较大，而瀑布的粒子也会有粒子的 4 种基本特性，并且粒子在高处也会受到地心引力的影响而产生加速度的力量，所以在计算瀑布粒子的时候，也就必须在每一个瀑布粒子上加上加速度的参数来增加逼真感。

8.2.5 雪花粒子

雪花特效在粒子系统里受地心引力的影响没这么大，所以它停留在空中的时间相对地也会较长，不过它所受的风力影响却可以大大地改变其每一个粒子的运动方向，其运动的方式如图 8.14 所示。

图 8.14　雪花特效在粒子系统里受地心引力与风力的影响

在大自然的世界里，风不可能只有向着同一个方向吹，风可能会从东吹到西，也有可能从西吹到东，而形成一股气流，所以如果雪花粒子要显得更逼真的话，我们可以将雪花粒子向着某一个方向吹（所有的粒子都是要加上这个值），在下一时间再往回走一个随机数单位（小于主风力的随机数值），而在同一个时间里再慢慢往下移，这样便形成了犹如雪花般的特效了。

在游戏刚开始发展的时候，有许多游戏就已经包括了滚动条的效果，而且横向滚动条的游戏都能够被许多人所接受，不过这些人却不能够了解这种原理要如何制作，在下节里，就让我们来深入探讨 2D 游戏里常见的横向滚动条的运作原理。

8.3 横向滚动条移动

横向滚动条的技术常被使用在 2D 的游戏中，如大型游戏机台上较为流行的“越南大战”系统游戏。现在还有一些游戏更结合了横向滚动条的技术与 3D 场景的特效，让 2D 的游戏场景看起来更显得逼真，如 PS 主机平台上的“恶魔城——月下夜想曲”。

8.3.1 循环贴图

其实横向滚动条最简单地作法就是将背景画成一张大型的图形，然后再依据显示模式

的大小来抓取要显示的部分，举例来说，如果显示模式为“640×480”，而背景图是一张“1024×480”的大型图形，如果要将图放在屏幕中的话，我们就会在屏幕上看到如图 8.15 所示的情况。

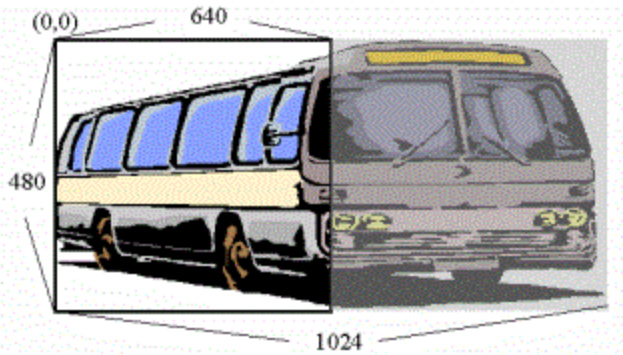


图 8.15　不同的显示模式所显示的图像

右边灰色屏蔽的部分是在屏幕上看不到的部分，所以我们只能在屏幕上看到公交车的左半部，如果要看到公交车的右半部的话，就必须将屏幕的画框移向公交车的右半部，但事实上这种作法是办不到的，因为我们不可能真的去移动屏幕的画框，所以如果要看到公交车的右半部，势必要去移动公交车的图才行，如图 8.16 所示。

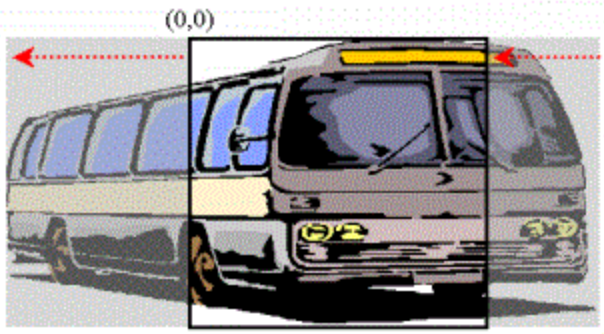


图 8.16　右边灰色屏蔽的部分是在屏幕上看不到的部分，我们只能在屏幕上看到公交车的左半部分

如果要观看背景图上的（X1,Y1）坐标，而且画框长为“W”、宽为“H”的话，我们可以从背景图的（X1,Y1）坐标点上取得长为 W，宽为 H 的图框，并且将它贴在屏幕的画框上（贴在显示内存上）。以 Direct Draw 的贴图函数为例，其语法如下所示：

```
画框.BltFast（画框上的左上角X坐标，画框上的左上角Y坐标，原始图,Rect
(X1,Y1,W,H)）
```

如此一来，便可以看到大型图形的全貌了，如此类推，我们也可以将大型图形的某一部分取出，然后再贴到要贴的任何位置上了，如图 8.17 所示。

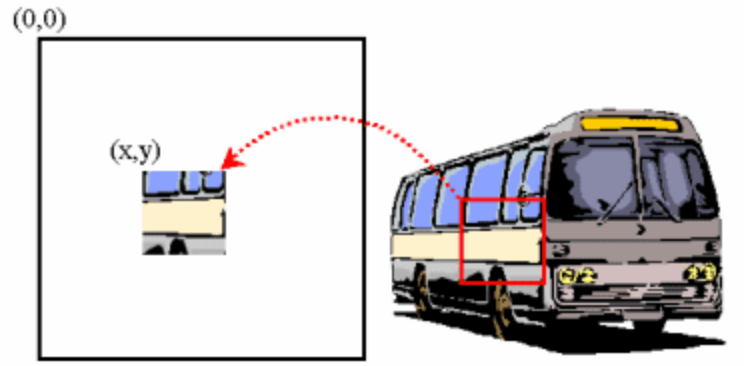


图 8.17　取出大型图形的某一部分再贴到要贴的任何位置上

其实我们还可以将这种做法衍生成另外一种表现方法，那就是“循环贴图”。所谓循环贴图就是将一张图的前页贴在自己的后页上，并且以循环的方式播放，形成一种生生不息的动画，如图 8.18 所示。

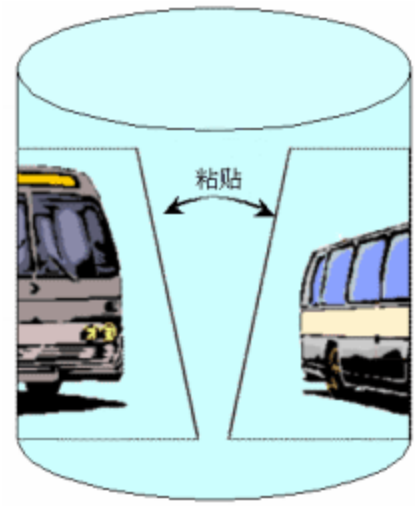


图 8.18　循环贴图

不过实际上是不可能将计算机里的图卷成圆筒状，所以只有将多余的部分再贴往画框中没有贴图的地方，即可做到这种效果，如图 8.19 所示。

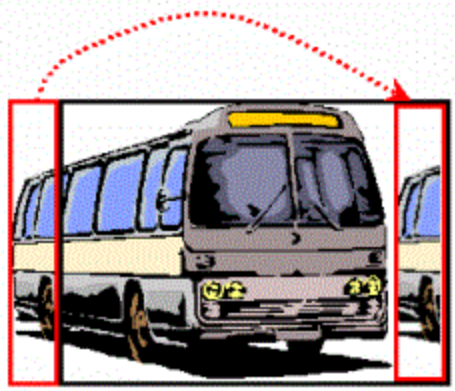


图 8.19　将多余的部分再贴往画框中没有贴图的地方

在这里利用前面所讲过的公式，将多出来的图块再复制到画框中的最右边，如此一来不就实现循环贴图的目的了吗？举例来说，假设背景图的长宽为 640×480，而屏幕的画框长宽也是 640×480，如图 8.20 所示。

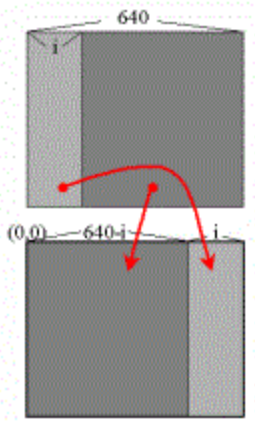


图 8.20　将多出来的图块复制到画框的最右边

从图 8.20 可以看出，如果要做循环贴图的话，势必要将左半部的图贴在画框中的右半部，其宽度为“i”值，而原来的左半部要贴在画框上的（640-i,0）的位置上；原来的右半部要贴在画框上的（0,0）的位置上，其程序代码如下所示：

```
For i = 0 To 640
   Blt 0, 0, srcPic, Rect (i, 0, 640 -i, 480)    '原来的右半部
   Blt 640 -i, 0, srcPic, Rect (0, 0, i, 480)    '原来的左半部
```

```
Next
```

如此一来，我们便可以将背景图做一次循环贴图，不过，如果我们要做无穷循环的话，我们只要将程序代码修改成如下所示：

```
i=0
Do
   Blt 0, 0, srcPic, Rect (i, 0, 640 - i, 480)    '原来的右半部
   Blt 640 - i, 0, srcPic, Rect (0, 0, i, 480)    '原来的左半部
   i=i+1
   If i>640 then i=0
Loop
```

将原来的程序代码修改完毕之后，背景图便可以一直重复不断地做循环贴图的动作了。

在了解到循环贴图的原理之后，接下来便可以做出许多不同的变化，例如将背景图由左向右移入、由右向左移出、由上往下移入、由下往上移出等等移入移出的效果，现在就来看看这些效果应该要如何才能做到。

例如，如果背景图要从屏幕左方往屏幕右方移入的话，就必须去抓取会显示在屏幕中的部分背景图，如图 8.21 所示。

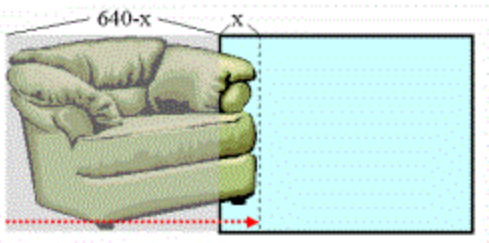


图 8.21　从屏幕左方往屏幕右方移入背景图

由上图可以得知背景图在屏幕中向右移动了“x”个单位，在这里我们可以利用前面介绍过的公式将部分的来源图取出，其程序代码如下所示：

```
Blt 屏幕左上角X坐标, 屏幕左上角Y坐标, 来源图, Rect（来源图左上角X坐标, 来源图左上角Y坐标, 来源图长, 来源图宽）
→Blt 0, 0, srcPic, Rect（640-x, 0, x, 480）
```

如此一来便可以取得要贴在屏幕上的部分背景图了，接着将 x 值编写成一个递增的循环便可以做出如同背景图由左向右移入的效果了，程序代码如下所示：

```
For x = 0 To 640
   Blt 0, 0, srcPic, Rect（640-x, 0, x, 480）
```

```
Next
```

如果将背景图由左向右移入的公式反相回来，我们便可以做到如同背景图自右向左移出的效果，程序代码如下所示：

```
For x = 640 To 0 Step -1
   Blt 0, 0, srcPic, Rect (640-x, 0, x, 480)
Next
```

同样的，如果要将背景图自上往下移入或自下往上移出的话，其方法都是一样的，不同的是原本只改变 X 坐标的数值，而现在只要将它改成 Y 坐标即可，如图 8.22 所示。

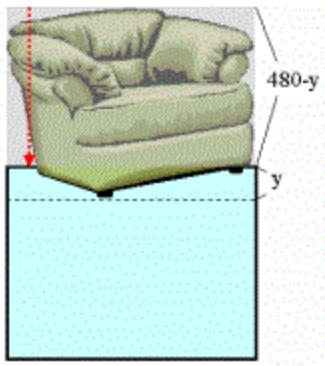


图 8.22　自上往下移入或自下往上移出背景图

其程序代码如下所示：

```
' 自上往下移入
For y = 0 To 480
   Blt 0, 0, srcPic, Rect (0, 480-y, 640, y)
Next
' 自下往上移出
For y = 480 To 0 Step -1
   Blt 0, 0, srcPic, Rect (0, y, 640, 480-y)
Next
```

接下来再看看一种不一样的算法，那就是对角移入与移出。

因为屏幕的横向与纵向的比例不可能会等于“1”（640/480），所以如果将 X 坐标与 Y 坐标各减 1 的话，则不能由这种运算法来看到背景图的全部画面，如图 8.23 所示。

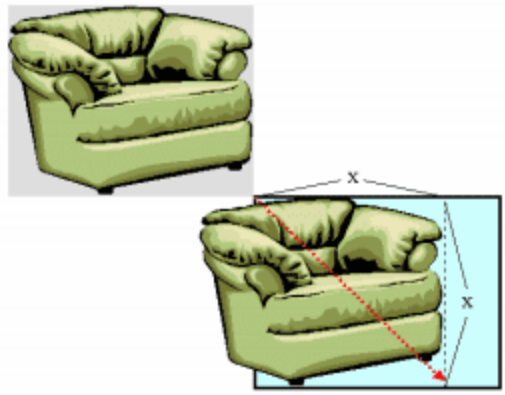


图 8.23　将 X 坐标与 Y 坐标各减 1 的话，则不能由这种算法看到背景图的全部画面

如果以对角的方式移入的话，就必须做到纵向坐标移动一个单位，横向坐标移动“x”个单位的方式来显示，而这个比例公式可以将它写成：

```
x:1=640:480 →x=640/480
```

简单地说，纵向坐标移动一个单位，横向坐标就要移动“640/480”个单位，如此一来就可以利用对角的方式将背景图移入到屏幕中了，其偏移量可以写成：

```
Offset_x=640/480
Offset_y=1
```

接下来要计算出应该要抓取来源图的哪一部分，其预备抓取的来源图大小，如图 8.24 所示。

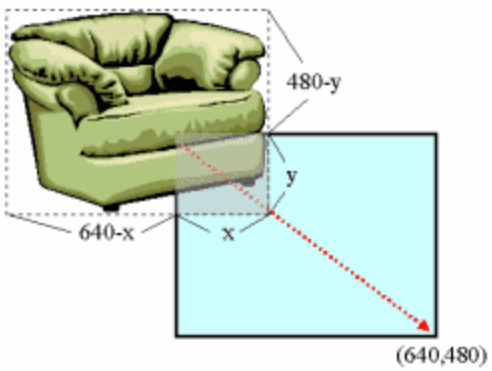


图 8.24　计算要抓取来源图的一部分

其程序代码如下所示：

```
For i=480 to 0 Step -1
        y=i
         x= (640/480) *y
        Blt 0, 0, srcPic, Rect (0, 0, x, y)
Next
```

其实只要记住这一种抓取部分来源图与贴在屏幕上位置的技巧，便可以做出许多不同的变化来，甚至可以加入下一章会谈到的物理原理，让 2D 贴图的技巧更加华丽。

在这个例子中，已知的数值为来源图的长宽和移动的偏移比例，所以我们可以利用这两个已知的值来求出“x”与“y”的坐标，其公式如下所示：

```
x= (640/480) *y
→x=1.3y
```

换句话说，如果 Y 坐标移动 1 个单位的话，相对地，X 坐标就要移动 1.3 个单位，而要取得来源图的范围则可以将公式写成如下所示：

```
Blt 0, 0, srcPic, Rect (640-x, 480-y, x, y)
```

我们将程序代码整理一下，就得到如下所示的程序代码：

```
For i=0 to 480
   y=i
   x=1.3*y
  Blt 0, 0, srcPic, Rect (640-x, 480-y, x, y)
Next
```

如此一来便能将背景图向着屏幕的对角移动了。同理，我们也可以利用这种公式将它由右下角移动到左上角，不同之处是抓取来源图是从左上角开始抓，如图 8.25 所示。

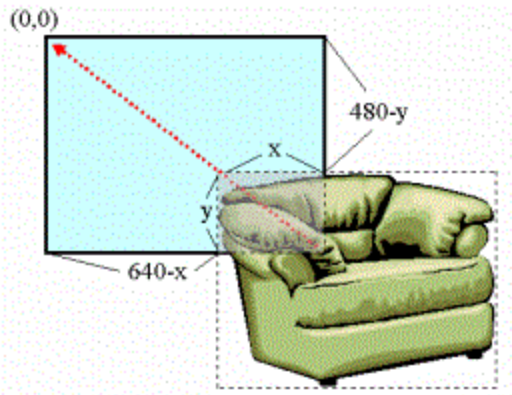


图 8.25　由右下角移动到左上角

其程序代码如下所示：

```
For i=480 to 0 Step -1
  y=i
  x=1.3*y
  Blt 0, 0, srcPic, Rect (0, 0, x, y)
Next
```

8.3.2 人物与背景

在 2D 游戏中，主角人物或者是敌人与背景图的互动是最常见的，通常主角或敌人是不可能直接通过所谓的障碍物，它们可能要跳起来或者是将障碍物击破，如图 8.26 所示。

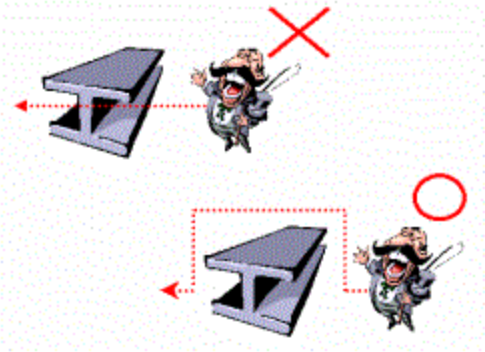


图 8.26　主角或敌人是不可能直接通过障碍物的，它们要跳起来或者将障碍物击破

这种必须要跳跃的障碍物，称之为“屏蔽点”。这种屏蔽点的目的是告诉玩家这个地方不可以直接通过，而在横向滚动条的游戏上应该如何才能让这些屏蔽点可以跟着背景图移动，如图 8.27 所示是一个简单的数组屏蔽图。

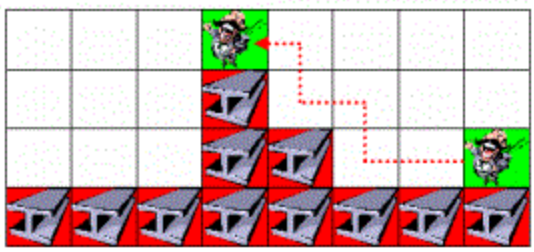


图 8.27　数组屏蔽图

在数组中设置障碍物的代表值为“1”，而可以让主角通过的数组值则设置成“0”，如

下所示：

```
A(8,4)={
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,1,1,0,0,0,
1,1,1,1,1,1,1,1}
```

假设游戏一开始是显示在第 4 行到第 8 行的数组图，如图 8.28 所示。

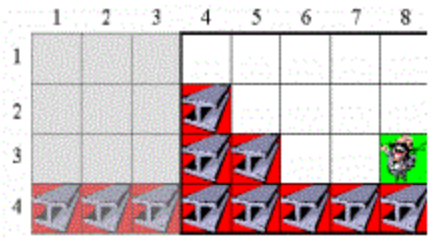


图 8.28　显示在第 4 行到第 8 行的数组图

然后我们将主角往前移动一格，而背景屏蔽则向后推一格，如图 8.29 所示。

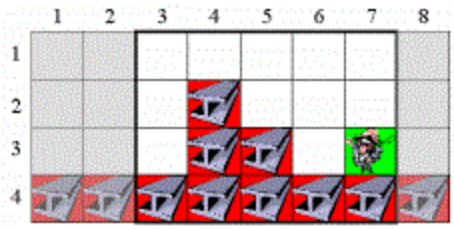


图 8.29　将主角往前移动一格，背景屏蔽则向后推一格

此时所呈现的是第 3 行到第 7 行的数组值，而现在所说的情况就是我们这里所要讨论的重点。

在这种情况下就是要求出可以显示的数组坐标值，如同上面的例子一样，游戏一开始是显示在第 4 行到第 8 行的数组值，所以我们将显示数组值的程序代码写成：

```
a=4
For i=a To a+4
   For j=1 To 4
Draw(i,j)
   Next
Next
```

如此一来便可以利用数组里的值（1 或 0）来判断是否要显示障碍物到屏幕上，如此类推，我们就可以做到显示所有数组移动后的画面了。

上面的例子算是比较简单的，但是如果背景图是一个横向与纵向都可能会改变的情况下，那么公式会不会更加复杂化呢？其实不然，只要修改前面所谈过的公式就可以了，而对于这种情况便可以迎刃而解。

举例来说，如果背景数组是如图 8.30 所示的话。

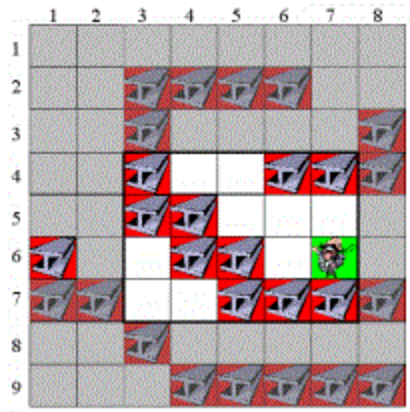


图 8.30　背景数组

主角原来是在（7,6）的数组坐标上，然后我们将主角移动到（5,5）的数组坐标上，相对的屏幕画框的左上角方块要从（3,4）移动到（1,3）的坐标上，如图 8.31 所示。

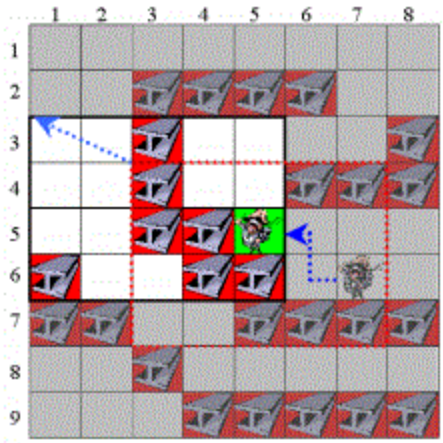


图 8.31　相对的屏幕画框的左上角方块要从（3,4）移动到（1,3）的坐标上

如此一来，我们就必须计算出正确的屏幕坐标点位置。从上面的例子中，已知的数值是人物的（X1,Y1）数组坐标，而屏幕左上角的方块坐标是（X2,Y2），而 X1 与 X2 的差值

为“4”，可以得到下列式子：

$X_2=X_1-4$

所以屏幕左上角的方块坐标就会为（X1.4,Y2）。如果“X1-4”的数值小于“0”的话，就必须将 X 坐标设置成 0，这样才不会出现“数组索引超出范围”的错误。

至于“Y2”坐标值的处理方式那就见人见智了，在这个范例中，我们将它设置成主角人物数组坐标的中间值，而屏幕的纵向坐标占了 4 格数组单位，所以我们将主角人物放在其中间的下一格，如图 8.32 所示。

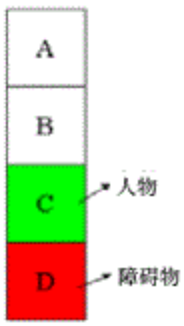


图 8.32　将主角人物放在中间的下一格

然后把人物数组坐标的公式编写成如下所示：

```
C=（A+D）/2+1
```

因为“D=A+3”，所以公式被修改成：

```
C=（2A+3）/2+1
```

而变量“A”是我们必须求的值，所以我们又可以将公式修改为：

```
C=（2A+3）/2+1
→A=C-5/2
```

我们将“C”的值代入上述的公式就可以得到答案：

```
A=5-5/2=3
```

我们将程序代码整理如下：

```
ManX1=5 '人物的X坐标数组单位
ManY1=5 '人物的Y坐标数组单位
MapX1=ManX1-4  '背景图的X坐标数组单位
MapY1=ManY1-5/2 '背景图的Y坐标数组单位
```

如此一来，我们就可以将上述的公式写成一个函数来供程序所调用，如同这一种情况，我们可以将它变化在各种大型的数组坐标上，只要我们使用上述算法，再一一求出必要的公式，我们就可以很容易地得到各种不一样的数组移动变化的函数了。

在我们介绍完单纯地移动背景之后，其实我们还可以在游戏中加上一个前景来增加游戏画面的立体感，接下来，就让我们来深入探讨这种 2D 游戏中的三维效果算法。

8.4 前景与背景的移动

在许多 2D 的游戏中，它们会使用一种如同建筑美术的图像表现法，我们称它为“透视图”，它可以用来加强 2D 游戏的画面立体感，让玩家不是在 3D 的显像技术里也能感受到游戏画面的立体感，以增加游戏画面的品质，本节可帮助美术人员作为 2D 游戏构图的参考。

8.4.1 透视图

透视图在建筑美术设计的领域里，它有 3 种较为特殊的表示法，分别称为“一点透视法”、“二点透视法”及“三点透视法”，接下来，我们就来一一介绍它们有何不同。

一点透视法

纵向的直线与消失点的水平线是呈垂直的，而纵向的直线与横线则呈平行，如图 8.33 所示。

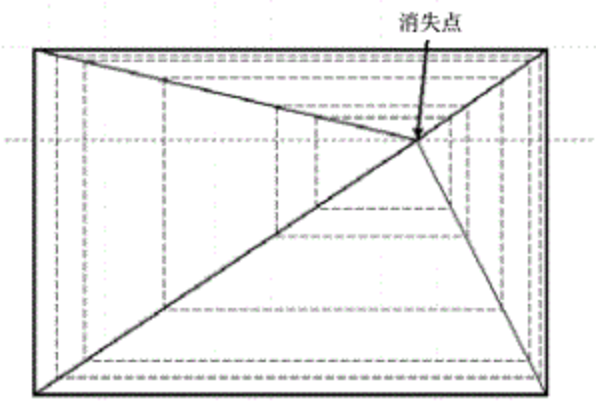


图 8.33　纵向的直线与横线呈平行

二点透视法

二点透视法就是一个画面中有两个消失点的透视图，通常我们用这种技术来表现建筑物正面与侧面同时存在的图画，如图 8.34 所示。

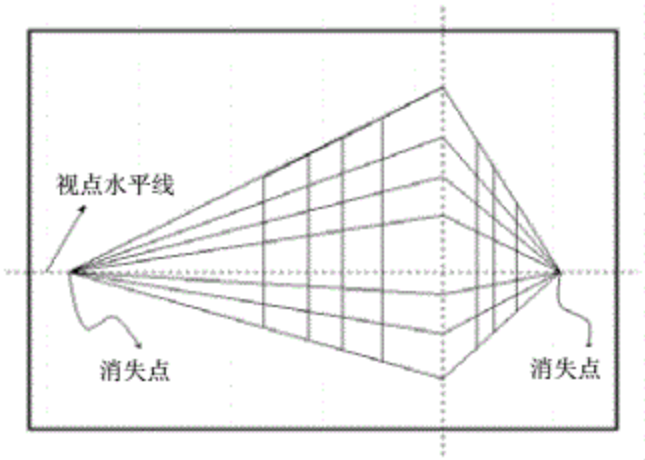


图 8.34　一个画面中有两个消失点的透视图

从图 8.34 中我们可以得知直线与视点水平线是呈垂直相交的，不过，它与一点透视法不同的是其横线与视点水平线并没有平行。

在二点透视法的技术中，我们可以将建筑物画成如图 8.35 所示。

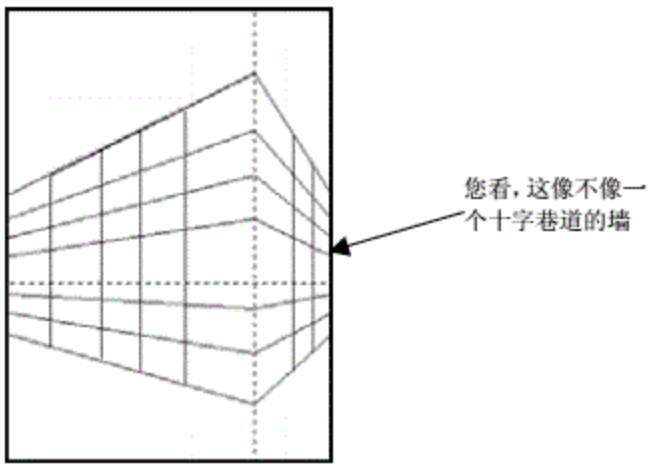


图 8.35　二点透视法中的建筑物

从图 8.35 中看来，虽然它没有明显的消失点，不过它还是地地道道的二点透视图，那另外两个消失点呢？其实这两个消失点是在图的外框中，从建筑美术透视学的角度来看，它是可以成立的。

三点透视法

顾名思义，三点透视法就是有 3 个消失点，以一个美术人员而言，它是 3 种透视图中比较难的一种，如图 8.36 所示。

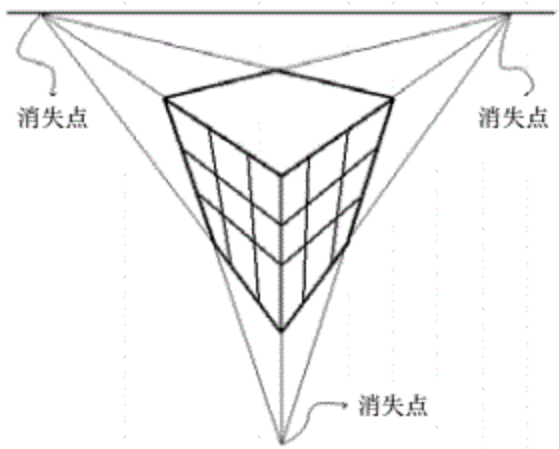


图 8.36　有 3 个消失点的三点透视法

不过，这种透视图表示法一般不会用在游戏中的动画中，所以我们在这里就不多作解释了，接下来，我们就来看看上述一点透视图与二点透视图在游戏中应该要如何表示。

8.4.2　游戏中的透视图

我们就先以一点透视法来讲解 2D 游戏中表示立体的奥妙吧！以一点透视法画出的图如图 8.37 所示。

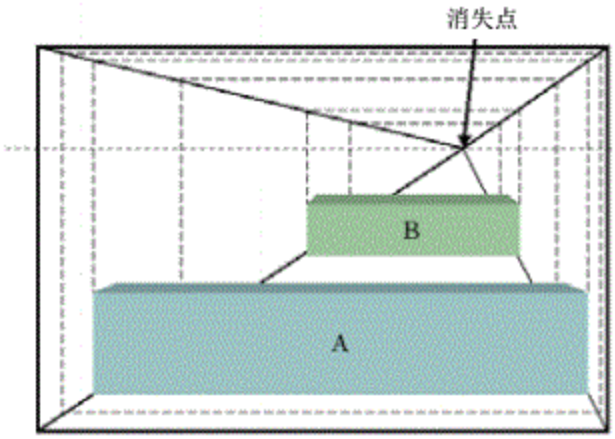


图 8.37　以一点透视法画出的图

我们以 A 面为前景，B 面为背景，相信各位读者都坐过火车或汽车吧！距离我们较远的物体，其离开我们视线的时间也就比较慢；而距离我们较近的物体，其离开我们视线的时间也就比较快。同样道理，当我们的视线从图 8.37 中的 A 面右边移向左边的时候，A 面的移动速度就会比较快，相对的 B 面的移动速度也就会比较慢，如图 8. 38 所示。

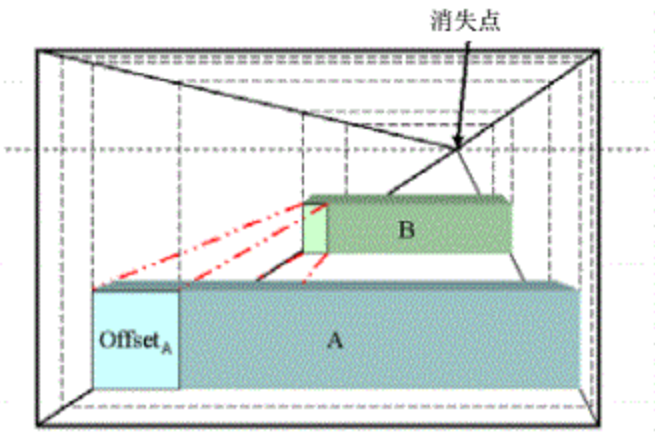


图 8.38　视线由 A 面的右边移向左边

而我们在游戏中要表现出前景与背景不同的前提下，我们就必须将前景与背景的移动速度设置为不同的值，如此一来，我们便能表现出远近的视觉效果了。

从图 8.38 中我们可以看出 A 面中的“$Offset_A$”比 B 面中的“$Offset_A$”还要大，所以

在同一时间，我们就必须将 A 面移动 $Offset_A$ 的长度，而 B 面则是与 $Offset_A$ 长度的等比例的移动。

如果 A 面的长度是“W_A”的话，“$Offset_A$”在 A 面长度中的比例就为“$Offset_A/W_A$”，而 B 面则是“$Offset_B/W_B$”，A 面与 B 面的比例是相等的，所以我们可以将其公式编写成如下所示：

$Offset_A:W_A=Offset_B:W_B$→解成表达式：

$Offset_B=Offset_A*W_B/W_A$

从上面公式中，我们已经知道的变量值为 W_A、W_B、$Offset_A$ 的值，所以我们便可以很轻易地算出 $Offset_B$ 的值，接下来，我们只要移动 A 面的 $Offset_A$ 值，同时再移动 B 面的 $Offset_B$ 值，如此一来，我们就可以看出前景与背景的立体变化了。

近年来，2D 游戏有采用斜角视觉技术的趋势，其目的是让 2D 游戏看起来更现实化，如果 3D 游戏采用 45° 角的拍摄效果，而斜角视觉的效果我们应该如何表示，其算法要如何编写呢？在下一节里，我们将深入讨论。

8.5 斜角视觉

近几年来，斜角视觉的场景效果在 2D 游戏中倍受好评，它也将 2D 平面的游戏带进 3D 立体的效果变化，其 PC 游戏中较为有名的为“仙剑奇侠传”，在当时，他们就是以这种斜角视觉的场景效果，虏获了不少玩家们的心。

8.5.1 图块

我们知道，游戏中的场景是由一定数量的图块（tile）所拼接而成的，就像是在铺设我们自己家中的地板瓷砖一样。在 2D 游戏中所采用的场景图块形状可分成两种：一种是“矩形图块”，另一种则是“菱形图块”，因此我们在编写场景编辑引擎的时候就应该按照图块形状的不同而编写不同的拼接算法，这两种图块拼接引擎的优劣性，我们在这里就不加以讨论了，不过有一点我们可以肯定的是这两种方法之间，谁也无法否定谁。

我们在这里使用少量的图块来构造一个较大的场景，这样做的优点是可以减少内存的消耗、便于计算从一处走到另一处所要消耗的时间或体力（通过率）、物体间的遮掩、动态场景的实现等等，我们可以利用定义图块的属性来完成一些现实模拟的计算。斜角视觉所采用的图块是由一些上、下、左、右都对称的菱形所构成的，其横向宽度与纵向宽度之比通常为 2:1，如图 8.39 所示。

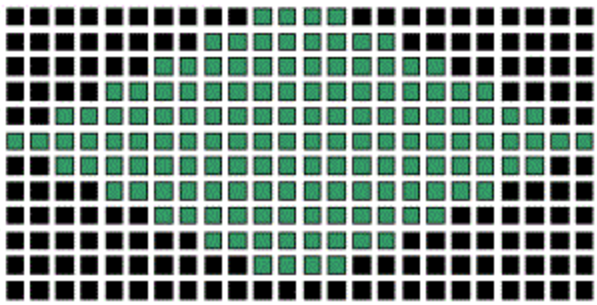


图 8.39　上、下、左、右都对称的菱形，其横向宽度与纵向宽度之比为 2:1

通常我们在常用的是图素宽为 32 个像素点、高为 16 个像素点的矩形图块，从图 8.39 中我们可以将四周黑色区域做透明化。如果我们细心一点的话，势必可以看出，在矩形图块的底部有一行像素点也是透明的，也就是说实际高度是 15 个像素点，为什么呢？原因是这样的，我们先来看看这两张矩形图块的合并图，如图 8.40 所示。

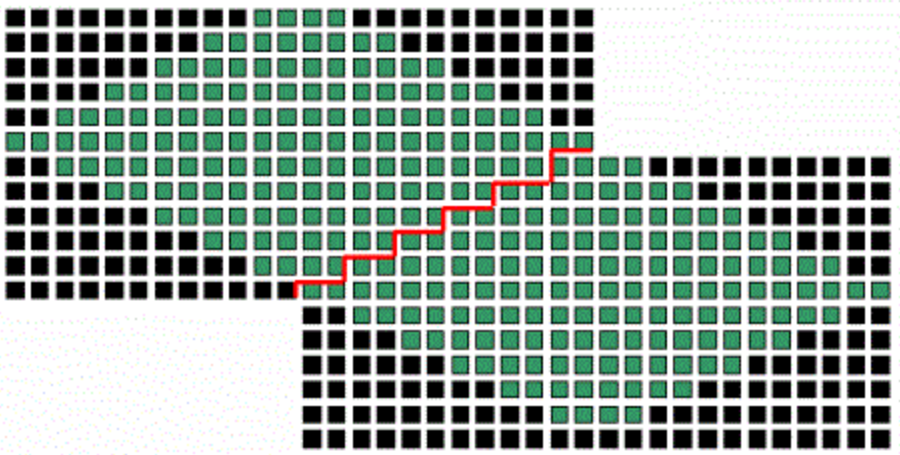


图 8.40　两张矩形图块的合并图

从图 8.40 中我们可以得知，从当前图块位置右移了 16 个像素点，再往下移 8 个像素点就是右下方图块的位置，其目的是将两个图块拼接。16 和 8 分别为横向宽和纵向高的一半，16 和 8 分别为 2^4 和 2^3，这对于图块的拼接与优化算法是多么好的两个数字。接下来就是如何让每一个图块定位，其方法也有很多种，最早的 RPG 游戏采用的方法是以左上角到右下角为 X 轴方向，而 Y 轴方向则为右上角到左下角方向。这种编号虽然很简单，不过场景的 4 个角落就成了黑暗区域，这就是美中不足的地方。另一种方法是直接用屏幕坐标来进行编号，由于图块是交错式的排列，这种排列方式带给我们编号上的困难。目前常用

编号方式有两种，分别为“奇偶行排列”与“奇偶行并成一行”，如图 8.41 所示。

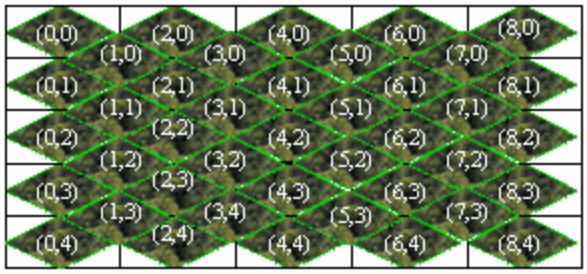


图 8.41　奇偶行排列

现在我们把所有横坐标为奇数的图块拿走，如图 8.42 所示，这样是不是与我们的直视角有些类似了呢！

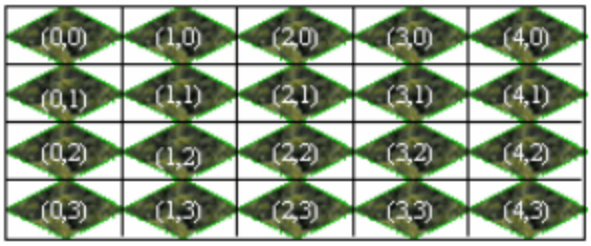


图 8.42　移去所有横坐标为奇数的图块

接下来，我再利用奇偶行并成一行的方式编写出坐标换算的引擎。

8.5.2 坐标换算

坐标换算是斜视角引擎中首要解决的问题之一，它是场景拼接的前提，这也就是说为什么斜角视觉比直角视觉计算还要复杂的原因之一。前面我们把奇数列的图块拿走之后，我们可以看到的是一个类似直视角这样的一个场景地图，如图 8.43 所示。

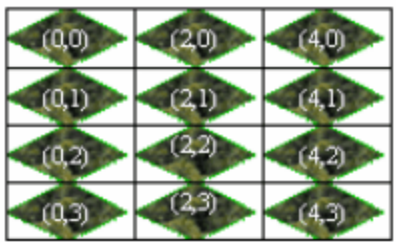


图 8.43　类似直视角的场景地图

说到这里，大家可能已经想到我们可以利用类似直角视觉的方法来进行坐标换算了。例如，地图中的某一点，它的绝对像素坐标为（x, y），而我们要算出它的绝对图块坐标为（tx, ty），首先该点一定是位于如图 8.43 所示的某个矩形区域中，假设是该点位于（2,1）的矩形区域中，其次该点可能落在该矩形区域里的 5 个区域内（菱形内和菱形外其他 4 个角落），如图 8.44 所示。

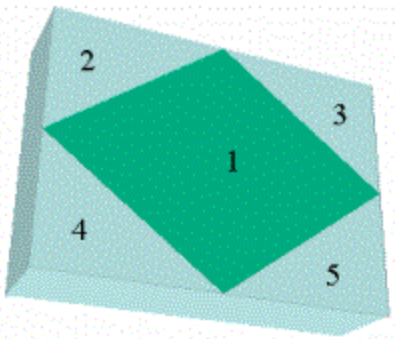


图 8.44　地图中的某一点落在该矩形区域里的 5 个区域内

最后我们再根据菱形的前后左右关系，就可以很方便地确定出该点所在图块的编号了。其演算的方法如下所示：

（1）首先我们将矩形区域里的 5 个区域内的数据定义为：

```
m_CellFigue[16][32] =
{
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2},
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,2,2},
  {1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2},
  {1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2},
  {1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2},
  {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2},
  {1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2},
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
  {3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4},
  {3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4},
  {3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4},
  {3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4},
  {3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,4,4,4,4,4,4,4,4,4},
  {3,3,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,4,4,4,4,4,4,4,4,4,4,4,4,4,4},
  {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,4,4,4,4,4,4,4,4,4,4,4,4,4},
```

```
  {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4}
};
```

（2）接下来，我们再定义小矩形图块的基本长宽：

```
int m_nCellWidth = 32;
int m_nCellHeight = 16;
int m_nCellWidthShift = 5;
int m_nCellHeightShift = 4;
// 根据像素坐标计算出绝对图块坐标
// int nPixelX  横向像素坐标
// int nPixelY  纵向像素坐标
// int nCellX  绝对横向图块坐标
// int nCellY  绝对纵向图块坐标
// BOOL bAbsolutePixel  传入的像素坐标是否是绝对坐标
```

（3）最后，我们便可以编写一个坐标换算的函数了：

```
void Cell (int nPixelX, int nPixelY, int &nCellX, int &nCellY, BOOL
bAbsolutePixel)
{
  // 先计算出传入像素点落在哪个（偶数列）图块所在的矩形区域内
  if (bAbsolutePixel)  // 运算绝对坐标
  {
    nCellX = (nPixelX >> m_nCellWidthShift) << 1;
    nCellY = nPixelY >> m_nCellHeightShift;
  } else// 是屏幕坐标，则要先换算成绝对坐标后再计算
  {
    nCellX = (((nPixelX-m_rcDest.left) +m_nPosX) >> m_nCellWidthShift)
<< 1;
    nCellY = ((nPixelY-m_rcDest.top) +m_nPosY) >> m_nCellHeightShift;
  }
  // 计算传入点在小矩形区域内的相对位置
  int nPx, nPy;
  if (bAbsolutePixel)  // 绝对坐标
  {
    nPx = nPixelX & (m_nCellWidth-1) ;
    nPy = nPixelY & (m_nCellHeight-1) ;
  } else// 相对坐标
  {
    nPx = ((nPixelX-m_rcDest.left) +m_nPosX) & (m_nCellWidth-1) ;
```

```
    nPy = ((nPixelY-m_rcDest.top)+m_nPosY) & (m_nCellHeight-1);
  }
  // nPx与nPy即为数组的相对坐标
  // 求出小矩阵内的位置
  switch ( m_CellFigue[nPy][nPx] )
  {
  case 1:
nCellX-; nCellY-; break; // 左上角
  case 2:
nCellX++; nCellY-; break; // 右上角
  case 3:
nCellX-; break;  // 左下角
  case 4:
nCellX++; break;  // 右下角
  }
}
```

上述函数中的“m_rcDest”是相对于屏幕的一个窗口（View Port），也就是说地图只在屏幕中的某个矩形区域内移动。另外“m_nPosX”和“m_nPosY”是该矩形窗口左上角的绝对像素坐标。

如此一来，我们便可以轻易地知道目前人物所在数组矩形图块的位置了。

斜角视觉的基本坐标运算讲解到这里，我们已经解决了如何计算定点所在图块的编号，这对于接下来的图块拼接演算占了非常重要的地位。在此之前我们还有一个小小的问题，那就是如果我们已知的值是一个图块编号的话，那么我们又要如何计算它的绝对像素坐标或相对像素坐标呢？还有我们又要如何计算其他相邻图块的编号呢？这些问题我们在以后编写路径搜索 AI 等其他特定的场合都会使用到，虽然目前我们还不会用到，但是这是属于坐标计算的范围，所以我们就在这里提前讨论了。

我们先来看看应该如何将绝对图块坐标换算成绝对像素坐标。在这里，我们再利用前面的图，如图 8.45 所示。

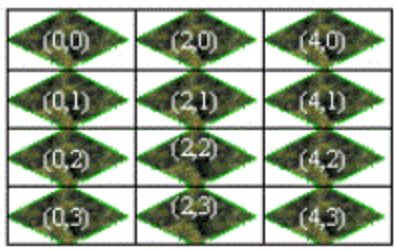


图 8.45　将绝对图块坐标换算成绝对像素坐标

我们已经知道了偶数列的图块有图 8.45 这样一个类似直角视觉的特性。那么同前面讨论的一样，如果定点的图块在偶数列的话，那么我们可以直接计算其坐标位置即可，但是如果定点是位于奇数列图块的话，我们则还需要在相对应的偶数列图块坐标上加上半个图块的偏移，其程序代码可修改为：

```
// 根据绝对图块坐标计算出像素坐标
// int nCellX  绝对横向图块坐标
// int nCellY  绝对纵向图块坐标
// int nPixelX  横向像素坐标
// int nPixelY  纵向像素坐标
// BOOL bAbsolute  计算绝对像素坐标还是相对（屏幕）像素坐标
void Cell2Pixel (int nCellX, int nCellY, int &nPixelX, int &nPixelY, BOOL
bAbsolute/*=TRUE*/)
{
  // 先计算偶数列图块的绝对像素坐标
  nPixelX = (nCellX >> 1) << m_nCellWidthShift;
  nPixelY = nCellY << m_nCellHeightShift;
  // 计算图块的绝对像素坐标
  if (nCellX & 1)
  {
    nPixelX += (m_nCellWidth >> 1);
    nPixelY += (m_nCellHeight >> 1);
  }
  if (!bAbsolute)
  {
    // 计算相对（屏幕）坐标
    nPixelX -= m_nPosX;
    nPixelY -= m_nPosY;
  }
}
```

接下来，我们再来看看应该如何计算相邻图块的编号，其图块的编号会有两种情况，一种是位于奇数列，如图 8.46 所示。

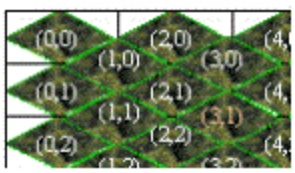


图 8.46　位于奇数列的相邻图块的编号

另一种是位于偶数列，如图 8.47 所示。

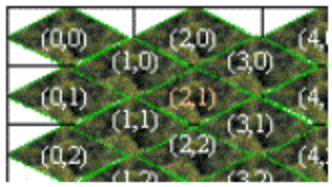


图 8.47　位于偶数列的相邻图块的编号

从图 8.46 和图 8.47 可以看出，我们在计算坐标的时候只要区分这两种情况就行了，其程序代码如下所示：

```
// 计算相邻图块编号
// 方向:
// 3 4 5
// 2  6
// 1 0 7
void NextCell (int &x, int &y, int nDirection)
{
  if ( x & 1)    // 奇数列
  {
    switch ( nDirection )
    {
    case 0 :
y = y + 1;
break; // 下
    case 1 :
x = x - 1;
y = y + 1;
break; // 左下
    case 2 :
x = x - 2;
break; // 左
    case 3 :
x = x - 1;
break; // 左上
    case 4 :
y = y - 1;
break; // 上
```

```
    case 5 :
x = x + 1;
break; // 右上
    case 6 :
x = x + 2;
break; // 右
    case 7 :
x = x + 1;
y = y + 1;
break; // 右下
    }
  }else // 偶数列
  {
    switch ( nDirection )
    {
    case 0 :
y = y + 1;
break; // 下
    case 1 :
x = x - 1;
break; // 左下
    case 2 :
x = x - 2;
break; // 左
    case 3 :
x = x - 1;
y = y - 1;
break; // 左上
    case 4 :
y = y - 1;
break; // 上
    case 5 :
x = x + 1;
y = y - 1;
break; // 右上
    case 6 :
x = x + 2;
break; // 右
    case 7 :
```

```
x = x + 1;
break; // 右下
    }
  }
}
```

如此一来，我们便可以很轻易地求出各相邻图块的编号。

8.5.3 组织图块

接下来，我们再来讨论应该如何组织图块单元的属性。斜角视觉与直角视觉都有一个本质的区别，那就是用于拼接地图的图块都是有高度的，而我们可以通过这一个属性来方便实现掩码算法。

首先我们先来看一些传统 RPG 中的地图格式，一般的地图可分成两层或三层，第一层我们称为“基本图块层”（Base Layer），如草地、路等，这一层是由一些完整菱形图块所组合而成的。第二层则是“边缘图块层”（Fringe Layer），这一层中的图块主要都是一些不完整的菱形图块，如树木、花草、石子等等，通过这些图块的缝隙我们可以看到第一层（Base Layer）的图形，这也就是实现遮掩效果的一大关键，如图 8.48 所示。

图 8.48　通过这些图块的缝隙我们可以看到第一层（Base Layer）的图形

其房子的墙角有些图块只有半个菱形，这就是所谓的“边缘图块”，而这些边缘图块（墙角）覆盖在其他基本图块（草地）之上，当人物走到墙角的时候，我们可以看到的只有没被遮住的半个人。举一个例子来说，例如长在草地上的小花，当人走在这里的时候，花草就会被人的脚遮住。有些游戏还会使用到第三层图形，而第三层图形是用于存放一些可以被捡拾的道具或宝物，如武器、药品等等，所以第三层图形则被称为“Object Layer”。

一般地图的每一个单元都占 2 个字节（Bytes），相当于 16 位（Bits），而第 1 个字节（Bytes）是用来存储图块单元的编号，第 2 个字节则是用来存储图块的标志属性，其中第 2 个字节

中的低 4 位是此单元的高度，第 5 位至第 8 位依次为图块单元的最高位（与第 1 位所组成的图块单元编号）、障碍标志（人物是否可以通过）、人物标志（此图块是否有人站在上面）、地图链接标志（当人走到这里时可以切换到其他地图上）。其各位所包含的意义如表 8.1 所示。

表8.1　地图单元格各位的含义

位	含义
15 bit	地图链接标志
14 bit	人物标志
13 bit	障碍标志
12 bit	编号高位
8～11 bit	单元高度
0～7 bit	图块单元编号低8位

从表 8.1 中我们可以看出图块编号占了 8 个位，换句话说，构成地图的图块最多不超过 255（$2^8-1=255$）块，对于早期的游戏来说，内存资源是相当节省的。不过，现在就不一样了，我们可以利用更丰富的图块来建构我们的地图，对于表 8.1 中的格式，我们可以做适当的扩展。如此一来，不仅可以提高游戏中的速度，更能够使游戏场景更加丰富。为了方便起见，我们还是暂时沿用表 8.1 中的地图格式。其地图结构程序代码可定义为：

```
struct Tile
{
  BYTE    index;
  BYTE    flag;
};
struct Cell
{
  Tile    base;
  Tile    fringe;
};
class Map
{
  BYTE    m_CellIdxFlg;       // 图块索引高位
  BYTE    m_CellRmrFlg;       // 障碍标志
  BYTE    m_CellNpcFlg;       // 人物标志
  BYTE    m_CellLnkFlg;       // 地图链接标志
  Cell     * m_pMapData;      // 地图数据
};
```

8.5.4 图块拼接

我们前面所介绍的内容其实都可以说是斜角视觉地图实现的基础，当我们了解了这些基础知识之后，接下来，我们就可以开始做场景的显示了。

我们已经知道，场景中的每一个图块都是以交错式的方式排列而成的，所以当我们在画矩形图块的时候，就应适当地考虑周边的情况，否则矩形图块四周就会产生锯齿的效果，如图 8.49 所示。

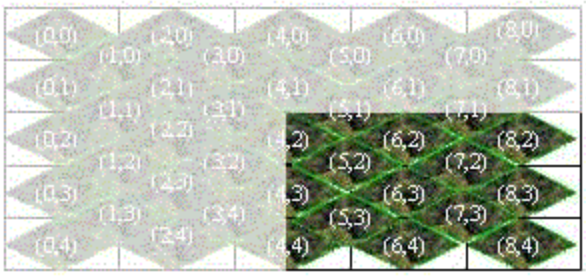


图 8.49　矩形图块四周产生锯齿的效果

如果从图块（2,1）开始画起，则图块（3,0）、（5,0）、…以及（1,1）、（1,2）、…就会被漏画。其实一开始我们就已经提到过，所有的斜线方向相邻的图块其位置都相差半个图块的宽度，而图块的次序与图块编号是一致的，为了不让场景产生锯齿的效果，我们就必须在每次画出图块的时候多画一些不完整的图块。所以我们在这个例子中，只需将显示的区域适当地加大一些就可以了。

为了方便计算，我们将起始图块向左上方向移动一些，例如，如果起始图块为（3, 4）的话，其图块就从（2, 4）开始，如果起始图块为（2, 4）的话，则图块就从（0, 4）开始，也就是说开始与结束都从偶数列开始计算。

8.5.5 人物遮掩

人物的遮掩我们可以将它分成两种情况，一种是人物与人物之间的遮掩，另外一种是人物与地图中的建筑、树木等障碍物之间的遮掩。

第一种情况的解决办法就是通过一个具有位置属性的基础图块，而此基础图块上又衍生出其他的图块，这样我们就可以从视觉的角度对人物的位置进行排序了，从远到近分别画出各图块与人物，如此一来，我们便可以实现人物的遮掩了。当然排序算法可按照个人的喜好而定。

至于第 2 种情况，相信大家一定还记得我们前面所说过的，每一个图块是有高度的，而图块的高度又是如何来定义的呢？我们来看看图 8. 50 这幅图。

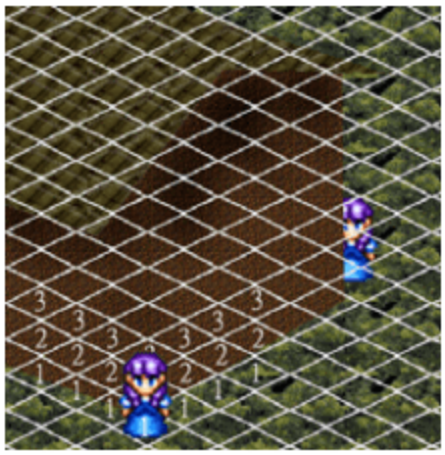


图 8.50　图块的高度

一般建筑物图块的高度与现实是一致的，图 8.50 中的房子从墙角往上的高度依次分别为 1、2、3、…，而这些是我们在编辑场景的时候就必须定义好的。人物也是有高度的，它的高度是从下往上依次递增，如图 8.51 所示。

图 8.51　人物的高度从下往上依次递增

这些编号与地图中图块的编号有些类似，只不过是上下倒过来罢了。这些高度就是为了确定图块遮掩而定义的，我们在显示一个场景的时候，一般会先画地图，接着再画人物，因为可能人物会被建筑物遮住，所以这时我们就要重画人物位置的部分地图场景，前面我们所讲的图块高度属性现在就可以派上用场了，从下往上依序比较人物与图块的高度，如果图块的高度大于人物的高度则图块就是遮住人物，所以此图块要重画；如果是人物遮住图块的话，则图块不须重画。

其实斜角视觉与直角视觉之间的优劣有许多层出不穷的新观念，例如现实的表现方面直角视觉似乎不如斜角视觉，而直角视觉却可以让人觉得地图不那么复杂混乱，这些优劣性的表现则是见仁见智了。

在 2D 的游戏中经常会遇到的是主角与敌人的砍杀，而这必须会使主角与敌人的图形碰在一起，这种原理我们称它为“碰撞”，在看完前景与背景的变化之后，接下来，我们再来看看物体碰撞的算法应该如何计算。

8.6 碰撞

在游戏中，碰撞是一种最基本的算法，其算法又可以分成好几种，例如，人物与敌人的碰撞、飞机与子弹的碰撞或者是为了某些特殊的事件而产生的碰撞等，换句话说，游戏中的碰撞是在某一种条件下而产生的，其目的可能会让人物失血、死亡、取得宝物等等。

8.6.1 平面碰撞

在这里我们先来探讨游戏中较为简单的平面碰撞算法，在前面我们已经提过，在游戏中的所有图素都是以一种几何图形来构成的（一般为矩形位图），而平面碰撞这一种算法首要的条件是必须取得矩形图的左上角坐标与右下角坐标，如图 8.52 所示。

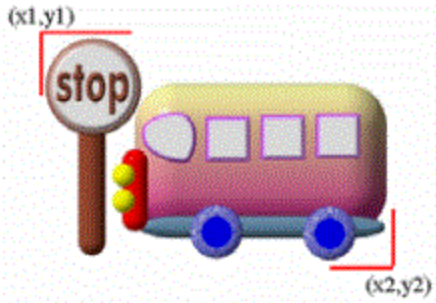


图 8.52　取得矩形图的左上角坐标与右下角坐标

如果我们要判断一个不定变量坐标是否碰撞到矩形图的话，我们只要判断这个不定变量坐标的 X 值是否在 X1、X2 之间和 Y 值是否在 Y1、Y2 之间，即可知道这个不定变量坐标是否碰撞到矩形图了，如图 8.53 所示。

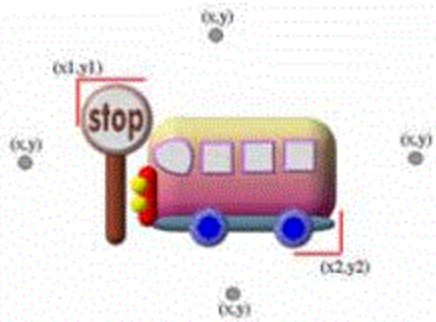


图 8.53　判断一个不定变量坐标的 X 值

在矩形图中判断不定变量的坐标是否存在的程序代码如下：

```
bool inx,iny;
if ( x>=x1 )
{
   if ( x<=x2 )
   {
      //不定变量坐标的 x 值在矩形图之内
      inx = 0;
   }else
   {
      //不定变量坐标的 x 值在矩形图的右方
      inx = 1;
   }
}else
{
   //不定变量坐标的 x 值在矩形图的左方
   inx = -1;
}
if ( y>=y1 )
{
   if ( y<=y2 )
   {
      //不定变量坐标的 y 值在矩形图之内
      iny = 0;
   }else
   {
      //不定变量坐标的 y 值在矩形图的下方
      iny = 1;
   }
}else
{
   //不定变量坐标的 y 值在矩形图的上方
   iny = -1;
}
if ((inx || iny) == 0)
{
   //不定变量坐标的坐标在矩形图之内
}
```

如此一来，我们可以知道不定变量坐标是否碰撞到矩形图。

相信各位读者可以看出，上面的程序代码真像老婆婆的裹脚布，又臭又长，其实我们这样写程序代码是为了让读者了解判断条件的方法，如果我们真正编写判断是否有无碰撞的程序代码时，只要编写成如下所示即可：

```
if ( x>=x1 && x<=x2 && y>=y1 && y<=y2 )
{
  //不定变量坐标的坐标在矩形图之内
}
```

是不是非常简单呀？这就是平面碰撞的基本条件公式。

在 2D 的游戏里，矩形图的碰撞判断是属于较简单但不精确的条件表达式，如果我们要求较为精确的碰撞判断，我们可以利用球面判断法，接下来，就让我们来看看球面的碰撞公式要如何编写。

8.6.2 球面的碰撞

球面是一个圆形的几何图形，它不像矩形可以利用四角的坐标来判断不定数坐标是否在球面之内，如果我们坚持要使用四角坐标来判断球面的话，那么将会产生如图 8.54 所示的情况。

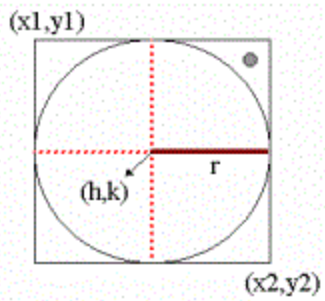


图 8.54　不定数坐标以四角碰撞而言，它已经算碰撞到了，但是实际上它还未碰撞到球面

所以在这个时候，我们可以利用数学公式的“圆方程序”来求出不定数坐标在球面的哪一个地方，其圆的方程序公式如下所示：

```
若圆心为（h,k），半径为r，则可得方程序 '(x-h)²+(y-k)²=r²'
```

例如，有一个球面的圆心为（2,1），半径为 2，其圆的方程序如下：

```
（x-2）²+（y-1）²=2²→（x-2）²+（y-1）²=4
```

其程序代码我们可以将它编写成：

```
void CheckCircle(int x,int y)
{
    int point,r2;
    r2 = 4;
    point = spr(x-2) + spr(y-1);
    if ( point == r2)
    {
        //不定数在球面边缘上
    }else if ( point > r2)
    {
        //不定数在球面外
    }else if ( point < r2)
    {
        //不定数在球面内
    }
}
```

您看很简单吧！编写好球面判断的方程序之后，我们只要代入不定数坐标的 X、Y 值就可以求出它位于球面的哪里了。

接下来，我们再来看一种比较特别的碰撞判断法。

8.6.3 人物的碰撞

上面所讲碰撞判断式都是一些基本的几何图形，而它们都可以使用一些数学公式来作为碰撞的基本条件，不过在游戏中，不管是主角、敌人或宝物它们都是没有固定角度的图形，如图 8.55 所示。

图 8.55　在游戏中，不管是主角、敌人或宝物都是没有固定角度的图形

如此一来，我们判断是否碰撞的精确度就更加困难了。实际上判断这一类碰撞的条件并不难，我们只要了解其中的道理，判断碰撞的程序代码编写起来就显得容易多了。

例子如，如果今天我们有两张没有固定角度的图形需要判断是否有无碰撞的话，我们势必要利用这两个图形的屏蔽图，什么是屏蔽图呢？屏蔽图就如图 8.55 右方的那一张黑白图，为什么我们要多做一张黑白屏蔽图呢？那是因为等一下我们要使用这一张黑白图来与另一张黑白图做 AND 的运算，其 AND 的布尔运算规则如表 8.2 所示。

表8.2　AND布尔运算

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

其目的是要让黑色与黑色撞在一起的时候，可以得到一个“1”值来作为是图形与图形碰撞到了，如图 8.56 所示。

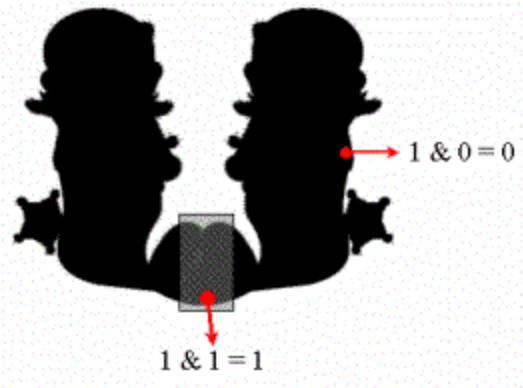


图 8.56　得到一个“1”值来当作是图形与图形碰撞到了

假设人物屏蔽图的长为“w”，宽为“h”，而背景图则存储在 DC 中，其判断是否有无碰撞的程序代码为：

```
void CheckTouch(HDC &srcDC)
{
    int touch;
    for (int i=0;i<w*h;i++)
    {
        for (int j=0;j<srcDC.Width*srcDC.Height;j++)
```

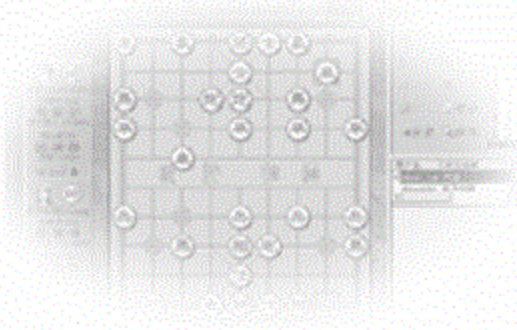
```
        {
            touch = DestDC.Pixel[i] & srcDC.Pixel[j];
            switch (touch)
            {
            case 0:
                //没有碰撞
                break;
            case 1:
                //有碰撞
                break;
            }
        }
    }
}
```


我们在这里要特别提醒读者，这一种判断碰撞的方法是利用一个像素与另一个像素的AND运算来做碰撞的标准，所以这一种判断碰撞的方法在执行速度上也就会比较慢，请尽量不要使用在大图形上。

谈到这里，我们几乎将市面上2D游戏所常用到的算法都介绍完了，相信读者对于2D的算法有了一定的认识了，而3D算法比2D算法复杂，对于初学者来说，3D的算法常常会伤透了初学者的脑筋。接下来，我们再来介绍一些较为常用的3D基本算法。

第 9 章

3D 基本算法



3D 的算法比 2D 算法复杂，对于初学者来说，3D 的算法真是伤透了脑筋，而本章将介绍基本的 3D 显像算法，例如基本的二维与三维坐标的转换，平行与远景三维坐标转换等，以及一些配合 DirectX 或 OpenGL 的 3D 算法。

9.1 3D 坐标系

从计算机坐标学来说，坐标转换是一门非常重要的课题，因为它的作用是用来产生对象让我们可以看到，其中不同的坐标系统所呈现出来的方式会有所不同，如同我们以不同的角度去观看一个特定的目标，从不同角度所看到对象的显示效果会有所不同。

9.1.1 坐标转换

不管是在 2D 或者是 3D 的环境中，我们为了能够明确地表现物体位于空间的位置，通常我们会使用一种坐标系统来表示它。坐标系统必须有一个基本的原点位置，从原点再延伸出去 2 个或 3 个坐标轴，形成一个特定的空间，这个空间即是我们所谓的 2D 空间或 3D 空间。

如果在空间中形成两个以上的坐标系统，我们就必须使用其中的一个坐标系统来描述其他不同的坐标系统，这些不同的坐标系统必须经过一些特殊的转换才能让这个坐标系统接受，而这种转换的过程我们就称为“坐标转换”。

对于计算机显像而言，它只能表现出 2D 空间的坐标系统，而在我们脑海中的三维虚拟空间坐标系就势必要在计算机屏幕上显示。因此我们就必须将三维空间中的物体转换成屏幕所能够接受的 2D 坐标系统。这整个过程，通常我们会使用“Model”、“World”及“View”这 3 种坐标系统与 4 种不同的转换方式来表现。接下来，我们就来介绍这些不一样的坐标系统与坐标转换的方法。

9.1.2 Model 坐标系统

Model 坐标系统即是物体本身的坐标环境，物体本身也有一个原点坐标，而物体其他的参考顶点是由原点所衍生出来的，如图 9.1 所示。

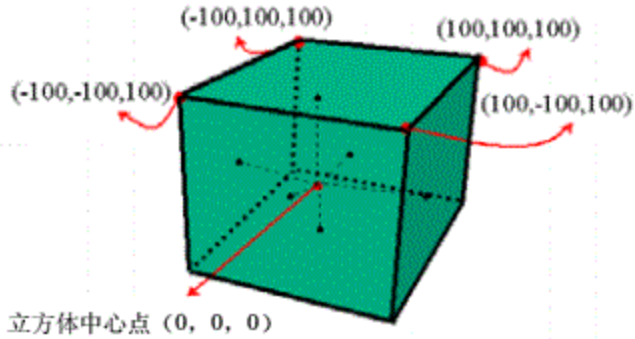


图 9.1　Model 坐标系统

如图 9.1 所示，（100,100,100）的顶点坐标是参考原点（0,0,0）所延伸出来的坐标点，这种由几何图形自己所参考出来的坐标系统，我们就称为“Model 坐标系统”。

9.1.3 World 坐标系统

在 3D 游戏中，一个场景可能由两个以上的对象所构成，而我们又要将这些对象摆放在特定的位置坐标上。大家应该不难发现，如果我们只使用“Model 坐标系统”来表现物体在 3D 世界的位置是不够的，那是因为“Model 坐标系统”是用来表示物体自己本身的坐标系统，而不能被其他的物体所使用，并且其他的物体本身也有自己的“Model 坐标系统”。在 3D 的世界里，有几个目标物体就会有几个 Model 坐标系统，而且这些 Model 坐标又不能表示自己在 3D 世界里的真正位置。所以我们就必须再定义出另外一个可供 3D 世界中的物体参考的坐标系统，并且使得所有的物体可以正确地被摆放在自己的坐标位置上，而这种另外定义出来的坐标系统我们则称为“World 坐标系统”。

9.1.4 View 坐标系统

当我们有了物体本身的“Model 坐标系统”与能够表现物体在 3D 世界的位置坐标的“World 坐标系统”以后，接下来，我们必须要有一个观看上述两者的坐标系统。如此一来，屏幕的显示才会有依据，而这个可以观看的坐标系统我们称为“View 坐标系统”。

9.1.5 坐标转换

在 3D 世界中，坐标转换过程是相当复杂的，它必须经过 4 个不同的转换步骤，最后才能显示在屏幕上，这才是我们所要的画面。坐标转换的流程是先将一个物体的“Model 坐标系统”转换至“World 坐标系统”，再将“World 坐标系统”转换至“View 坐标系统”

之中：接着再通过“Projection transform”将“View 坐标系统”计算出投影空间的坐标，最后再参考 ViewPort 中的参数，将位于截割区内的坐标进行最后的二维转换，以至于显示在屏幕当中。虽然这整个过程相当复杂，不过，读者不必太担心，这些过程已经有一些开发工具可以提供坐标转换的低阶运算，而且转换的速度相当快，我们只要调用特定的指令，再加上硬件设备的支持，这些复杂的转换过程可以快到让我们感觉不到，例如 Direct3D 及 OpenGL 开发工具。

在我们了解 3D 坐标系统的定义之后，下面我们来讨论一下物体在 3D 世界里应该要如何移动、旋转及缩放。

9.2 坐标矩阵

通常我们会使用矩阵来进行坐标转换的工作，因为矩阵的表示较容易记忆与判断，例如 Direct3D 与 OpenGL 等开发工具，它们也都是利用矩阵的方式来让我们进行坐标的转换。在计算机图形学里，矩阵的表示方式是以 4×4 矩阵来呈现的，因为这种矩阵的表示方式可以用来表示平移（Translation）、旋转（Rotation）及缩放（Scaling）等 3 种转换功能，而这 3 种转换功能就已经包含了 3D 世界的转换形式了。

这一种矩阵的运算对象及所产生的结果坐标，我们称之为“齐次坐标”（Homogeneous Coordinate）。

9.2.1 齐次坐标

“齐次坐标”具有 4 个不同的元素，简称“四元素”，其表示法为（x,y,z,w）。如果将齐次坐标表示成 3D 坐标的话，其表示法则为（x/w,y/w,z/w）。通常 w 元素都会被设成“1”，其用意是用来表示一个比例因子，如果表现在某一个坐标轴的话，它则可以用来表示该坐标轴的远近参数。不过，在这个时候 w 元素则会被定义成距离的倒数（1/距离）。如果我们要表示无限远的距离时，我们可以将 w 元素设置成“0”，而 z-buffer 的深度值也是参考此值而来的。

在前面我们所讨论的坐标转换过程，一开始的物体坐标值为（x,y,z），而 Direct3D 或 OpenGL 开发工具会将它设置成(x,y,z,1)的齐次坐标；而接下来的“World 坐标系统”、“View 坐标系统”、“投影矩阵坐标系统”的转换都是利用这个齐次坐标来进行运算的。

或许读者会问：“齐次坐标要如何才能转换成屏幕坐标呢？”其实在投影矩阵坐标系统的转换过程中，在截割运算进行之前，齐次坐标会被转换为 3D 坐标。例如投影矩阵坐标系统转换之后的坐标为（x,y,z,w），那么在截割运算之前，其齐次坐标会被转换成 3D 坐标值（x/w,y/w,z/w），最后再将（x/w,y/w）的坐标对应到 render target，并且转换成只有二

维的屏幕坐标了。

3D 的坐标转换通常包括 3 种转换运算，分别为“平移”、“旋转”及“缩放”，接下来，我们来探讨这 3 种 3D 坐标的转换方法。

9.2.2 矩阵平移

所谓“矩阵平移”即是物体在 3D 世界里向着某一个向量方向移动，如图 9.2 所示。

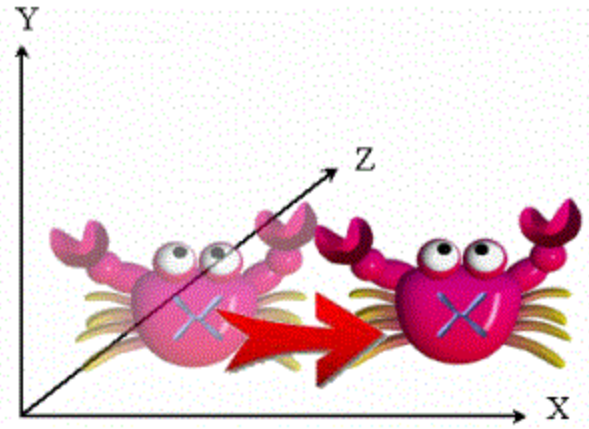


图 9.2　物体在 3D 世界里向着某一个向量方向移动

例如，某一个物体的顶点坐标为（x,y,z），而它的平移向量为（tx,ty,tz），到达目的后的顶点坐标为（x',y',z'），其矩阵平移运算的表示法如下所示：

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 100tx \\ 010ty \\ 001tz \\ 0001 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

9.2.3 矩阵旋转

矩阵旋转的定义则是 3D 世界里的某一个物体绕着一个特定的坐标轴旋转，如图 9.3 所示。

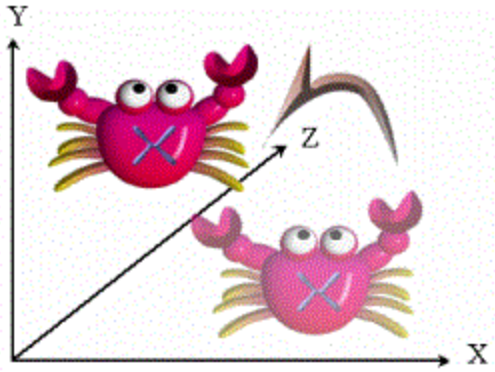


图 9.3　3D 世界里的物体绕着一个特定的坐标轴旋转

旋转的原则是以原点为中心，并向着 x 坐标轴、y 坐标轴或者是 z 坐标轴以逆时针方向旋转 ϕ 个角度，最后我们可以得到旋转后的顶点坐标（x',y',z'）。

x、y、z 坐标轴的旋转矩阵如下所示：

（1）绕着 x 轴旋转：

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

（2）绕着 y 轴旋转：

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

（3）绕着 z 轴旋转：

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

如果要顺时针方向旋转的话，我们可以将ϕ角度设置成负值。

9.2.4 矩阵缩放

“矩阵缩放”即是物体沿着某一个轴进行一定比例缩放的运算，如图 9.4 所示。

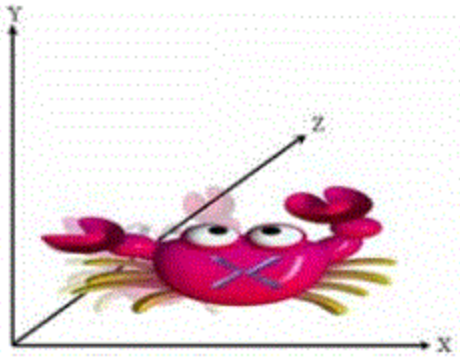


图 9.4　物体沿着 X 轴放大

在矩阵的缩放过程中，物体的顶点距离原点越近，其位移的数值则会越小，而位于原点上的顶点则不会受到位移的影响。例如，我们的顶点坐标为（x,y,z），在三个轴上的缩放值为（ηx, x，η y,）的比例，最后我们得到的顶点坐标为（x',y',z'），其矩阵的表示法如下所示：

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \eta x & 0 & 0 & 0 \\ 0 & \eta y & 0 & 0 \\ 0 & 0 & \eta z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

9.2.5 矩阵的结合律

在 3D 世界里，我们可以使用前面所讨论的平移、旋转、缩放来实现许多变化的效果，例如顶点坐标乘上平移矩阵之后，再乘上旋转矩阵，我们就可以实现物体在 3D 世界里的平移和旋转的变化效果了。不过，要实现矩阵的这种变化效果，我们势必要一一乘上应该的运算矩阵，才能得到最后的顶点坐标，这对于特定的矩阵相乘来说，这种转换的过程运算实在是太复杂了，例如平移矩阵为 A、旋转矩阵为 B、缩放矩阵为 C，而原来的顶点坐

标为K，最后得到的顶点坐标为K'，其矩阵相乘的公式如下所示：

```
K'=CBAK
```

从这个例子来看，我们必须将顶点坐标K乘上平移矩阵A，然后得到的值再乘上旋转矩阵B，最后再将得到的值再乘上缩放矩阵C，如此一来才能得到最后的顶点坐标K'值。如果一个矩阵要做16次乘法运算的话，那么三个矩阵势必要做48次乘法运算，这种过程是不是很繁杂呢？

其实我们可以将这种特定的矩阵相乘过程简化，因为矩阵相乘运算符合数学上的“结合律”。简单地说，我们可以将A、B、C三个矩阵先结合成另一个矩阵，如下所示：

```
μ=CBA
K'=μK
```

如果以后要使用这种特定的矩阵相乘时，就可以将顶点坐标乘以这一个运算好的矩阵即可，好处就是我们只要做16次乘法运算就行了，其过程是不是简化多了呢？

看完了矩阵转换的原理之后，接下来，我们将这些原理代入Direct3D中试试看。

9.2.6 Direct3D矩阵

在Direct3D的定义中，矩阵被声明成一个名为“D3DMATRIX”的数据结构，如下所示：

```
typedef struct _D3DMATRIX {
    union {
        struct {
            float       _11, _12, _13, _14;
            float       _21, _22, _23, _24;
            float       _31, _32, _33, _34;
            float       _41, _42, _43, _44;
        };
        float m[4][4];
    };
} D3DMATRIX;
```

而取得其元素的方式，除了直接存取成员变量之外，我们也可以在C++编辑器的最前面定义一个“D3D_OVERLOADS”的常数。如此一来，我们便可以直接使用索引值来取得元素内的值了。

其索引值是从“0”开始的，而且先定义行（row），再定义列（column），如下所示：

```
#define D3D_OVERLOADS
```

```
#include <d3d.h>
D3DMATRIX mat;
//下面3种表示方式的意思都是一样的
mat._13 = 0.8f;
mat[1][3] = 0.8f;
mat(1,3) = 0.8f;
```

举一个例子来说，如果我们要建立一个平移的矩阵，其表示法如下：

$$\begin{bmatrix} 1004.0 \\ 0108.0 \\ 00116.0 \\ 0\,0\,0\,1 \end{bmatrix}$$

在程序中，我们可以将它编写成：

```
D3DMATRIX mat = {
      1,   0,   0,   D3DVAL(4.0)
      0,   1,   0,   D3DVAL(8.0)
      0,   0,   1,   D3DVAL(16.0)
      0,   0,   0,     1
};
```

其表示法是不是非常地可视化呢？当我们有了矩阵的表示法后，接下来，我们再来看看向量在 D3D 中如何表示。

9.2.7 向量表示法

在 Direct3D 定义中，向量被声明成一个名为“D3DVECTOR”的数据结构，如下所示：

```
typedef struct _D3DVECTOR {
    float x;
    float y;
    float z;
} D3DVECTOR;
```

类似地，我们也可以看 C++编辑器的最前面定义一个“D3D_OVERLOADS”的常数来方便我们对矩阵及向量做一些运算，下面举几个比较基本的矩阵及向量运算函数：

矩阵相乘

```
//将矩阵pM1与矩阵pM2相乘之后的结果传给矩阵pOut
D3DXMATRIX* D3DXMatrixMultiply(
  D3DXMATRIX* 错误! 超级链接引用无效。,
  CONST D3DXMATRIX* 错误! 超级链接引用无效。,
  CONST D3DXMATRIX* 错误! 超级链接引用无效。
);
```

向量相乘

```
//将向量pV1与向量pV2相乘之后的结果传给向量pOut
D3DXVECTOR3* D3DXVec3Cross(
  D3DXVECTOR3* 错误! 超级链接引用无效。,
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。,
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。
);
```

两个向量进行 dot product 的运算

```
FLOAT D3DXVec3Dot(
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。,
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。
);
```

计算向量的长度

```
FLOAT D3DXVec3Length(
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。
);
```

计算向量的单位向量

```
D3DXVECTOR3* D3DXVec3Normalize(
  D3DXVECTOR3* 错误! 超级链接引用无效。,
  CONST D3DXVECTOR3* 错误! 超级链接引用无效。
);
```

向量相加

```
D3DXVECTOR3* D3DXVec3Add(
  D3DXVECTOR3* 错误！超级链接引用无效。,
  CONST D3DXVECTOR3* 错误！超级链接引用无效。,
  CONST D3DXVECTOR3* 错误！超级链接引用无效。
);
```

其实 Direct3D 还提供了许多非常好的运算函数，它们都是采用低阶的语法所编写而成的，其执行运算的速度相当的快，使用起来也相当简便。不过，它的函数指令实在是太多了，所以我们在这里就不再详加讨论了，有兴趣的读者可以参考其他 D3D 相关的书籍。

在前面我们介绍了投影的一些基本原理，而这种投影的原理在 3D 运算中占了非常重要的地位，例如我们要如何将现实中的三维坐标映射在计算机屏幕上的二维坐标系统中，下面我们将讨论一些 3D 投影的原理与技巧。

9.3 投影转换

在现实生活中，我们是在一个以三维表现的空间里，而计算机屏幕中的坐标系统却只能表示二维的空间。如果我们要将现实中的三维空间表现在计算机的二维空间中，就必须将三维坐标系统转换成二维坐标，并且将 3D 世界里的坐标单位映射到 2D 屏幕的坐标单位上，我们才能在计算机屏幕上看到所谓的 3D 世界，而这整个转换的过程我们称之为“投影”。

在计算机图形学中，有许多能够使用的线性或非线性的方式可以把 3D 空间的物体映射到 2D 的平面上。不过，最让我们感兴趣的投影方式则是“平行（正交）投影”和“透视投影”这两种，而且这两种投影方式是现在最常使用的投影技术。接下来，就让我们分别来讨论这两种投影方式要如何进行三维坐标的转换。

9.3.1 平行投影

当我们省略掉三维空间中的一维元素之后，我们就可以得到一个平行投影的图形坐标。这时，三维空间中的所有顶点都会从三维空间映射到 2D 平面的并行线上，因此我们就称这种方式为“平行投影”，如图 9.5 所示。

我们可以在投影线与投影面交叉角度的基础上更进一步地细分平行投影。如果交叉的角度是直角的话，则称之为“正交投影”（orthographic）；如果不是直角的话，则称为“倾斜投影”（oblique）。

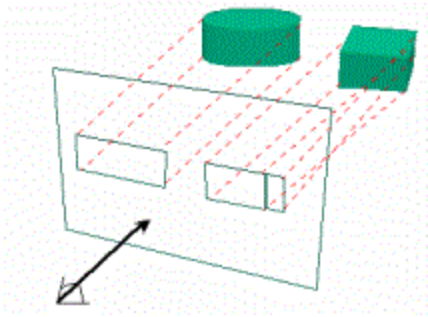


图 9.5　平行投影

这种投影的方式跟工程绘图有类似的地方，通常我们都会使用“正交投影”（顶视、前视和侧视）来转换三维坐标，这是因为它所呈现出来的画面与我们在现实生活中所看到物体的距离的视觉效果是一样的。

如果我们选择的投影面平行于坐标系统上的 X 与 Y 轴的平面时，投影线则会平行于 Z 轴；因此，平行投影的转换将会除掉空间中所有顶点的 z 坐标单位。对于我们人类的视觉而言，我们都会对观察者在世界中的位置和方位（通常指的是摄像机位置和方位）所创造出来的投影感到兴趣。而在这些情况下，投影面则不需要平行于坐标系统中的 X 与 Y 轴面。

其实还有许多方式可以用来描述摄像机所投影出来的影像。我们可以指定 3D 世界中观察者（摄像机）的顶点坐标以及描述观察者在 3D 世界中的参考方位的向量，或者不用指定向量来表示其顶点向量。可以通过这三个角度的 α 、 β 、 γ 来表达其同样的内容，因而描述观察者的方位。任意投影的转换过程都可以利用 3D 世界中所有摄像机的表达方式来进行转换，而这种转换程必须通过两个步骤才能实现。

在这种投影的转换过程中，首先是执行“仿射转换”，这一步的目的是对空间进行转换以实现投影面能够映射到 X 与 Y 轴的平面上，接着第 2 个步骤再执行上面我们所说的平行投影的转换之后才能完成投影的需要。

正如我们上面所看到的一样，对于平行投影而言，基本的原理是将 3D 顶点的 z 坐标去掉。但是，去掉 z 坐标之后，便失去了所有原始 3D 空间的深度信息。为了避免这种情况的影响，我们必须考虑“透视投影”。

虽然平行投影有这种缺点，不过，它还是在许多 3D 应用的领域中被广泛地使用，例如在 CAD 中的应用。平行投影保留了图像中的并行线和对象的实际大小，这个重要的性质带给平行投影立足于 3D 投影的地位。

9.3.2 透视投影

透视投影所建立的对象投影图像的大小必须依赖对象与观察者的距离。在透视投影中

要表现这种效果其实并不困难，就如同在第 8 章中所说的 2D 透视图一样，如图 9.6 所示。

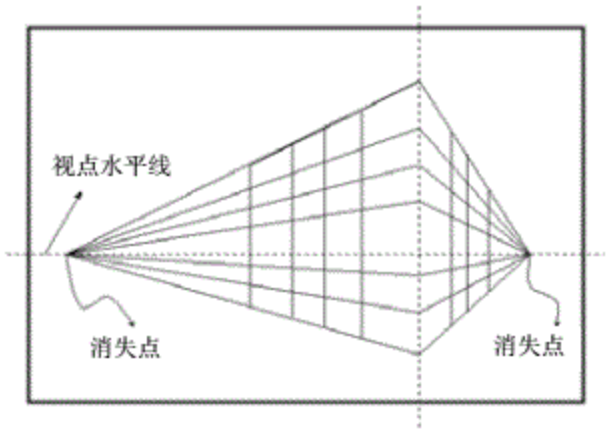


图 9.6　2D 透视图

如图 9.6 所示，最先与我们联系起来的是一条空荡荡且笔直的街道，以及街道两旁消失在无穷远的建筑物。透视投影是让我们以场景的现实视觉表现的特点来产生图像。毕竟，如果道路不是汇集到一点或是建筑物的距离不是离我们越远而越小的时候，这种街道看起来就会显得非常的不自然。

我们可以利用观察者的眼睛去直视远方的一个点，而光线从所有对象上反射回来，并且汇聚到这个点上，然后再经过模拟透视投影的转换，使得每一条光线在映射到眼睛之前，它们就已经与观察者面前的平面相交叉。如果我们能够找到交叉的横断面，并且描绘那里的点，观察者的注意力就会被欺骗，认为从描绘的点那里发射出的光线实际是来自与空间中其原始的位置，而让观察者感觉好像看到真正的 3D 立体空间一样，如图 9.7 所示。

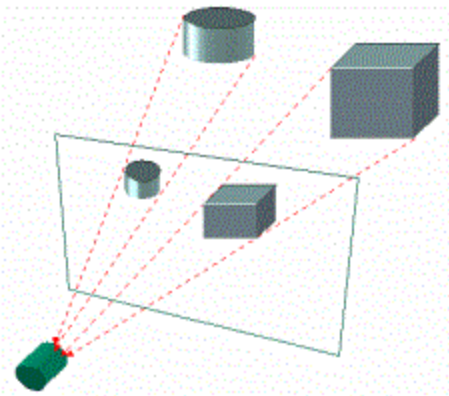


图 9.7　视觉欺骗形成的 3D 立体空间

在此我们可以使用前面所谈到的方法，选择一个与参考坐标系统内的 X 与 Y 轴平行的投影面。在这种情况下，我们不难发现原点与图像上的顶点之间的关系，如图 9.8 所示。

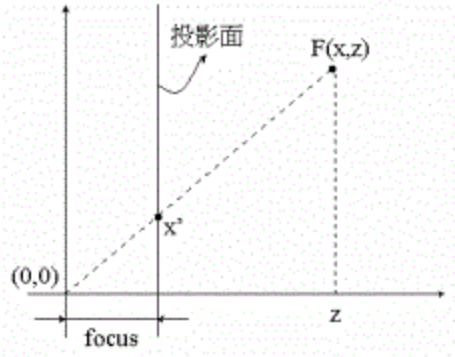


图 9.8 原点与图像上的顶点之间的关系

首先我们观察一下平面的显示，观察者的眼睛位于参考坐标系统的原点上，而观察者的眼睛与投影面的距离我们称之为“focus”（焦点距离）。其目的是要确定哪些顶点可以在光线从 F 点发射到观察者眼睛的时候所产生的投影面，所以我们就必须在屏幕上的这个投影面上描绘物体。

说到这里，您是不是对透视投影的概念更加深刻了解了呢？接下来，我们要来分析如图 9.9 所示两个三角形之间的关系。

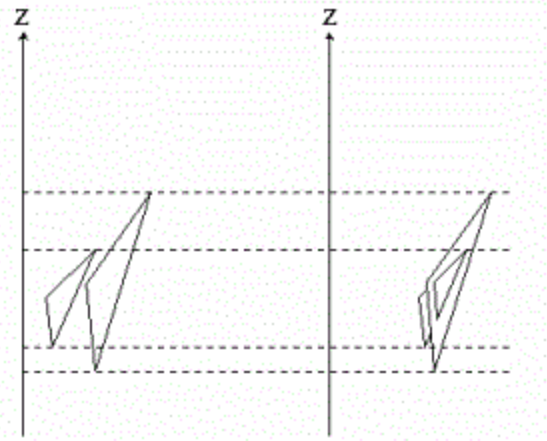


图 9.9 原始空间中一个多边形遮掩另一个多边形

从图 9.9 中，我们可以得到两个已知的事实，那就是两个大小不同的三角形在这个坐标系统上的起点（两者都在原点上）是相同的，以及这两个三角形的正切值也是相同的，所以我们可以推算出如下的公式：

$$\frac{x'}{focus} = \frac{x}{z} \Rightarrow x' = \frac{x * focus}{z}$$

因为它们都有相同的 Y 值，所以我们可以利用下列这两个公式来描述 3D 描绘的情况：

$$y' = y * focus/z$$
$$x' = x * focus/z$$

从上面的例子可以得知，我们考虑的情况只有利用观察者位于坐标系统上的原点处，它是沿着 z 轴方向看到的情况。不过，如果我们不是利用这种情况的话，它则会与平行投影一样必须先进行一个仿射转换，再将原始的空间转换到这里所描述的透视转换，如此一来，才可能进行透视投影。

在透视转换之后，将物体映射到屏幕上时，势必要得到 x 轴与 y 轴的坐标值，但是不需要 z 轴的坐标值，所以我们可以将顶点的 z 坐标去掉。然而，当我们试图映射具有多个表面的物体时，我们就必须知道物体上所有多边形的 z 深度，如此一来，我们才可以推论出什么是可见的，什么是不可见的。

如上述范例，我们将 z 轴上的坐标定义成 z'=z，如此才可以保留各个多边形的深度值。不过，由于 x 轴与 y 轴的坐标进行了投射转换，而在 z 轴的深度值不变的情况下，其深度可能会发生一些相应的变化。简单地说，如果原始空间中一个多边形遮掩另一个多边形的时候，在经过投影之后，实际上前者会出现在后者的后面，而被后者所遮掩。

为了避免这种情况发生，我们就必须对 z 轴上的深度值做非线性的转换。例如“z'=-C/z”，而这里的 C 是一个常数向量。

透视投影的呈现效果是非常直接了当的，然而，当我们在计算焦点距离长短变化的同时，我们就必须去理解它的物理意义，这样才能选择合适的 C 常数向量。

如图 9.10 所示，焦点距离能够确定视角的区域。焦点距离越小则视角宽度越广，而焦点距离越大则视角宽度越窄，这种情况有助于我们将距离当作是屏幕图像的度量单位。例如，当我们选择的焦点距离是 160 个单位，而显示分辨率是 320×320 的像素单位，视角的宽度则为 90º。

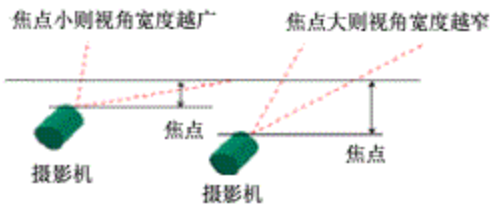


图 9.10　通过焦点距离确定视角的区域

透视转换所产生的图像可能一开始看起来会有一点不自然的失真效果，所以我们必须要改善其顶点坐标的真实感。在 3D 世界里，视角宽度在 75°~85° 的焦点距离其效果是最好的。当然，这也要取决于场景和屏幕中的几何结构才行。

通常用矩阵的形式来表示透视转换的过程是非常困难。实际上，我们也只能使用矩阵乘法来处理线性的转换。然而透视转换是非线性的，所以在这里我们不宜使用矩阵来转换透视投影。

如果说我们坚持使用矩阵来表示透视投影的话，我们势必要订定一种特殊的约定。对于平移的转换来说，我们也是使用了一种较小的约定，那就是在原来 3×3 的矩阵上增加了一个额外的元素。而这种元素要能够表达透视转换的约定，也就是说这个元素是为了保留“齐次坐标”。

我们在顶点的 X、Y、Z 坐标后增加一个坐标向量“1”，如果在矩阵转换的期间，该坐标向量会得到不同的值而被重新归一化。换句话说，即是向量中的每个元素都乘上一个常数，而最后的元素又成为 1。假设使用这种方法之后，就能够将透视转换表示成如下所示的矩阵：

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/\text{focus} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} x & y & -1 & z/\text{focus} \end{bmatrix}$$

其得到的结果向量最后一个元素不是 1，而我们再乘上“1/z/focus”来归一化，如下所示：

$$x \quad y \quad -1 \quad z/\text{focus} = \begin{bmatrix} x \bullet focus/z & y \bullet focus/z & -focus/z & 1 \end{bmatrix}$$

上述式子其目的是为了要取决于对 z 坐标的需求，当我们不再需要 z 坐标的时候，我们则可以去掉它，最后矩阵会发生一些变化，如下所示：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/\text{focus} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

我们将向量中的 z 坐标等于 0，其变化如下所示：

$$[x \bullet focus/z \quad y \bullet focus/z \quad 0 \quad 1]$$

如果观察者位置上的 z 坐标为负值的话，这个位置则不会被选为坐标系统的原点坐标。因为如此一来，转换公式与转换矩阵会因为这样而产生不同的形式，在此时我们应该依赖特别的转换形式来取代这种情况。

一般而言，我们可以利用点数区分的方式来处理各种透视的转换（如一点透视或两点透视），或者我们也可以利用摄像机的观察方向和屏幕法线的角度来区分。在这里，我们只利用观察者方向垂直于投影面的一点透视来进行转换的形式，因为它足以解释透视转换的特殊之处。为了达到 3D 绘图应用的目的，我们或许需要处理稍微复杂一点的透视转换，而在大多数情况下，我们可以容易地换算出前面讨论的仿射转换。

透视转换还有几个值得我们注意的地方，比方说，z=0 的顶点将会被投影转换映射平面的哪一个坐标上。关于这一个问题，我们可以从上面的公式中看出，当 z 坐标等于 0 的时候会导致计算的错误，所以这一个问题在 3D 的绘图中是根本就是不可能存在的顶点。

另一个问题，当 z 坐标为负值的时候，其转换会产生负坐标的情况，如此一来，我们在平面上看到的物体（或物体的一部分）将会被翻转过来。不过，带有负 z 坐标的物体会映射在观察面的后方，简单地说，就是躲在观察者的身后，如此一来，在投影平面上我们也就看不到这个物体了（至少是它的一部分）。其实为了解决这一类问题，惟一的办法就是在演算的过程中，确保所有的顶点都没有无效的 z 坐标。

接下来，再来思考一下如何利用矩阵形式来表示透视转换。从上述所讨论的所有转换形式来看，一个顶点要被映射在平面的时候，它必须按照我们接下来所说的步骤进行平面映射。

首先，观察者的位置坐标与方位先进行仿射转换，在我们进行透视转换之前，必须先执行裁剪的动作，其目的是要除去物体后面看不到的顶点。然后我们再进行透视转换的工作。以传统的矩阵形式来表达裁剪是非常困难的，正如我们前面所讨论的那样，当我们用一个矩阵来表达几个连续转换的时候，矩阵通常是非常难以表达的。然而，在这种情况下，我们需要的不只是一个矩阵，因为在这里旋转与透视都被裁剪给隔开了，如果我们在确定没有顶点在物体后面之后，那么我们就可以免除裁剪的动作，如此一来，我们才可以利用矩阵的形式来表达每一个转换，并且运算它们的级联矩阵。然而，在其他情况中，裁剪的动作是不能被避免的，如此一来，我们则必须将转换的动作分隔在两个阶段上，其阶段分别是 3D 裁剪之前与 3D 裁剪之后。

说了这么多的裁剪方法之后，接下来，我们就来深入讨论 3D 裁剪在三维运算中应该如何才能进行。

9.4 裁剪

在上一节里看到在透视转换中存在着另一种对坐标的约束，那就是我们不能在屏幕空间中转换 z=0 的那些顶点，而且具有负 z 坐标值的物体位置会映射在观察者的后方，因此这个物体是不可见的。为了避免这个问题的发生（如被 0 除或者物体某些部分被透视转换所裁切掉了），所以必须确保只有在有效顶点被转换之后，再进入透视平面的空间。上述两种情况，对于第一种情况，必须先执行 2D 或屏幕边界的裁剪；对于第 2 种情况而言，我们就必须使用 3D 物体的裁剪。在本节里，我们就来介绍如何处理这两种情况，并且讨论所有可以裁剪的方法。

9.4.1 2D 裁剪

裁剪是对于某种图形做出修剪的动作，其目的是为了让看不见的顶点不加以描绘，以提升执行的速度，裁剪算法同时也是执行裁剪图形（2D 区域或 3D 区域）的规范。通常，我们很难找出一种适合任意形状和任意裁剪体的方法，这主要是受物体形状的约束。

对于 2D 裁剪而言，我们可以定义出 3 种不同的裁剪方法。第一种是在光栅处理阶段之前，先进行裁剪的动作，这种方法较适用于处理简单的图形单位，例如被矩形裁剪区域所约束的多边形。第 2 种裁剪的方法是发生在某些情况下，尤其对复杂的图形来说，在光栅处理阶段的同时，进行裁剪较为合适。而在得到图形的屏幕坐标之后，首先要检查它是否在合适的边界内，如果在合适的边界内时，我们才可以进行映射图形的处理。第 3 种裁剪的情况是当裁剪区域中的几何结构比较复杂的时候，我们可以选择一些较大的缓冲区光栅图形，然后从缓冲区中选择在该复杂裁剪区域内部的图形，如图 9.11 所示。

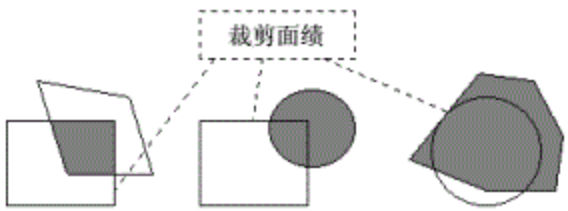


图 9.11　从缓冲区中选择在该复杂裁剪区域内部的图形

注意：投影转换把物体图形坐标从 3D 空间映射到屏幕的 2D 空间上，这些图形就必须在 2D 空间中被绘制，使观察者可见。既然我们考虑的是由一系列整齐地镶嵌的小方块组成的光栅图形，因此我们要确定这种几何像素的模拟描述要如何能够被正确地转换成一系列的离散图形，而这种过程我们称之为“光栅处理”

(rasterization)。

第一种裁剪的方法是在分析寻找和描述裁剪区域内部的图形所耗费的资源是否太多，如果耗费的资源不多时，其工作的过程是相当良好的。

当裁剪范围大于多边形的时候，例如寻找和分析描述圆形和矩形的交叉点时，我们则可以利用第 2 种裁剪方法，其方法是在光栅处理过程中进行裁剪的工作。

当在运行中的预裁剪和验证图形的范围大于裁剪区域范围的时候，则可以考虑使用第 3 种裁剪方法。例如，当裁剪区域是圆形或椭圆形的时候，我们就可以采用这种裁剪方法。

通常我们都会以矩形裁剪区域（计算机屏幕长宽）来进行裁剪的工作，而且这种简单地裁剪几何图形也适用于其他的投影算法，所以我们主要都会集中考虑使用第一种裁剪方法。接下来，我们讨论应该如何在矩形区域内裁剪点、线段和多边形。

9.4.2 点的裁剪

在裁剪的 3 种方法中，使用起来最方便也是最快速的方法就是使用矩形裁剪区域，其区域以 4 个有限线段来描述，它限制了屏幕最小和最大的水平坐标以及最小和最大的垂直坐标。对于这个区域而言，点的裁剪可以利用这 4 个有限线段的约束来检查点的坐标是否应该要被裁剪。

尽管这种做法已经相当的简化了，但是我们还是可以在区域的限制线段通过坐标起点的时候，来改善其图形的边线。

9.4.3 线段的裁剪

我们可以将线段的裁剪扩展到上述方法，在描绘每一个图形之前，我们每一次都要去检查它是否在边界内。然而，在光栅处理的同时，这种裁剪的方法则会增加光栅处理内循环的复杂性，而且在图像位图内以点的地址来优化线段的绘制，比起屏幕坐标来绘制图形还要优化。

这种线段裁剪的方法可以分析图形的相对约束位置，并且找出符合这一点的图形部分。在线段裁剪的情况下，它需要找出其交叉点或线段与裁剪边界的交叉点，通常不是直接就能够把这种交叉发生的地方看得一清二楚的，如图 9.12 所示。

这一种简单的裁剪方法是由 Cohen 和 Sutherland 在 20 世纪 70 年代所提出的算法，它是通过沿着边界进行裁剪，而将可能被裁剪的部分定义成定位交叉点，然后作为裁剪图形的另一个边界，如此一来，保证可以找到符合所有情况的线段端点，如图 9.13 所示。

如图 9.13 所示，线段 AB 第一次被边界 B_1 裁剪，所以我们找到交叉点 I_1，并且进一步以边界 B_2 来处理 I_1B 的线段，这种裁剪的方法也就是我们说的线段裁剪了。这种裁剪的方法非常简单，它可以被用于任意多边形区域内裁剪其线段，而不一定非得使用矩形裁剪不可。

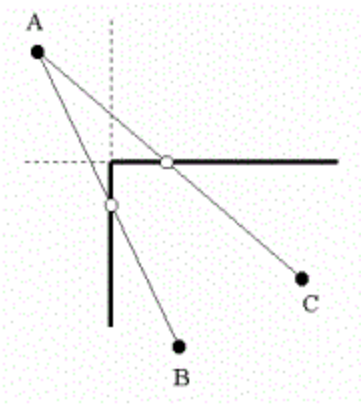


图 9.12　线段与裁剪边界的交叉点

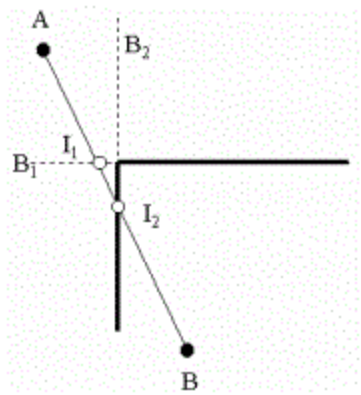


图 9.13　找到符合所有情况的线段端点

当我们需要找出多个交叉点的时候，这些顶点中某些部分可能是无效的，只要是交叉点有无效的情况发生，我们则要避免去计算这些无效的交叉点。当我们可以推断出裁剪线段会在裁剪区域的内部或外部时，我们则可以用此条件来排斥或接受某一条线段。例如，当裁剪线段两个端点的水平坐标小于最小可接受的屏幕坐标时，这种线段就可以安全地被排斥掉。同样的，这种做法也可以被使用在其他的边界和水平坐标之间。

其实还有一种更简单的接受或排斥裁剪线段的方法，而且这一种方法还可以加速环节判断的比较，这种方法我们称为“区域外编码”（region outcodes）。

我们为平面分配了一个位的编码样式，通过这个编码位的样式之后，每一个编码位就可以注明图形是否会出现在矩形裁剪区域的哪一个位置了，如图 9.14 所示。

如图 9.14 所示，在这种技术的典型例子中，外编码的第一位描述了在裁剪矩形上方的区域，如果我们对线段的两个端点分配这种外编码的话，我们则可以通过这个位的运算来

检查各顶点在平面上的组合位置。如果某些位对两个端点来说是“置位”（都为“1”）的话，这个区域内的整个线段会出现在裁剪区域的外部，这个运算的过程，我们可以利用“AND”来做校验。如果运算后的结果不是“0”的话，裁剪线段则可以安全地被排斥掉。另一方面，如果任意一个位对裁剪线段的端点是置位的话，我们则要进行裁剪处理，如此一来，只有线段没有被置位的情况下（全部为“0”），我们才可以接受这个线段，这个运算的过程，我们可以利用“OR”来做校验。外编码也帮助我们可以确定裁剪线段交叉了哪一个裁剪边，以及我们必须要在哪里运算其交叉点。

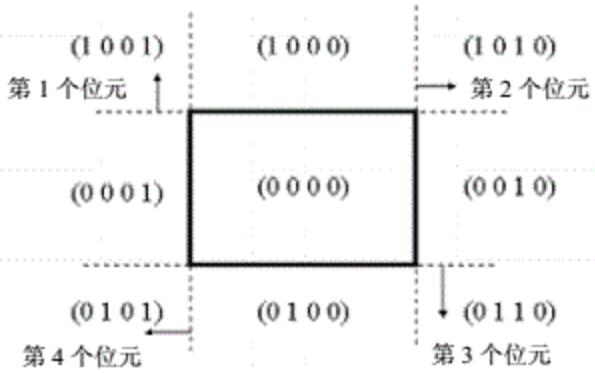


图 9.14　注明图形每一个编码位在矩形裁剪区域的位置

只要是线段不能被接受或排斥的时候，就必须进行裁剪的处理。正如上面所说的那样，这整个过程都要涉及到寻找裁剪线段的交叉点，而在垂直和水平裁剪边的情况下，要寻找与线段的交叉点就显得容易多了，如图 9.15 所示。

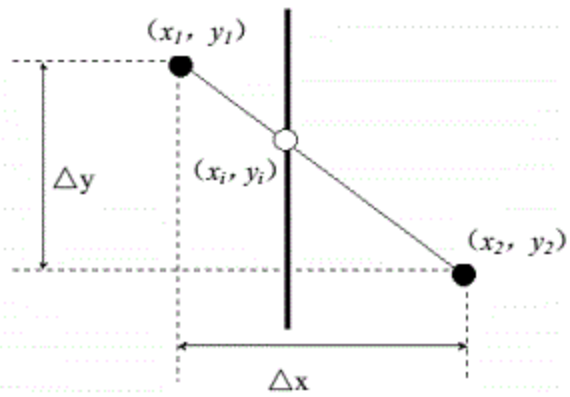


图 9.15　线段的交叉点

在图 9.15 中，其解决方法涉及到必须分析两个同样的三角形之间的关系。由于交叉点

是属于已知 X_i 坐标的裁剪边，因此剩下的工作是寻找对应的 Y_i 坐标，我们可以这么计算：

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} \Rightarrow y_1 = y_2 - \frac{\Delta y \bullet (x_2 - x_1)}{\Delta x}$$

如上述表达式我们可以得知垂直边 Y_i 求 X_i 的情况。每一个顶点都涉及到一次乘法和一次除法。在这个时候，线段可能已经乘上了每一个顶点所定义的值。例如，当我们在处理多边形的明暗时，除了空间坐标之外，我们也必须处理光线的强度值，还有纹理多边形要处理纹理坐标等等。

对于图形的应用而言，不涉及到乘法或除法的方法通常是可以增加运算的速度。在裁剪这种情况中，我们可以使用“二元搜寻”（binary search）这种方法来运算，因为它只会涉及到一些基本的整数运算。因此，这种方法就可以提高我们搜寻交叉点的速度。

二元搜寻算法通常是用于寻找线段的中间值，通过反复减小线段空间的大小，直到找出最后交叉点的坐标。我们将这种算法应用到裁剪的问题上，这种算法则可以寻找线段的中点并沿着裁剪边比较它的位置。在寻找交叉点的过程中，我们可以得到两个线段，而其中一个不符合的线段则会被去除，在下一次寻找交叉点的时候，它则会去使用剩下的范围再进行二元搜寻，直到这个范围小于原始的线段为止，如图 9.16 所示。

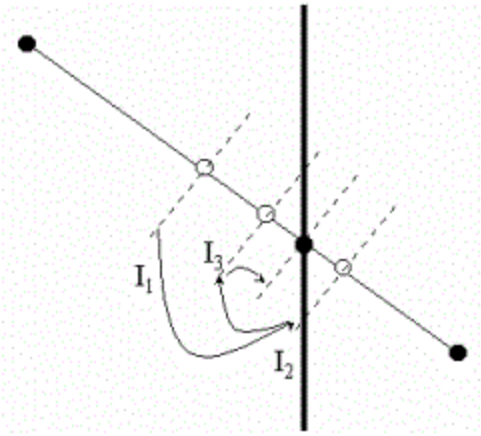


图 9.16　寻找交叉点

例如，如图 9.16 所示的线段 AB 被分割为两部分，从端点 A 到中点 I_1，以及从 I_1 到另一个端点 B。通过比较中点的 x 坐标和裁剪边的 x 坐标，我们可以推断出裁剪边位于中点的右侧，即线段 I_1B。剩下的线段 AI_1 则被去除。其算法会不断重复去除不必要的线段，直到中点的 x 坐标和裁剪边的 x 坐标可以相等或足够接近为止。

尽管算法在这里可能涉及到好几次去除的动作，但是在每一次去除的过程中，新的线

段空间只有原先的一半，因此这种去除的动作就会很快地终止。使用此算法运算其中点的时候，它只需要利用非常基本的运算，如下所示：

$$X_{mid} = \frac{X_1 + X_2}{2}$$

$$Y_{mid} = \frac{Y_1 + Y_2}{2}$$

正如我们看到的那样，我们只需要使用加法和除法即可算出线段的中点在哪里。然而，需要特别注意的是二元搜寻终止的条件。由于整数的截断，二元搜寻可能会陷入无穷循环之中，以至于不能完全等于裁剪线段的 x 坐标，并且达到精准的正确值，所以我们必须仔细地考虑中点计算的特性并且确保不会出现意料之外的情况。

9.4.4 多边形的裁剪

接着我们来讨论一下沿着矩形边缘区域来裁剪多边形。如同前面所介绍的线段裁剪一样，多边形裁剪会涉及到多边形边缘和裁剪区域的边缘上寻找相交的交叉点，它与线段裁剪不同的是在交叉点作为裁剪的结果后，多边形的形状可能会产生很大的变化，如图 9.17 所示。

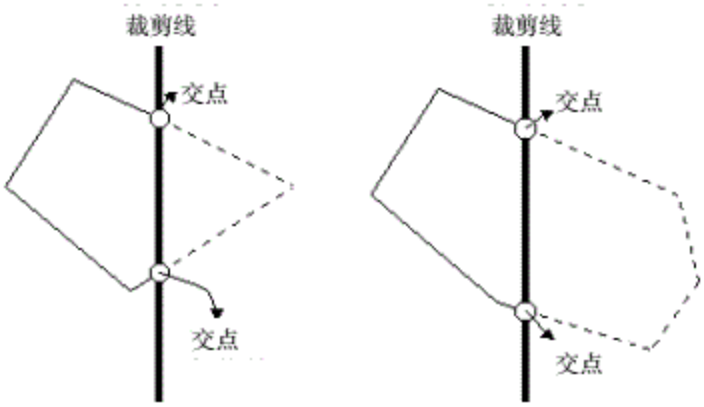


图 9.17　在交叉点裁剪时，多边形的形状发生的变化

在裁剪直线时，我们要使用的方法非常具有概括性，可以沿着任意多边形的裁剪区域来裁剪此多边形。在矩形屏幕中，将每一个裁剪边缘计算裁剪例程一次，最后获得符合所有强制标准的多边形，如图 9.18 所示。

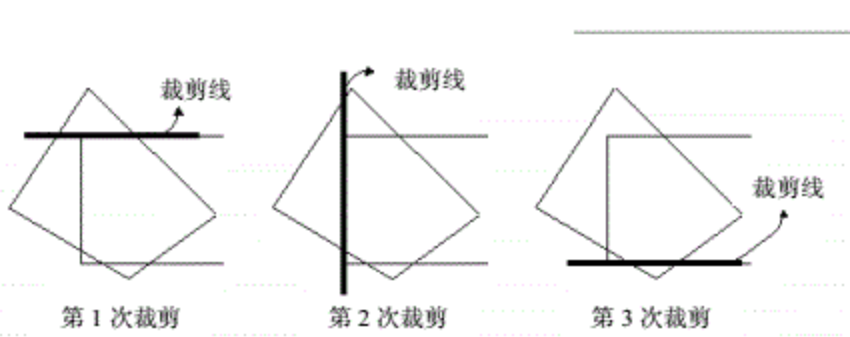


图 9.18　直线的裁剪过程

在每一次裁剪中，我们得到了多边形改小的问题，在这里，我们建立了精确且符合强制标准的新多边形。我们已经能够求出寻找线段和裁剪边交叉点的技术。接下来，我们再来讨论一下，我们应该如何以单个有限线段来裁剪一系列边缘所形成的多边形。

一个多边形通常会以一种具有一定顺序的顶点所组成，每一对连接的顶点描述了一条边。只要其中一条边与裁剪边发生了交叉，我们就要找出一个交叉点，而它也必然成为裁剪后的多边形的新顶点，如图 9.19 所示。

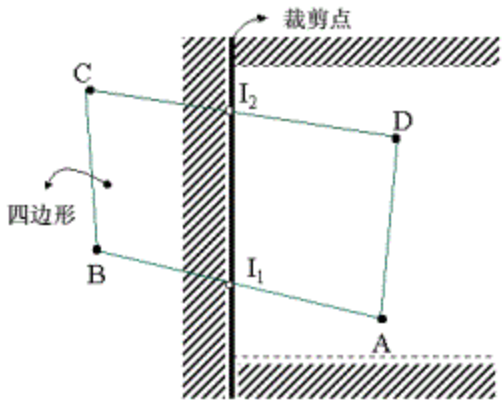


图 9.19　找出交叉点

从图 9.19 中，我们可以看到多边形 ABCD 在裁剪后变成了 AI_1I_2D。为了证明这个事实，我们以在多边形描述中出现的顺序来考虑多边形的边。当 AB 边跨过裁剪线的时候，从图 9.18 中可以看到多边形 ABCD 在裁剪后变成了 AI_1I_2D。当 AB 边跨过裁剪线的时候，就必须在多边形中增加另外一个新顶点，然后再顺着多边形的顺序再穿回到裁剪线，如上例，新的交叉点也就是 I_1、I_2，而在交叉点之间的裁剪线则限制了裁剪区域的范围，并且定义出裁剪多边形的新边 I_1I_2。

我们必须知道属于两条边的每个顶点，以及多边形的描述，这些都必须要有一定的顺序。在每个顶点与裁剪中间会产生两个不一样的顶点，而这两个不一样的顶点又会形成多

边形的另一条边，因此在分析边的时候，就必须考虑这一个问题。当一条边被裁剪所接受的时候，它就必须从结果列表中复制它的第 2 个顶点，假设第 1 个顶点已经先被考虑过了，而第 2 个顶点又超出裁剪区的时候，就必须找出其交叉点，并复制到结果列表上。当一条边被删除时，也不需复制任何顶点。

当原来的多边形完全被裁剪过后，下一步就要传递给光栅去处理。光栅处理会把多边形当作是一系列的图形线，并且将它们绘制出来。不过这里的裁剪边水平线却没有任何光栅处理例程的附加信息。既然任意水平线与其他两个边都有共同的端点，因此这个图形线便能够从邻边上找到，不过我们可以不把水平线传递给扫描例程处理，所以也不会失去它的连续性，如图 9.20 所示。

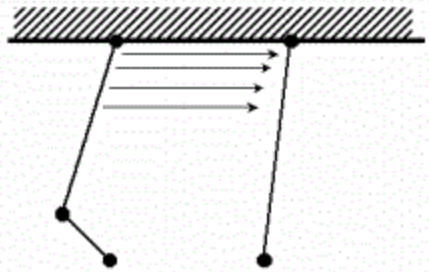


图 9.20　从邻边上查找图形线

在这里我们必须提到的是这一种算法不仅仅可以用在凸多边形上，其实它也可以被使用在凹多边形。然而，在大多数情况下，凹多边形可能被裁剪分割成多个部分，如图 9.21 所示。

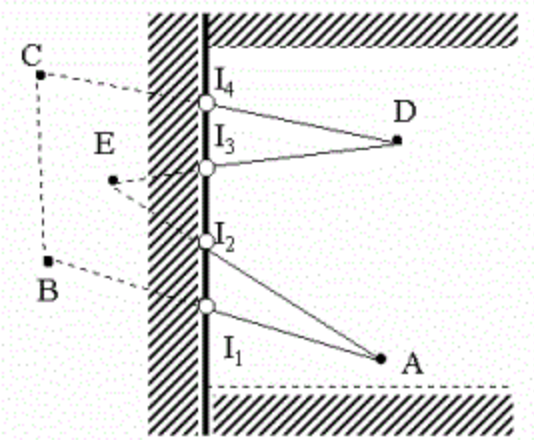


图 9.21　凹多边形被裁剪分割成多个部分

多边形 ABCDE 被裁剪成两块：AI_1I_2 和 DI_3I_4。我们不难看出，在这种情况下，算法计算的情况就是将单个多边形合并了两块 AI_1I_2 和 DI_3I_4，最后产生的这个多边形，严格说来它不是一个单纯的多边形，因为 I_1I_4 和 I_3I_2 的边在某一个线段是共享的，而这种多边形通常被称作“弱简单（weakly-simple）多边形”，因为许多几何算法对此种情况不会产生任何特殊的变化，所以我们则将这种情况归纳于单纯的分化例子，而且不需要做任何图形的变动或只需要做很小的变动就能够正确地绘制弱简单多边形。

至此，我们已经将 2D 裁剪的部分全部讲完了，下节中再来深入讨论 3D 裁剪的各种算法推论。

9.4.5 3D 裁剪

如同前面所讨论过的一样，透视投影只适合被使用在空间中所有顶点的子集上。因为透视转换会倒转与观察者坐标的距离，对 z=0 的顶点来说，它的结果会是无穷远，而且它也会忽略观察者后面的顶点。所以我们势必要利用 3D 裁剪的技术来确保只有有效顶点才能进行透视转换。

透视转换限制了观察者前面顶点的世界空间，在上一节中，我们看到了沿着水平线和垂直线所裁剪的图形，而在这里我们只要稍微概括一点不同的形式就能够以同样的解决方法来实现视平面裁剪的目的，惟一不同的是必须去考虑三维坐标而不是二维坐标。

在透视转换之前，算法会在观察者前方的平面上执行裁剪，目的是为了删除在某些方面可能会引发除以零的这种非法计算的顶点。虽然在该平面上已经非常严格地限制所有有效的顶点，不过仍然有可能以很大的绝对坐标值被映射到屏幕空间上。实际上您不需要太担心，因为这种情况是不太可能发生，因为坐标太大以至于会超出存储它们的变量位大小，如图 9.22 所示。

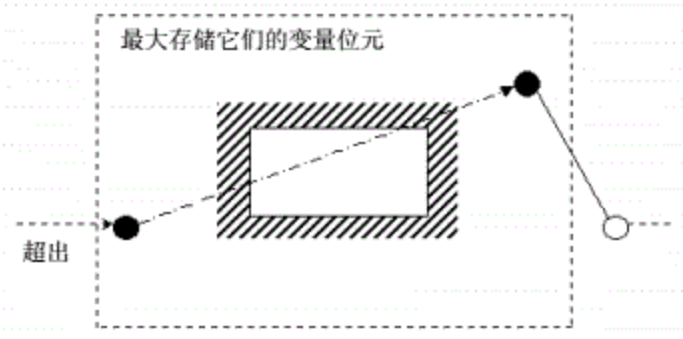


图 9.22　坐标太大将超出存储它们的变量位大小

还有存储在有符号表达式中的数值就很容易遇到溢位，因为负数的表示是为了很大的

正数而定的，在数值很大的情况下依然被投影顶点的坐标改变了符号，因而这些顶点会分别出现在平面的不同区域上。如果碰巧是用来描述某些图形顶点的话，则投影会失去它原有的连贯性，这个图形可能会突然占掉屏幕空间相当大的区域。例如一条直线如果在顶点上溢位时，它可能会从屏幕的左边到右边或从上方到下方，以至于跨越整个屏幕平面。

如上面的这种情况，我们可以利用修补方式来尝试移动裁剪面远离视平面。其实这个办法还是不能解决根本的问题，而只是将不可避免的顶点移到更远的地方去。不过在世界空间大小有限的情况下，这个解决的办法就已经足够了。

请注意，投影算法的最终目的是让顶点能够在屏幕上被绘制，所以我们可以在空间中根据它们被投影到屏幕上的能力来分离这些无效的顶点。在这里所有被投影到屏幕内部的顶点称为“视体”（view volume），而投影到屏幕外的顶点当然也就在空间中视体的外部，我们将这些不能被显示在屏幕内的顶点问题给联系起来，并且在空间中导入某些条件，以避免这个问题的发生。

在平行投影与透视投影的这两种情况下，势必要找到物体上的所有视体。而在这两种情况下，投影技术可以将屏幕边界的点、视体内的点和其余的点分割开来。在平行投影的情况中，投影线会相互平行并且与投影面正交，穿过屏幕边界的直线会在屏幕投影面的前方，因而定义出一个类似棱柱的视体。在透视投影的情况下，投影线与观察者的视线是相交的，穿过屏幕边界的投影线在空间中也形成了一个类似金字塔的视体，有时候也可以把它称作“视锥”，视锥同样也会限制观察者前方较近的裁剪面，如图 9.23 所示。

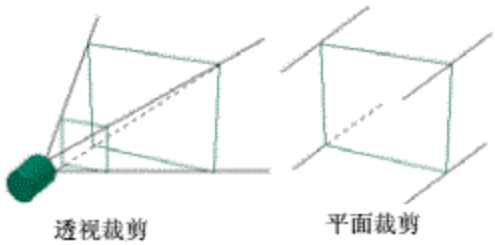


图 9.23　透视裁剪与平面裁剪

对于使用平行投影的算法来说，2D 屏幕边界裁剪会限制这个类似棱柱的视体。使用透视转换算法则可以通过某些处理的办法来限制这个类似金字塔视体（视锥）的坐标，而不需要屏幕边界的裁剪，因为视体裁剪的方式可以确保所有的图形顶点被投影到屏幕内部。

除了确保透视转换与光栅处理能够被正确应用之外，视体裁剪也会限制一起进行处理图形的数量。当然能够早期确定这些不需要出现在图像中的顶点，对于往后要做的处理过程就会越少。在透视转换的过程中，由于使用了距离反转的技术，所以它会使得远离观察者的图形在屏幕上看起来非常的小。在这些运算的情况下，既然这些物体的视觉效果是微不足道，那么我们就有理由不去花费太多的时间进行绘制。通常我们也会在视体中加入后

裁剪面，这么一来就可以删除一些不想要的物体。换句话说，我们就可以不用再特别去绘制它。

通过透视视体的裁剪，又多出了一个复杂的问题，那就是不得不通过可能在空间中任意位置的平面裁剪。在所有转换算法还未发生的情况下，我们能够利用裁剪边和裁剪面的简单几何结构来进行裁剪，然而限制视体的 6 个面中只有两个简单的面，而这两个面即是“前裁剪面”和“后裁剪面”，其他 4 个面的位置运算就显得非常不方便。要解决这一类问题的方法通常会付出一些运算性能上的代价，尽管我们能够在理论上分析上解决交叉点的问题，但是通常这种方法可能非常复杂，所以不能够被采用。接下来看看几个可以用来处理视体裁剪问题的方法。

在这个问题上所面临的主要复杂性就在于视体几何结构上计算的困难，因此第 1 个要解决的问题就是必须去尝试改正视体几何的结构。可便于计算的视体几何视域角度必须为 90º，如果在这种情况下，其平面便会形成一种透视视锥，这时问题就会显得简单多了，如图 9.24 所示。

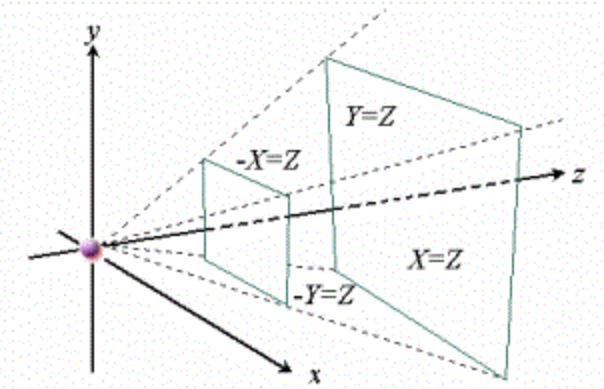


图 9.24　一种平面透视视锥

从图 9.24 中可以看出，其他 4 个平面的公式可写成：

```
Y=Z，-Y=Z，X=Z，-X=Z
```

现在我们通过这些平面的运算公式就可以方便地用在裁剪计算上了。不过一般会发生的情况是选择的视域角度会小于 90º，在这些情况下，我们可以尝试缩放原始空间来得到一个新的视体，而裁剪就能够通过简单的几何结构而执行运算了，一旦这个步骤结束后，空间再缩放回原始的构造就可以了。

我们可以选择一个足以代替这种多重处理过程的步骤，如此一来便可以决定在简单视体内裁剪，而进一步在较小的视角域中使用透视转换。但是如果这么做的话，那么我们也可能会失去 3D 裁剪的重要特性。也就是说被投影的顶点可能会位于屏幕的外部，因此在

这里必须执行 2D 屏幕边界裁剪才行。

当然如果您直接去进行这两种复杂的裁剪处理其实并不合理，因为它们可能会浪费掉相当多的运算资源，因此就必须导入另一种简化的办法那就是在执行视体裁剪的阶段时，我们就必须沿着裁剪面的前后进行裁剪，而剩下的几个面只要去执行简单的接收和删除测试就可以了。最后当图形投影到屏幕空间上时，再进行 2D 裁剪处理。

如同上面的论点一般，当我们在执行第 1 个步骤的时候，就要避免通过前裁剪面裁剪时可能会发生“除零”的问题，而且也要避免所有顶点可能发生的溢出问题。只有在 3D 空间中跨越很长的距离，而且仍然在视体和坐标最大数值边界内的顶点，它们还是有可能会发生溢出的情况，不过通常我们都会假设这种情况是不存在的。

既然可以得到等式“Y＝Z，-Y＝Z，X＝Z，-X＝Z”所描述的裁剪面，那么我们也就可以很容易观察出在体外的所有空间，其不等式将写成如下所示：

```
Y>Z, -Y>Z,X>Z, -X>Z
```

如果有某些图形上的所有顶点都符合上面不等式的话，我们就可以将这个图形抛除掉了。

在 3D 空间中个别的图形通常与某些复杂的对象会组合在一起，而这些对象可能是由几百个图形所组合所成。如果某些顶点被抛除了，其他的顶点就可能也要同样被抛除，在这里，我们可以利用这个特性，对对象上的图形执行抛除的测试。对于这些测试来说，我们需要一些精确的对象属性，而这些属性则可以告诉我们它是否可以被抛除。例如，当对象的中心在视体外的时候，这个中心就会不符合使用了，因为对象的某些部分可能还在视体之内，我们可以利用另外一种更好的方法来解决这种问题，那就是以某些简单几何结构的边界来封闭这个对象。如果边界体与视体没有交叉的话，这种抛除就可以安全地被执行。通常我们会使用的边界体有两种，这两种边界体分别是“边界盒”或“边界球”，如图 9.25 所示。

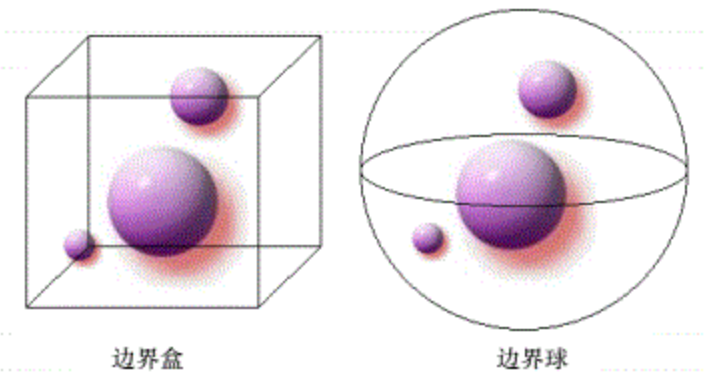


图 9.25　“边界盒”与“边界球”

边界盒可以表示对象的最小和最大空间坐标，但边界球的半径要由从对象中心算起的最远点来决定。

如果我们使用边界盒来作为边界体的话，我们就能够用来检查来自视体中所有可能的抛除顶点。举例来说，当边界盒最小的 x 坐标大于最大 z 坐标时，在 X=Z 平面外的对象就可以被抛除。尽管我们已经使用复杂对象的边界盒来抛除个别的图形，不过，这种边界盒又可能产生许多顶点的多边形。相应地，对于顺序顶点的计算又会更加复杂了。

同样的，边界球也可以使用如上述的方式来处理裁剪的动作。在这种情况下，我们必须计算从裁剪面到球心的距离。如果这个距离大于球的半径时，对象则可以被抛除。

在介绍完所有裁剪的算法原理后，相信您对于裁剪有了更深刻的印象，接下来再看看一种可以将隐藏面消除的算法。

9.5 消除隐藏面

到目前为止，我们还忽略了一个非常重要的问题，那就是屏幕上的一些图形可能会被另一些图形所挡住。例如在描绘一个由多边形面所组成的三维物体时，那么它的某一部分必然是会被挡住的，而屏幕上的显示必须是可以看见的东西。比方说，对于一个立方体而言，无论从哪个方向进行透视处理，我们最多只能看到其中三个面，因此就必须想出一种方法来决定哪些面是可以看到的。

如果使用从屏幕到世界的视觉处理方法的话，很自然地就可以保证只有图形上正确的部分才能显示于屏幕上。在这种视觉处理中，它的可见性是在屏幕的每一个图形上进行判断。当我们从眼睛中发出一条射线，而这一条射线穿过一个特定图形时，那么首先与这条射线相交的表面在这一个图形上就是可见的。从这个表面反射的光线能够进入我们的眼睛，让我们可以很清楚地看见这个物体。

在本节中将深入讨论隐藏面消除的一些演算方法。在计算机图形学中，由于最容易被使用到的图形就是多边形，所以下面要讨论的许多技术都是专门针对多边形。

9.5.1 背面剔除(back culling)算法

许多三维物体中，它们所占据的空间都被一些连续的表面所包围，在观察这些物体的时候，我们只能看到这些包围表面中的正面部分，而物体的背面则无法看到。“背面剔除算法”就是将这些看不到的背面多边形去除掉，如图 9.26 所示。

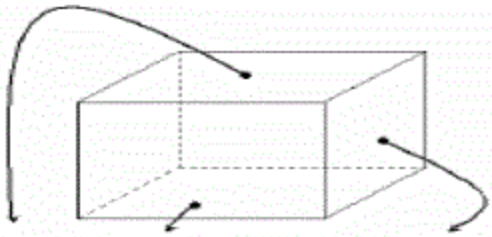


图 9.26　只显示三维物体能看得到的其中三面

如果位于表面上的任意两个点的联机都没有超出边界的话，那么这个多面体就是一个凸多面体，而凹多面体就没有这样的特性，如图 9.27 所示。

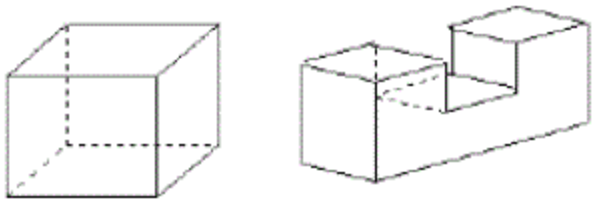


图 9.27　凸多面体与凹多面体

如果我们对一个多边形模型执行背面剔除，并且这个模型是一个凸多面体的话，那么在经过这样的处理后，就已经消除了所有的隐藏表面。由于这些物体的形状，所以隐藏的多边形就是组成背面的多边形。但是在消除凹多面体隐藏面的同时，这一种通用的技术就会出现一些特殊的问题；问题就是有可能某些物体的正面会被其他多边形的面所遮挡住，如图 9.28 所示。

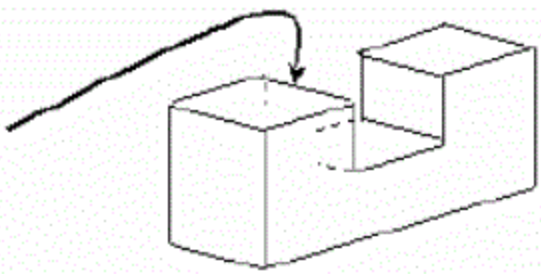


图 9.28　这个面被其他的面挡住了

此时位于背面的多边形仍然是不可见的，我们是可以很明确地消除它们， 这样做虽然不能解决这一类的问题，但是至少也降低了其复杂性。

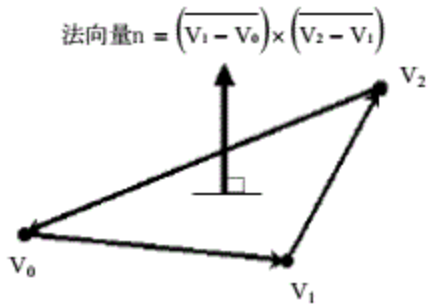
这一种推理也可以被应用在光线投射算法中。我们可以完成隐藏面消除的工作要完全

归功于这种算法的自然本性，而且背面剔除算法可以减少场景的复杂度，使我们不用再考虑那些复杂的隐藏面，更可以加快视觉处理的运算过程。

现在可以设计一套方法来决定一个多边形平面是位于物体表面的正面还是背面。其实使用法向量是在描述一个多边形的朝向来决定多边形平面是属于正面还是背面。当一个多边形的法向量与观察方向之间的夹角大于 90° 的时候，就表示这个多边形平面是位于物体的背面。

在数学上的“矢量积”和“纯量积”可以帮助我们解决上述的问题。首先计算位于一个多边形平面上的某两个向量的矢量积，使其得到这个多边形的法向量，而这两个向量则可以通过多边形顶点的差值来得到。接着，计算观察方向与法向量之间纯量积的符号，由此决定它们之间是否形成了大于 90° 的角度。如果其角度大于 90° 的话，这个多边形就要被剔除掉，而不用再考虑进行视觉投影的处理过程了。

在这种运算的过程中，值得我们注意的是两个向量的矢量积结果也是一个向量，而这个由矢量积所得到的向量与先前的两个向量成正交。换句话说，根据多边形平面上所形成的这两个向量，我们可以得到两种可能的法向量，而它们的方向正好相反，如图 9.29 所示。



法向量 n 的方向为 $V_0V_1V_2$ 平面的正面

图 9.29　法向量的方向

如图 9.29 所示，如果在连续顶点 V_0,V_1 和 V_2 上建立两个向量，那么法向量的方向将会产生在这两个向量的顺序中。我们可以很简单地将多边形表面的正面与顶点的反时针顺序联系起来，这就是定义表面法向量的基本原则。

说到这里，我们已经介绍了两种不同的视觉处理过程，并且还有两种不同的投影转换，而使用这些技术可以在成像过程中的某一特定阶段里完成背面剔除的演算。对于平行投影而言，投影线都具有相同的方向，并且与观察者的方向是一致的。就在投影阶段之前，观察方向的向量会指向 Z 轴方向，例如我们可以利用（0,0,1）来进行描述，同时也就减少了纯量积的计算量，其结果就等于法向量的 z 分量。如此一来就只需要计算法向量的 z 分量值就可以了。

对于透视投影而言，投影线会相交在观察者的眼中，它的方向是不同的。我们可以在

世界或观察空间中的任意一点上构造一个指向观察者眼睛的向量，并且使它指向该点的方向，这样就得到了这一点的观察方向了，如图 9.30 所示。

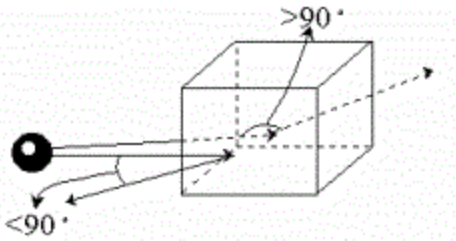


图 9.30　在世界或观察空间中的任一点构造一个指向观察者眼睛的向量

不过一旦物体经过透视转换，再从观察空间映射到屏幕空间之后，用来执行背面剔除的观察方向是恒定的，我们只需要对法向量的 z 分量进行简单地计算就没问题了。

背面剔除法其实可以在映射过程的开始阶段执行，也可以在世界空间甚至对象空间中进行处理。当然越早剔除掉背面的多边形，就可以减少执行多余的操作运算。在这里，您还要考虑到对于顶点所进行的 3D 转换；如果可以剔除掉不属于表面前方顶点的话，在对象空间中执行背面剔除运算将会节省其运算过程。我们可以对每一个顶点设置一个布尔变量，当某个顶点的多边形位于表面前方的时候，就将该顶点的布尔变量值设为“true”，在检查完所有多边形之后，就可以将没有被设置成“true”的顶点剔除掉，如此一来便可以避免对被剔除的顶点执行 3D 转换，因而减少了运算量。如果不选择执行上述操作的话，那么同样可以在世界空间中来进行背面剔除的运算。在世界空间中执行剔除运算是可以避免对一些多边形进行运算量很大的裁剪计算。但是剔除运算本身的运算量会比在光栅化执行之前多一些。

除了在运算时计算多边形表面的法向量之外，我们还可以在程序的映射循环之前预先计算好这些向量，并且将它们与每一个多边形表面联系起来。为了在世界空间中得到法向量，我们可以将顶点先转换到世界空间中。谈到这里，有一点要特别注意的是为了达到对一个向量变化的目的，我们只能去转换其线性的部分（也就是旋转与缩放转换），而平移转换就没有必要使用。因为它并不会影响向量值。将一个法向量从一个空间转换到另一个空间中的运算量往往要比直接在目标空间中建立该法向量还要多，但是由于光线处理的过程也要使用到单位法向量，所以便可以将两种运算的目的合并起来。然而在一个空间中建立一个单位法向量的运算量又要比从另一个空间中转换过来的运算量大上的许多，因此我们还是要对法向量进行转换。

同理，我们可以将其应用在屏幕到世界的视觉处理上。正如上面所说的，我们可以对这种方式进行一些修改，避免计算光线和背面多边形的交叉点。由于在这种视觉处理的方法中，我们没有明确地把坐标转换到视觉空间中，所以它留下的只是可能剔除的世界和对象空间。由于这种光线的方向很难被提前看见，因此它们可能正好与观察者不可见的多边

形发生相交，如果如此，在对象空间中就没有剔除的意义了。

9.5.2 排序

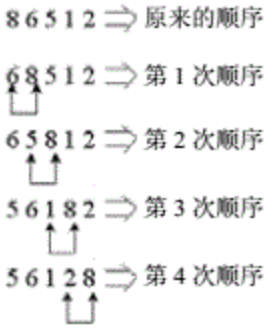
如同前一节看到的一样，背面剔除法对多边形模型中的隐藏面消除来说是不太够的。一般情况下，在映射世界空间到屏幕的时候，我们要找到所有被遮挡起来的多边形是比较困难的。而要解决这一个问题，每一个多边形可以相应到所有其他的多边形上进行反复裁剪，并检查得到的表面深度信息。如果表面沿着观察方向比裁剪多边形更远的话，那么它就一定会被遮挡住，所以我们就要将它剔除掉。在这一个处理的过程结束时，便会得到一系列新的多边形，而它们都是可见的。

对以上的方式来说，它的运算量还是会很大，而且不一定实用。其实我们还可以利用另外一种方法来剔除掉这些看不到的顶点，方法就是利用显示卡硬件的缓冲区来加速其运算（Z-Buffer）。不管场景中的多边形有没有挡住其他的多边形，只要按照从后面到前面的顺序光栅化图形就可以正确地显示所有可见的图形了。简单地说，那就是将离观察者最近的一个多边形最后进行光栅化处理，这种方法就称为“画家算法”。

在这种算法中，我们必须注意到当多边形光栅化处理的运算数据消耗太多时，例如使用比较复杂的纹理映射和光线，再使用从后面到前面的顺序贴图就不太好了，因为多边形会因而被遮挡住，使得运算资源会白白浪费大量的工作，所以在这个时候，我们可以使用前面所说的一些方法来映射图形可能会较为适当。

为了得到从后面到前面的顺序，我们可以沿着观察的方向，按照多边形的深度信息对它们进行排序。如果使用透视转换的话，在空间中就不能使用排序方法，因为与观察方法相对的不同多边形会发生一些变化。

为了沿着观察方向进行排序，我们必须找到一个多边形比较的条件。最简单的方法就是比较一个多边形上所有顶点的最大 z 坐标（离观察者的距离），也可以比较多边形顶点 z 坐标的平均值，其实排序的算法还有很多种，而最直截了当的方法就是使用“冒泡排序”（bubble sort）。在这种算法中，列表中会将一个物体与其前面的物体进行比较，如果不满足排序的条件，便将它们的位置进行交换。如果从列表中的第 1 个物体开始对每一个物体进行比较，那么对于单个物体来说，就会逐渐被交换到正确的位置，就像一个气泡被逐渐推到水面一样。我们将一个物体移动到正确的位置只需要进行 n-1 比较，排序过程如下所示：



这样一来便可以很轻易地分出什么是可见和不可见的了。

9.5.3 八叉树

在上一节已经看到，一般多边形模型隐藏面消除的代价极为昂贵。然而在大多数情况下，我们要处理的对象有一些特殊的属性，利用这些属性可以减轻隐藏面消除的工作量。例如在以“高度值运算”（Height-field）来处理表面的地形曲面时，就可以很容易地获得多边形视点从前面到后面的顶点顺序。由于这种表面方法具有矩形自然的属性，地形就可以被分割成许多正方形的单元格。

再来讨论另一种情况，那就是当观察者在一些单元边界内位于虚拟地形表面上或表面的上方。我们能够把整个地形分割成 4 个规则的子地形，如图 9.31 所示。

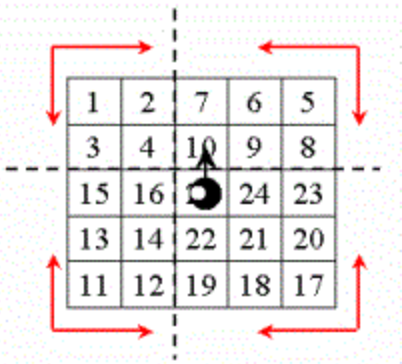


图 9.31　把整个地形分割成 4 个规则的子地形

由于这种表达方法的规律性，这 4 个子地形在观察者屏幕上的投影几乎没有任何交叉，只有少数由于透视转换的缘故，交叉点才有可能出现由小到大映射的子地形所得到的修复动作。

在每一个分割区域里，我们通过最远的多边形而得到由后面到前面的顺序，一行一行

或一列一列地朝向观察者所在的位置进行处理。包含观察者所在位置的多边形，最后才会被进行光栅处理。例如，我们可以将地形单元的光栅处理顺序编排为：

```
第1次：1、2、3、4
第2次：5、6、7、8、9、10
第3次：11、12、13、14、15、16
第4次：17、18、19、20、21、22、23、24、25
```

这种编排能够有效地保证在任意投影地形的可见性。然而在每一个地形单元里，正如我们所看到的一样，它们可能是由两个三角形所组合而成的。所以光栅处理的顺序将会依据观察者在场景中角度的不同而有所改变，如图 9.32 所示。

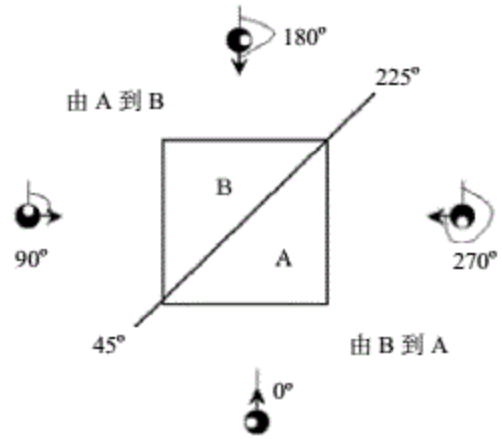


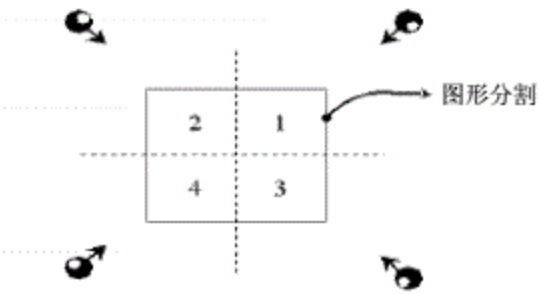
图 9.32　光栅处理顺序将根据观察者在场景中的不同角度而改变

从图 9.32 中我们可以很清楚地看到当观察者方向范围在 45° 到 225° 的时候，多边形 B 会在 A 之后被映射。否则当观察方向不同的时候，则以相反的顺序来映射多边形（A 在 B 之后被映射）。如果观察者的角度正好在 45° 或者 225° 的时候，映射的顺序就无所谓了，因为多边形的投影在屏幕上是没有交叉的。

这种有规律的特性，我们也可以将它作为由后到前寻找地形顺序的依据，而这种特性就可以使用“八叉树”的存储规则，或者也可以用在空间占用的矩形来描述映射。在这两种情况中，我们能够利用类似高度场的方法，扩展成为处理三维顶点而不是二维顶点。

“八叉树”的处理规则就是利用递归结构的方式来进行，在每个细分的层次上有着同样规则的属性。因此在每个层次上可以利用同样的编排顺序，以获得整个结构元素由后到前的顺序依据。

“四叉树”是“八叉树”在一个平面上的特例，为了简化这种情况，我们可以利用图 9.33 再编排一个四叉树的顺序规则来。



视点从 90°~180° 时，其图形顺序为：3142
视点从 270°~0° 时，其图形顺序为：2143
视点从 180°~270° 时，其图形顺序为：4231
视点从 0°~90° 时，其图形顺序为：1234

图 9.33　四叉树的顺序规则

如图 9.33 所示，四叉树的顺序规则取决于观察者的方位，这些方位一共分成 4 种不同的编排顺序，而每一种编排顺序都使用在特定的方位范围内。

如同上述的方式，我们可以使用在八叉树的规则编排上，并且通过它将地形平面扩展到三维，最后可得到 8 种不同的编排顺序，而每一种编排顺序都可以应用在不同的方位角度上。

9.5.4 二元空间分割树

二元空间分割树是一种空间分割的方法，称为 Binary Space Partitioning Tree，简称“BSP Tree”。

BSP Tree 是由 Fuch 和 Kedem 在 1980 年提出的，通常被应用在平面的绘图上。因为物体与物体之间有位置上的相联性，所以当平面每一次重绘的时候，我们就必须考虑平面上的各个物体位置之间的关系，然后再加以重绘。

对于这些要求，BSP Tree 采取的方法就是在一开始将数据文件读进来的时候，就将整个资料文件中的数据（即墙的数值）先建成一个二元树的数据结构，如图 9.34 所示。

二元树会建立许多 NODE 的数据结构，而 NODE 中的数据结构代表的是一面墙的意思，此二元树右方的子树，称为“FRONT”，它是用来存放位于此 NODE 前方的所有物体；此二元树左方的子树，称为“BACK”，它是用来存放位于此 NODE 后方的所有物体，所以当地形数据被读进来的时候，BSP Tree 也会同时被建立，而且 BSP Tree 也只会建立一次。

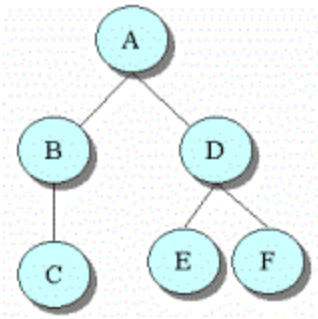


图 9.34　将资料中的数据生成二元树

当视点开始移动的时候，平面景象就必须被重新绘制。而重绘的方法就是以视点为基础，对此 BSP Tree 加以分析，只要在 BSP Tree 中，且位于此视点前方的话，它就会被存放在一个串行当中，最后我们只要按照串行的顺序一个一个地将它们绘制在平面上就可以了。

在 3D 游戏中，BSP Tree 的运算非常复杂，不过还可以利用另外一种方式来绘制多边形，那就是我们利用深度测试来显示室内环境及场景。深度测试是一种可以决定哪个对象在场景中离摄影机最远。在深度测试之后，它将依据最远到最近的顺序来绘制多边形，所以可以正确地看到每一个对象。

如图 9.35 所示，“B”区域会显示在 Y 多边形的上面，而“A”区域会显示在 X 多边形的上面，如果不做深度测试的话，在“A”与“B”两个区域里，我们可能会看到 X 多边形某个部分在 Y 多边形的上方，且 Y 多边形某个部分在 X 多边形的上方。

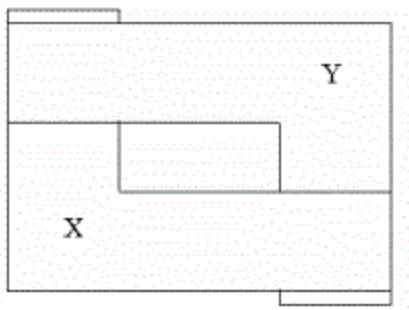


图 9.35　X（Y）多边形的某一部分在 Y（X）多边形的上方

如果使用 OpenGL 或 Direct3D 的函数库去显示此多边形的话，它们就会自动做深度测试（Z-Buffer），但是它们这样做没有效率，而且执行速度会非常的慢，更何况执行速度一直是 3D 绘图的最大缺陷，所以这就是为何要使用二元空间分割树的原因。

如同前面介绍的一样，二元空间分割树的技术可以利用树节点（Nodes）来快速和有

效率的排序多边形，使其减轻系统的负担，而且在计算机运行的时候，我们又可以很简单地了解摄影机到最远程的多边形之间的关系。

如图 9.36 所示，这个场景就需要深度测试。如果我们不使用深度测试的话，A 和 E 的墙在映射后可能会遮住 D 与 B 这两面墙，或是 E 面墙的显示会令人不满意，这种情况如同前面所说的一样，不合理的图就会在这里产生，所以我们就需要在这里建立一个二元空间分割树来改善这个问题。

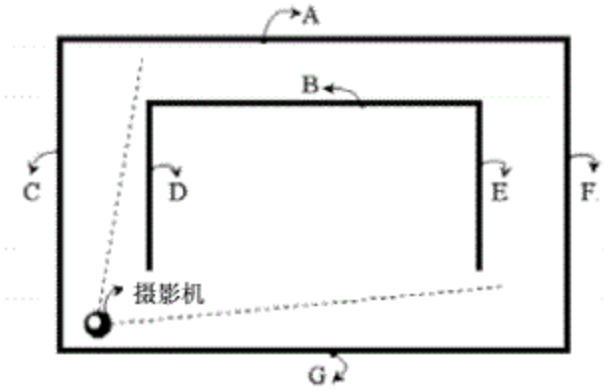


图 9.36　摄影机到最远程多边形的关系

当二元空间分割树在开始建立的时候，它必须要先完成下面几项任务：

（1）从函数中选择一个多边形。
（2）通过平面方程序（Ax+By+Cz+D=0）转换多边形为平面数据。
（3）通过函数分类所有的多边形（前方、后方、重叠、穿越）。
（4）分割跨越平面的多边形。
（5）如果任何一个多边形在平面的前方，我们就回到步骤（1）并处理这些多边形。
（6）如果任何一个多边形在平面的后方，我们就回到步骤（1）并处理这些多边形。
（7）在目前的节点中加入与平面重叠的多边形于串行中。

在步骤（1）里，可能很难取舍一个可用的多边形让我们进行以下的每一个步骤。当我们要选择一个多边形的时候，就必须先将所有的多边形通过一种条件式的函数来选择所需的多边形。在这个范例中，我们选择了多边形 B 为主 Node。在步骤（2）中，将多边形 B 转换成一个平面多边形，而这个平面多边形则是由两个向量无限地延伸下去，而且等一下就要使用这一个平面多边形来分类场景中的所有多边形。

现在有了这个平面多边形 B 之后，在步骤（3）里就必须将所有的多边形加以分类。在这个范例中，A 多边形在 B 的后面，D 与 E 多边形在 B 的前方，而 F 多边形则跨越 B。

我们已经通过这个方式分类出所有多边形，并且使用平面方程序来取得这些点到面的

距离，接着再以距离为依据划分出其他的多边形。在步骤（4）中，需要分割任何跨越主 Node 的多边形，如果我们完成了跨越主 Node 多边形的分割以后，应该可以看到如图 9.37 所示的情况。

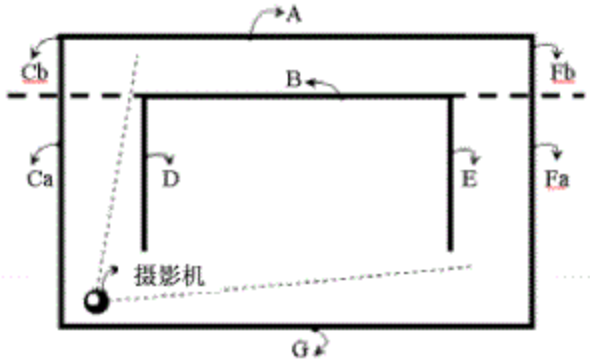


图 9.37　跨越主 Node 多边形的分割

请您特别留意，在步骤（5）与步骤（6）中，它们会要求在平面前方和后方的多边形再回到步骤（1）中运算，这种技术称为“递归”，如果这些多边形只在平面的前方和后方时，自然就要执行步骤（5）与步骤（6）的动作。如果已经执行了这些步骤后，我们还是不能建立一个完整的节点结构，因为按照上面所说的那样，必须在最后的步骤中，加入任何与目前节点平面重叠的多边形到节点串行中，如此一来才算是完成建立第 1 个分割节点的结构串行，如图 9.38 所示。

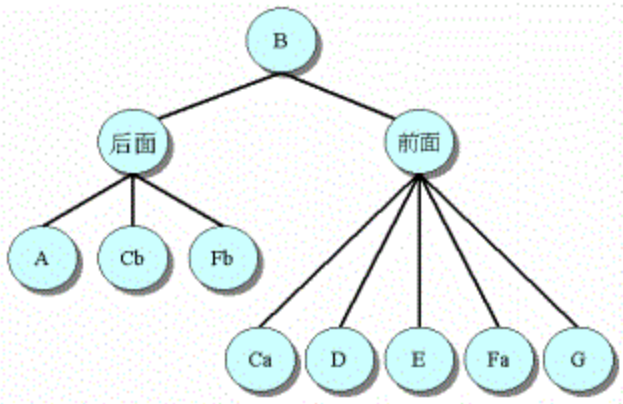


图 9.38　较详细的二元树

我们是不是很容易将二元树结构给分割出来了！不过，这样的二元树结构还是不足够，所以还必须做出更多的切割才行。接下来，就以平面前方的多边形再做另一次的分割。在这里，我们再利用 2D 平面为主 Node 来做分割的动作。分类和切割后的结果，如图 9.39

所示。

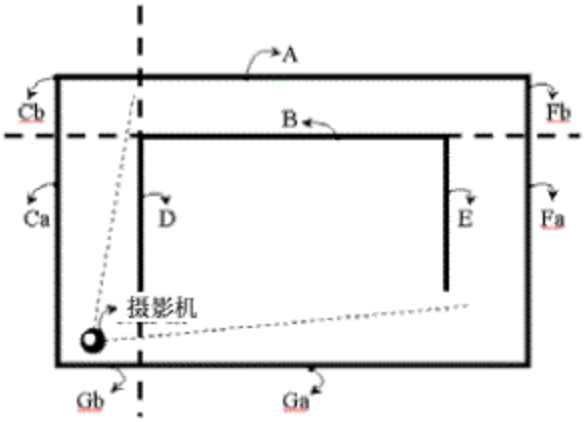


图 9.39　分类和切割后的结果

现在的二元空间分割树看起来就如图 9.40 所示。

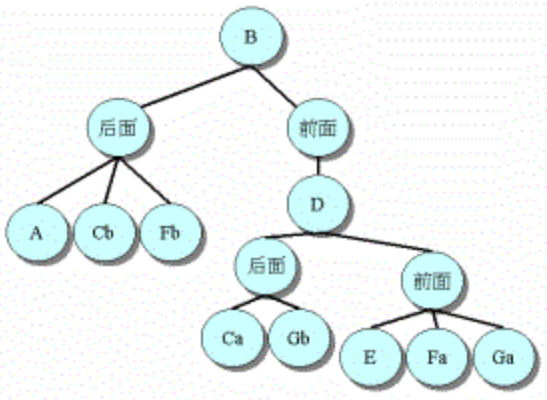


图 9.40　二元空间分割树

如果再继续分割上述二元树结构的话，其二元树结构就会变成如图 9.41 所示。

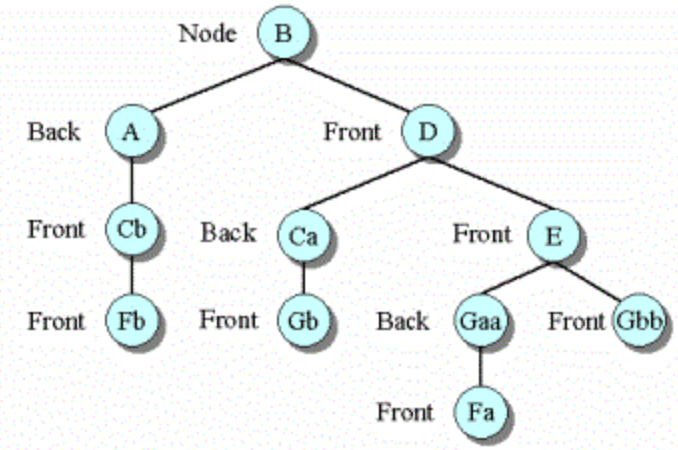


图 9.41　二元树结构

当二元树结构全部分割以后，我们一共可以得到 11 个节点，而每一个枝节都有一个父节点。实际上，除了少数多边形之外，这里没有做太多的分割运算。当在计算机上进行映射图形的时候，我们将使用这个分叉树来作为映射的基本原则。

当映射多边形的时候，我们可以利用摄影机的位置与每个平面做对比，然后再以距离为基本的显示顺序原则。

一开始，基本节点是 B 多边形，接着再执行下面的步骤：

（1）求出平面与摄影机之间的距离。

（2）如果距离大于 0 的话：

- 移向后面节点并回到步骤（1）。
- 显示目前节点内的多边形。
- 移向前面节点并回到步骤（1）。

（3）如果步骤（2）是错误的话：

- 移向前面节点并回到步骤（1）。
- 显示目前节点内的多边形。
- 移向后面节点并回到步骤（1）。

完成上面的步骤之后，将绘出节点的顺序排列如下所示：

```
Node A, Node Cb, Node Fb, Node Fa, Node Gab, Node Gaa, Node E, Node Gb,
Node Ca, Node D
```

我们已经介绍完 BSP 的分割原则了，接下来再来介绍另外一种 3D 映射常常会用到的

演算技术。

9.5.5 细节层次

在实时 3D 真实感游戏的绘制过程中，如果要得到某种特定视觉效果的话，绘制图像算法的选择性就会被限制住。因而要实现 3D 映射的实时性，只有从需要绘制的 3D 场景本身着手。在当前的真实感图形学中，需要绘制 3D 场景的复杂度非常高，而且一个复杂的场景也有可能会包含几十甚至几百万个多边形，所以要实现这种复杂场景的绘制是很困难的。

其实我们可以利用一种简单的方式来解决这一类问题，那就是利用减少场景的复杂度来提高图像的绘制速度，而“细节层次”（Level of Detail，简称 LOD）这一门用来显示和简化场景的技术就是在这种背景下产生的。

在场景中，我们可以合并这些可见而不损失画面的视觉效果。细节层次技术最初是为了简化多面体网格物体而设计的一种算法，这些复杂的多面体网格往往是扫描真实 3D 物体以后才会得到的。为了真实反映原物体的表面 3D 变化，在扫描过程中所采取的取样点就非常密集，而这些密集的网格则为绘制 3D 场景带来极大的困扰，因此人们就开始着手研究复杂多面体网格的简化演算。

细节层次绘制简化的技术就是在不影响画面视觉效果的条件下，逐步简化景物的表面细节来减少场景的几何图形所产生的复杂性，并且又可以提高绘制算法的效率。该技术通常会对一个原始多面体模型建立出几个不同逼近程度的几何模型，与原模型相比，每个模型均会保留一定的层次细节，当观察者从近处观察物体的时候，则采用精细的模型图素；当观察者从远处观察物体的时候，则采用较粗糙的模型图素。这样对于一个复杂的场景而言，我们可以减少场景的复杂度，而且绘制图像的速度也能够大幅度的提高，这是层次细节显示和简化技术的基本原则。不过值得注意的是当视点连续变化的时候，两个不同层次的模型之间就会存在一个明显的跳跃，所以必须要在相邻层次的模型之间产生一种光滑的视觉处理，使得绘制的图像呈现高度真实感。层次细节技术的研究主要是集中在如何建立原始网格模型不同层次细节的方法，以及如何建立相邻层次多边形网格模型之间的几何形状处理演算。

对于原始网格模型不同细节层次的模型建立，我们可以假设场景的模型都是以三角形的网格来呈现（在实际应用中，为了绘制方便，3D 场景一般都被转化为三角形网格状），因此便可从网格的几何特性着手，而这种技术可以归纳出 3 种不同的基本简化演算，分别是“顶点删除”、“边界压缩”及“面的收缩”。

顶点删除

“顶点删除”技术是删除网格中的一个顶点，然后再对它的相邻三角形做出一个空洞，

这个空洞就当作是三角的剖分，以保持网格的一致性，如图 9.42 所示。

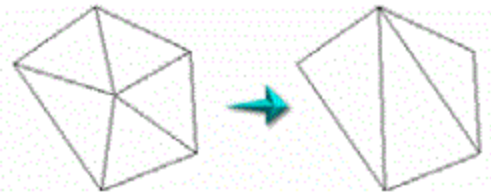


图 9.42　将中间顶点删除

边界压缩

“边界压缩”技术是将网格上的一条边压缩成一个顶点，这个顶点与该边相邻的两个三角形一起退化掉，而再把它的两个顶点融合成一个新的顶点，如图 9.43 所示。

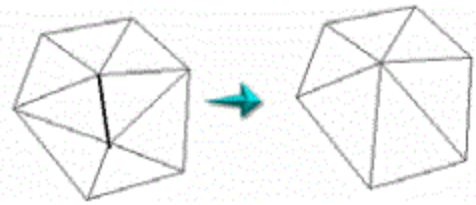


图 9.43　将中间顶点删除

面的收缩

“面的收缩”技术是将网格上的一个面收缩成一个顶点，让该三角形本身的和与其相邻的 3 个三角形一起退化掉，而它的 3 个顶点则收缩成另一个新的顶点，如图 9.44 所示。

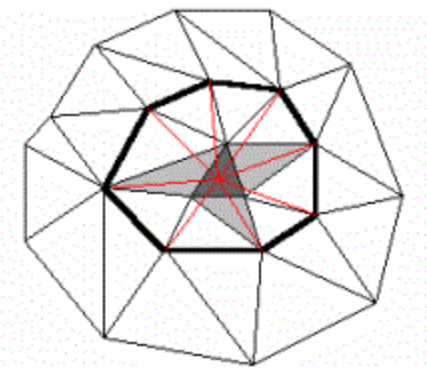


图 9.44　将中间的面收缩成一点

利用上面的基本演算，只要确定每一次演算网格场景所带来的误差值，用这个误差值来计算原始网格上的每一个基本元素的误差值当作是它的权利值（不会被删除的权利），然后再开始循环进行网格基本简化的演算。在每一次循环中，选取最小的权利值来执行简化的动作，并更新变化的网格信息，且重新计算改变基本网格元素的误差值插入到队列中，再开始下一个循环，直到队列的最小误差达到使用者的设置值，或者是使用者希望简化的网格数。

通过上述方法，我们可以建立原始场景不同细节层次的模型，这些建立的模型就具有一定层次细节，而且相对于原始网格。它们彼此之间的误差是逐步递增的，这样的模型可以使用在层次细节的显示上。建立相邻层次多边形网格模型的几何形状光滑处理，基本方法就是通过对应网格基本元素的位置来实现光滑的处理，这个问题的关键就是要如何得到两个相邻层次的多边形网格模型之间的对应关系。

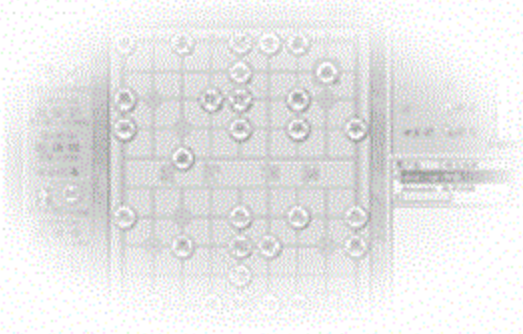
对于“顶点删除”和“面的收缩”演算而言，可以在对象与其相邻基本元素之间建立起相对应的关系；对于“边压缩”演算而言，只要简单地将压缩边上的两点与压缩后的新点建立起相对应的关系就可以了。在有了这些相对应的关系后，便可以通过这种关系来实现光滑面的处理。

细节层次简化是实时 3D 图形学中应用比较多的一项技术，通过这种技术，我们可以简化场景的复杂度，同时也可以采用不同分辨率的模型来显示复杂场景的不同物体，以满足实时 3D 的要求。

在实时 3D 的算法中，我们已经将一些常用的公式与原则讲解完了，在后面的章节里，我们再来介绍一套游戏中较为常用的物理模拟算法。

第 10 章

常用数学与物理算法



对于一套游戏来说，物理分子系统可以增添游戏的丰富性，可是有许多读者对物理分子的演算却一知半解。本章将介绍基本的物理分子系统的概念与算法，即使读者不能理解物理原理，也能套用算法，实现自己的目的，像是粒子爆炸、碰撞、风动效果、重力加速度等物理现象的模拟。

10.1 常用数学运算公式

不管是在2D或者是3D的系统中，我们可能会使用复杂的数学公式来运算物体的运动，而这些复杂的数学运算公式对于一个初学者来说，实在是太困难了，可是我们又考虑到它在游戏中的重要性，所以在本章中列举了几个较为常用，而且在游戏开发过程中必须使用到的数学运算公式，并且讨论它们的演算方式与用法。

10.1.1 内积

内积是力学与3D图形学中的知识。在3D图形学中，内积用于计算两个向量之间角度的余弦，如图10.1所示。

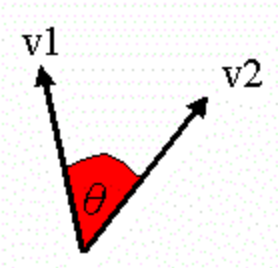


图10.1　向量间的夹角

在进行光照计算与背面消隐时，我们可能会使用到它，当然在力学中也不例外。接下来，我们来看看这两个向量之间夹角的余弦内积应该如何计算。

在求内积之前，首先我们来了解应该如何计算向量的长度（已知向量的大小）。这个问题的关键在于计算两点之间的距离，我们将它分成两种不同的维数系统来求得向量的长度。

2D系统

定义一个向量(x,y)，其向量长度的计算公式为：

$$向量长度=\sqrt{(x_2+y_2)}$$

先定义两个向量。向量 A(x1,y1)及向量 B(x2,y2)。其内积的公式为：

```
A*B= (x1,y1) * (x2,y2) = (x1*x2+y1*y2)
```

3D 系统

定义向量(x,y,z)，其向量长度的计算公式为：

$$向量长度=\sqrt{(x_2+y_2+z_2)}$$

当算出向量的长度之后，接下来，便可以开始求两个向量之间的内积了。

定义两个向量。向量 A(x1,y1,z1)及向量 B(x2,y2,z2)。

```
A*B= (x1,y1,z1) * (x2,y2,z2) = (x1*x2+y1*y2+z1*z2)
```

我们已经可以求出向量的内积了，不过我们却还不了解这个内积的值到底要做什么用。其实这个值可以用来计算两个向量之间的夹角余弦。其夹角余弦公式如下所示：

```
cos(θ)=(v1*v2)/(v1向量长度* v2向量长度)
```

那么该内积运算值的范围就会在-1~1 之间。如果这两个向量都指向同一个方向的话，那么该内积运算的值就会为“1”；如果这两个向量方向正好相反的话，那么内积的值就会为“-1”；如果这两个向量互成直角的话，那么内积的值就会为“0”，内积运算的值实际上就是可以告诉我们这两个向量之间的相似性，所以现在您应该明白为什么我们要求两个向量的内积了吧！

当要判断一个多边形是否会面向摄影机的时候，只需要计算多边形的内积就可以了。在判断多边形是否会面向摄影机的运算时，必须取得多边形中的两个重要的向量，一个是该多边形的法线向量，另一个是从摄影机到该多边形的顶点向量。如果该内积运算值小于 0 的话，那么这就表明了该多边形是正对着摄影机，也就是说我们正在看着该多边形的正面。如果该内积运算值大于 0 的话，那么这就表示该多边形是背对着摄影机，也就是说我们正在看着该多边形的背面。

在介绍完向量的内积之后，接下来，再来看看一般被使用在 3D 系统中的“叉积”。

10.1.2 叉积

在 3D 系统中（见图 10.2），叉积与内积都是 3D 图形学与力学的好帮手，而且在各个地方都有可能会使用到它。

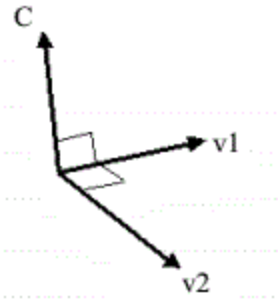


图 10.2　3D 系统坐标

它是以 v1 与 v2 作为输入向量，并且求出返回向量 c，而返回向量 c 则会垂直于两个输入向量 v1 与 v2。输入向量 v1 与 v2 的长度和它们夹角之间正弦乘积会等于向量 c 的长度。所以，如果两个平行输入向量值为 0 的话，其向量 c 的方向就会指向任意方向。其公式如下所示：

```
向量c的长度＝向量v1的长度*向量v2的长度*sin(θ)
```

叉积的计算

首先定义两个输入向量 v1(x1,y1,z1)与 v2(x2,y2,z2)，而输出向量为 v3(x3,y3,z3)。其计算公式如下所示：

```
x3=(y1*z2)- (y2*z1)
y3=(z1*x2)- (z2*x1)
z3=(x1*y2)- (x2*y1)
```

叉积的作用

在 3D 图形学中，经常需要计算一个多边形的法向量。在 3D 空间里，从一个点发出的两条线段，只要这两条线段不在同一条直线上，那么它们就可以确定一个平面的存在。对于一个多边形来说，如果已知同一个顶点的两条边就可以确定该多边形所在的平面了，如图 10.3 所示。

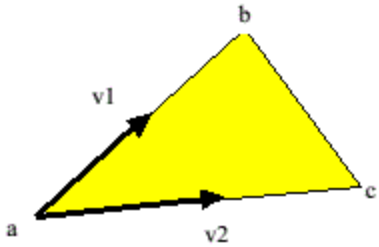


图 10.3　两边确定一个平面

因此只要计算同一个顶点两条边的向量叉积，就可以求出该多边形的法向量了。

如果要求这两条向量边的时候，一定要注意它们的先后顺序，如果取反了的话，计算出来的法向量则会与实际的法向量相反。一般而言，我们都是按照逆时针的方向来取得各顶点的坐标，只要记住这个顺序，就可以正确的求出边的向量了。如同 OpenGL 开发函数一样，必须按照逆时针方向来取得各顶点的坐标。当然，也可以改成顺时针方向来取得各顶点，但是一定要记住，所设置的法向量是什么方向，以免尔后在使用法向量的时候会发生意想不到的错误。

10.1.3 四元数

四元数最早是为了扩展复数的应用而产生的，后来有人发现四元数也可以应用在计算机图形学上，作为表示旋转的方法之一。

通常复数的形式，可以将它写成：

```
xi+y
```

其中 x 是虚部，y 是实部。不可思议的是 i 的平方根会等于 1，这种情况是不可能存在的，所以称这种方程序为“虚数”。

假如“xi + y”是一个复数的话，那么可以将它作为四元数中一组很特别的复数。四元数不仅只有一个虚部，而是具有 3 个不同意义的虚部。四元数的形式我们可以将它写成：

```
xi + yj + zk + w
```

而这里的 i、j、k 平方根都会等于 1。在本节中，我们要忽略掉四元数中平方根的形式，并且将虚数写成：

```
[w,(x y z)]
```

旋转（Rotations）

四元数可以用来表示所有形式的旋转。正如在前面所看到的那样，四元数的形式仅需要 4 个浮点数来构成，所以它只是给对应的矩阵添加了第 4 个数。这个属性可以帮助我们处理旋转的运算，但是应该如何才能取得“[w,(x y z)]”的旋转结构呢？

答案就在这里，首先需要计算四元数的平均值，而四元数的平均值与向量有点相似，它可以作“$w^2 + x^2 + y^2 + z^2$”平方根的计算。如果运算出来的平均值为 1 的话，就得到所谓的“单位四元数”。

四元数并不是天生就具有旋转的特性，因此还要做一些转换四元数或其他必要的工作才可以让四元数具有旋转的特性。我们可以将空间的四元数转换成“欧拉角”或“轴与角”的表示法，并且转换成可供开发工具（OpenGL 或 DirectX）所使用的矩阵。

下面将四元数转换成旋转矩阵的形式：

$$\begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x_2-2y_2 \end{bmatrix}$$

这是惟一需要变换的矩阵，因为不管是 OpenGL 还是 Direct3D 开发工具，它们都不接受任何指定旋转矩阵以外的东西。例如，我们比较喜欢使用 OpenGL 的旋转函数“glRotate”，而不是使用“glMultMatrix”来处理旋转的话，应该将四元数转换成“轴与角”的表示形式：

```
s=x²+y²+z²
Ax=x/s
Ay=y/s
Az=z/s
α=cos(W)
```

如此一来就可获得对应于 glRotate 函数的轴与角的参数。

转换四元数

当然，我们可能也会利用四元数转换的常规表示法。将矩阵转换成为四元数并不是经常要做的事情，因为矩阵本身根本就不是实现旋转最实用的方式，所以要采用另一种“欧拉角”的表示方式。我们希望使用“欧拉角”的表示方式，因为它们可以很容易地表示 3 个轴与角的旋转组合，并且让轴与角的旋转容易地转换成四元数，如下所示：

```
s=sin(α/2)
w=cos(α/2)
```

```
x=sAx
y=sAy
z=sAz
```

组合旋转

在讨论为什么要使用四元数之前，先来讨论一个需要解决的问题。当要组合两个旋转矩阵的时候，可以使用两个四元数相乘来实现。这两个 Q1 与 Q2 四元数的相乘形式，可以将它定义成：

```
[w1*w2-V1-V2, (w1*V2 + w2*V1 + V1×V2)]
```

在上述方程序中，“*”代表的是纯量乘法、“×”代表的是向量的叉积，而“ϕ”是向量的内积。向量 V1 与 V2 是由 Q1 与 Q2 的 x、y、z 所分组而成的。请注意，矩阵在这种情况下，四元数的乘法是不可进行交换的。

四元数的好处

现在获得了所有关于四元数的重要信息，不过我们却还没有明白为何要使用四元数的原因。其实我们使用四元数的原因主要有两个理由，一个是我们可以避免“万向节锁”的情况发生，另一个是在旋转时它可以允许平滑插值（*SLERP*）的产生。

万向节锁是一种容易影响到欧拉角表现的现象，简单地说，它意味着我们将在某些时候会失去角度上的自由性，而这都要归并于欧拉角必须要用球坐标系统来表示。当物体绕着某一个轴旋转时，它则不能够再适应其他的轴了。例如，物体在空间中的某一个位置上，在那里刚好会有两个轴的作用相互抵消掉了，看起来就好像其中一个轴断掉一样。而四元数就不会遭受万向节锁的困扰，因为它不用分离三个轴来表示旋转。

使用四元数的第 2 个理由就是它可以实现两个状态之间的平滑插值。的确，四元数是可以支持球形线状插值的，这意味着这些顶点沿着球体表面传播就像是从一个方位移动到另一个方位一样。如果在两个轴或角度之间做一个线性插值的话，可能会在一个不是单位长度的轴那里停止，而在某种情况下，轴实际上已经穿越了原点，所以应用程序不得不处理绕着零长度轴旋转的问题。这不是一个好现象，因此必须要避免使用球形线性插值。

10.1.4 两点间距离

两点之间距离的计算其实很简单。这一例程适用于任何范围的数字中，因此可以把它应用在任何 2D 或 3D 的程序中。

在 2D 系统里，先定义两个点 A 和 B，它们的坐标分别为(x1,y1)与(x2,y2)，而两点之

间的距离公式为：

```
x=x2－x1
y=y2－y1
两点距离＝(x²+y²)^0.5
```

在 3D 系统中，先定义两个点 A 和 B，它们的坐标分别为(x1,y1)与(x2,y2)，而两点之间的距离公式为：

```
x=x2－x1
y=y2－y1
z=z2－z1
两点距离＝(x²+y²+z²) ^0.5
```

正如所看到的一样，这些公式有平方根运算，如果想提高程序代码执行速度的话，就应该尽量去避免平方根运算。如果有必要的话，可以仅仅执行一个平方根就可以了。

如果不需要得到非常精确的距离值，还可以考虑使用查表法来避免平方根的运算。比方说，在球体间进行碰撞的测试时，只是希望知道它们有没有发生碰撞而不要求碰撞的范围大小，那么就可以不使用平方根来进行运算。

上述这些是在后面设计游戏时将会使用到的数学公式，相信您对于 2D 或 3D 的数学演算公式有所了解了吧！

在下一节中，将开始深入讨论一些运行在游戏中的物理原则。

10.2 熟悉的物理例程

不管哪一类电子游戏，我们都能得到物理学被运用在其中的影子，例如赛车游戏中，车辆横行于马路上的速度运算；或球类游戏中，球对于地面的反应。如果今天我们不将现实生活中的自然现象加入到游戏中的话，游戏本身看起来就显得非常不真实。例如在游戏中，如果水的流向由下往上移动的话，这是很不合理的，由此可见物理学在游戏中占有很重要的地位。

10.2.1 速度

对于每个人来说，“速度”是一个相当熟悉的名词。所谓“速度”其意思是说单位时间内所改变距离的量。例如，当我们在开车的时候，可以把车开到每小时 100 公里，这就是速度的描述，而它的含义就是在一个小时之内，车子可以跑 100 公里远的距离。当在游戏中要表现速度的时候，只要在物体坐标位置上加上一个速度常量，这个物体则会在游戏

中以匀速度向着速度常量所指的方向前进。例如，在 2D 环境中，假设某一个物体的坐标位置为（x,y），而将它的速度常量设置为“α”，其速度的方程序可以将它写成：

```
x=x+α
y=y+α
```

如果一直重复不断地执行上述方程序，在 2D 环境中，则可以看到这个物体向着某个方向前进了，如图 10.4 所示。

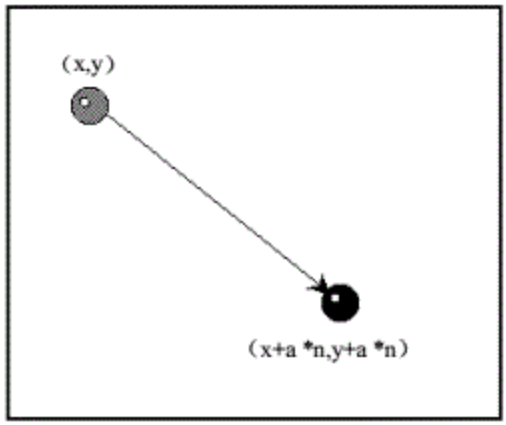


图 10.4　物体向着某个方向运动

在上述例子中，可以看到这个物体的 x 与 y 坐标都加上一样的 α 速度常量。事实上，它们可以分别加上不同的速度常量，如图 10.5 所示。

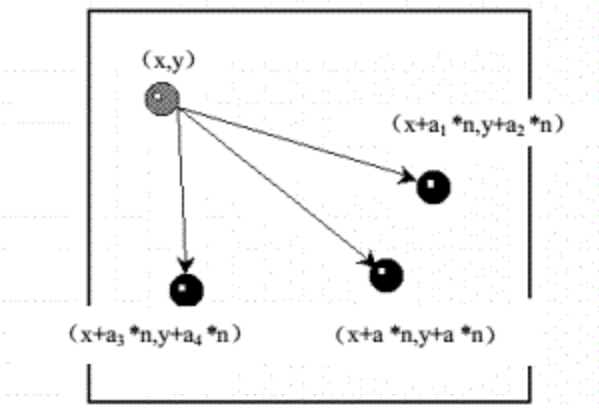


图 10.5　不同的速度常量

如果这个速度常量为负值的话，物体则会向相反的方向移动，如图 10.6 所示。

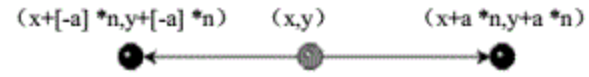


图 10.6　物体的速度与移动方向

如同上面所说的那样，这个物体加上了一个速度常量之后，它则会在空间里以等量的速度前进。不过，在现实生活中，物体很少会以匀速度进行运动，换句说话，物体的速度是可增可减的。接下来，我们就来讨论这种可以让物体的速度递增或递减的物理变化。

10.2.2 加速度

加速度是由速度所衍生出来的物理变量，它是单位时间内速度的变化速率。例如，当我们踩下车子的供油踏板时，车速则会不断地改变，直到速度稳定为止。

速度是单位时间内物体移动的一个常量，而加速度则是速度的一个变量，当物体在空间中移动的速度越来越快或越来越慢时，我们只有靠加速度这个变量来决定，其变化情况如图 10.7 所示。

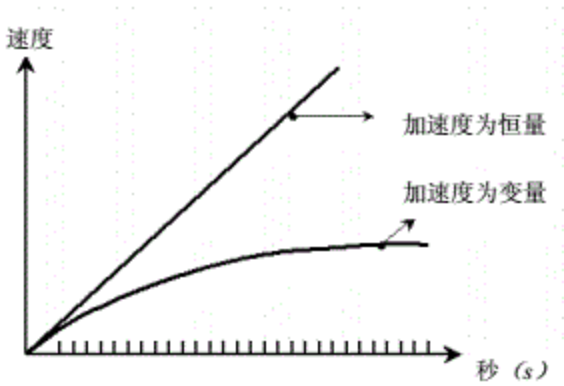


图 10.7　物体运动加速度与速度之间的变化

所以为求真实性，因此必须在物体的速度常量上加上一个加速度常量，如下所示：

```
a = a +k
x=x+ a
y=y+ a
a 为速度。
k为加速度常量。
```

如此一来，便可以很简单地看出速度 a 再加上加速度 k 之后，其速度 a 的值会越来越大，这不就符合前面所说的条件吗？物体在空间的移动量如图 10.8 所示。

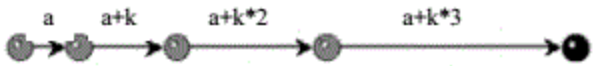


图 10.8　物体在空间中的移动量

如同上述方式，又可以将加速度应用在如同汽车在加速以后，再到慢慢停止的运动一样，如图 10.9 所示。

图 10.9　汽车加速与停止运动的轨迹

其实此方程序是很容易理解的。只要在物体的速度达到最高值或某一种条件成立之下，再将原本累加的加速度值慢慢递减回来，如下所示：

```
If go=True Then
 a = a +k
Else
 a = a -k
End If
x=x+ a
y=y+ a
```

10.2.3 动量

在计算机游戏中，很多物体可能会被设想成为汽车、飞弹、飞行器或其他现实生活中的物体。这些物体在现实生活中是利用材料所制造而成的，因此它们具有一定的质量。重量是对力的描述，但是质量并不能代表重量，而质量会对加速度发生作用。不管怎么样，在游戏中为了呈现物体运动的真实感，所以这些物体就应该具有某种程度的虚拟质量，换句话说，当这些物体在运动的时候，它们必须具有一定的动量。

一般而言，动量是物体的质量和速度的乘积，简单地说，它就是具有移动物体的特性，而这种特性会与物体的质量与速度有关。

```
动量＝质量×速度
```

当某个物体以一定速率运动时，如果想把它们停下来就必须花很大力气。例如要将一列以每小时 2 公里速度前进的火车停下来，这会比以每小时 1000 公里的速度前进的子弹停

下来还要困难许多，因为火车的质量会远远大于子弹的质量。如果一列火车以每小时 2 英里的速度撞击我们，那么我们可能会被火车给压得扁扁的。

在游戏中，我们可以随心所欲的制造任意质量给空间中的物体。不过如果希望可以建立一个真实度极高的游戏时，我们就应该在游戏中理性设置这些物体的质量，如此一来才能符合与现实生活相似的物理特性。

在这里所谈论关于动量的所有内容都是要引导游戏中物体与物体相互碰撞而产生的结果。在加入这项工作之前，我们必须先来了解一下“动量守恒定律”的原则。在物理学中的守恒指的是物体在一次运动后能量的保持，而这种能量是可以转换的，不过它还是会一直存在，因此守恒定律的基本原则就是“能量即不能产生，也不能被消灭”。 在一个物体碰撞另一个物体的情况下，由于动能无法被释放，所以它会一直被保持着。无论如何，如果有两个物体撞在一起的时候，动量守恒公式则为：

```
M1V1=M2V2
M1是第1个物体的质量。
M2是第2个物体的质量。
V1与V2是它们相对速度。
```

在了解到物理动量守恒的原则之后，也许我们不可能准确地遵循这种规律，不过却可以在游戏中尝试做出更真实的模拟碰撞了。

10.2.4 重力

在大自然中存在着一股很大的力量，这个力量可使得我们不会从地球上飘流到太空中，而且可以让我们稳稳当当地站在地表上，这种力量称为“重力”。

在游戏的虚拟空间里，为了做到现实的真实感，我们也把空间里的所有物体加上一个重力的单位。一般而言，重力是一个向下的力量，当物体要往上飘的时候，重力则会依据物体的运行方向再加上一个往下的力。例如将球由 A 地抛向 B 地的时候，因为球的运动方向与重力之间的关系，球的运动路线会形成一个抛物线，如图 10.10 所示。

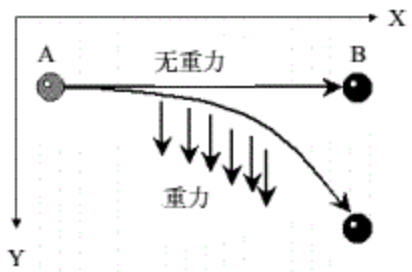


图 10.10　由于重力球在运动时会形成一个抛物线的轨迹

如果是在无重力的情况下，我们可以将球的运动方式写成如下形式：

```
x=x+α
y=y+0
α为速度常量。
```

为了游戏的真实感，在这里，我们再加入重力的表现，可以将公式写成：

```
x=x+α
y=y+M
α为速度常量。
M为重力常量。
```

如此一来，便可以在游戏中表现出与现实生活一样的物理现象了。在以上的范例中，这个物体是以一个速度恒量的值在做运动，虽然已经加上重力的常量，不过现实生活中，所求出的这种运动方式还是不及格的；因为还是少加了一个物理量，这个物理量就是“重力加速度”。

所谓重力加速度简单地说，它就是除了重力之外，物体在往下掉的过程中，它也会加入一个加速度的作用力，如图 10.11 所示。

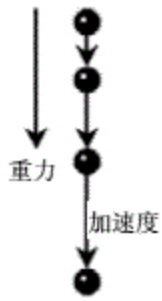


图 10.11　重力加速度示意图

这种重力加速度的力是因为重力对物体施力而产生。物体在高空中，因为重力的影响而会往下掉，这种力就如同前面所说的加速度原理一样。在物体还没有到达地面之前，它所掉下的速度会越来越快。由于这种会加速物体往下掉的力量是因为重力的关系，所以这种力量称为“重力加速度”。如同前面所说的一样，在物体上加入一个重力常量之后，我们必须还要再加上一个加速度常量，如下所示：

```
x=x+α
β=β+k
y=y+M+β
α为速度常量。
```

M为重力常量。
β为y值速度常量。
k为β的加速度常量。

这样才能够更加真实地表现物体由高空中往下掉的视觉效果了。

10.2.5 爆炸

爆炸的物理现象在游戏中经常被使用，不管是游戏中的魔法攻击、飞弹爆炸，或是飞机失事的效果显示，这些都必须要利用爆炸的画面来衬托出视觉的效果。其实爆炸的物理现象是很容易做到的，只要将前面所讲过的物理现象加以合并使用，就能够做出如同现实生活中的爆炸效果了。在现实生活中爆炸是属于一瞬间将某个物体冲破的现象，而被冲破的物体会变成许多小块状的物体散落四处。在游戏中，我们可以利用两种表示方法来描述爆炸的效果，一种是属于静态的表现，它是利用美工图素的变化来描述爆炸，也就是美工人员必须画出一张连续的爆炸图以供游戏中显示，如图 10.12 所示。

图 10.12　连续的爆炸图

另一种爆炸的效果是属于动态的，它是利用粒子的运动方式来描述爆炸的过程，而粒子的运动过程，必须利用物体移动的规律来描述。

如果在游戏中要做出极为真实的爆炸效果，那么势必要利用相当多的物理定理才行，接下来就为您介绍这些可以用来做出爆炸效果的物理原则。

刚才提到在爆炸效果的运动过程中，一开始它是将各种块状物体在一瞬间由一个圆心向外围圆周的任意方向拓展，形成一个爆炸最初期的现象，如图 10.13 所示。

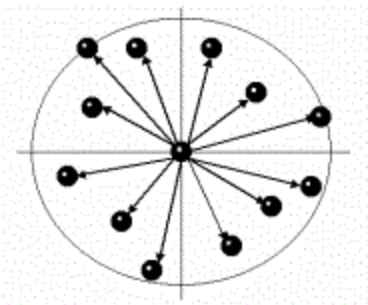


图 10.13　粒子会向着任意方向移动

要实现这种效果其实很简单，我们只要将每一个粒子向着圆的外围移动就可以了。在移动这些粒子之前，先来了解一个非常好用的圆周方程序。另外再来看一下 XY 坐标系统的定义，如图 10.14 所示。

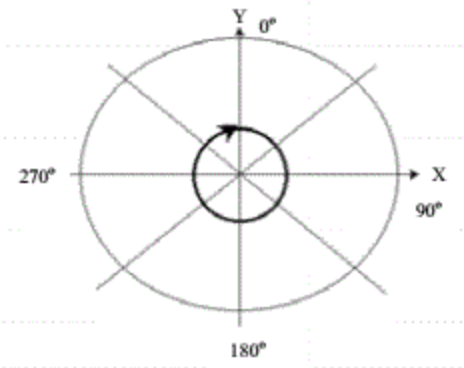


图 10.14　二维坐标系的角度

接下来，我们来看看数学上三角函数的变化，如表 10.1 所示。

表10.1　三角函数表

角度	0°	90°	180°	270°
Sin	0	1	0	-1
Cos	1	0	-1	0

由表 10.1 可以得知，空间中的 XY 坐标变化，如果将点坐标加上三角函数变化的话，就可以很轻易地画出一个圆的圆周，如下所示：

```
x=x+r*cos(θ)
y=y+r*sin(θ)
θ为角度的变化。
r为圆的半径。。
```

如果利用上面的公式将 θ 的角度由 0° 一直累加到 360° 的话，我们就可以画出一个圆了。程序代码如下：

```
r=5
For i=0 To i<=360
  x=x+r*sin(i)
  y=y+r*cos(i)
Next i
```

当我们要做出爆炸的最初期效果时，可以将每一个粒子的点坐标加上这种画圆周的公式。

在此，因为角度以随机数来显示，所以每一个粒子就会在 0°~360° 之间取得一个运行方向，然后再将半径由 0 慢慢累加到特定的值，使得空间中的每一个粒子会按照我们所设置的方程序沿着任何角度运动了，如图 10.15 所示。

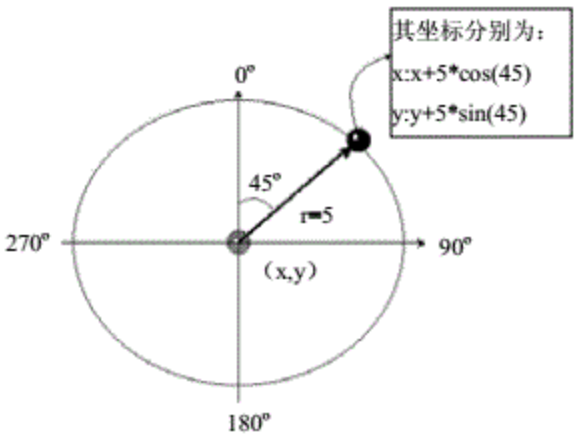


图 10.15　粒子运动方向

可以将运算粒子运动方向的函数写成：

```
Sub PosMove(x As Integer, y As Integer, r As Integer, arr As Integer)
    x=x+r*cos(arr)
y=y+r*sin(arr)
End Sub
```

我们已经可以做出控制粒子运动方向的方程序，接下来为了实现爆炸瞬间的效果，则在每一个粒子运动的过程中加上一个加速度的力，而这个加速度的力与前面所说的有点不同，前面所说的加速度都是越来越快，但是在这里，因为爆炸的瞬间力量是很大的，所以粒子的加速度会由快而变到慢。上面的方程序中，我们可以看出圆的半径是用来控制粒子运动的速度，如果将半径加上一个固定的常数，如“r=r+1”，这个粒子则会按照速度恒量的原则向着我们所设置的方向前进，不过为了让粒子的移动速度产生变化，所以必须要改变这个固定常数的量，又因为爆炸的瞬间是最快的，所以我们可以将方程序写成：

```
k=5       '爆炸的运动加速度
r=10      '爆炸的瞬间值
r=r+k     '爆炸的运动速度
```

```
k=k-1    '递减运动的加速度
```

其实现过程可用下列函数表示：

```
Sub AddSpeed(r As Integer, k As Integer)
    r=r+k
    k=k-1
End Sub
```

有了这个函数之后，就可以轻易地表现出爆炸的运动效果，如图 10.16 所示。

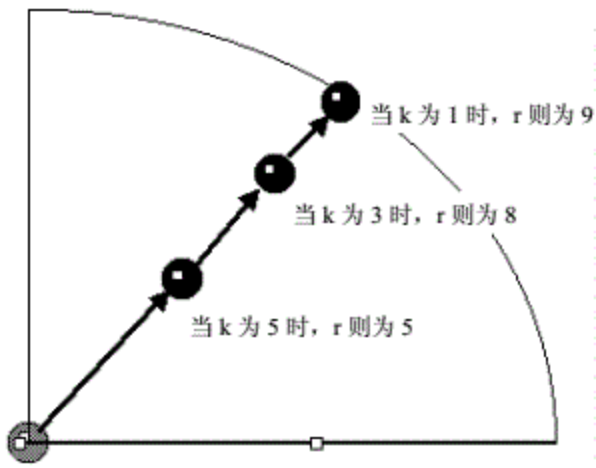


图 10.16　爆炸的运动效果

注意：如果要计算出更精确的运动加速度，我们可以将半径与加速度的常数设置成浮点数，以提高数值的精确度。

当可以呈现爆炸粒子最初期的运动方式之后，因为爆炸粒子也会受到重力影响，所以接下来还要将每一个粒子加上一个重力作用力。

如果没有将每一个粒子加上重力的话，那么粒子就要向着我们所设置的方向一直不停地前进，直到速度小于 0 为止。而粒子就会停留在空间中的某一处，这与现实是不相符的，所以您必须还要将重力的常量加在每一个粒子上。

如同前面所说的一样，重力在空间中是一个向下的作用力，所以当粒子在运动的同时，我们就必须再加上一个向下的力量，如图 10.17 所示。

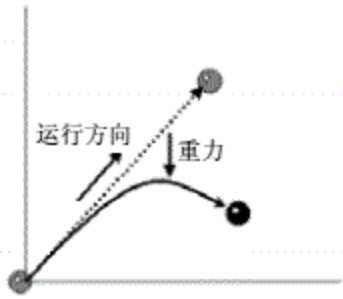


图 10.17　物体运动的方向受重力的影响

前面所讲的粒子移动程序代码如下所示来，然后在 Y 坐标的方程序中加入一个向下的重力 M：

```
Sub PosMove(x As Integer, y As Integer, r As Integer, arr As Integer,
M As Integer)
    x=x+r*cos(arr);
y=y+r*sin(arr);
y=y-M;
End Sub
```

这样一来，便可以做出与现实爆炸相类似的效果了，如图 10.18 所示。

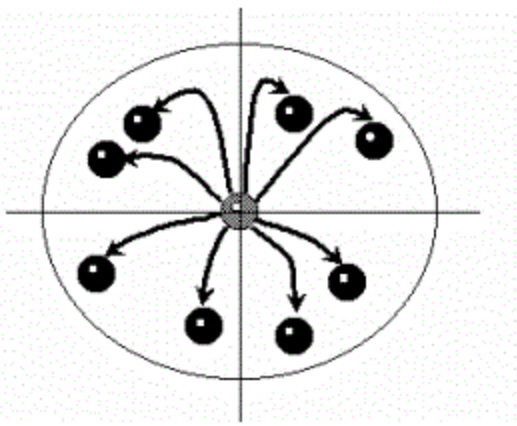


图 10.18　每个粒子都会受到重力的影响

爆炸的效果是不是很容易做出来呢！接下来，我们再来看看在现实生活中较为常见的折射运动的运算。

10.2.6 折射

一般而言，折射是一个光学反应的术语，通常它是用来描述光线通过某种介质时的反应，如图 10.19 所示。

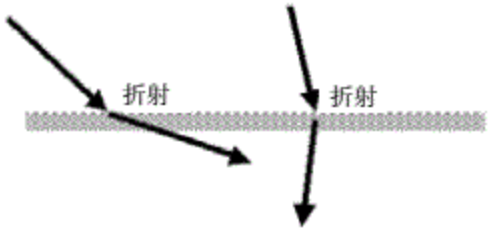


图 10.19　光线通过某种介质时的折射反应

不过，在这里我们要讨论的不是这种光学的反应，而是物体在做规律运动的时候，当它碰撞到其他的物体之后会做出一种反射的动作。在现实生活中，我们都知道物体碰撞到另外一个物体时，它们都会做出适当的反射动作。例如，球碰撞到墙壁的时候，球的运作方向则会因为墙的平面而有所改变，如图 10.20 所示。

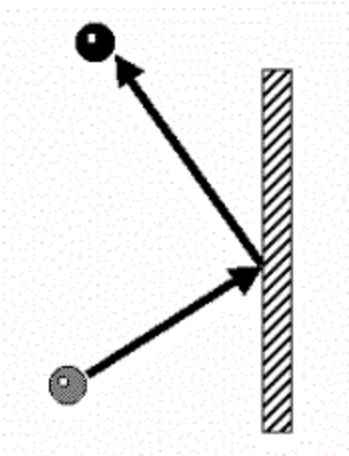


图 10.20　球碰撞到墙壁时，球的运作方向

这种反射动作在大自然中有一定的规则变化，而我们就是要在这里深入讨论这一种物理反应。 一般而言，在游戏中会经常看到如上面所说的物理反射动作，例如撞球游戏，当母球被推向桌边的时候，母球在碰撞到桌边后，便会做出反弹动作，而这种反弹动作，就称为“折射”。下面接着介绍折射的运算方式。

如同上面所讲的一样，当物体在碰撞到墙壁的时候，它会做出一个特定的反射。或许您会问道：“那物体在碰撞到墙壁之后，它到底会被弹到哪里呢？”

其实道理很简单，我们只要在这种反射中，求出它的反射角，就可以知道物体会被弹到何处。当物体碰撞到墙壁之后，物体的运动方向与墙壁的交点会形成一条垂直于墙壁的法线，如图 10.21 所示。

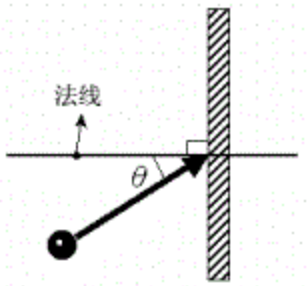


图 10.21　物体运动方向与墙壁的交点形成一条垂直于墙壁的法线

接下来，再将物体运行的方向映射到这条法线的另一端，如图 10.22 所示。

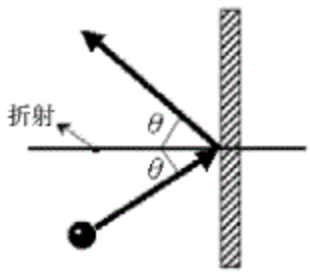


图 10.22　将物体运行的方向映射到法线的另一端

如此一来，便可以得到两个一模一样的角度，而物体反射运动的折射便是利用这个角度来换算的。在上面的陈述中，相信您不难发现，如果要计算物体碰撞到墙壁后的运动方向，我们只要得到这条法线和法线与物体运动方向的角度，便可以求出另一个反射的运动方向了。

首先考虑 θ 的角度应该如何计算。先将上述的图修改成如图 10.23 所示。

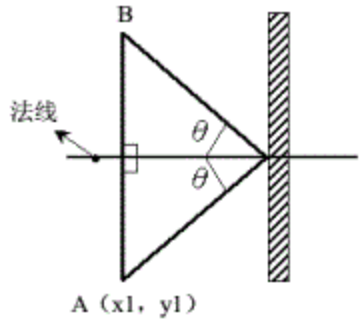


图 10.23　计算 θ 角

从图 10.23 中，可以轻易地看出 B 点坐标值（x1,-y1），不过这个坐标值似乎对物体的运动没有用，而我们所要求的是 θ 的角度，它才是对物体运动过程中有用的数值。在图 10.23 中，下半部的三角形可以将它看成如图 10.24 所示。

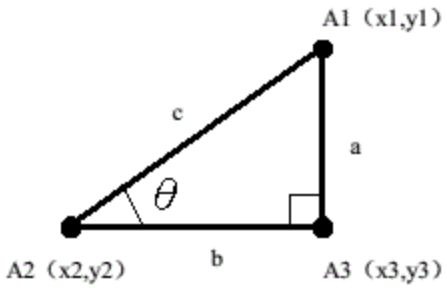


图 10.24　三角形的坐标值

以这个三角形来说，我们可以得到如下所示的三角函数方程序：

```
tanθ=a/b=(y3-y1)/(x3-x1)
```

如此一来，便可以轻易得到“tan θ ”的值了。不过在这里，还是没有得到 θ 的角度值，所以还要利用反三角函数来求得 θ 角度的值，可以利用程序开发工具所提供的“asin”函数来求得 θ 角度，如下所示：

```
tanθ=a/b→θ=asin(a/b)=asin(((y3-y1)/(x3-x1)))
```

其中，asin 为 Visual Basic 的反三角函数。

现在便可以取得角度 θ 的值了。其实我们要求角度 θ 的值，还有一种比较容易计算的方法，如图 10.25 所示。

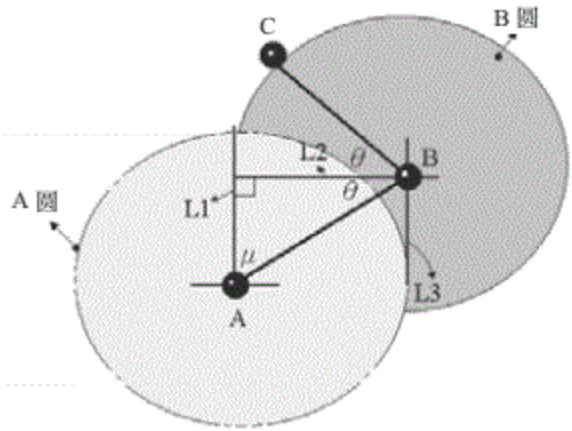


图 10.25　计算角度 θ 的值

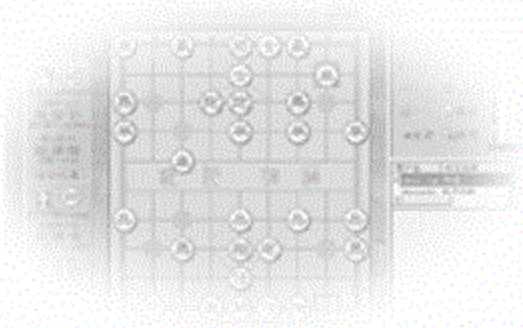
如同前面所说的一样，当物体要从 A 点移动到 B 点时，可以利用圆周角度与半径的关系来实现物体移动的行为。在图 10.25 中，已知的角度为“μ”，而 L1 又与 L2 互相垂直，所以“θ”就会为“90 μ”，因此便可以得到法线另一端的映射角了。当物体碰撞到 L3 的时候，则将圆心移到 B 点，如此一来便形成另外一个 B 圆。在 B 圆里，物体的移动角度则成为另一边的映射角度“θ”，所以在这里便可以再运行物体反射后的运动。

在反射运动的过程中，为了显示出物体运动时真实感，还可以加入加速度的常量与摩擦力（与施力方向相反的力）的反应。因为前面已经讲述过了，所以在这里我们就不再加以讨论了，请您自行加入这些物理常量。

现在已经介绍完游戏中常用的物理规则了，相信您对物理的公式有了更加深刻的印象。接着在下一章我们再利用前文所提及的一些常用算法来制作一套小游戏范例，以及介绍一些常见的小游戏之制作过程。

第 11 章

游戏绘图实例



在前两章中，所讨论的是一些 2D、3D 的贴图原理与方法，然而要将这些原理与方法实际应用制作出可执行的算法，则还有一些必须注意的细节。因此在本章中，我们使用 Visual Basic 来通过编程实现前两章所讨论过的一些贴图方式，您也将了解到实际制作与理论之间的一些差距在哪里。

11.1 基本动画与贴图

在第 8 章中，我们曾经讨论过一连串的 2D 绘图算法，这些算法只是各种动画的基本运作原理，对于一个专业的程序设计人员来说，只要给他算法，无论使用何种程序语言，他都可以进行无穷的运用，甚至创建出新的算法，在程序设计领域中，后期对于算法的了解将会比对程序语言本身的了解重要。

本章将通过第 8 章的一些算法进行程序设计，虽然算法的设计并不局限于使用何种程序语言，然而基于初学者的角度考虑，我们选择 Visual Basic 来作为设计程序所使用的工具，因为它对于初学者来说最方便而且容易上手，如果您熟知其他程序语言，可以自行将本章的内容用您所熟知的程序语言编写程序。

11.1.1 星球运行

前面介绍的动画制作最基本原理就是连续移动的画面，凭借绘图对象在绘图画布上（例如窗口上）连续不断的移动，就可以实现动画的效果。单纯的直线移动较容易制作，然而若要模拟真实世界中某些事物的移动，就得套用一些相关的数学或物理公式。这也衍生出另一个观念：程序语言、算法、数学等相关学科的相互结合，才有可能设计出最高效的程序，而是着重在程序语言功能上的介绍。

下面第 1 个实际制作的例子是星球运动，要真正模拟星球运动并不容易，必须考虑到引力、加速度等问题，然而我们可以适当的简化，使用圆周运动来加以模拟，圆周运动基本上可以由半径、角度与转动速度 3 个要素来决定，下面直接使用图 11.1 来进行说明。

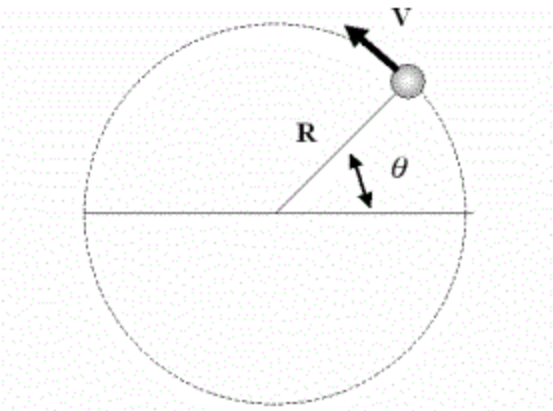


图 11.1　圆周的半径、角度与转动速度

在程序设计中，通常以逆时针作为旋转角度的正方向；然而角度、半径与速度如何反映在绘图坐标上，我们必须凭借旋转的圆心坐标加上球的投影横坐标与纵坐标，以求出球目前的 XY 坐标，而旋转的切线速度则以每次旋转的角度代替，所以我们可以使用图 11.2 中的坐标来表示。

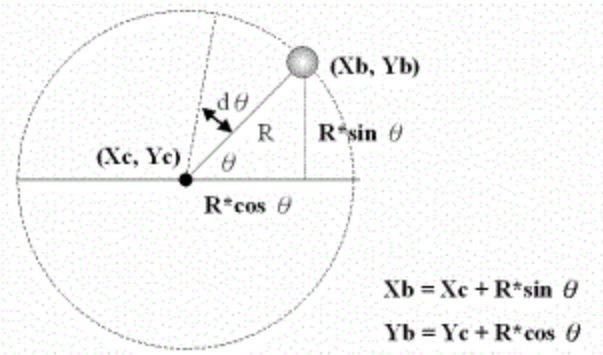


图 11.2　旋转角度、半径与速度的坐标

我们将基于以上的公式来仿真星球的运行，程序中将会有一颗恒星、两颗行星与一颗卫星，您可以打开范例文件 BasicAni.vbp 执行，查看执行结果，如图 11.3 所示。

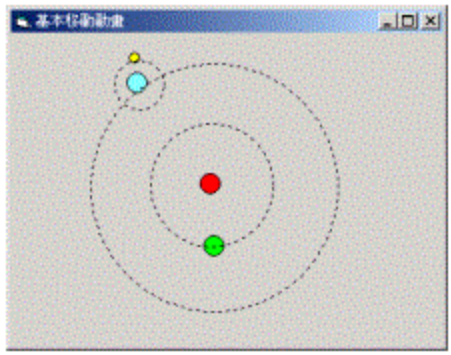


图 11.3　BasicAni.vbp 的运行结果

以下是我们的范例程序代码：

```
01 ' 程序：基本移动动画
02 ' 范例文件：BasicAni.vbp
03
04 Dim ang1, ang2, ang3, r1, r2, r3
05 Dim m_ang1, m_ang2, m_ang3
06 Dim xcenter, ycenter
07
08 Private Sub Form_Activate()
09     Randomize
10     '球的起始角度
11     ang1 = Int(Rnd() * 360)
12     ang2 = Int(Rnd() * 360)
13     ang3 = Int(Rnd() * 360)
14     '球的移动角度
15     m_ang1 = Int(Rnd() * 10) + 5
16     m_ang2 = Int(Rnd() * 15) + 10
17     m_ang3 = Int(Rnd() * 20) + 10
18     '球的运行中心
19     xcenter = Shape4.Left
20     ycenter = Shape4.Top
21     '球的运行半径
22     r1 = 50
23     r2 = 100
24     r3 = 20
25 End Sub
```

```
26
27 Private Sub Timer1_Timer()
28     Dim x, y As Double '计算球坐标
29     Const PI As Double = 3.14159  ' 圆周率
30     x = r1 * Sin(ang1 * PI / 180)
31     y = r1 * Cos(ang1 * PI / 180)
32     Shape1.Left = Shape4.Left + x
33     Shape1.Top = Shape4.Top + y
34     ang1 = ang1 + m_ang1
35
36     x = r2 * Sin(ang2 * PI / 180)
37     y = r2 * Cos(ang2 * PI / 180)
38     Shape2.Left = Shape4.Left + x
39     Shape2.Top = Shape4.Top + y
40     Shape7.Left = Shape2.Left -10
41     Shape7.Top = Shape2.Top -10
42
43     ang2 = ang2 + m_ang2
44
45     x = r3 * Sin(ang3 * PI / 180)
46     y = r3 * Cos(ang3 * PI / 180)
47     Shape3.Left = Shape2.Left + x
48     Shape3.Top = Shape2.Top + y
49     ang3 = ang3 + m_ang3
50 End Sub
```

基本上，这就是第 8 章中所提及利用对象的坐标改变来制作动画的方法，对于一些简单的动画都可以使用这种方式，动画播放的速度可以使用 Timer 组件来控制，它可以在指定的时间间隔内执行所编写的程序代码指令。

11.1.2 贴图动画

利用图形对象的坐标移动只能制作简单的动画，如果要制作较精细的动画，必须利用到贴图的技巧，在第 8 章介绍过基本的动画贴图技巧，包括一维图像单元格的贴图与二维图像单元格的贴图，在此将二维影像的贴图作为算法设计的参考，所使用的图片如图 11.4 所示。

图 11.4　二维影像的贴图

我们播放的顺序将由左向右、自上向下，为了能够指定播放的图像单元格为图片中的哪一块区域，必须使用绘图函数辅助计算图像单元格播放的位置，所使用的是 Visual Basic 中所提供的 PaintPicture 函数，其参数的指定说明如下所示：

```
PaintPicture(source, dx, dy, dwidth, dheight, sx, sy, swidth, sheight,
opcode)
```

- soucre：绘图来源对象。
- (dx, dy)：目标区坐标。
- (dwidth, dheight)：目标区绘图区域大小。
- (sx, sy)：来源区坐标。
- (swidth, sheight)：来源区图形区域大小。
- opcode：vb 句柄。

其中 opcode 先使用 vbSrcCopy，表示进行来源图片的复制，在实际制作时会先将图片加载 Visual Basic 的 PictureBox 组件，并预先将之设置为“不可视”，然后再使用 PaintPicture 函数进行绘图区域的计算，并将图形绘制到窗体上，同样地可以使用 Timer 组件来控制动画播放的速度，如图 11.5 所示是在窗体上的组件配置。

图 11.5　使用 Timer 组件控制动画播放速度

每一个图像单元格的大小为 128×96，为了配合图片的背景，我们也将窗体的背景设置为黑色，此程序的范例文件为 PictureAni.vbp，程序代码如下：

```
01 ' 程序：贴图动画
02 ' 范例文件：PictureAni.vbp
03
04 Dim x As Integer, y As Integer ' 贴图来源坐标
05
06 Private Sub Form_Activate()
07     x = 0
08     y = 0
09     Picture1.Visible = False ' 设置PictureBox为不可视
10 End Sub
11
12 Private Sub Timer1_Timer()
13     ' 将图片绘制到窗体上
14     Form1.PaintPicture Picture1.Picture, 96, 48, 128, 96, x, y, 128,
96, vbSrcCopy
15
16     ' 计算绘图来源X坐标
17     x = x + 128
18     If x = 640 Then
19         ' 计算绘图来源Y坐标
20         y = y + 96
21         x = 0
22         If y = 480 Then
```

```
23            y = 0
24         End If
25     End If
26 End Sub
```

其实这种方式也有其必须负担的成本，因为我们必须额外花费时间计算绘图来源区域，然而其好处是可以直接加载整张图片。您也可以将前面的图片切割为数个图片文件，在播放时再依序加载，如此可以省去计算绘图来源区域的时间，然而却必须花费时间加载图片，两者各有其优缺点，真正在进行游戏制作时必须与其他程序设计人员商量用统一的动画播放方式。

11.1.3 横向滚动条贴图

在第 8 章我们曾经介绍过横向滚动条贴图，对于一个有边界的横向地图而言，只要使用一个贴图方块，并判断方块是否在地图边界之内；而对于一个无边界循环的单向滚动条播放动画而言，必须有两个贴图方块，这两个贴图方块的作用我们用图 11.6 作说明。

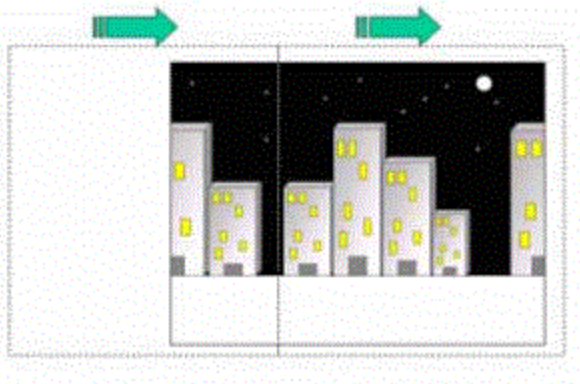


图 11.6　两个贴图方块

我们假设地图会不断地转动，则贴图时右边的显像方块所指定的图片来源区域会逐渐变窄，消失的部分则在左边的方块再次贴出，其道理就如同播放幻灯片，将图片的尾端与前端连接起来，再不断地循环播放，如图 11.7 所示。

对于这样的滚动条动画，我们只要两个贴图指令并配合固定时间播放就可以实现。需要注意的就是图片的接合问题，为了突显动画的效果，可以在循环中加上一个人物作为位移的对比，如图 11.8 所示为我们制作的程序效果预览图。

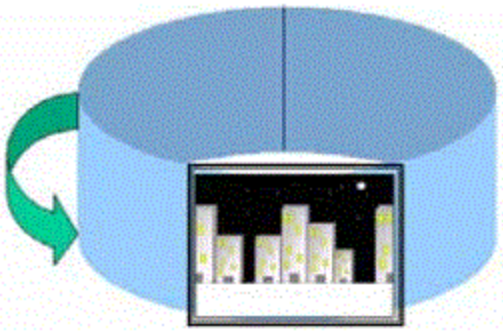


图 11.7　将图片的尾端与前端连接起来

图 11.8　程序效果预览图

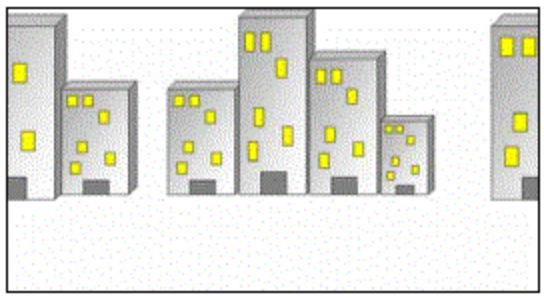
在图中的人物事实上是静止不动的，由于背景循环的关系，使得人物像是在进行走动，利用背景与前景的位移关系制作出动态效果，程序代码如下：

```
01 ' 程序：横向背景循环转动
02 ' 范例文件：HorScroll. vbp
03
04 Dim x As Integer
05 Dim w As Integer, h As Integer
06
07 Private Sub Form_Activate()
08     w = Picture2.Width
09     h = Picture2.Height
10     x = 10
11     Image1.Visible = False
12     Picture2.Visible = False
13 End Sub
14
15 Private Sub Timer1_Timer()
16     ' 先贴左边方块的图
```

```
    Form1.PaintPicture Picture2, 0, 0, x, Form1.Height, _
       533 -x, 0, x, Form1.Height, vbSrcCopy
    ' 再贴右边方块的图
    Form1.PaintPicture Picture2, x, 0, w, h, _
       0, 0, w, h, vbSrcCopy
    ' 贴上人物以突显动画效果
    Form1.PaintPicture Image1, Image1.Left, Image1.Top, 115, 86, _
       0, 0, 120, 85, vbSrcCopy
    ' 循环贴图
    x = x + 10
    If x > 533 Then
       x = 10
    End If
End Sub
```

横向滚动条动画的制作还可以作进阶的变化，以这个例子为例，我们可以利用数个背景播放的时间差，制作出具有远近感的背景转动效果。例如天空云彩的转动应该较慢，而不是随着前景一同移动，此时必须使用两张背景贴图和 4 个贴图区，背景的选用如图 11.9 所示。

远景背景图



近景背景图

图 11.9　背景图

转动速度的决定可以使用两个 Timer 组件，也可以使用变量递增值的不同来实现，下面使用后者来设计程序，以下为程序代码：

```
' 程序：具有远近感的横向背景转动
' 范例文件：HorScroll2.vbp

Dim x1 As Integer, x2 As Integer
Dim w1 As Integer, h1 As Integer
Dim w2 As Integer, h2 As Integer

Private Sub Form_Activate()
    w1 = Picture1.Width
    h1 = Picture1.Height
    w2 = Picture2.Width
    h2 = Picture2.Height
    x1 = 10
    x2 = 10
    Image1.Visible = False
    Picture1.Visible = False
    Picture2.Visible = False
End Sub

Private Sub Timer1_Timer()
    ' 远景图，先贴左边方块的图
    Form1.PaintPicture Picture1, 0, 0, x1, h1, _
       533-x1, 0, x1, h1, vbSrcCopy
    ' 远景图，再贴右边方块的图
    Form1.PaintPicture Picture1, x1, 0, w1, h1, _
       0, 0, w1, h1, vbSrcCopy
    ' 近景图，先贴左边方块的图
    Form1.PaintPicture Picture2, 0, h1, x2, h2, _
       533-x2, 0, x2, h2, vbSrcCopy
    ' 近景图，再贴右边方块的图
    Form1.PaintPicture Picture2, x2, h1, w2, h2, _
       0, 0, w2, h2, vbSrcCopy
    ' 贴上人物以突显动画效果
    Form1.PaintPicture Image1, Image1.Left, Image1.Top, 115, 86, _
       0, 0, 120, 85, vbSrcCopy
    ' 循环贴图
    x1 = x1 + 1
    If x1 > 533 Then
       x1 = 10
```

```
40      End If
41          x2 = x2 + 10
42      If x2 > 533 Then
43          x2 = 10
44      End If
45 End Sub
```

您可以执行范例文件来浏览这个程序执行后的效果，基本上纵向滚动条的制作方式也是如此，只不过转动的方向由水平变为垂直方向，您可以自己尝试制作纵向滚动条的效果。

11.1.4 互动地图转动

地图转动其实是比连续的背景图转动更容易制作，我们先从基本的横向地图转动开始设计，背景图与显像窗格如图 11.10 所示。

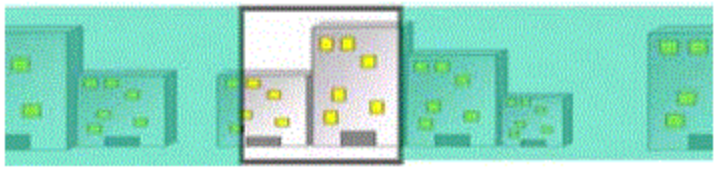


图 11.10　背景图与显像窗格

基本上只要判断图片的哪些区域需贴到显像窗格之中就可以了，而必须注意的是这个地图是有边界的，而不是像前面的背景图循环贴图，所以必须判断窗格是否已达左右边界，我们利用窗格的中心与边界的距离来判断，只要使用一个变量就可以了，您可以先执行范例文件 GraphScroll.vbp 来浏览横向地图的转动效果，如图 11.11 所示为程序执行结果的画面之一。

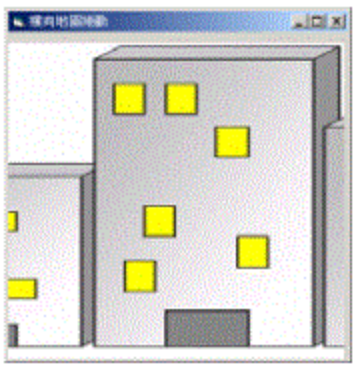


图 11.11　请使用左右方向键来操作地图转动

横向地图转动的实际制作相当简单，下面是程序的源代码：

```
' 程序：横向地图转动
' 范例文件：GraphScroll.vbp

Dim Xc As Integer
Dim w As Integer, h As Integer

Private Sub Form_Activate()
    Xc = 800
    w = Picture1.Width
    h = Picture1.Height
    Picture1.Visible = False
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    ' 指定横向地图的区域进行贴图
    Form1.PaintPicture Picture1, 0, 10, w, h, _
       Xc-w / 2, 0, w, h, vbSrcCopy

    If KeyCode = 39 Then ' 如果按下右方向键
       Xc = Xc + 10
    ElseIf KeyCode = 37 Then ' 如果按下左方向键
       Xc = Xc-10
    End If

    ' 判断是否遇到地图的左右边界
    If Xc < w / 2 Then
       Xc = w / 2
    ElseIf Xc > 1600 -w / 2 Then
       Xc = 1600-w / 2
    End If
End Sub
```

介绍完横向地图转动的方式，接下来制作二维地图的转动就相当容易了，我们首先必须准备一张大地图，而在贴图时每次只显示其中一个小区域，这是二维地图的基本转动原理，所准备的大地图如图 11.12 所示。

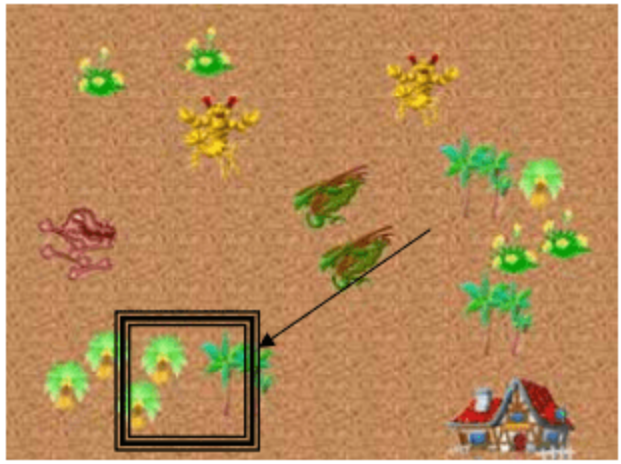


图 11.12　每次只显示地图的一小块区域

同样使用键盘来进行地图转动的操作，而我们也必须判断显像方块是否已抵达地图的上下左右任一边界，如果遇到边界就不该再继续转动。判断的方式同样使用显像方块的中心坐标，如此我们只要使用两个变量，图 11.13 是地图转动程序的执行结果，其中加入了一个小人物作为操作中心点。

图 11.13　地图转动程序执行结果

实际上人物是静止的，在地图转动时由于背景的移动，使得结果看起来像是人物在移动；请注意这个程序还没有加入障碍物的判断，它只用来演示地图转动的效果，稍候将介绍如何判断障碍物以决定是否能前进，这个程序的实现代码如下：

```
01 ' 程序：二维地图转动
02 ' 范例文件：MapScroll.vbp
```

```
03
04 Dim Xc As Integer, Yc As Integer
05 Dim w As Integer, h As Integer
06 Private Sub Form_Activate()
07     Xc = 400
08     Yc = 300
09     w = Picture1.Width
10     h = Picture1.Height
11     Picture1.Visible = False
12 End Sub
13
14 Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
15     ' 贴上地图部分区域
16     Form1.PaintPicture Picture1, 0, 0, w, h, _
17         Xc-w / 2, Yc-h / 2, w, h, vbSrcCopy
18     ' 贴上人物
19     Form1.PaintPicture Picture2, 50, 50, 73, 97, _
20         0, 0, 73, 97, vbSrcCopy
21
22     ' 判断按下的按键
23     Select Case KeyCode
24         Case 38 ' 按上
25             Yc = Yc-10
26         Case 40 ' 按下
27             Yc = Yc + 10
28         Case 37 ' 按左
29             Xc = Xc-10
30         Case 39 ' 按右
31             Xc = Xc + 10
32     End Select
33
34     ' 判断是否遇到左右边界
35     If Xc < w / 2 Then
36         Xc = w / 2
37     ElseIf Xc > 800-w / 2 Then
38         Xc = 800-w / 2
39     End If
40
41     ' 判断是否遇到上下边界
```

```
42     If Yc < h / 2 Then
43         Yc = h / 2
44     ElseIf Yc > 600-h / 2 Then
45         Yc = 600-h / 2
46     End If
47 End Sub
```

在进行二维地图转动时，请注意在窗口程序中 Y 轴坐标的设置是以下方为正方向，所以当按向下箭头键时，应该增加显像方块的 Y 坐标值，而不是减少。

11.1.5 通过障碍物

前一个范例在示范地图转动时，并没有加入判断是否碰到障碍物的功能，障碍物的判断可以先用数组来设置障碍物的位置。每次在移动人物之前就先比较数组中的元素值，看看下一个移动的位置是否有障碍物存在，我们先以一个最简单的障碍物判断范例来说明如何用程序实现，所选定的地图如图 11.14 所示。

图 11.14　钢筋为障碍物，人物遇到障碍物会无法通过

根据图 11.14，可以设置一个二维数组来记录障碍物的位置，数组设置如下所示，其中标示为 1 表示该处存在障碍物：

```
1, 1, 1, 1, 1
0, 0, 0, 0, 1
0, 0, 1, 0, 0
0, 1, 1, 0, 0
```

```
1, 1, 1, 1, 0
```

所有制作的程序将使用键盘进行操作，程序在每一次按下按键时，就必须进行一次数组元素检查，看看下一个位置元素值是否被标示为 1，所以必须用两个变量来记录人物当前的位置，这个简单的障碍物判断程序代码如下：

```
' 程序：通过障碍物
' 范例文件：BlockMove.vbp

Dim block(4, 4) As Integer
Dim px As Integer, py As Integer

Private Sub Form_Activate()
    Dim t0, t1, t2, t3, t4, i
    ' 设置数组记录障碍物的位置
    t0 = Array(1, 1, 1, 1, 1)
    t1 = Array(0, 0, 0, 0, 1)
    t2 = Array(0, 0, 1, 0, 0)
    t3 = Array(0, 1, 1, 0, 0)
    t4 = Array(1, 1, 1, 1, 0)
    For i = 0 To 4
        block(i, 0) = t0(i)
        block(i, 1) = t1(i)
        block(i, 2) = t2(i)
        block(i, 3) = t3(i)
        block(i, 4) = t4(i)
    Next i

    ' 人物起始位置
    px = 4
    py = 4
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    ' 判断按下何键，并判断下一步是否有障碍物存在
    ' 如果不存在障碍物，则往下移动一格
    Select Case KeyCode
        Case 38 ' 按上箭头键
            If py-1 > -1 Then
                If block(px, py-1) = 0 Then
```

```
35                  py = py-1
36                  Image1.Top = Image1.Top-80
37              End If
38          End If
39      Case 40 ' 按下箭头箭
40          If py + 1 < 5 Then
41              If block(px, py + 1) <> 1 Then
42                  py = py + 1
43                  Image1.Top = Image1.Top + 80
44              End If
45          End If
46      Case 37 ' 按左箭头键
47          If px-1 >-1 Then
48              If block(px-1, py) <> 1 Then
49                  px = px-1
50                  Image1.Left = Image1.Left-80
51              End If
52          End If
53      Case 39 ' 按右箭头键
54          If px + 1 < 5 Then
55              If block(px + 1, py) <> 1 Then
56                  px = px + 1
57                  Image1.Left = Image1.Left + 80
58              End If
59          End If
60    End Select
61 End Su
```

这个程序还没有使用背景转动与贴图，如果要加上背景转动，可将前面的背景转动范例再结合数组值来进行判断，我们所使用的背景图如图 11.15 所示。

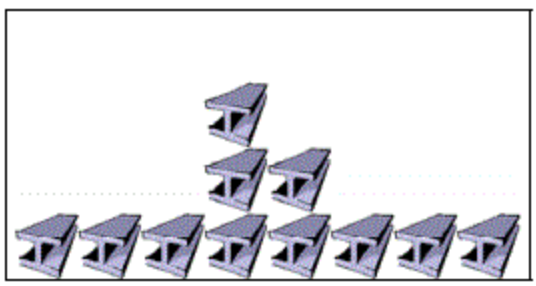


图 11.15　背景图

根据这个背景图，可以定义出一个数组来记录每一个障碍物的位置，数组定义如下所示：

```
0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 1, 0, 0, 0, 0
0, 0, 0, 1, 1, 0, 0, 0
1, 1, 1, 1, 1, 1, 1, 1
```

不过一旦结合背景转动功能将会多出一项考虑，就是我们在按下按键进行操作时，究竟是该移动人物还是转动背景图？如果处理不好的话，会发生贴图的残像问题。在这个程序中的实现方法是，如果人物在背景图的后半区域活动时，就转动背景图，如果人物在背景图的后半区域活动时，就移动人物，如此就不会在贴图时出现残像的问题，判断的依据是人物在数组中的索引位置，程序执行结果如图 11.16 所示。

图 11.16　可跳过背景障碍物的程序执行效果

以下是这个程序的代码：

```
01 ' 程序：横向滚动条结合障碍超越
02 ' 范例文件：BlockScroll.vbp
03
04 Dim Xc As Integer, Px As Integer, Py As Integer
05 Dim Xm As Integer, Ym As Integer
06 Dim w As Integer, h As Integer
07 Dim block(7, 3) As Integer
08
09 Private Sub Form_Activate()
```

```
    Dim t0, t1, t2, t3, t4, i
    ' 设置数组记录障碍物的位置
    t0 = Array(0, 0, 0, 0, 0, 0, 0, 0)
    t1 = Array(0, 0, 0, 1, 0, 0, 0, 0)
    t2 = Array(0, 0, 0, 1, 1, 0, 0, 0)
    t3 = Array(1, 1, 1, 1, 1, 1, 1, 1)
    For i = 0 To 7
        block(i, 0) = t0(i)
        block(i, 1) = t1(i)
        block(i, 2) = t2(i)
        block(i, 3) = t3(i)
    Next i

    Xc = 450  ' 显像区域中心
    w = Picture1.Width
    Xm = 225  ' 人物的位置
    Ym = 150

    Px = 7  ' 人物在数组中的位置
    Py = 2
    h = Picture1.Height
    Picture1.Visible = False
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
        Case 39 ' 如果按下向右键
            If Px + 1 < 8 Then
                If block(Px + 1, Py) <> 1 Then
                    Px = Px + 1
                    ' 判断该移动人物或背景
                    If Px > 3 Then
                        Xc = Xc + 75
                    Else
                        Xm = Xm + 75
                    End If
                    MoveChar
                End If
            End If
```

```
49        Case 37 ' 如果按下向左键
50            If Px-1 >-1 Then
51                If block(Px-1, Py) <> 1 Then
52                    Px = Px-1
53                    ' 判断该移动人物或背景
54                    If Px > 2 Then
55                        Xc = Xc -75
56                    Else
57                        Xm = Xm-75
58                    End If
59                    MoveChar
60                End If
61            End If
62        Case 38 ' 按向上键
63            If Py-1 >-1 Then
64                If block(Px, Py-1) = 0 Then
65                    Py = Py-1
66                    Ym = Ym-75
67                    MoveChar
68                End If
69            End If
70        Case 40 ' 按向下键
71            If Py + 1 < 4 Then
72                If block(Px, Py + 1) = 0 Then
73                    Py = Py + 1
74                    Ym = Ym + 75
75                    MoveChar
76                End If
77            End If
78    End Select
79 End Sub
80
81 ' 贴图函数
82 Private Sub MoveChar()
83    Form1.PaintPicture Picture1, 0, 0, w, h, _
84        Xc-w / 2, 0, w, h, vbSrcCopy
85    Form1.PaintPicture Image1, Xm, Ym, 75, 75, _
86        0, 0, 153, 147, vbSrcCopy
87 End Sub
```

如果不想花费精力去判断人物或背景的移动时机，则方法是将人物固定在显像方块的中心，而每次只移动背景，而为了遇到边界时不致于出现移动人物的麻烦，您必须在地图的四周加上边界，此时在地图与人物的关系如图 11.17 所示。

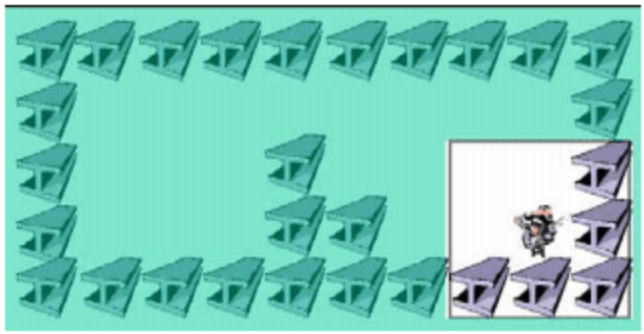


图 11.17　地图与人物的关系图

当然此时的地图与显像方块之间的关系已经不是单纯的一维地图转动，而是二维地图了，不过我们用来记录障碍物的数组可以不用改变，判断方式也大致相同，其程序的执行结果如图 11.18 所示。

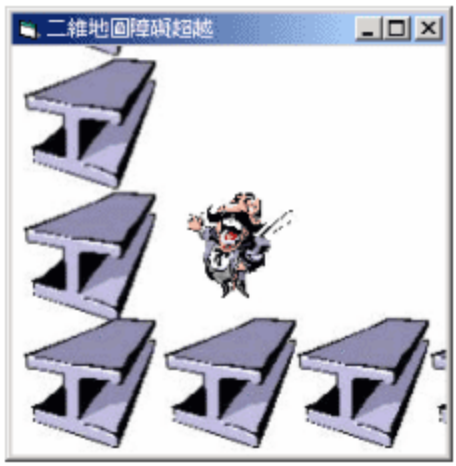


图 11.18　设置了边界的地图，此时已经是二维地图

程序代码并没作多大改变，主要是加入了上下地图转动与改变了一些贴图的位置，程序代码如下：

```
01 ' 程序：二维地图与障碍超越
02 ' 范例文件：BlockScroll2.vbp
03
```

```
Dim Xc As Integer, Yc As Integer, Px As Integer, Py As Integer
Dim Xm As Integer, Ym As Integer
Dim w As Integer, h As Integer
Dim block(7, 3) As Integer

Private Sub Form_Activate()
       Dim t0, t1, t2, t3, t4, i
    ' 设置数组记录障碍物的位置
    t0 = Array(0, 0, 0, 0, 0, 0, 0, 0)
    t1 = Array(0, 0, 0, 1, 0, 0, 0, 0)
    t2 = Array(0, 0, 0, 1, 1, 0, 0, 0)
    t3 = Array(1, 1, 1, 1, 1, 1, 1, 1)
    For i = 0 To 7
       block(i, 0) = t0(i)
       block(i, 1) = t1(i)
       block(i, 2) = t2(i)
       block(i, 3) = t3(i)
    Next i

    Xc = 660  ' 显像区域中心
    Yc = 160
    w = Picture1.Width

    Px = 7  ' 人物在数组中的位置
    Py = 2
    h = Picture1.Height
    Picture1.Visible = False
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
       Case 39 ' 如果按下向右键
          If Px + 1 < 8 Then
             If block(Px + 1, Py) <> 1 Then
                Px = Px + 1
                ' 移动背景
                Xc = Xc + 75
                MoveChar
             End If
```

```
            End If
        Case 37 ' 如果按下向左键
            If Px-1 >-1 Then
                If block(Px-1, Py) < > 1 Then
                    Px = Px-1
                    ' 移动背景
                    Xc = Xc-75
                    MoveChar
                End If
            End If
        Case 38 ' 按向上键
            If Py-1 >-1 Then
                If block(Px, Py-1) = 0 Then
                    Py = Py-1
                    ' 移动背景
                    Yc = Yc-80
                    MoveChar
                End If
            End If
        Case 40 ' 按向下键
            If Py + 1 < 4 Then
                If block(Px, Py + 1) = 0 Then
                    Py = Py + 1
                    ' 移动背景
                    Yc = Yc + 80
                    MoveChar
                End If
            End If
    End Select
End Sub

' 贴图函数
Private Sub MoveChar()
    Form1.PaintPicture Picture1, 0, 0, w, h, _
        Xc-w / 2, Yc, w, h, vbSrcCopy
    Form1.PaintPicture Image1, 100, 80, 65, 65, _
        0, 0, 153, 147, vbSrcCopy
End Sub
```

在这一节中我们介绍了基本的动画原理与各种贴图技巧，然而这些技巧只能用于一些

小游戏的制作。例如在场景或背景地图规模不大的时候，因为我们都是使用一整张地图来作为背景转动。然而这种方式不适用于场景变化多的游戏中，因为若采用此方法势必要用大量的图片文档。

就拿我们所制作的最后一个例子来说，背景几乎全为白色，而障碍物的样式也只有一种，此时我们可以采用重复贴图的方式，将障碍物贴在指定的位置即可。这样做有个好处，就是我们可以指定数组值随时改变场景，而不用为每一种地图制作一张图档。采用此方法的另一个好处是，障碍物可以拥有更多种变化，例如当数组元素设置为 1 时表示障碍物 A，数组元素设置为 2 时表示障碍物 B，而当数组元素设置为 0 时就表示没有障碍物。

采用重复贴图的方式可以减少图片文件的调用与增加场景的变化，然而对程序设计人员来说就必须额外增加许多判断的程序代码，我们将这一部分的内容与斜角地图的说明合并在下一节中进行说明。

11.2 斜角地图

曾经在第 8 章说明过斜角地图的制作原理，您可以使用一整张地图来制作斜角地图，然而您可能必须为每一个场景制作地图。本节采用重复贴图的方式来制作斜角地图，这可以减少图片档案的调用并增加场景的变化，然而您必须额外增加许多贴图与判断的程序代码。

11.2.1 制作透空图

由于地图必须重叠拼接，所以要使用贴图的方式制作斜角地图，必须先了解如何制作透空图，也就是在贴图时，图片的背景是透明的。因此在重复贴图时才不致于使得背景覆盖了其他的图片，您可以从图 11.19 中比较出两者的不同之处。

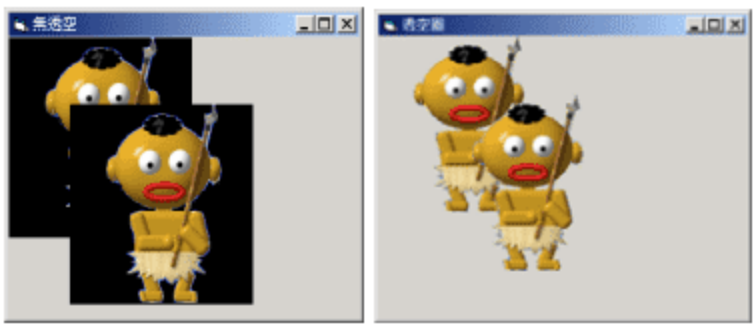


图 11.19　重复贴图的区别

透空图的制作原理是利用屏蔽，并使用绘图函数对图片与屏蔽中每一个像素进行逻辑 AND 运算。我们知道进行 AND 运算时，只要操作数中有一个为 0，则运算结果为 0。在进行贴图时如果像素值为 0 表示该点为黑色，所以将原图中的某点像素与黑色像素进行 AND 运算，则显示出来的部分就会是黑色。白色像素的 RGB 值为(255, 255,255)，在内存中表示位值全为 1；所以，如果原图中的某点与白色像素进行 AND 运算，其结果为保留原图中的颜色值。基于这个基本原理，以上图为例，原图与屏蔽图应如图 11.20 所示。

原图

屏蔽后效果

图 11.20　原图与屏蔽效果

制作透空图的方法有许多种，下面介绍实现起来最简单的一种，首先我们将屏蔽图贴到背景图上，并与背景图上的每一点进行 AND 运算，其结果会如图 11.21 所示。

图 11.21　屏蔽与背景中每一点像素进行 AND 运算

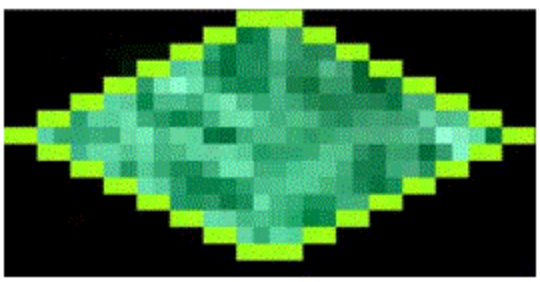
此时屏蔽图已经成为背景图的一部分，接下来我们要将原图中每一点像素与上图进行运算，并去除原图的背景。我们观察到原图中的背景被设置为黑色，所以我们在贴图时，必须将背景的黑色部分与原图中的黑色部分反相为白色，然后再进行 AND 运算，如此就可以完成透空图的制作。

大部分的贴图函数都会提供在贴图时进行相关的逻辑运算指令，使用 Visual Basic 的贴图函数进行贴图时，我们可以指定 vbSrcAnd 与 vbSrcInvert 两种参数，用于在贴图时进行 AND 运算与反相运算，制作透空图的编序代码如下：

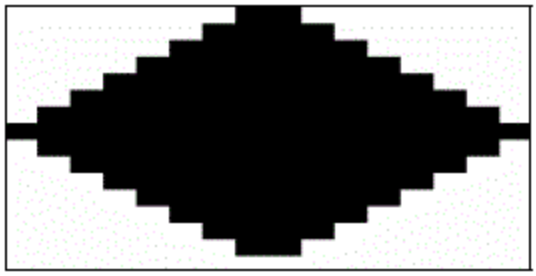
```
01 ' 程序：制作透空图
02 ' 范例文件：Transparent.vbp
03
04 Private Sub Form_Load()
05     Picture1.Visible = False
06     Picture2.Visible = False
07
08     Form1.PaintPicture Picture1, 0, 0, 300, 300, _
09         0, 0, 300, 300, vbSrcAnd
10     Form1.PaintPicture Picture2, 0, 0, 300, 300, _
11         0, 0, 300, 300, vbSrcInvert
12
13     Form1.PaintPicture Picture1, 50, 50, 300, 300, _
14         0, 0, 300, 300, vbSrcAnd
15     Form1.PaintPicture Picture2, 50, 50, 300, 300, _
16         0, 0, 300, 300, vbSrcInvert
17 End Sub
```

11.2.2 斜角地图拼接

了解透空图的制作之后，接下来我们可以开始制作第 8 章所介绍的斜角地图，因为拼接时必须使用透空图，所以地图中每一个小方格的制作都必须经过两次贴图的动作。下面所使用的地图方格图片大小设置为 32×16 像素，但贴图时会将之贴为 64×32 的大小，也就是原图的两倍，如此地图中的方格才不致于过小，我们所使用的方格放大图如图 11.22 所示。



地图方格



屏蔽图

图 11.22　方格放大图

为了让您看得出拼接时的边界，在地图方格的原图中我们先将周围以较明显的绿色加以标示出来了，在实际的游戏制作过程中若去除边界颜色，看起来就会有如大型地图。我们使用循环来实现地图的贴图与拼接操作，并加上人物的移动，使用键盘进行人物移动操作，移动的方式和效果图如图 11.23 所示。

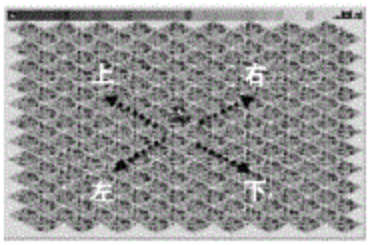


图 11.23　移动的方式和效果图

地图看来很大，其实也只使用了两张小图片而已，而为了简化程序代码的编写，人物可直接使用 Image 组件来显示，而不是使用贴图函数，完整的程序代码如下：

```
' 程序：斜角地图
' 范例文件：RpgMap.vbp

Dim X As Integer, Y As Integer

Private Sub Form_Activate()
    ' 打开自动重绘
    Form1.AutoRedraw = True
    Picture1.Visible = False
    Picture2.Visible = False
    ' 人物起始位置
```

```
12      X = 4
13      Y = 5
14
15      Dim i As Integer, j As Integer
16
17      For i = 0 To 8
18          For j = 0 To 10
19              ' 贴上第1排方格
20              Form1.PaintPicture Picture2, i * 64, j * 32, 64, 32, 0, 0,
32, 16, vbSrcAnd
21              Form1.PaintPicture Picture1, i * 64, j * 32, 64, 32, 0, 0,
32, 16, vbSrcInvert
22              '拼接第2排方格
23              Form1.PaintPicture Picture2, i * 64 + 32, j * 32 + 16, 64,
32, 0, 0, 32, 16, vbSrcAnd
24              Form1.PaintPicture Picture1, i * 64 + 32, j * 32 + 16, 64,
32, 0, 0, 32, 16, vbSrcInvert
25          Next j
26      Next i
27  End Sub
28
29  Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
30      Select Case KeyCode
31          Case 37 ' 左
32            X = X-1
33            Y = Y + 1
34          Case 39 ' 右
35            X = X + 1
36            Y = Y-1
37          Case 38 ' 上
38            X = X-1
39            Y = Y-1
40          Case 40 ' 下
41            X = X + 1
42            Y = Y + 1
43      End Select
44
45      ' 移动人物
46      Image1.Left = (X-0.5) * 32
```

```
47      Image1.Top = (Y + 1) * 16
48  End Sub
```

为了简化程序逻辑，我们在第 08 行打开了程序的自动重绘功能，如此窗口在被覆盖之后地图也不致于消失。您在这个地图中也看到了地图拼接时的一个问题，就是地图周围会出现锯齿状，我们可以在周围补贴上一些地图方格以解决这个问题，而在这里我们则使用更有效的方法，由窗口外围开始贴图，让地图超过窗口可显示的范围，这样就不必额外在周围方格的贴图动作上花时间，这个程序的改进结果如图 11.24 所示，而这次我们已将方格周围的边界线去除掉了。

图 11.24　无锯齿、无边界的斜角地图

下面是这个程序的代码，您可以看到程序代码几乎没有什么改变，而只是调整了当中的一些设置值：

```
01  ' 程序：无锯齿的斜角地图
02  ' 范例文件：RpgNoEdge.vbp
03
04  Dim X As Integer, Y As Integer
05
06  Private Sub Form_Activate()
07      ' 打开自动重绘功能
08      Form1.AutoRedraw = True
09      Picture1.Visible = False
10      Picture2.Visible = False
11      ' 人物的起始位置
12      X = 8
13      Y = 8
14
```

```
15    Dim i As Integer, j As Integer
16
17    '拼接与贴图
18    For i = 0 To 8
19       For j = 0 To 10
20          Form1.PaintPicture Picture2, i * 64-32, j * 32-16, 64, 32,
0, 0, 32, 16, vbSrcAnd
21          Form1.PaintPicture Picture1, i * 64-32, j * 32-16, 64, 32,
0, 0, 32, 16, vbSrcInvert
22
23          Form1.PaintPicture Picture2, i * 64, j * 32, 64, 32, 0, 0,
32, 16, vbSrcAnd
24          Form1.PaintPicture Picture1, i * 64, j * 32, 64, 32, 0, 0,
32, 16, vbSrcInvert
25       Next j
26    Next i
27 End Sub
28
29 Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
30    Select Case KeyCode
31       Case 37 ' 左
32         X = X-1
33         Y = Y + 1
34       Case 39 ' 右
35         X = X + 1
36         Y = Y-1
37       Case 38 ' 上
38         X = X-1
39         Y = Y-1
40       Case 40 ' 下
41         X = X + 1
42         Y = Y + 1
43    End Select
44
45    ' 移动人物
46    Image1.Left = X * 32
47    Image1.Top = (Y-1) * 16
48 End Sub
```

11.2.3 有障碍物的斜角地图

使用贴图方式制作地图的好处是，可以随时改变地图上的景物配置，而无需重新制作地图。下面使用数组来制作障碍物地图，数组元素值为 0 表示没有障碍物，大于 0 表示有障碍物。我们可以为每个不同种类的障碍物进行编码，因此只要改变量组中的元素值，就可以改变地图上的景物配置。

然而首先必须解决的问题是坐标定位的问题，您必须将绘图坐标中的每一个点与数组中的元素索引相配合，如此才不致于有移动判断错误的问题，制作完成的效果与坐标定位方式如图 11.25 所示。

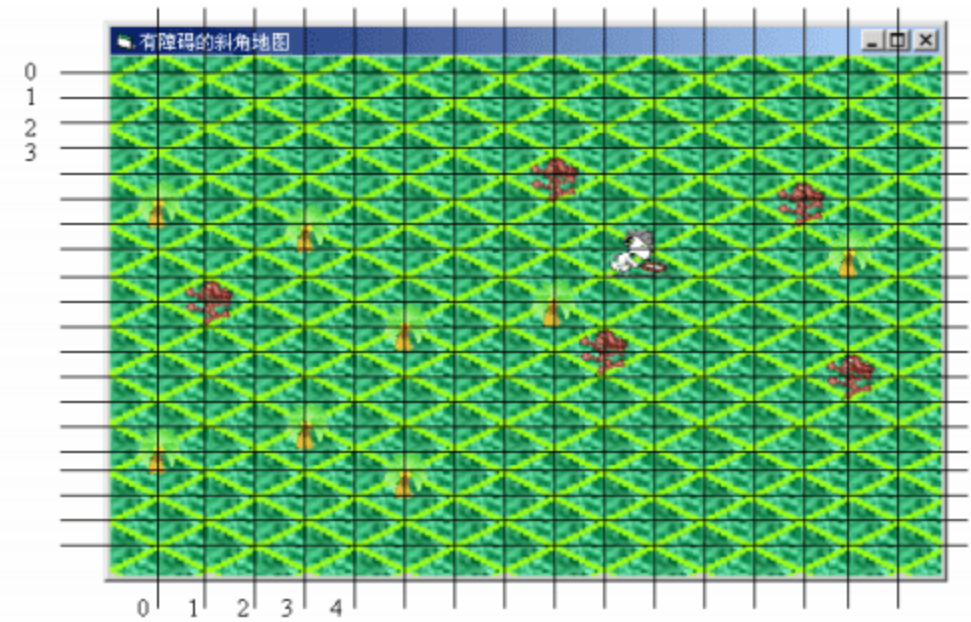


图 11.25　坐标定位与数组索引的对应

在进行判断人物的移动时，只要对数组的元素值进行检查，如果不为 0 就表示有障碍物，所以画面上的人物就不能移往该窗格，如图 11.26 所示为更换了无边界背景之后的执行效果。

图11.26　少了方格线，已有游戏的感觉了

场景中只有两种障碍物，数组元素值设置为 1 表示骷髅头，设置为 2 表示树木，在实际制作游戏的过程中您可以设置更多种类的障碍物，以下是这个程序的实现源代码：

```
01 ' 程序：有障碍物的斜角地图
02 ' 范例文件：RpgBlock.vbp
03
04 Dim X As Integer, Y As Integer
05 Dim ax As Integer, ay As Integer
06 Dim block(16, 20) As Integer
07
08 Private Sub Form_Activate()
09     Dim i As Integer, j As Integer
10     Dim arr(9)
11     Dim test(2, 2) As Integer
12
13     ' 打开自动重绘功能
14     Form1.AutoRedraw = True
15     Picture1.Visible = False
16     Picture2.Visible = False
17     Picture3.Visible = False
18     Picture4.Visible = False
19     Picture5.Visible = False
20     Picture6.Visible = False
21
22     ' 人物的起始位置
23     X = 8
24     Y = 8
```

```
25
26      ' 在数组中设置障碍物位置
27      ' 1 表示骷髅头
28      block(8, 10) = 1
29      block(1, 9) = 1
30      block(5, 17) = 1
31      block(8, 4) = 1
32      block(9, 11) = 1
33      block(13, 5) = 1
34      block(14, 12) = 1
35      ' 2 表示树木
36      block(0, 6) = 2
37      block(3, 7) = 2
38      block(5, 11) = 2
39      block(8, 10) = 2
40      block(0, 16) = 2
41      block(3, 15) = 2
42      block(5, 17) = 2
43      block(14, 8) = 2
44
45      ' 贴上草地
46      For i = 0 To 8
47          For j = 0 To 10
48              Form1.PaintPicture Picture2,i*64-32,j * 32-16,64,32,_
49                  0, 0, 32, 16, vbSrcAnd
50              Form1.PaintPicture Picture1, i * 64-32,j*32-16,64,32, _
51                  0, 0, 32, 16, vbSrcInvert
52
53              Form1.PaintPicture Picture2, i * 64, j * 32,64,32, _
54                  0, 0, 32, 16, vbSrcAnd
55              Form1.PaintPicture Picture1, i * 64, j * 32,64,32, _
56                  0, 0, 32, 16, vbSrcInvert
57          Next j
58      Next i
59
60      ' 贴上障碍物
61      For i = 0 To 16
62          For j = 0 To 20
63              Select Case block(i, j)
```

```
64              Case 1  ' 贴上骷髅头
65                  Form1.PaintPicture Picture4,(i+0.5)*32,(j-1)*16,_
66                      32, 32, 0, 0, 300, 300, vbSrcAnd
67                  Form1.PaintPicture Picture3,(i +0.5)*32,(j-1)*16,_
68                      32, 32, 0, 0, 300, 300, vbSrcInvert
69              Case 2  ' 贴上树木
70                  Form1.PaintPicture Picture6,(i+0.5)*32,(j-1)*16,_
71                      32, 32, 0, 0, 300, 300, vbSrcAnd
72                  Form1.PaintPicture Picture5,(i+0.5)*32,(j-1)*16,_
73                      32, 32, 0, 0, 300, 300, vbSrcInvert
74          End Select
75      Next j
76  Next i
77 End Sub
78
79 Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
80      ' 判断障碍物与边界
81      Select Case KeyCode
82          Case 37 ' 左
83              If X-1 >= 0 And Y + 1 <= 20 Then
84                  If block(X-1, Y + 1) = 0 Then
85                      X = X-1
86                      Y = Y + 1
87                  End If
88              End If
89          Case 39 ' 右
90              If X + 1 <= 16 And Y + 1 >= 0 Then
91                  If block(X + 1, Y-1) = 0 Then
92                      X = X + 1
93                      Y = Y-1
94                  End If
95              End If
96          Case 38 ' 上
97              If X-1 >= 0 And Y . 1 >= 0 Then
98                  If block(X-1, Y-1) = 0 Then
99                      X = X-1
100                     Y = Y-1
101                 End If
102             End If
```

```
103         Case 40 ' 下
104             If X + 1 <= 16 And Y + 1 <= 20 Then
105                 If block(X + 1, Y + 1) = 0 Then
106                     X = X + 1
107                     Y = Y + 1
108                 End If
109             End If
110     End Select
111
112     ' 移动人物
113     Image1.Left = (X + 0.5) * 32
114     Image1.Top = (Y-1) * 16
115 End Sub
```

这个程序存在一个潜在的问题，因为我们是使用数组来标示障碍物的存在，在某些场景中障碍物不多时，数组中多数的元素值将会是 0，而形成所谓的“稀疏数组”，这些内容为 0 的元素相当于没有存储任何的信息，但仍占有一定的内存空间，尤其是在地图越来越大的时候，这个情况可能会更严重。

在过去内存容量不大且价格昂贵的时候，内存空间的使用必须相当有效率，稀疏数组的问题就必须加以解决，这是数据结构所讨论的范围，在现在内存容量越来越大时，虽然这会浪费内存空间，然而却可以简化程序设计的逻辑，不过当内存空间成为设计程序时的考虑因素时，稀疏数组的问题仍然必须依赖数据结构来加以解决。

本节介绍了斜角地图上的一些实际制作过程，然而我们的障碍物与人物仍然是平面绘图，如果您考虑了立体障碍物的存在、地图的转动、立体人物的走动时，程序编写的基本原理是不变的，只不过您必须处理更多的细节。在更深入的应用中，您甚至可以制作起伏的地形背景，而这又牵涉到更多的算法，这已经超出本书讨论的范围，虽然如此，在这里所介绍的内容已足以让您设计出一些中规模的 2D 游戏。

下一节中我们将进行粒子运动方面的讨论，模拟真实世界中一些粒子活动的状态。

11.3 粒子运动

想要模拟自然界的粒子运动，我们必须对基本的物理移动有所认识，不过自然界的物理现象有时过于复杂，此时我们可以利用一些随机数或其他方式来加以模拟，以简化程序设计的过程。

11.3.1 爆炸烟火

粒子运动一个很好的例子是烟火的爆炸，当烟火爆炸时，会产生无数的烟火碎片，每一个碎片就是一个粒子，每个粒子拥有各自的位置、水平初速度、垂直初速度、颜色与生命周期等，粒子的信息描述越详细，烟火的模拟就可以越逼真，为了简化范例说明的逻辑过程，我们将每个粒子的信息定义如下：

```
Private Type pot
    state As Boolean  ' 是否存活
    x As Integer      ' 碎片目前的X位置
    y As Integer      ' 碎片目前的Y位置
    Vx As Integer     ' 碎片的水平速度
    Vy As Integer     ' 碎片的垂直速度
    color As Long     ' 绘制碎片的不同颜色
End Type
```

粒子是否存活就表示其是否燃烧殆尽，基本上每个粒子的燃烧时间应该是不同的，不过由于它是在屏幕上模拟显示效果，我们简化为粒子超出窗口范围就表示不存活；至于粒子起始位置则以随机数决定，然后则根据爆炸时所获得的水平速度、垂直速度与重力加速度来决定，每个粒子的水平初速度与垂直初速度本应由动量守恒定律来决定，您若要去计算每一个粒子的质量，将会使程序的设计过程变得更加复杂，在这个程序中我们直接使用随机数来模拟爆炸时所获得的速度与方向；另外就是烟火的颜色，这时取随机数 0～15 的数，每个数表示 QBColor 函数的一个指定值，因此每个粒子将可获得不同的颜色，而制造出五彩缤纷的烟火效果，基于以上的原理，每个粒子起始状态的代码如下：

```
01 ' 起始烟火
02 Private Sub StartParticles()
03     Dim i, x_center, y_center As Integer
04     Randomize
05     ' 设置爆炸点，窗口预设为 640×480
06     x_center = Rnd() * 320
07     y_center = Rnd() * 240
08     ' 以循环设置粒子的信息
09     For i = 0 To Max-1
10         fire(i).x = x_center
11         fire(i).y = y_center
12         ' 爆炸，设置粒子存活
13         fire(i).state = True
14         ' 以随机数决定粒子颜色
```

```
        fire(i).color = QBColor(Rnd() * 15)
        ' 以随机数决定粒子水平与垂直初速度
        fire(i).Vx = Rnd() * 40-Rnd() * 40
        fire(i).Vy = Rnd() * 40-Rnd() * 40
    Next
End Sub
```

在烟火爆炸之后，我们必须检查每个粒子是否燃烧完毕，由于已将粒子燃烧完毕简化为粒子是否超出窗口，所以设计程序时也转变为判断粒子是否超出窗口，如果超出则设置粒子为不存活状态。当所有的粒子都不存活时，我们必须重新启动粒子，也就是重新施放烟火，下面是该程序实现的代码片段：

```
'检查状态
Private Sub CheckState()
    Dim i As Integer
    Dim Play As Boolean
    For i = 0 To Max-1
    ' 如果还有碎片存活，就离开循环
        If fire(i).state = True Then
            Play = True
            Exit For
        End If
    Next

    If Not Play Then
    ' 设置时间为0，并重新初始碎片状态
        time = 0
        StartParticles
    Else
        time = time + 1
    End If
End Sub
```

至于每一次烟火粒子的移动，就是单纯的重力加速度模拟了，我们使用循环来清除上一次绘图并重新绘制每个粒子以及对每个粒子进行位置计算，下面是程序实现的代码片段：

```
' 移动烟火
Private Sub ShowMove()
    ' 清除前一次绘图
    Form1.Cls
    Dim i As Integer
```

```
    For i = 0 To Max-1
       If fire(i).state Then
          ' 绘制所有碎片
          Form1.PSet (fire(i).x, fire(i).y), fire(i).color
          ' 如果超出屏幕，表示碎片燃烧完毕
          If (fire(i).x > 640 Or fire(i).x < 0 Or fire(i).y > 480)
Then
             fire(i).state = False
          End If
          ' 水平移动与垂直移动
          fire(i).x = fire(i).x + fire(i).Vx
          fire(i).y = fire(i).y + fire(i).Vy
          ' 重力加速度
          fire(i).Vy = fire(i).Vy + 9 * time
       End If
    Next
End Sub
```

至于烟火的施放周期，自然是利用 Timer 来控制了，对每一次的时间间隔，都必须检查粒子状态、移动粒子并计算位置，前面已经将这两个过程写为函数，因此每次只要调用上面两个函数即可：

```
' 定时播放
Private Sub Timer1_Timer()
    CheckState
    ShowMove
End Sub
```

这个程序的范例文件为 FireRock.vbp，您可以自己打开执行来浏览结果，前面已经线给出了主要的程序代码，完整的程序代码就不在这里列出了，如图 11.27 所示为程序执行时的播放效果。

图 11.27　五彩缤纷的烟火（注，此图则于印刷问题，效果不明显）

11.3.2 雪花效果

烟火的粒子系统是比较容易仿真的，虽然每个粒子必须处理的信息较多，然而由于经过适当的简化，在物理现象的模拟上并不困难。接下来我们所要讨论的雪花效果，虽然每个雪花粒子的基本信息较少，但为了逼真地模拟出雪花飘落的效果，反而必须花费一些功夫处理物理现像的细节，首先我们来看看一个雪花粒子的基本定义：

```
01 ' 定义雪花粒子
02 Private Type snow
03     x As Integer  ' 雪花的 X 位置
04     y As Integer  ' 雪花的 Y 位置
05     size As Integer ' 雪花的大小
06 End Type
```

在上面的定义中，每个粒子只拥有三个基本信息，因为每个雪花受到空气阻力的影响，重力加速度反而不是影响飘落速度的主因。事实上每个雪花几乎是以等速度落下，而这个速度取决于雪花的大小，大片的雪花应该拥有较快的下落速度，而小片的雪花应该慢慢落下。当风吹动的时候，风力对每一片雪花的影响也不根同，大片雪花应该不容易被吹动，而小片雪花受风力的影响会较大。所以综上所述，雪花的大小将是模拟的效果是否逼真的主要因素。

在程序开始时，我们使用随机数来决定每个雪花的位置，我们将雪花一律先产生在窗

口的上方，因此在落下至窗口可视范围的时间就不会相同，而雪花的大小也是根据随机数来决定，实现的程序代码如下：

```
Private Sub Form_Activate()
    Dim i As Integer

    Picture1.Visible = False
    Picture2.Visible = False

    ' 开始每个雪花的位置与大小
    Randomize
    For i = 0 To Num
        s(i).x = Rnd() * 640
        s(i).y = Rnd() * 480-480
        s(i).size = Rnd() * 5 + 5
    Next i
End Sub
```

雪花的飘落受风力与重力的影响，我们使用雪花的大小信息作为风力与重力影响的权数（weight）。而当雪花落下至超出窗口范围时，则重新将之产生至窗口的上方再次落下，如此就可制出雪花循环飘落的效果，程序实现的代码如下：

```
' 移动雪花
Private Sub Timer1_Timer()
    Dim i As Integer

    ' 清除上一次的绘图
    Form1.Cls
    ' 绘制每一个雪花
    For i = 0 To Num
        Form1.PaintPicture Picture2, s(i).x, s(i).y, _
            s(i).size, s(i).size, 0, 0, 50, 50, vbSrcAnd
        Form1.PaintPicture Picture1, s(i).x, s(i).y, _
            s(i).size, s(i).size, 0, 0, 50, 50, vbSrcInvert

        ' 水平位移，考虑每个粒子的不动的风动影响
        s(i).x = s(i).x + wind * 10 / s(i).size
        ' 垂直位移，考虑每个粒子不同空气阻力
        s(i).y = s(i).y + s(i).size * 0.5

```

```
        ' 如果超出屏幕，则重新从上方飘落
        If s(i).y > 480 Then
            s(i).x = Rnd() * 640
            s(i).y = Rnd() * 480-480
        End If
    Next i
End Sub
```

至于刮风的时机与风力的大小，我们也是用随机数来决定，在每一次刮风之后，使用随机数决定发出 Timer 事件的下一次时间间隔，下面是程序实现的代码片段：

```
' 刮风了
Private Sub Timer2_Timer()
    Randomize
    ' 以随机数决定风速
    wind = Rnd() * 6-3
    ' 以随机数决定下次风吹的时间
    Timer2.Interval = Rnd() * 4000
End Sub
```

对于雪花效果模拟的主要程序实现过程，前面已经给出了，您可以打开范例文件 Snow.vbp 来执行以观看结果，如图 11.28 所示为执行时的一个效果图。

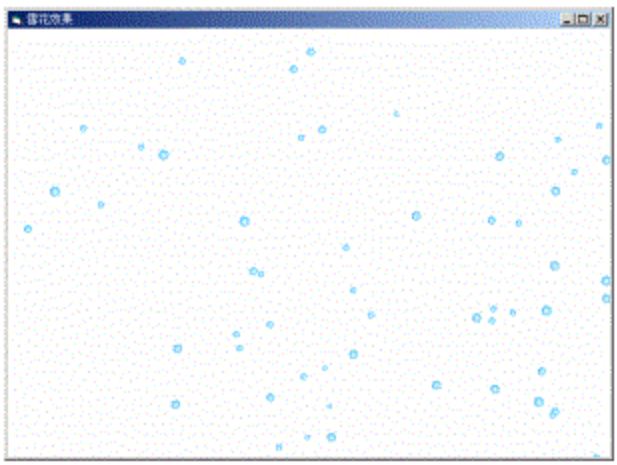


图 11.28　雪花效果，每个粒子的大小并不相同

与雪花粒子类似的是烟粒子，因为它们同样受到风力的影响，而烟上升的速度则受空气阻力影响，也几乎是做等速度上升运动。所以烟粒子的仿真与雪花粒子的仿真极其类似，

差别在于一个是上升，一个是下降，而烟的产生处则集中于一点，并必须加上扩散的效果，您可以自己按上面的讨论结果设计粒子的运动。

11.3.3 瀑布粒子

这里最后一个制作的粒子系统是瀑布粒子，瀑布粒子就是水分子的流动。在流动的过程中每个粒子会不断地产生挤压与碰撞，要完全模拟水分子的流动、挤压与碰撞过程会使得程序的设计变得相当复杂，一个粒子所必须包括的信息也会增加，然而通过一些其他的模拟方式，仍然可以制作出相当接近的瀑布流动效果。首先为每一个粒子定义如下：

```
' 瀑布粒子
Private Type water
    x As Single      ' X坐标
    y As Single      ' Y坐标
    Vx As Single     ' 水平速度
    Vy As Single     ' 垂直速度
    time As Integer  ' 下落时间
End Type
```

为了让程序的执行结果富有变化性，所以在粒子下落的过程中再设置一层障碍物，在碰到障碍物时粒子的垂直下落速度将变为 0，直到离开障碍物再继续下落，下面您就可以看到利用下落时间（time）来控制碰到障碍物的效果。

我们使用 1000 个粒子来模拟水分子的运动，首先由于每个粒子流动的速度不同，我们先将粒子产生于窗口之外，如此它们进入窗口的时间就不同。而水粒子不可能只在一个平面移动，由于挤压的效应，水流动时会形成一层厚度，在 Y 方向使用随机数来模拟，而以不同的水平速度形成粒子挤压时交相出现的效果。其粒子开始方式的程序代码如下：

```
Private Sub Form_Activate()
    Dim i As Integer

    wcolor = QBColor(14)     ' 水的颜色
    bcolor = QBColor(15)     ' 障碍物的颜色
    Timer1.Interval = 50     ' 播放时间

    Randomize
    For i = 0 To NUM
        w(i).x = Rnd() * -50     ' 粒子先产生在窗口外部
        w(i).y = Rnd() * 5 + 15 ' 以随机数模拟粒子上下挤压而产生的效果
        w(i).Vx = Rnd() * 5 + 2 ' 以随机数决定粒子水平速度
```

```
        w(i).Vy = 0              ' 垂直速度先设为 0
        w(i).time = 0            ' 下落时间先设为 0
    Next i
End Sub
```

粒子的水平移动速度是固定的，而落下效果与碰到障碍物的实现操作，就是重力加速度与时间的控制问题。当碰到障碍物时，只要将下落时间设置为 0，重力加速度就不会对粒子的垂直位置产生影响，实现的程序代码如下：

```
Private Sub Timer1_Timer()
    Form1.Cls  ' 清除上次绘图
    Form1.Line (0, 20) - (50, 20), bcolor      ' 第1个障碍物
    Form1.Line (50, 100) - (150, 100), bcolor ' 第2个障碍物

    ' 绘制与移动水粒子
    For i = 0 To NUM
        Form1.PSet (w(i).x, w(i).y), wcolor
        w(i).x = w(i).x + w(i).Vx
        ' 如果超出第1层障碍物
        If w(i).x > 50 Then
            w(i).y = w(i).Vy * w(i).time / 100 + w(i).y ' 下落距离
            w(i).Vy = 9.8 * w(i).time  ' 加速度
            w(i).time = w(i).time + 1  ' 下落时间
        End If

        ' 如果遇到第2层障碍物
        If w(i).y >= 100 And w(i).x < 150 Then
            w(i).y = 90 + Rnd() * 5  ' 以随机数产生水面厚度
            w(i).time = 0  ' 暂停下落时间
        End If

        ' 如果超出窗口显示范围
        If w(i).y >= 250 Then
        ' 重新开始水粒子
            w(i).x = Rnd() * -50
            w(i).y = Rnd() * 5 + 15
            w(i).Vx = Rnd() * 5 + 2
            w(i).Vy = 0
            w(i).time = 0
        End If
```

```
    Next i
End Sub
```

凭借一些随机数设置与简化，我们也可以将模拟复杂的水分子流动效果，当然如果要求更精细的模拟，所使用到的物理学知识将会更多，这涉及每个人对物理学科了解多少的问题了。如图 11.29 所示是程序执行的效果图。

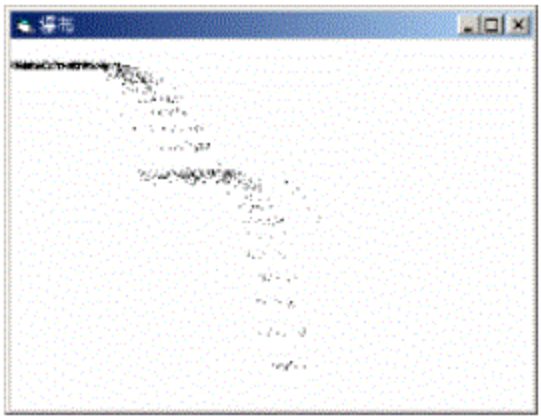


图 11.29　瀑布效果

对于粒子系统的介绍就到此先告一段落，其实您可能也发现在实际制作时都是类似的，只要第 1 步对粒子信息的包装正确，接下来进一步对物理现象的模拟就不是难事了。要在程序中更逼真的仿真粒子，您就必须对现实世界中的粒子运动更加了解，必要时您还可以结合对象导向的观念，以包装更多的粒子信息。

11.4 立体坐标与投影效果

我们曾经在第 8 章与第 9 章中对立体坐标作了说明，在前两章节中是在介绍如何进行坐标转换，而在这一节中，我们将直接使用坐标转换公式，并制作一些基本的立体图形效果，以让您了解如何在 2D 平面屏幕上显示 3D 的效果。

11.4.1 立体坐标转换

对于立体坐标，我们所经常使用的是直角坐标，也就是以 X、Y、Z 轴来定位立体空间中的一点，而另一个经常使用的立体坐标系则为极坐标，使用 r、θ、a 来描述空间中的一点，直角坐标与立体坐标的示意图如图 11.30 所示。

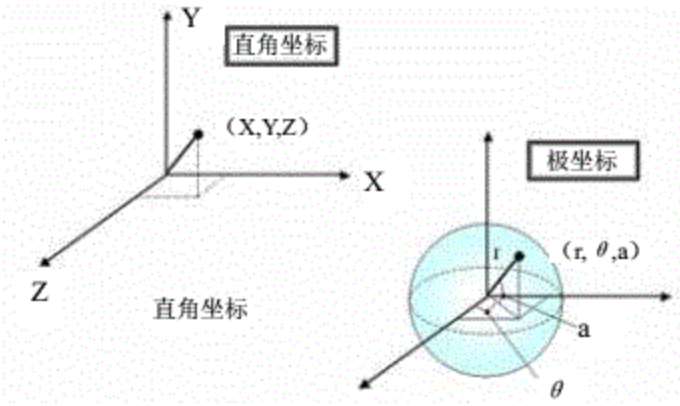


图 11.30　直角坐标与立体坐标示意图

其中 X、Y、Z 与 r、θ、a 的互换公式，我们可以配合三角函数来进行转换，转换公式如下所示：

$$x = r\cos\theta\sin a$$
$$y = r\sin\theta$$
$$z = r\cos\theta\cos a$$

在数学上要显示立体空间中的一个坐标点时，使用极坐标会比使用直角坐标来得方便，然而我们一般比较熟悉的是直角坐标，因为它比较直觉与简单。而在计算机屏幕绘图上，也是采用直角坐标，其中最基本的立体坐标投影平面坐标方式，就是将 Z 轴设置为 0，我们可以使用一个著名的心脏线公式来说明，这个公式如下：

$$r = a(1-\sin\theta)$$

这个公式使用极坐标，针对每一个 r、θ、a 画出空间中每一个点，将会得到一个立体图形，如果将这个立体图形投影至 XY 平面，看起来会像是一个心脏的形状，因此称之为心脏线公式，我们可以利用程序画出这个图形，使用程序的现代码如下：

```
Private Sub Form_Activate()
    Dim i As Integer, j As Integer
    Const PI As Single = 3.14159
    Dim r As Single, x As Integer, y As Integer
    Dim Xc As Integer, Yc As Integer

    Form1.BackColor = vbWhite
    Form1.AutoRedraw = True
```

```
    ' 绘图中心
    Xc = 200
    Yc = 50

    ' 从 0° 到 360°
    For i = 0 To 360
        For j = 0 To 360
            r = i * PI / 180 * (1-Sin(j * PI / 180))
            x = r * Cos(j * PI / 180) * Sin(i * PI / 180) * 20
            y = -r * Sin(j * PI / 180) * 20 ' 向下为Y方向，取值为正值
            PSet (x + Xc, y + Yc)  ' 绘点
        Next j
    Next i
End Sub
```

如图 11.31 所示为程序的执行效果图。

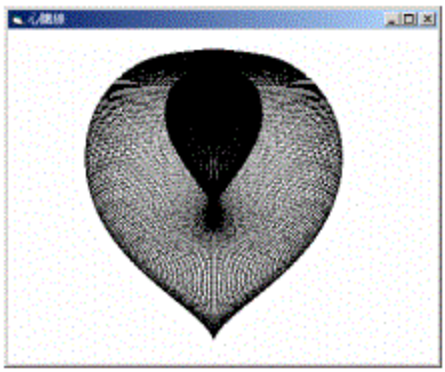


图 11.31　程序实现的心脏线公式绘图

11.4.2 立体坐标旋转

前面介绍的是没有旋转的立体坐标投影，您也许会想要旋转上面这个图形以观察每一个角度的显像结果，前面曾经谈过立体坐标旋转公式的计算方式，如图 11.32 所示为旋转关系图。

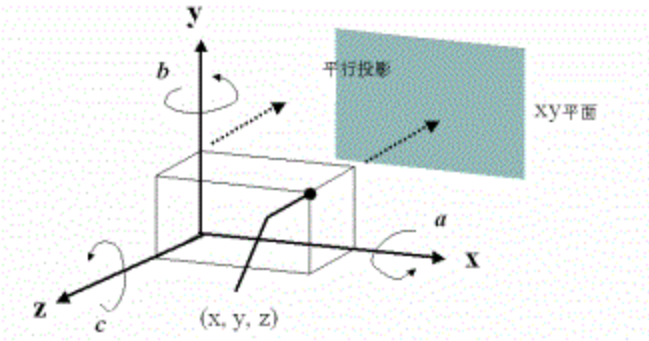


图 11.32　旋转关系图

公式如下：

$$\left.\begin{aligned} x_1 &= x\cos(a) + z\sin(b) \\ y_1 &= y \\ z_1 &= -x\sin(b) + z\cos(b) \end{aligned}\right\}$$ 绕 *Y* 轴旋转 b 角度

$$\left.\begin{aligned} x_2 &= x_1 \\ y_2 &= y_1\cos(a) - z_1\sin(a) \\ z_2 &= y_1\sin(a) - z_1\cos(a) \end{aligned}\right\}$$ 绕 *X* 轴旋转 a 角度

$$\left.\begin{aligned} x_3 &= x_2\cos(c) - y_2\sin(c) \\ y_3 &= x_2\sin(c) + y_2\cos(c) \\ z_3 &= z_2 \end{aligned}\right\}$$ 绕 *Z* 轴旋转 c 角度

以上所列的最后三条公式即是立体图形旋转后在空间中的直角坐标，而我们可以将之投影至 XY 平面上以便于屏幕绘图，所以 Z 轴坐标可以不必理会，也就是设置为 0，此时可求出公式如下所示：

$$\left.\begin{aligned} x_1 &= x\cos(a) + z\sin(b) \\ y_1 &= y \\ z_1 &= -x\sin(b) + z\cos(b) \\ x_2 &= x_1 \\ y_2 &= y_1\cos(a) - z_1\sin(a) \\ x_3 &= x_2\cos(c) - y_2\sin(c) \\ y_3 &= x_2\sin(c) + y_2\cos(c) \end{aligned}\right\}$$ 这是投射到 *XY* 平面的坐标值

这种方法称之为平行投影，也就是不考虑立体对象远近感的问题，适用于表示小型的立体对象，我们以一个简单的旋转正立方体程序来实现以上的公式制作过程。立方体的边长为 100，并以原点为中心，我们以变量 a、b、c 来控制立方体的旋转，您可以先打开范例文件 3DCubic.vbp 来操作执行的效果，如图 11.33 所示为操作时的几个参考画面。

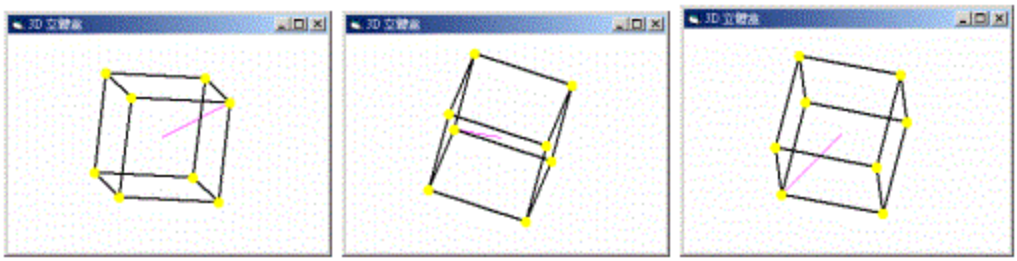


图 11.33　操作参考画面

我们特地将立方体的顶点以不同的颜色标示出来，而事实上在绘图时，也只是利用绘制直线的函数连接各顶点以绘出立方体，其中我们特别加上一条由中心至第 1 个顶点的直线，以了解目前立方体的旋转角度，vbp 关于程序的实现过程，我们直接在以下程序代码中使用批注进行说明：

```
01 ' 程序：立方体坐标旋转
02 ' 范例文件：3DCubic.vbp
03
04 Dim a As Single, b As Single, c As Single
05 Dim p(2, 7) As Single
06 Dim Xc As Single, Yc As Single ' 绘图中心
07 Const PI As Single = 3.14159
08 ' 旋转后的顶点投影
09 Dim x(7) As Single, y(7) As Single, z(7) As Single
10 ' 旋转角度变量
11 Dim da As Single, db As Single, dc As Single
12
13 Private Sub Form_Activate()
14     Dim i As Integer
15     Dim arr0, arr1, arr2
16
17     Timer1.Interval = 200  ' 播放时间间隔
18
19     Xc = 150
```

```
    Yc = 100

    ' 立方体旋转初始角度
    a = PI / 36 * 45 ' 45°°
    b = PI / 36 * 45 ' 45°°
    c = PI / 36 * 45  ' 45°

    ' 以数组记录立方体的立体坐标
    arr0 = Array(1, 1, -1, -1, 1, 1, -1, -1) ' X
    arr1 = Array(1, -1, -1, 1, 1, -1, -1, 1) ' Z
    arr2 = Array(1, 1, 1, 1, -1, -1, -1, -1) ' Y

    For i = 0 To 7
        p(0, i) = arr0(i)
        p(1, i) = arr1(i)
        p(2, i) = arr2(i)
    Next i
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    '用上、下、左、右箭头键和PageDown、PageUp键来操作立方体
    Select Case KeyCode
        Case 38  ' 上
            da = PI / 36
            db = 0
            dc = 0
        Case 40  ' 下
            da = -PI / 36
            db = 0
            dc = 0
        Case 37  ' 左
            db = PI / 36
            da = 0
            dc = 0
        Case 39  ' 右
            db = -PI / 36
            da = 0
            dc = 0
        Case 33  ' Page Up
```

```
59            dc = PI / 36
60            da = 0
61            db = 0
62        Case 34  ' Page Down
63            dc = -PI / 36
64            da = 0
65            db = 0
66        Case Else ' 按下任一键停止旋转
67            dc = 0
68            da = 0
69            db = 0
70    End Select
71
72 End Sub
73
74 Private Sub Timer1_Timer()
75    Dim x1 As Single, x2 As Single, x3 As Single
76    Dim y1 As Single, y2 As Single, y(7) As Single
77    Dim z1 As Single, z2 As Single, z(7) As Single
78    Dim i As Integer
79
80    Form1.Cls  ' 清除上一次绘图
81
82    ' 计算各个顶点的XY平面投影坐标
83    For i = 0 To 7
84       x1 = p(0, i) * Cos(b) + p(1, i) * Sin(b)
85       y1 = p(2, i)
86       z1 = -p(0, i) * Sin(b) + p(1, i) * Cos(b)
87       x2 = x1
88       y2 = y1 * Cos(a) -z1 * Sin(a)
89       x(i) = (x2 * Cos(c) -y2 * Sin(c)) * 50 + Xc
90       y(i) = (x2 * Sin(c) + y2 * Cos(c)) * -50 + Yc
91    Next i
92
93    ' 设置定位参考线绘图粗细为 1
94    Form1.DrawWidth = 1
95
96    ' 定位参考线
97    Line (Xc, Yc)- (x(0), y(0)), QBColor(13)
```

```
98
99      ' 设置边长绘图粗细为 2
100     Form1.DrawWidth = 2
101
102     ' 线制边长
103     For i = 0 To 2
104         Line (x(i), y(i))- (x(i + 1), y(i + 1))
105         Line (x(i + 4), y(i + 4))- (x(i + 5), y(i + 5))
106         Line (x(i), y(i))- (x(i + 4), y(i + 4))
107     Next i
108
109     Line (x(3), y(3))- (x(0), y(0))
110     Line (x(7), y(7))- (x(4), y(4))
111     Line (x(3), y(3))- (x(7), y(7))
112
113     ' 设置顶点绘图粗细为 10
114     Form1.DrawWidth = 10
115
116     ' 绘制顶点
117     For i = 0 To 7
118         Form1.PSet (x(i), y(i)), QBColor(14)
119     Next i
120
121     ' 旋转立方体
122     a = a + da
123     b = b + db
124     c = c + dc
125
126     ' 避免长时间执行溢位
127     If a > 2 * PI Or a <-2 * PI Then
128         a = 0
129     End If
130     If b > 2 * PI Or b <-2 * PI Then
131         b = 0
132     End If
133     If c > 2 * PI Or c <-2 * PI Then
134         c = 0
135     End If
136 End Sub
```

如程序中的注释说明，您可以使用键盘来进行立方体的旋转操作，要注意的是由于我们将坐标投影到屏幕上，所以您所看到的坐标与实际所设置的坐标是相反的，也就是近点被投影为远点，而远点会被投影为近点，由于这是一个不具远近感的立方体，所以目前比较看不出远近的效果，下面将介绍具有远近感的立方体绘图公式。

11.4.3 具有远近感的立方体

以平行方式投影的立方体在 XY 平面上看来不具有远近感，如果以非平行方式投影立方体的坐标就可以制作具有远近感的立方体图形，我们前面曾经介绍过单点与两点透视法，就是利用这种投影方式来制作。

非平行方式投影坐标时，需指定透视的消失点，而投影平面与立方体对象的位置也会影响投影大小，投影平面的距离就相当于我们观察立方体的位置，所以我们必须额外考虑观察位置的 x、y、z 三个平移位置，下面以 l、m、n 三个变量来表示，一个非平行投影的示意图如图 11.34 所示。

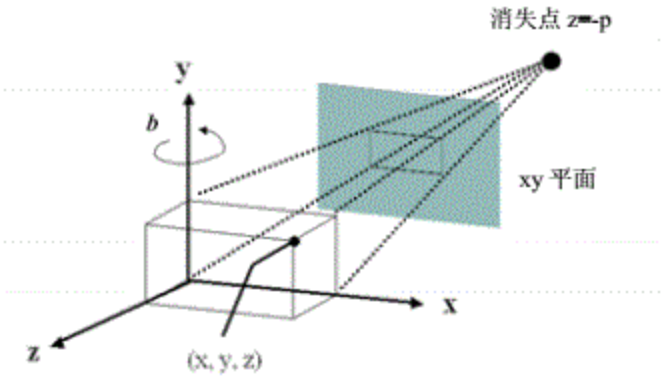


图 11.34　非平行投影的示意图

观察物体远近时通常使用定点观察，而不用旋转 x 与 z 轴，所以前面的立体坐标公式可以简化，在考虑了观察点位置之后，可以得出以下的公式：

$$x_1 = x\cos(b) + z\sin(b) + l$$

$$y_1 = y + m$$

$$h = -x\sin(b)/p + z\cos(b)/p + n/p + l$$

$$x = x1/h$$

$$y = y1/h$$

基于这个公式，可以将前一个范例加以改写，使立方体在旋转时同时能显示远近感，您可以先打开范例 3DCubic2.vbp 来执行并观察执行的结果，如图 11.35 所示为执行效果图。

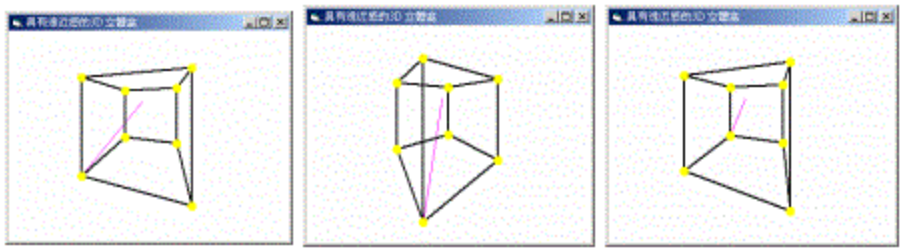


图 11.35　执行效果图

以下为这个程序的程序代码：

```
01 ' 程序：具有远近感的3D立方体
02 ' 范例文件：3DCubic2.vbp
03
04 Dim a As Single, b As Single, c As Single
05 Dim p(2, 7) As Single
06 Dim Xc As Single, Yc As Single
07 Const PI As Single = 3.14159
08 Dim x(7) As Single, y(7) As Single, z(7) As Single
09 Dim db As Single  ' 绕 Y 轴旋转变量
10 Dim l As Single, m As Single, n As Single
11
12 Private Sub Form_Activate()
13     Dim i As Integer
14     Dim arr0, arr1, arr2
15
16     Timer1.Interval = 200  ' 播放时间
17
18     ' 绘图中心
19     Xc = 150
20     Yc = 80
21
22     a = 0  ' 0°
23     b = PI / 36 * 45 ' 45°
24     c = 0   ' 0°
25
26     l = 2    ' 观察相对x位置
```

```
    m = 50   ' 观察相对Y位置
    n = 50   ' 观察相对Z位置

    ' 以数组记录顶点
    arr0 = Array(100, 100,-100,-100, 100, 100,-100,-100) ' X
    arr1 = Array(100,-100,-100, 100, 100,-100,-100, 100) ' Z
    arr2 = Array(100, 100, 100, 100,-100,-100,-100,-100) ' Y

    For i = 0 To 7
        p(0, i) = arr0(i)
        p(1, i) = arr1(i)
        p(2, i) = arr2(i)
    Next i
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    ' 利用左右方向键控制立方体的旋转
    Select Case KeyCode
        Case 37  ' 左
            db = PI / 36
        Case 39  ' 右
            db =-PI / 36
        Case Else
            db = 0
    End Select
End Sub

Private Sub Timer1_Timer()
    Dim x1 As Single, x2 As Single, x3 As Single
    Dim y1 As Single, y2 As Single, y(7) As Single
    Dim z1 As Single, z2 As Single, z(7) As Single
    Dim i As Integer, h As Single

    Form1.Cls ' 清除上一次绘图

    ' 计算绘图坐标
    For i = 0 To 7
        x1 = p(0, i) * Cos(b) + p(1, i) * Sin(b) + 1
        y1 = p(2, i) + m
```

```
66          h =-p(0, i) * Sin(b) / 100 + p(1, i) * Cos(b) / 100 + n / 100
+ 1
67          x(i) = x1 / h + Xc
68          y(i) = y1 / h + Yc
69      Next i
70
71      Form1.DrawWidth = 1
72
73      ' 参考线
74      Line (Xc, Yc)- (x(0), y(0)), QBColor(13)
75
76      Form1.DrawWidth = 2
77
78      ' 绘制边长
79      For i = 0 To 2
80          Line (x(i), y(i))- (x(i + 1), y(i + 1))
81          Line (x(i + 4), y(i + 4))- (x(i + 5), y(i + 5))
82          Line (x(i), y(i))- (x(i + 4), y(i + 4))
83      Next i
84
85      Line (x(3), y(3))- (x(0), y(0))
86      Line (x(7), y(7))- (x(4), y(4))
87      Line (x(3), y(3))- (x(7), y(7))
88
89      Form1.DrawWidth = 10
90
91      ' 绘制顶点
92      For i = 0 To 7
93          Form1.PSet (x(i), y(i)), QBColor(14)
94      Next i
95
96      ' 旋转立方体
97      b = b + db
98
99      ' 避免长时间执行溢值
100     If b > 2 * PI Or b <-2 * PI Then
101         b = 0
102     End If
103 End Sub
```

其实这个程序可用来显示单点透视与两点透视，如果 Y 轴旋转角度为 0 时就是单点透视，不为 0 时就是两点透视，从前面的范例执行结果中应该能看出两点透视的效果，我们改变一下观察点的 m 位置并将旋转角度设置为 0，就可以看出单点透视的效果，如图 11.36 所示。

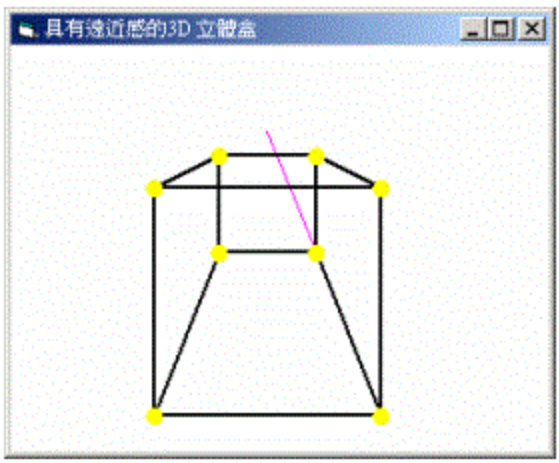


图 11.36　单点透视效果

11.4.4 旋转的心脏线

我们回过头来看看心脏线的问题，前面将心脏线投影至 XY 平面上并仅从正面观察，我们可以利用立体坐标公式的转换，制作一个可利用键盘操作以旋转图形的心脏线程序，如图 11.37 所示为程序执行的效果参考图片，您可以从几个不同的角度了解心脏线公式的奥妙。

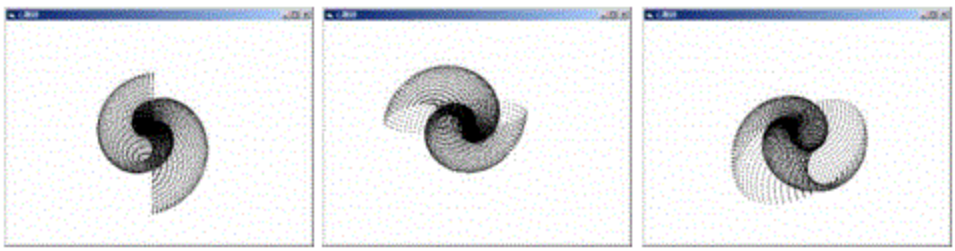


图 11.37　程序执行的效果参考图片

仔细观察程序的执行结果之后，您会发现这个心脏线公式真正的立体图案，就有如两个交叠的螺旋贝壳，凭借这个程序的辅助，我们就可以将难以想象的数学公式以实际的图

形显示出来，以下为程序的实现代码：

```
' 程序：旋转心脏线
' 范例文件：Heart2.vbp

Const PI As Single = 3.14159
' 旋转角度变量
Dim a As Single, b As Single, c As Single
Dim da As Single, db As Single, dc As Single

Dim r As Single, x As Integer, y As Integer
Dim Xc As Integer, Yc As Integer

Private Sub Form_Activate()
    Form1.BackColor = vbWhite
    Form1.AutoRedraw = True
    Form1.DrawWidth = 2
    Timer1.Interval = 200

    ' 绘图中心
    Xc = 250
    Yc = 180
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    ' 以上、下、左、右箭头键和PageDown、PageUp键来操作
    Select Case KeyCode
        Case 38  ' 上
            da = PI / 36
            db = 0
            dc = 0
        Case 40  ' 下
            da =-PI / 36
            db = 0
            dc = 0
        Case 37  ' 左
            db = PI / 36
            da = 0
            dc = 0
        Case 39  ' 右
```

```
            db =-PI / 36
            da = 0
            dc = 0
        Case 33  ' Page Up
            dc = PI / 36
            da = 0
            db = 0
        Case 34  ' Page Down
            dc =-PI / 36
            da = 0
            db = 0
        Case Else ' 按下任一键停止旋转
            dc = 0
            da = 0
            db = 0
    End Select

End Sub

Private Sub Timer1_Timer()
    Dim i As Integer, j As Integer
    Dim xt As Single, yt As Single, zt As Single
    Dim X1 As Single, Y1 As Single, _
        X2 As Single, Y2 As Single, Z1 As Single

    Form1.Cls ' 清除屏幕

    ' 从0° ~360°
    For i = 0 To 90
        For j = 0 To 90
            r = i * PI / 45 * (1-Sin(j * PI / 45))
            xt = r * Cos(j * PI / 45) * Sin(i * PI / 45) * 15
            yt = r * Sin(j * PI / 45) * 15
            zt = r * Cos(j * PI / 45) * Cos(i * PI / 45) * 15
            X1 = xt * Cos(b) + zt * Sin(b)
            Y1 = yt
            Z1 =-xt * Sin(b) + zt * Cos(b)
            X2 = X1
            Y2 = Y1 * Cos(a)-Z1 * Sin(a)
```

```
78            x = X2 * Cos(c)-Y2 * Sin(c) + Xc
79            y =-X2 * Sin(c)-Y2 * Cos(c) + Yc
80            PSet (x, y)   ' 绘点
81        Next j
82    Next i
83
84    ' 旋转
85    a = a + da
86    b = b + db
87    c = c + dc
88
89    ' 避免长时间执行溢位
90    If a > 2 * PI Or a <-2 * PI Then
91        a = 0
92    End If
93    If b > 2 * PI Or b <-2 * PI Then
94        b = 0
95    End If
96    If c > 2 * PI Or c <-2 * PI Then
97        c = 0
98    End If
99 End Sub
```

对于立体图形的绘制操作，我们就先到这里告一段落。接下来开始介绍一些碰撞的实例应用。

11.5 碰撞

碰撞也是物理现象的一种，我们在第 8 章中也介绍了一些碰撞的算法，在这一节中将设计一些基本的碰撞程序，例如多边形碰撞、打砖块游戏等。

11.5.1 多边形碰撞

有时候一些复杂的图形可以使用最简单的方法来实现，多边形碰撞屏幕保护程序就是一个简单的例子，我们只要计算顶点的碰撞与移动，再依序连接四个顶点，就可以完成一个简单的屏幕保护程序，而我们这次并不在每次绘图前就清除屏幕，而是凭借定时的屏幕清除动作，使程序的执行结果有更多的复杂变化，如图 11.38 所示为程序预览效果图。

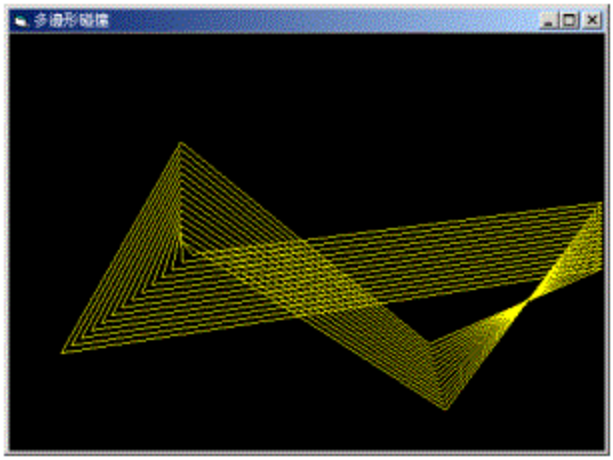


图 11.38　多边形碰撞程序预览

您只要将这个程序设置为开始执行时全屏幕显示，就可以成为屏幕保护程序，每当顶点遇到上下边界时，就改变 Y 速度方向，碰到左右边界时就改变 X 速度方向，这就是碰撞的基本原理，这个程序相当简单，我们直接用注释方式在程序中进行说明，程序代码如下所示：

```
' 程序：多边形碰撞
' 范例文件：ScreenSaver.vbp

' 定义顶点
Private Type pot
    x As Integer
    y As Integer
    Vx As Integer
    Vy As Integer
End Type

Dim p(3) As pot
Dim w As Integer, h As Integer

Private Sub Form_Activate()
    Dim i As Integer

    w = Form1.Width / 15  ' 窗口宽度
    h = Form1.Height / 15 ' 窗口高度
```

```
20
21      Timer1.Interval = 100  ' 定时绘图
22      Timer2.Interval = 5000 ' 定时清除屏幕
23      Form1.BackColor = vbBlack  ' 设置背景为黑色
24
25      '开始4个顶点
26      Randomize
27      For i = 0 To 3
28          p(i).x = Rnd() * w
29          p(i).y = Rnd() * h
30          p(i).Vx = Rnd() * 10-Rnd() * 10
31          p(i).Vy = Rnd() * 10-Rnd() * 10
32      Next i
33  End Sub
34
35  ' 定时绘图
36  Private Sub Timer1_Timer()
37      Dim i As Integer
38
39      ' 进行边的绘制
40      For i = 0 To 2
41          Line (p(i).x, p(i).y)- (p(i + 1).x, p(i + 1).y), QBColor(14)
42      Next i
43
44      Line (p(3).x, p(3).y)- (p(0).x, p(0).y), QBColor(14)
45
46
47      For i = 0 To 3
48          ' 移动顶点
49          p(i).x = p(i).x + p(i).Vx
50          p(i).y = p(i).y + p(i).Vy
51
52          ' 判断是否碰到边界
53          If p(i).x <= 10 Or p(i).x >= w-15 Then
54              p(i).Vx =-p(i).Vx
55          End If
56          If p(i).y <= 10 Or p(i).y >= h-30 Then
57              p(i).Vy =-p(i).Vy
58          End If
```

```
    Next i
End Sub

' 定时清除屏幕
Private Sub Timer2_Timer()
    Form1.Cls
End Sub
```

11.5.2 打砖块

其实打砖块也是运用到碰撞的原理，只不过在判断碰撞时不仅考虑到边界碰撞的问题，对于打砖块来说，下边界已经去除，而使用球是否碰撞到棒子来作为球是否反弹的依据。在此我们不考虑棒子的左右移动是否对球的 X 速度分量造成影响，而纯以弹性碰撞来进行设计，下面是打砖块的基本架构程序代码，这个程序中还没有将砖块加入其中：

```
' 程序：打砖块架构程序
' 范例文件：Brick.vbp

Dim ballX As Integer, ballY As Integer
Dim barX As Integer, barY As Integer
Dim w As Single, h As Integer
Dim dx As Integer, bdx As Integer, bdy As Integer

Private Sub Form_Activate()
    Timer1.Interval = 50

    ' 窗口的大小
    w = Form1.Width / 15
    h = Form1.Height / 15

    ' 球的中心坐标
    ballX = w / 2
    ballY = h / 2-50
    ' 棒子的中心坐标
    barX = w / 2
    barY = h-35
    ' 球的XY速度分量
    bdx = 5
    bdy = 5
```

```
25
26      ' 设置球的位置
27      Shape1.Left = ballX-Shape1.Width / 2
28      Shape1.Top = ballY-Shape1.Height / 2
29      ' 设置棒子的位置
30      Shape2.Left = barX-Shape2.Width / 2
31      Shape2.Top = barY-Shape2.Height / 2
32  End Sub
33
34  Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
35      Select Case KeyCode
36          Case 37 ' 左
37              dx =-20
38          Case 39 ' 右
39              dx = 20
40          Case Else ' 按任一键停止移动棒子
41              dx = 0
42      End Select
43  End Sub
44
45  Private Sub Timer1_Timer()
46      ' 移动棒子
47      Shape2.Left = Shape2.Left + dx
48
49      ' 如果棒子碰到左右边界则无法再移动
50      If Shape2.Left <= 0 Then
51          Shape2.Left = 0
52      ElseIf Shape2.Left >= w-Shape2.Width Then
53          Shape2.Left = w- Shape2.Width
54      End If
55
56      ' 移动球
57      Shape1.Left = Shape1.Left + bdx
58      Shape1.Top = Shape1.Top + bdy
59
60      ' 如果碰到棒子则反弹
61      If Shape1.Top + Shape1.Height >= Shape2.Top Then
62          If Shape1.Left > Shape2.Left-Shape1.Width And _
63              Shape1.Left < Shape2.Left + Shape2.Width Then
```

```
64            bdy =-bdy
65        End If
66    End If
67
68    ' 如果球遇到左右边界则往x反方向移动
69    If Shape1.Left <= 0 Or Shape1.Left >= w-Shape1.Width Then
70        bdx =-bdx
71    End If
72    ' 如果球遇到上边界则往x反方向移动
73    If Shape1.Top <= 0 Then
74        bdy =-bdy
75    End If
76
77    ' 如果球掉出窗口则重新发球
78    If Shape1.Top > h Then
79        ballX = w / 2
80        ballY = h / 2-50
81        Shape1.Left = ballX-Shape1.Width / 2
82        Shape1.Top = ballY-Shape1.Height / 2
83    End If
84 End Sub
```

如图 11.39 所示为打砖块基本架构程序运行的效果图。

图 11.39　一个打砖块程序的基本架构

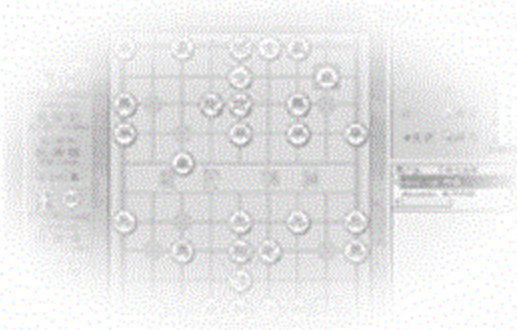
接下来我们只要再进行砖块的绘制与碰撞即可完成一个简易版的打砖块程序。球是否碰到砖块的判断方式其实是相同的，至于砖块的绘制我们可以使用一个二维数组来记录砖

块的位置，如果数组值不为 0，表示该处存在砖块，当碰撞到时就将之设置为 0，表示砖块被消去。而凭借设置数组元素值为不同的内容，我们可以绘制各种不同样式的砖块，就如同前面一节中在地图上绘制不同的障碍物。

对于一些基本的 2D 绘图算法，已经作了详细的介绍，而这里所使用到仅有键盘操作，在下一章中，我们将介绍摇杆操作与音效处理的一些基本原理。

第 12 章

音效与操作装置



游戏过程中绝不可缺少的是音效部分，本章将讨论如何将音效加入程序之中，并配合本章内容提供几个音效范例（使用 DirectX），以实际体验混音、立体音效等效果，并且介绍玩家键盘、鼠标与摇杆等操作装置的工作原理，有哪些类型、程序中应如何控制这些操作装置。

12.1 音效处理

在游戏设计领域中，音效的好坏足以决定一个游戏的成败。在游戏进行的过程中，如何搭配适当的背景音乐或是动作音效，也是一个重要的问题。更进一步您还可以搭配 3D 效果的音效，让玩家在进行游戏时能如身历其境，增加与游戏情境的结合。

本章是从程序设计的角度来进行音效的讨论，音效的制作绝不仅是单纯的将音乐播放出来，为了让您能对音效的融合有更深入的认识，我们将以 DirectX 对音效的划分与处理方式来介绍。因为包括了许多观念，包括处理混音与 3D 等方面所必要的信息。

12.1.1 声音文件

在 DirectX 中拥有两个音效处理的主要成员：DirectSound 与 DirectMusic。前者用来处理 Wave 格式的声音文件，后者则可用来处理 Midi 格式的声音文件，而游戏中所使用的声音文件通常也是以这两种格式为主。

我们知道，一般 Wave 格式的声音文件所占容量会比较大，以我们所收听的 CD 音乐品质来说，一首 3 分钟的歌曲，大约会占据 30MB 左右的空间，一张光盘通常以 650MB 为基本单位，这也就是为什么一般的音乐 CD 最多只能容纳约 15 到 20 首的歌曲（以一首约 2 到 4 分钟来计算）。

如果游戏中对于音效的品质要求极高，或是想让游戏中的音乐成为卖点之一（像是巴冷公主中的原住民吟唱，见图 12.1），通常就会采用 Wave 格式的声音文件，或是更进一步的提供音效盘（通常就是一张音乐 CD 盘），让玩家可以在游戏进行时置入播放或单独使用于随身听或是音响之中。

图 12.1　对于音效品质较高的游戏，还会额外提供音效盘

利用软件或硬件的计算能力，进一步仿真 Midi 音效播放时中间搭配的和弦效果，使得 Midi 音效也能提供悦耳的音乐。这类加强 Midi 音效的软件或硬件通常称之为“音效合成器”。它的工作原理就是将 Midi 音效加以仿真，并转换为 Wave 格式再通过声卡播放出来。

早期的 Midi 音效必须使用硬件的支持，如果声卡上没有内建音效合成器，则无法达到好的音效播放效果。不过有些声卡厂商本身会提供软件音效合成器，让我们可以利用软件来加强 Midi 的播放效果，然而使用软件仿真的坏处就是必须额外花费 CPU 的开销。

近期的一些游戏在设计时会采用 DirectX 技术，其中的 DirectSound 与 DirectMusic 就是用来处理 Wave 与 Midi 声音文件，它们也提供了软件音效合成器的功能，如果玩家的声卡已内建硬件音效合成器，则会直接使用硬件的音效合成功能。假如声卡上不支持合成器功能，则会使用 DirectSound 与 DirectMusic 的软件合成功能，DirectMusic 可以将 Midi 格式的档案仿真为高品质的 Wave 文件进行播放，而 DirectSound 则提供 3D 音效的播放功能。接下来我们所要进行的就是对这两个成员的基本介绍。

12.1.2 DirectSound 对象成员

一般人的观念，对于音效播放可能都只局限于文件本身或是播放程序，然而 DirectSound 对于一个音效的播放可区分为数个对象成员，我们仅介绍几个较为具体的成员，分别是声卡（DirectSound）、2D 缓冲区（DirectSoundBuffer）、3D 缓冲区（DirectSound3DBuffer）与 3D 空间倾听者（DirectSound3DListener），如图 12.2 所示。

图 12.2　DirectSound 对象成员

当然我们都知道要播放音效的话，计算机必须安装有声卡，而 DirectSound 即是将声卡作为一个装置对象，一个对象负责处理一组音效运算，声卡对象就等于是一个功能丰富的声卡，即使真正的声卡上所没有的硬件功能（例如音效合成器），声卡对象也可以自行仿真。

通常玩家的计算机上应该只会安装一块声卡，所以在使用 DirectSound 时也只会使用到一个声卡装置对象，在多任务操作系统中，会使用到声卡的程序并不只有游戏本身。而且在只有一块声卡的情况下，以往您必须亲自处理声卡与其他程序共享的协调问题，不过在使用 DirectSound 则无需担心这个问题，它会自行处理声卡的共享协调问题。

声音文件原本可能是在硬盘或是光盘中，要播放音效时必须先将之加载到内存，内存的位置可能是在声卡上或是主存储器中，不过在 DirectSound 中您不用担心这个问题。您只要将音效加载到缓冲区对象中，而缓冲区对象就相当于声卡对象上的内存，至于要如何

使用声卡上的内存或主存储器，DirectSound 则会自行判断，如果硬件内建有内存的话，则会尽量使用它来建立缓冲区。

除了提供基本的 2D 音效之外，DirectSound 还提供有 3D 音效的仿真功能，3D 缓冲区用来存放 3D 声音文件，DirectSound 将声卡对象具体化为一个实体的声卡，而 2D 缓冲区与 3D 缓冲区则具体化为这个声卡上所提供的 2D 音效芯片与 3D 音效芯片。

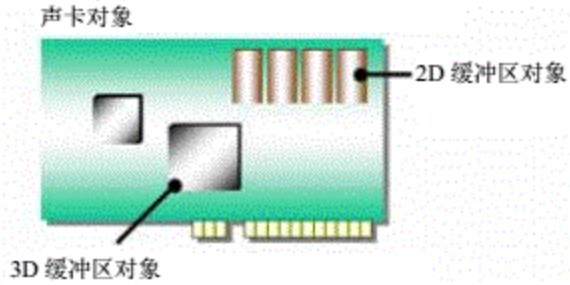


图 12.3　具体化的 DirectSound 对象示意图

对于 3D 音效而言，因为倾听者位置的不同，所听到的音效感觉就会有所不同，举例来说，音源播放的方向如果在倾听者的前方或后方时，所听到的声音方向或音量大小感觉就会不相同。在过去要运用 3D 音效往往必须使用多声道喇叭或支持多声道输出的声卡，然而 DirectSound 将倾听者也具体化为一个对象，通过设定 3D 倾听者对象的位置信息，玩家只要使用耳机或一般的喇叭，就可以体会到 3D 音效的效果，而程序设计本身并不用复杂的公式或算法。

所以您应该可以体会到 DirectSound3DBuffer 对象也是一个具体化的对象，如果将 3D 缓冲区中的数据作为一组立体空间中的喇叭，而 3D 倾听者相对于这些喇叭不同的位置，就会有不同的音效感觉，如图 12.4 所示。

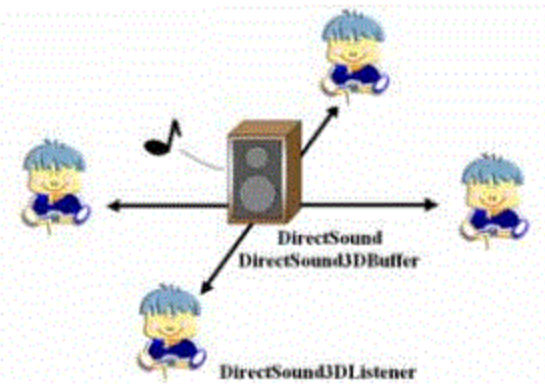


图 12.4　DirectSound3DListener 对象的具体示意图

12.1.3 音效缓冲区

在 DirectSound 中的音效缓冲区有 2D 缓冲区与 3D 缓冲区两种，缓冲区用来载入声音文件，一个缓冲区只用来载入一个声音文件。所以您也可以将一个缓冲区视为一个声音文件，2D 缓冲区可以进行平面的音效播放，而 3D 缓冲区则还拥有计算与控制 3D 音效的功能。

一般来说，玩家的计算机会配备两个喇叭，一个是用来播放左声道，另一个是用来播放右声道，而 2D 缓冲区可以直接控制左右声道声音的强弱，当然最基本的整体音量大小、播放速度控制，2D 缓冲区都具备，除了基本的单一音效文件播放功能之外，DirectSound 也提供多个音效文件同时播放的功能。根据此功能，我们可以将数个声音文件适当的加以组合并同时间播放，就可以完成许多种不同的播放效果。

举例来说，在同一个背景音乐下，当玩家行经森林时，就可以将虫鸣鸟叫的声音文件混音，而当玩家行经集市时，就可以将集市的人声、叫卖声等声音文件混音，如此一来我们就不必特地为不同的场景制作不同的声音文件。您可以执行配套光盘中的 Mixer.exe，并选取两个不同的 Wave 档案进行播放（您的计算机必须安装有 DirectX），以体会混音效果，如图 12.5 所示。

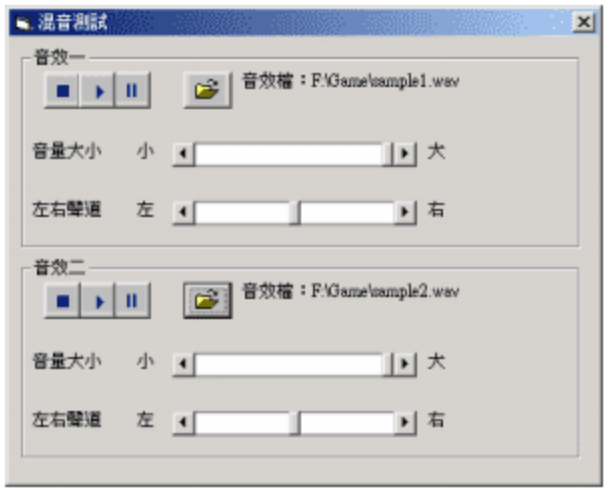


图 12.5　配套光盘所提供的混音测试程序

如果要制作 3D 音效播放效果，您就必须建立 3D 缓冲区，在 DirectSound 中 3D 缓冲区是以 2D 缓冲区作为基础，必须将 2D 缓冲区设定为具有控制 3D 音效的能力，且从中取得 3D 缓冲区对象，之后您就可以使用此缓冲区来进行 3D 音效的控制，而另一个重要的对象则为 3D 倾听者，它同样从 2D 缓冲区中取得，如图 12.6 所示。

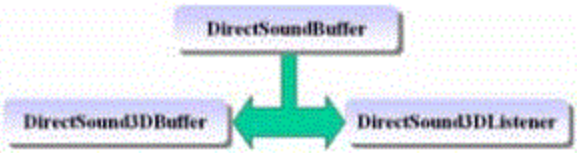


图 12.6　从 2D 缓冲区中取得 3D 缓冲区对象与 3D 倾听者对象

3D 音效在播放时有几种可能会发生的情况，一是音源不动而倾听者移动，二是倾听者不动而音源移动，三是音源与倾听者一起移动。这也是为什么必须取得 3D 缓冲区与 3D 倾听者的原因。3D 缓冲区就相当于音源，而 3D 倾听者则表示玩家相对于音源的位置，我们将前两种可能的情况用图 12.7 来表示。

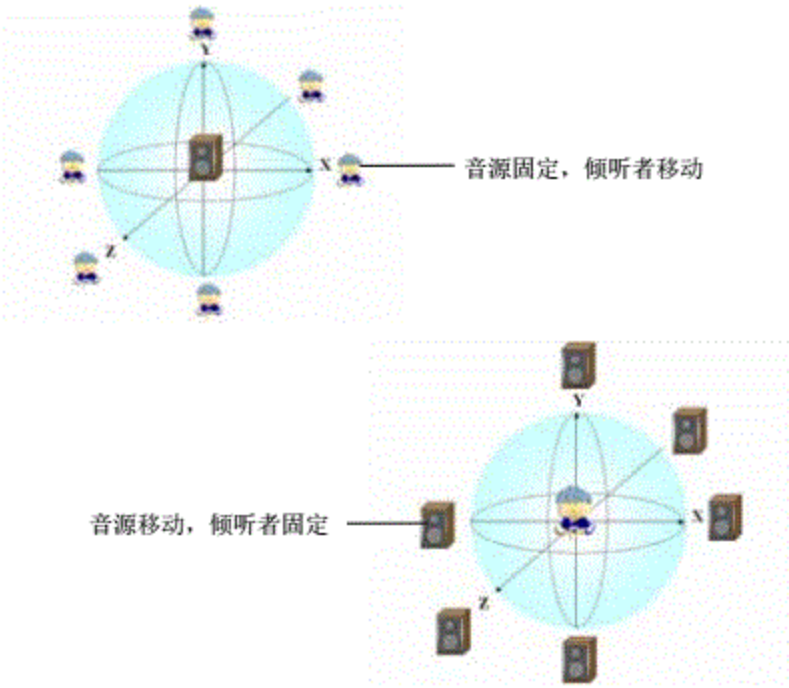


图 12.7　音源与倾听者之间的关系

图 12.7 中，显示了 3D 音效的范围可能为立体空间中任何一个位置，且也显示了程序设计时坐标的定位方式。而事实上如果您要使用 DirectSound 进行音效程序设计时，您会发现3D缓冲区对象与3D倾听者对象其实可以是相同的对象，它们所拥有的设置几乎相同，但事实上 3D 缓冲区并不从事音效的播放与控制，而是用来设定位置与声音传播范围等相关信息，在您设定这些信息之后，播放的操作仍然交给 2D 缓冲区对象来执行。

在播放 3D 音效时，我们知道如果位于音源正面所听到的声音会较大，而位于音源后

方所听到的声音会较小，而越远的地方声音越小，在 DirectSound 中我们则以声音传播范围来进行设定，一个声音传播范围的示意图如图 12.8 所示。

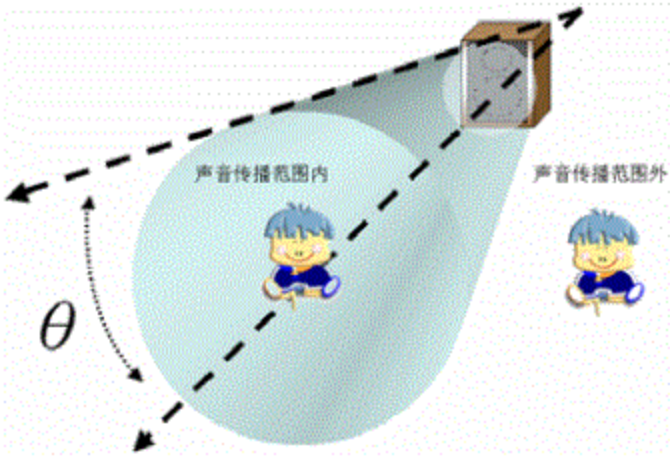


图 12.8　声音传播范围示意图

当倾听者位于声音传播范围内时，所听到的声音会比较清楚，如果倾听者位于声音传播范围外时，所听到的声音会比较小，而声音也具有一定的扩散性，所以我们可以利用声音传播范围的角度来设定。在 DirectSound 的设定中，还可以设定内层声音传播范围与外层声音传播范围。内层声音传播范围所听到的声音会比外层声音传播范围所听到的声音大，外层声音传播范围也可以设定它的音量，而如果倾听者位于外层声音传播范围之外，则表示完全听不到声音。

为了让您能体会 3D 音效的效果，我们制作了一个 3D 音效的示范程序，您可以执行程序 3DSound.exe 来倾听每一个不同位置所得到的结果（您的计算机必须安装有 DirectX），如果您的声卡只支持二声道，则使用耳机会比使用喇叭的效果更逼真。如图 12.9 所示为程序执行时的参考画面。

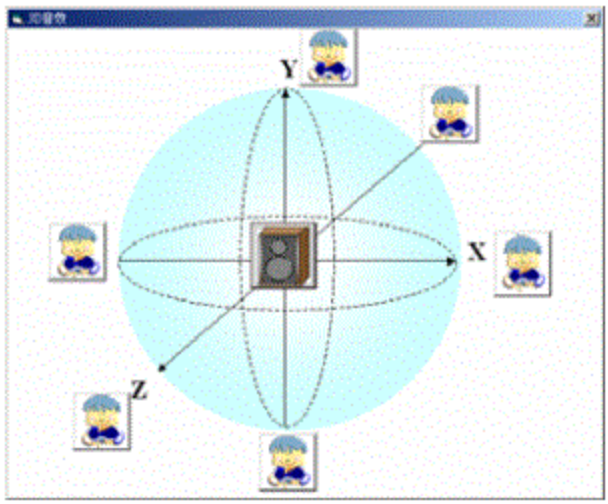


图 12.9　您可以使用此程序来了解 3D 音效的效果

12.1.4 DirectMusic 对象成员

DirectMusic 是在 DirectX6 之后才加入到 DirectX 家族之中的，它专门用来播放 Midi 格式的声音文件，并具有软件音效合成器，可以将 Midi 模拟成 Wave 文件，而使得播放时的音效更为美妙。甚至达到 Wave 文件播放时的效果，由于使用软件音效合成器，所以即使玩家的声卡是古董级的声卡，同样也可以享受高品质的音效合成效果。

其实 DirectMusic 很像是个音效合成器对象，因为它仍会使用到 DirectSound 对象，只不过这个动作被隐藏起来而由程序自行处理。所以设计程序时并不会直接进行声卡对象的动作设置，如图 12.10 所示为 DirectMusic 对象成员之间的关系图。

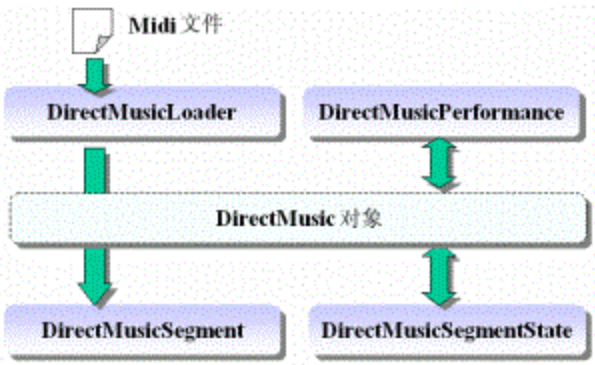


图 12.10　DirectMusic 对象成员

DirectMusicLoader 是用来载入 Midi 文件，文件会交给 DirectMusicSegment 对象，一个 DirectMusicSegment 相当于 DirectSound 中的缓冲区对象，一个 DirectMusicSegment 对象表示一个 Midi 文件，当我们使用 DirectMusicPerformance 进行音效播放时，必须指定要播放哪一个 DirectMusicSegment 对象。

事实上您连 DirectMusic 对象也不会接触到。因为我们在进行程序设计时，会先从 DirectMusicPerformance 对象开始，此时就会自动建立一个 DirectMusic 对象，这个对象是隐藏的，我们并不会直接接触。无论是音效的播放、停止都会通过 DirectMusicPerformance 对象进行操作，播放的时候会传回一个 DirectMusicSegmentState 对象，当中会包括目前档案的播放信息。

使用 DirectMusic 也可以同时播放多个 Midi 文件，而每一个 Midi 文件必须要有自己的 DirectMusicSegment 对象与 DirectMusicPerformance 对象；为了让您了解 DirectMusic 的音效执行效果，我们制作了一个简单的演示程序，您可以尝试进行音效的合成，程序名称为 MidiPlayer.exe（计算机必须安装有 DirectX 程序），如图 12.11 所示为程序执行时的参考画面。

图 12.11　您可以利用此程序体会 DirectMusic 的效果

关于音效播放的介绍在此先告一段落，我们要着重介绍的是一些通用的概念，而关于程序的设计则依所使用的程序语言而有所不同，其基本的播放操作方式是相同的，实际上您只要配合游戏执行时的逻辑适当加入音效即可，这并不会太难。

在下一节中，将介绍游戏进行中的一些操作装置，包括键盘、鼠标与摇杆操作，我们从程序设计的角度来介绍，这些已经熟悉的操作装置的工作原理。

12.2 操作装置

一般游戏可操作的设备有 3 种：键盘、鼠标与摇杆。或许您立刻想到一些特殊的装置，例如枪枝造型的操作装置、球拍造型的操作装置，甚至于推杆造型的操作装置（在进行撞

球游戏时使用），然而在 DirectX 的成员 DirectInput 对象看来，这些装置仍然被归类为摇杆的一种。这有个好处，因为您无法预测玩家会使用什么样的装置，我们只要对 DirectInput 对象进行设置，至于它要如何控制不同的装置则无需去费心。

12.2.1 键盘与鼠标

DirectInput 对于操作装置是以“轴”与“按钮”来定义，它将操作装置分为 3 类：“键盘”、“鼠标”与“摇杆”。键盘即是计算机标准的输入装置，而各类鼠标（无论是使用轨迹球或光学鼠标）、数字板或触摸屏幕，都归为鼠标类操作装置，至于其他的装置，则都归为摇杆类操作装置。

键盘不像鼠标或摇杆具有方向性，所以它属于没有轴的操作装置，而键盘基本上具有 101 个以上的按键，所以键盘在 DirectInput 的定义上，就属于“无轴”、“多按钮”的操作装置。使用 DirectInput 来操作键盘比使用 Windows 事件处理来操作键盘拥有更多的优点，由于 DirectX 的成员都可以直接存取目的装置，而无需通过 Windows 的信息，所以使用 DirectInput 直接存取键盘装置，会比使用 Windows 事件处理反应快。对于一些需要高速反应操作的游戏（例如实时的 3D 格斗游戏），就会使用 DirectInput 来进行键盘操作，而另一个好处是可以运用更多的组合键，以完成更多的键盘组合操作。

一般的鼠标通常具有 2~3 个按键，由于鼠标是利用轨迹球的滚动或光线的相对位移（指光学鼠标）来决定其移动量，所以鼠标的操作具有方向性，但是鼠标的移动并没有原点依据，它所采用的是“相对轴”，所以在 DirectInput 的定义中，鼠标属于“多按钮”、使用“相对轴”的操作装置。

在 DirectInput 中对于鼠标移动是使用具有 4 个成员的结构来表示，其成员包括 x、y、z 与 buttons，其中 x、y 分别表示 X 轴与 Y 轴的移动，而 z 则表示移动量，buttons 是个 0～2 的数组，用来表示鼠标左、右与中键。 DirectInput 并不使用 Windows 的信息产生事件，使用 DirectInput 来进行操作装置的管理有两种方式，一种是直接取得装置的状态，另一种是设定缓冲区。两者各有其优点，实时数据可以读取装置的实时信息，但在程序忙碌时可能读取错误的操作或遗漏了信息，而使用缓冲区的话，每个操作信息都会被存储在缓冲区中。而程序再由缓冲区中读取，如此操作信息不会遗失，但您必须自行处理缓冲区，缓冲区过大则会导致操作有延迟的效应，缓冲区过小则操作信息仍有可能被遗漏。

12.2.2 摇杆

许多摇杆是专门为游戏而设计的，因此并没有所谓的标准摇杆。在 DirectInput 中对摇杆的定义为“多轴”与“多按钮”的操作装置，无论该摇杆长的多怪异，操作方式多复杂，都将其归于轴与按钮的操作，在 DirectInput 中定义有两种摇杆类型，一种为 6 个轴 32 个按钮的一般摇杆，一种为 24 轴 128 个按钮的新型摇杆。

一般来说，我们进行的 2D 或 3D 游戏，使用的摇杆多为支持 6 个轴的一般摇杆，这 6 个轴主要是 3D 立体中的 X、Y、Z 轴，以及绕这 3 个轴的旋转轴，如图 12.12 所示。

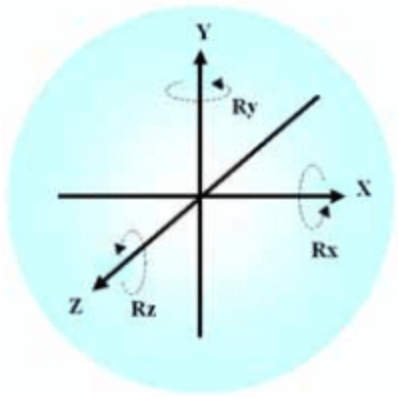


图 12.12　6 轴一般摇杆的轴判断方式

在图 12.12 中旋转轴的方向是使用右手来判断，姆指指向直角坐标轴的正方向，而其余 4 指就是旋转轴正方向，不同摇杆对这 6 个轴的设计位置可能不相同，以我们手上的摇杆为例，它的轴方向如图 12.13 中定义。

图 12.13　一个摇杆可能的定义方向

在图 12.14 中我们将滑杆拿来作轴的操作，除了轴与按钮之外，摇杆上还会设计有准星帽（Point.of.View，简称 POV），轴操作要定义在滑杆或是准星帽上可以由玩家自行切换，至于切换的判断则是由摇杆的驱动程序完成，在程序设计时只要专心于轴与按钮的操作即可，而不用理会到底玩家的摇杆如何设置，这也就是为什么玩家的摇杆即使奇形怪状，我们仍无需担心的原因。

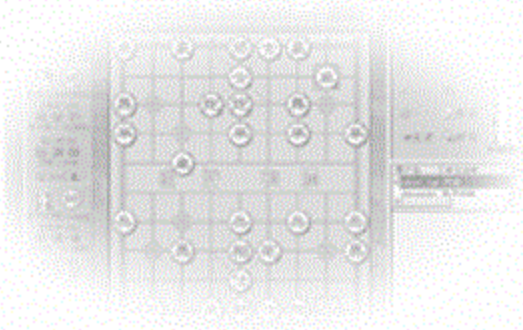
图 12.14　各种类型的摇杆

至于游戏设计时采用何种装置，则视游戏的类型而有所不同，使用键盘或鼠标操作是标准的选择，如果游戏在使用摇杆时会是比较好的选择时，您也只能站在建议的角色。因为玩家不一定会拥有摇杆，通常最好是进行侦测，如果侦测到摇杆已连接，则提示玩家使用摇杆，否则的话就预设使用键盘或鼠标。

基本上摇杆的需求不高的话，所需的摇杆价格通常不高，基于营销的角度来看，也可以在游戏包装时附赠购买者摇杆，如此不但可以吸引购买者，若玩家在游戏过程中因方便的操作装置而更能融入其中的话，对游戏产品本身也有正面的帮助。

第 13 章

团队合作



在开发一套大型游戏时，必须要由许多人通力合作来共同完成。对于一家公司来说，一个部门会有许多人加入开发游戏，而各部门与各部门之间也必须相互协调合作。本章将探讨一个游戏开发团队应该要如何来合作与沟通。

13.1 开发团队的任务

一款游戏的成功与否，其关键在于一个决策者对于游戏开发团队任务的指派。一般而言，我们会先将一个开发团队的人物角色分配到“最恰当”的状况，不过基于成本的考虑与人力资源的不足，有时候可能会由一个人来扮演很多任务的角色，或由某一些人来扮演跨越任务的角色。不管怎么说，一套游戏的开发团队就必须要存在某些特定的角色成员，至于游戏开发团队的任务到底是什么呢？而任务中的角色应该要如何来扮演呢？请看我们细说分明。

13.1.1 团队的任务角色

尽管人力资源的不足或成本不能负荷的情况下，我们还是可以将这个游戏开发团队的任务分成五大类，而这五大类的任务则足以应付游戏开发上的各项细节流程。其五大类任务的分类如表 13.1 所示。

表13.1 任务分类与主要角色的关系

任务分类	主要角色
管理与设计	系统分析
	软件规划
	策划管理
	游戏设定
程序设计	程序总监
	程序设计
美术设计	美术总监
	美术设计
音乐创作	音乐作曲家
	音效处理员
测试与支持	游戏测试
	支持技术

其人力资源的分配与任务规划的指派可以将这五大类的分类架构绘制成如图 13.1 所

示的金字塔形状。

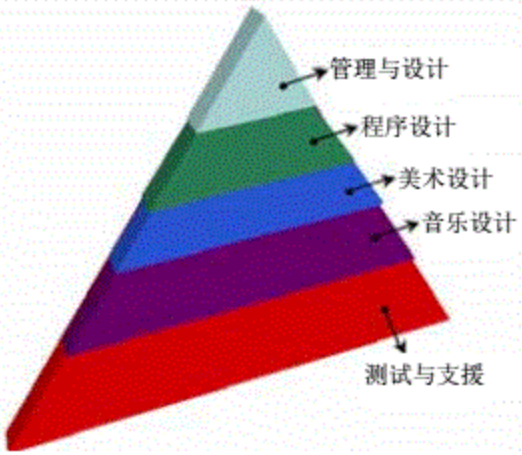


图 13.1　人力资源与任务规划分类架构图

如同前面所讲述的一样，开发团队中有些任务的位置上可能会安排某一些人，或者是有些任务的位置被空荡荡地放置在旁边，或者有些任务位置的席位可能会由许多人来扮演，甚至有些人会担任两个不同的角色。实际上如果没有严格的规范时，上述这些情形还是可以被接受的，不过在这种情况下，某些角色就必须扛起其他沉重的工作。

以早期的游戏制作来说，游戏设计者可能是由一个人来扮演起这五大类的任务角色，尽管是在策划游戏、程序设计、美工设定、音乐创作，甚至于连测试的工作都只是由个人来完成，不过我们还是将这种情况只能局限在制作小型游戏中。从现在的游戏格局来看，我们相信一款足以在游戏产业市场上生存的游戏产品，它不是由一个人就可以完成的任务，虽然在制作游戏的过程中，可能还有许多不同的因素要考虑，不过在人力资源与开发成本的限制上，我们还是建议在开发游戏之前，先建构起这个开发团队的金字塔，这样才是游戏制作的最佳途径。

13.1.2 游戏的主要灵魂角色

游戏最主要的灵魂是可以管理一套游戏的制作流程与设计一系列项目的管理设计人员，它的主要工作任务就是在于管理所有的团队人力资源、建构游戏中整体上的基本架构，以及设计编写游戏中的所有细节重点。

在一家有规模的公司里，游戏的灵魂角色通常是由公司中一个可以统筹整体游戏规划的人员来扮演，他是一套游戏的精神来源与一组游戏开发团队的最终向心目标。在一套游戏的开发过程中，管理者必须要在技术性的管理职位上具备起相当有技巧性的管理或训练

团队的能力，有时候他也必须要扮演起开发团队部门之间“和事佬”或者是“决策者”的身份，并且能够善于运用各种人力资源，达到开发游戏团队能力的最顶峰。

但很不幸的是，这种理论在一般我们所了解的游戏工作团队中，是很难看到这种人的存在。因此这种角色通常便由一个程序设计师来扮演，这可是一件非常危险的事。一般来说，游戏开发团队中最容易迷失自己的角色便是程序设计师。如果程序设计师一经迷失方向的话，他即会将整个游戏开发团队推向一个无底的研发深渊当中，这是非常可怕的。更何况一个程序设计师通常会因为太过于精于自己的技术，因而没有考虑到整体的人际关系与成本进度，最后可能会导致游戏开发团队的士气低落与成本不断地超出预算。

基于上述的说法，我们可以了解到一个游戏开发团队就必须要有一个具有管理、沟通与训练能力的领导核心。

13.1.3 游戏与设计者之间的桥梁

在一个游戏开发团队中，其工作压力与心理问题最大的就是程序设计师，它也是最会抱怨与烦燥的一种角色。通常在策划人员想要将游戏设计得尽善尽美的境界时，程序设计师就必须要花上大把的时间来实现策划人员的构思，而且在一个没有管理者的开发团队之中，程序设计人员便会更容易迷失自己，更何况一个游戏开发团队可能还会拥有两个以上的程序设计师，那不就更加天下大乱了吗？其实程序设计师的任务性质相当单纯，他只要由决策者与设计人员所规划出来的策划书来开发其应用程序就可以，而其他琐碎的事情应该是交由管理者或决策者来处理或解决。

谈到这里，相信您也发现了一个严重的问题，那就是开发一套游戏就必须要有这么多的程序设计师，如果他们的意见又不合时，那又要怎么办呢？我们都知道，人与人之间难免会有一些意见不合或冲突的时候，基于这一个理由，我们可以在这些程序设计群中，推举一个可以管理众人的“总监”角色。

以程序设计师来说，“总监”这个角色占有极为重要的地位。如同前面所说的一样，程序设计师是一个非常难以管理的族群，因为管理者要去考虑到整个开发团队的士气与默契和一些琐碎的事情，所以管理者根本没有办法再去管理这些个别的程序设计师，因此管理者就必须从这些人当中推选一个可以帮他管理这些程序设计师族群的人，通常这个角色是程序小组中技术最好的一个，而且他还必须要有将程序全面整合化的能力，简单地说，这个程序总监的角色，对上要以管理者的决策为主，对下就必须要有管理程序设计小组与整合程序的能力。

13.1.4 创造出游戏美感的艺术家

对于一个游戏开发团队来说，其工作性质与研发意见不像程序人员这么复杂，以其工作性质来看，美术人员只要非常单纯地绘出策划人员所要的画面与图素，而且在修改图像

时也不会占去开发游戏太多的时间（一般来说，会花上游戏开发太多时间的是程序设计的部分）。

美术人员就像是一个艺术家一样，他们只要去遵循策划人员所议定出来的主题，依照自己的想法来绘制游戏中的各种画面与图素，不过在这里，值得一提的是，一款游戏画面的完成是不能单靠一个美术人员的力量就可以搞定的，因此以一个游戏开发团队来看，美术人员的需求量就会相对地增加许多。而在这么多美术人员的管理规划前提之下，我们就必须从美术部门中，区分出一种与程序总监相等地位的“美术总监”来。

“美术总监”的管理工作性质与程序总监大致是相同的，惟一不同的是美术总监只要去统一游戏整体的绘制风格，且具有将各美术人员所绘制出来的图素整合起来的能力。

13.1.5 游戏音乐家

游戏开发团队中，工作性质算是最单纯的就非音乐（效）人员莫属了。游戏的声音部分，可以将其性质分成两种，一种是游戏中可以令人感动，甚至足以影响一个玩家情绪的音乐作曲家，另一种是创造出游戏中各式各样稀奇古怪声音的音效技术师。

音乐作曲家的工作就是创作出游戏中所有剧情演变的音乐，这是一件非常个人化的行为。简单地说，音乐作曲家只要做出可以附和游戏画面或剧情情绪的音乐就可以了。

在一套品质不错的游戏中，我们可以发现它在游戏中一些小细节的音效足以影响我们对游戏的评价，虽然没有了这些音效也不会影响到一款游戏的品质，不过它却可以提升玩家对游戏的观感。举例来说，当我们在玩一款以恐怖为主题的游戏时，如果没有听到一些可怕的声音，似乎就不怎么刺激了。不过如果在游戏中放上一些诡异的风吹声，或是一些踏在腐朽的木板上所发出的嘎嘎声，如此一来，无形中便增加了游戏的恐怖临场感，而音效技术师便是这些听了令人毛骨悚然音效的创造者。

13.1.6 测试与支持

测试与支持的分类是一种不需要具有特殊专业能力的人员所构成的，其工作的性质是在于帮忙测试游戏的优劣性与错误。在游戏制作的初期，策划人员可以请程序设计师编写一个较为简单的测试软件来提供测试人员测试游戏用，这些人员在游戏制作初期，人数是最少的。不过在游戏制作完成的距离越近，这些测试人员的人数也就会相对的越来越多，目的就是要让游戏编写人员了解到更详细的错误信息。在这个分类中，这些人也具有一种支持的特性，其支持的特性就是在于游戏工具的操作与计算机硬件设备的维护，使得公司的运作可以更顺畅。

在游戏的开发制作过程中，这些人员的分配是一门很大的学问。而且每一个人也都有自己的情绪与意见，这些不同的意见与看法如果不仔细解决的话，那么它们势必会演变成日后问题的冲突点。一经爆发，势必会影响到整个游戏开发团队的士气与精神，而一个成

功的管理者或决策者就必须要在这里面去学习经验，从经验中总结教训，以免日后会重蹈覆辙。

13.2 良好的工作环境与士气

一个游戏开发团队最为核心的主力是在于每一个人员身上所秉持的理念、精神与士气，而借着游戏开发团队的理念和良好的工作环境，它便可以造就出一个工作团队的精神与士气。所以我们可以将这种良好的工作环境，再加上一个良好的士气，其结果再与优质的产品划上一个等号。

良好的工作环境+良好的士气＝良好的产品

13.2.1 良好的工作环境

“工作环境”在现实生活中，指的是一个工作的场所，不过笔者在这里所要谈到的工作环境并不是一种地方或场所，而是一种无形的族群关系。再解释清楚一点，它就是一种人与人之间相互信任的关系。

在人与人之间可以互相信任的情况下，工作默契与士气便会在这里慢慢地形成了，我们相信如果没有这种良好的工作环境，人与人之间就会发生一些不必要的摩擦，而这些小小的摩擦甚至于可以毁掉整个策划及团队的士气与精神。这种道理也可以解释在公司与员工之间，一个可以信任员工的公司，在不影响工作进度及成本的考虑之下，甚至可以放手让员工自行去发挥，而员工在这种没有被绑死的状态之下，更能够将工作发挥到淋漓尽致。

13.2.2 士气的提升

一个游戏开发团队的士气足以影响到游戏本身开发的进度与品质。在这里，笔者要提供您一个特别的观念，那就是士气的提升并不需要利用到每一个游戏开发人员的冲动上。事实上，这种做法不但不能提升士气，而且会将游戏开发人员给宠坏，就如同一个什么都可以得到的小孩一样，你要他不要捣蛋似乎是一件很难的事。

举例来说，由于游戏开发人员都非常喜欢以自己的行为模式来做事，如果一个管理者或是决策者以他这种行为来当成对别人的榜样时，这个游戏开发团队便会很快地被瓦解了，或者是某一些人会认为如果让他们这样的做的话，公司就必须要付更多的金钱代价，不然他们就中止工作。这对于一个游戏开发团队或公司来说，它是一种相当大的打击，甚至在游戏还未开发完成之前，游戏团队就在这个战场上“伤的伤”、“逃的逃”了。如果不改善这种问题，就算是换上一批新血，最后还是会发生同样的问题。我们相信能够维持良好的

游戏团队士气就是以一种公平的心去对待团队中的每一个人。

笔者曾经看过一个例子，这个例子是在描述两个人在一家公司工作的事，这两个人同样呆在一家公司里工作，A 君是 B 君的上司，有一天 A 君与 B 君同样接到公司的一项任务，并且交待由他们两个人去做，A 君与 B 君在接到这项任务的时候，A 君以是 B 君的上司为由，吩时 B 君要全力以赴地去做，而自己却落得空闲，B 君心想：“反正是为公司做事嘛！算了！”几天下来，B 君将这一项任务做出了相当不错的成绩，在这时候 B 君心想 A 君毕竟是自己的上司，所以他认为这项任务的成绩还是先让 A 君了解一下比较好，B 君就这样将任务的成绩全部向 A 君报告，想不到 A 君竟然拿着这样的成绩去向公司报告，而公司又对 A 君这种工作态度赞赏有佳，甚至于在 B 君的耳边说为了提升 A 君这种工作士气，公司决定增加 A 君的薪资收入，B 君听到真是快要气炸了。事情过了不久之后，B 君便离开了这家公司，而这家公司在过了几年之后因业绩不彰而倒了，最后听说 A 君还继续在别家公司里做着相同的事情。

从上面的例子来看，一个成功的决策者绝对不会单单以一个人的行事模式来评断一个开发团队的士气，毕竟每一个人的生活方式都不一样，有的人喜欢这样、有的人却不喜欢这样，所以成功的决策者会在开发团队中取得一个公平的待遇与机遇来提升开发团队中的士气。

13.2.3 工作时间

在游戏产业中，形成了一股特有的风潮，那就是工作团队都没有一个固定的工作时间，甚至于彻夜不睡觉等奇特的现象。

其实如同这种不合常理的工作时间看来，隐约地看到它是一种时间安排不良的后果。而工作团队工作的时越久，它所消耗的士气与精神也就相对地增加，等到士气与精神跌到谷底之后，那么这个工作团队只有到崩溃的阶段了。这种不良的现象也就衍生出另外一种情况，那就是有些人是早上看不到人，等到晚上才看得到他的人，如果游戏的设计或程序一发生问题时，这个工作团队总是不能将所有的人集合起来处理，导致游戏开发的进度会严重的落后，而且这些人的脸永远看起来似乎都是以一种“没睡饱”的精神来面对工作伙伴，这不仅仅影响到了团队的士气，而且它是一种浪费成本与人力的现象。

我们认为严格管制员工的工作时间，这也可以减轻员工白天所受的工作压力。并且在有足够的休息时间之后，第 2 天便会更有精神地面对任何的挑战。

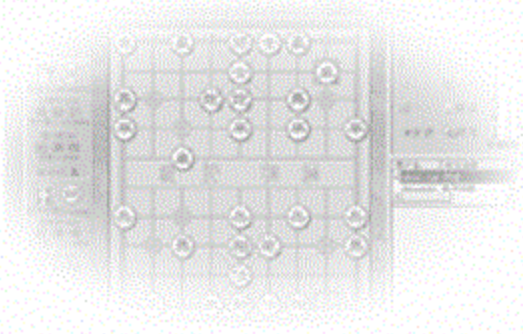
综合上述的种种问题因素，我们可以看出，一个成功的游戏开发团队在于其工作伙伴上，必须要取得它之间相互信任的态度，在开发团队的士气上也必须严格考虑，不要把个人的看法与行为当作是管理一个开发团队的标准，那是非常不理智的。

其实对于一个游戏开发团队来说，最难扮演的就是管理者或决策者的角色，这个角色必须要会去掌控游戏的开发时间、抓住工作团队的心、提升工作团队的士气、严格控管工

作团队的行为模式、尽量避免过度的浪费人力、严格考核开发的制作成本，以及建立起各部门之间人际关系的桥梁，以免除一些游戏开发过程中不必要的风险。

第 14 章

引擎的开发



在游戏的开发过程中，程序代码的开发进度最难以掌控，而且最耗时间。如果我们想要掌控整个游戏的程序代码的开发进度，最好的方式就是在于游戏引擎的延续性。本章将介绍游戏引擎的开发历程与游戏引擎未来的发展及延续性的问题。

14.1 游戏引擎简介

当我们在阅读各种游戏介绍的时候，经常会遇到“游戏引擎”（Game Engine）这个特殊的名词，而游戏引擎在游戏中到底扮演什么样的角色？它对于游戏未来发展到底会产生哪些影响呢？在本节中，很快为您揭晓。

14.1.1 什么是游戏引擎

在现实生活中，“引擎”是一种牵动车子的一个重要机械功能。我们都知道，引擎可以称得上是车子的心脏，它可以用来决定车子的稳定性和特有的性能，而车子的行驶速度与驾驶者操纵的优越感都必须建立在引擎的基础上。在游戏中，引擎的地位也是如此，玩家在游戏中所体验到的剧情、角色、美工、音乐、动画及操作方式等内容都是由游戏的引擎直接所控制的，所以游戏引擎可以说是游戏中在扮演发动机的角色。我们将游戏中所有的元素当成是舞台上的演员，而游戏引擎的责任就是在后台指挥着这些演员进行有秩序地工作的角色，换句话说，游戏引擎就是用在控制所有游戏功能的主程序。

游戏引擎主要的工作是进行游戏中的物理演算、碰撞运算、物体成像、玩家角色的操作，以及播放正确的音量和声音输出等必要的功能。其实不管是在2D或3D的游戏上，还是角色扮演、实时战略、冒险解谜、动作射击等不同类型的游戏上，它们都必须通过这种控制作用的程序代码才能运作，而这些程序代码则是游戏引擎的主要核心。

在游戏引擎经过多年来的不断演进变化，如今的游戏引擎已经被发展成一种由许多子系统所共同构成的复杂框架系统，其系统从建立模型、画面成像、行为动画、光影处理、分子特效、物理演算、碰撞侦测、数据管理、网络联机，以及其他专业性的编辑工具与套件等等，它几乎可以涵盖整个开发过程中所有的重要环节，使得游戏可以得到更好的华丽画面与流畅度。接下来，将介绍游戏引擎具有哪些重要的关键环节。

14.1.2 牵引着引擎的最重要环节

与上面所说的一样，游戏引擎在游戏中必须处理一些复杂的演算与运算，而这种复杂的演算与运算都会直接影响到游戏的品质与流畅度，换句话说，一套成功的游戏，背后就

必须要有一套功能完整的游戏引擎来支持。下面是游戏引擎中极为重要的几个环节。

光影处理

首先来介绍游戏引擎中“光影处理”的环节。光影处理即是用来处理游戏场景中光源对游戏中的人、地、物所影响变化的效果。游戏中的光影效果完全是依靠游戏引擎来控制的，例如光线折射与反射等基本的光学原理，以及动态光源、彩色光源等复杂的光学效果，这些都必须通过游戏引擎的程序技术才得以实现。

行为动画系统

目前市面上的游戏所采用的行为动画系统可以将它区分成下列两种：

- 骨骼行为动画系统

骨骼行为动画是利用内建的骨骼数据来带动物体而产生行为运动，这种行为动画系统比较节省计算机系统的资源。

- 模型行为动画系统

模型行为动画则是在模型的基础上直接进行行为动作的变化。而游戏引擎则是将这两种行为动画系统植入到游戏中，使得动画师可以方便为角色设计一些丰富的动作造型。

物理系统

游戏引擎的另一项重要的功能就是它可以提供物理系统的演算，这种物理的演算可以让物体在运动的时候遵循某些特定的规律。例如当玩家所操作的角色跳起来时，游戏引擎内定的重力值就可以决定其角色能跳多高，以及角色在落下时的速度会有多快。另外如同子弹的飞行轨迹、车辆的行驶运作等等，这些也都是由游戏引擎中的物理系统所决定的。

碰撞侦测

碰撞侦测是由游戏引擎中的物理系统所分化出来的，它是用来侦测游戏中各种物体的物理边缘是否碰撞在一起了。举例来说，当两个物体碰撞在一起的时候，这种碰撞侦测的技术可以避免它们相互穿透，如此一来，便可以确保当游戏中角色撞到某个物体的时候，其角色不会穿过这个物体，而且这个物体也不会依附在角色的身上，因为碰撞侦测会根据物体与物体之间的特性来确定两者的位置和相互作用的关系。

画面成像

画面成像也是游戏引擎最重要的功能之一，当我们将游戏中所有模型制作完毕之后，美工人员会依照角色中不同的面，将特定的材质贴在模型上，最后再通过游戏引擎中的画面成像技术将这些模型、行为动画、光影及特效等效果在系统中运算出来，并且将运算的结果显示在屏幕上。画面成像在游戏引擎的所有环节中最复杂，而且它的运算结果是可以直接影响到我们最后所输出的游戏画面，所以画面成像的技术在游戏引擎中，它实为最重要的一环。

网络与输入

游戏引擎还有一个重要的任务，那就是它必须负起玩家与计算机之间沟通的责任。尤其游戏的输入是来自玩家的键盘、鼠标、摇杆或其他外部的输入信号，所以游戏引擎就必须要包含处理玩家输入信号的技术。另外如果某些游戏还支持网络机制的话，则网络功能也会被包含在游戏引擎中，功能就是用来管理服务器端（Server）与客户端（Client）之间的网络通讯。

经过上述这些枯燥乏味的介绍以后，相信您至少可以了解到一个观念，那就是游戏引擎相当于游戏中的一个框架，而这个框架在建立完成之后，所有游戏相关人员只要通过这个游戏引擎便可以将所有的对象数据合并在一起了。因此在游戏的开发过程中，游戏引擎的制作过程通常会占用非常多的时间，以国外的例子来看，他们在制作游戏引擎的时候，通常会花掉发展整套游戏$^{3}/_{4}$的时间。相对的，他们所花费掉的成本也非常可观，甚至有的人会说：“如果当初意识到制作游戏引擎要付出这么大代价的话，我们根本就不可能去做这种傻事。因为没有人可以预料到几年之后的市场究竟是如何，这个游戏引擎到底还能不能用。”

正如上述所说的一样，如果要开发一套游戏引擎的话，我们就必须针对“研发成本”、“开发周期”和“降低风险”这三方面来考虑。正因为如此，现在的游戏业者也因此而越来越偏向两种趋势，一种游戏业者是完全投入游戏引擎的开发，另一种是游戏开发者会向第三者购买现成的引擎来制作自己的游戏，基于这两种趋势，从游戏产业来看，游戏引擎的开发便形成了另外一个独特的游戏市场基础。

14.2 游戏的引擎进化发展史

在游戏产业发展最辉煌的时期，对于每一家游戏厂商而言，它们几乎都只关心要如何尽量多开发出一些新的游戏，并且费尽心思要将这些游戏推销给玩家。尽管当时大部分的

游戏都显得有点简单而且粗糙，但是每一款游戏平均所开发的时间最少也要长达到八、九个月以上，一方面受到当时成像技术的限制，另一方面也因为每一款游戏几乎都要从头编写新的程序代码，以至于造成了大量的程序代码不断地重复产生。

在游戏产业的发展过程中，慢慢地，有一些经验较老道的游戏开发者就开始研究一种较为“偷懒”的方法，那就是延续了上一款类似题材游戏中某些部分的程序代码来作为新游戏的基本框架，如此一来，便可以节省更多的开发时间与成本了。在国外，有人曾经这么说过：“单位产品的成本因生产力水平的提高而降低，自动化程度越高的手工业者，最终会将那些生产力较低的手工业者给淘汰出局。”同样道理，游戏引擎的概念就是在这种机械化作业的背景下诞生的。

14.2.1 游戏引擎的诞生

每一款游戏都有属于自己的引擎，不过一套游戏的引擎要能够真正获得其他人的肯定，并且成为尔后制作游戏的一项标准，实在是不多。游戏引擎在历经十多年的发展历程中，我们可以了解到其实推动游戏引擎最大的功臣是来自于3D的游戏，尤其是第一人称的3D射击游戏。尽管2D引擎也有着相当久远的历史，例如“Planescape：Torment”（“异域镇魂曲”）、“Baldur's Gate”（博德之门）、“Icewind Dale”（冰风之谷）等等，但是上述这些游戏引擎的应用范围毕竟还是被局限在“龙与地下城”风格的角色扮演游戏中，虽然它们各自都有着独特的玩法与目的，但是这对于整个游戏引擎的技术发展而言，却起不了什么作用，这也就是为什么较为单纯游戏类型的引擎很少进入引擎授权市场的原因了，如飞行类游戏、体育类游戏、实时策略游戏等等，即使游戏开发者利用第三方的引擎也很难从其中获得游戏中理想的效果，例如，以“Age of Empires”（世纪帝国）的引擎来制作“Star Wars：Galactic Battleground”（星球大战：帝国战场），那不就像是“牛头不对马”了吗？(一个是2D游戏，一个是3D游戏)。

因此，游戏引擎的历史主要就围绕在动作射击游戏的类型上。我们可以说动作射击游戏与3D引擎之间的关系，就好像是一对孪生兄弟一样，它们一同诞生，一同成长，并且又可以互相为对方提供各方的发展动力。

14.2.2 引擎对于游戏的冲击

严格地说，游戏引擎诞生于公元1992年。在公元1992年的时候，3D Realms公司与Apogee公司公布了一款只有2M（2048 KB）的小游戏——“Wolfenstein”（德军总部），如果您是一个经验老道的玩家，那么您一定还记得刚接触到“德军总部”的那种兴奋心情吧！在这里，我们可以以“革命性”这一个极为煽动色彩的名词来形容“德军总部”在整个计算机游戏发展史上所占据的地位。

“德军总部”这款游戏开创了第一人称射击游戏的大门，更重要的是，它是首推在游

戏中以X与Y轴的基础上再增加了一个Z轴坐标，由宽度与高度所构成的平面上增加了一个向前与向后的深度空间，这个具有Z轴的游戏画面对于一些习惯于2D平面游戏的玩家来说，那可是一股巨大的冲击啊！

“德军总部”的3D引擎作者就是在“ID Software”公司的首席程序设计师——“John Carmack”（约翰卡马克），他正是借着这款Wolfenstein的3D引擎而在游戏界里站稳了游戏引擎的地位。事实上，“德军总部”并非是第一款采用第一人称视角的游戏，在“德军总部”发行的前几个月，Origin公司就已经推出了一款第一人称视角的角色扮演游戏——“Ultima Underworld”（地下创世纪），这款游戏是采用了与“Wolfenstein”类似的技术，但是它与Wolfenstein的3D引擎之间却有着相当大的差别。

举例来说，“地下创世纪”的引擎里有支持斜坡的运算，意思就是说在游戏中，我们可以看到地板与天花板有着不同的高度，使其可以区分出地形上不同的层次。而玩家又可以在游戏中进行跳跃的动作，也可以在游戏中做出抬头或低头的动作，这些特性是Wolfenstein的3D引擎无法做到的部分，而且从画面上来看，“德军总部”比较接近于漫画的风格而不是传统的图形画面。

从成像技术的层次上来看，Wolfenstein的3D引擎的确是比不上“地下创世纪”的引擎，但是Wolfenstein的3D引擎却更充分的表现出第一人称视角的特点，而且游戏中的快速火爆节奏，使得玩家一下子就记住“第一人称射击游戏”这个新名词了。继“德军总部”之后，“ID Software”公司又发行了另一款名义上称为“德军总部”续集的——“Rise of the Triad”（龙霸三合会），这款游戏在Wolfenstein 的3D引擎基础上增加了许多重要特性，包括跳跃、抬头和低头的动作。

在游戏引擎诞生的初期，ID Software公司同样又发行了另一款非常成功的第一人称射击游戏——“Doom”（毁灭战士）。Doom的引擎技术则大大地超越了Wolfenstein的3D引擎。“德军总部”中的所有物体大小都必须是固定的，而且游戏中所有的路径角度都只能呈现直角，也就是说它只能在游戏中利用笔直方式将主角进行前进或后退的动作，不过，这些限制则在“毁灭战士”中都得到了突破。尽管“毁灭战士”游戏的关卡还是维持在单一的平面上，也就是说它没有“楼上楼下”的概念，不过它的引擎却可以在墙壁的厚度上做出任意的变化，而且路径的角度也可以设定成任何的角度，这使得尔后的楼梯、升降平台、塔楼和户外等各种场景变成了一种可能实现的技术。

由于Doom的引擎在本质上还是以二维的空间为主，因此它还是可以做到同时在屏幕上显示大量的角色，而且不会直接影响游戏的执行速度，而这个特点便创造出一种疯狂且刺激的动作游戏风格。到目前为止，“Doom”除了“Serious Sam”（重装武力）系列能够相比之外，相信没有一款3D引擎能够在同一个时间之内显示一大批敌人（玩家的画面中），而且游戏依然还可以保持一定的流畅度。虽然Doom引擎在游戏画面上缺乏了足够的细腻感，但它却能够表现出惊人的环境效果，其纯熟的设计技巧也是令人赞叹的。说到这里，不得不提的是，Doom引擎可称得上是在游戏业界中第一个被授权，且被用在其他游戏中的

3D引擎。接下来，我们就来介绍一下Doom引擎被其他游戏采用的过程。

14.2.3 游戏引擎被其他游戏所采用

在1993年底，Raven公司采用修改过后的Doom引擎，并且开发了一款名为“ShadowCaster”（投影者）的游戏，这也就是游戏史上第一个成功的将第三方引擎拿来开发自己游戏的技术。在1994年，Raven公司又采用Doom的3D引擎来开发“Heretic”（异教徒），在这款游戏中，它为Doom引擎增加了飞行的特性，并且成为尔后角色能够跳跃的前身。

而Raven公司在次年（1995年）又采用Doom的3D引擎技术开发出了另外一套游戏——“Hexen”（毁灭巫师），这款游戏又加入了新的脚本技术、音效技术，以及一种类似多线式的关卡设计，使得主角可以在游戏中不同的关卡之间自由移动。

ID Software公司在制作“毁灭战士”系列的时候，它本身在引擎与游戏市场上就获得相当大的成功，其游戏大约在全球卖了350多万套，而授权费又为其公司带来了一笔可观的收入。在此之前，游戏引擎只是作为一种自产自销的开发工具，而且从来就没有一家游戏厂商考虑过要依靠游戏引擎来赚钱，不过由于Doom引擎的成功，这无疑是它为游戏产业打开了另一片新天地。

14.2.4 游戏引擎的大转变

游戏引擎在发展变化过程中，在1994年，3D Realms公司开发了一款名为Build的3D引擎，而它也是一个游戏引擎重要的里程碑。3D Realms公司利用Build的引擎技术推出了一款家喻户晓的“Duke Nukem 3D”（毁灭公爵）。

“毁灭公爵”的游戏就已经具备了所有第一人称射击游戏的所有标准技术，例如360°环视、跳跃，以及蹲下和游泳等特性，此外它还将“异教徒”里的飞行特性换成了喷气背包，甚至它还在引擎中加入了角色缩小等令人耳目一新的技术。在Build引擎的开发过程中，它先后诞生过14款游戏大作，例如“Redneck Rampage”（火爆乡巴佬）、“Shadow Warrior”（影子武士）和“Blood”（血兆）等等，因此Build引擎也为3D Realms公司带来了好几百万美元的额外收入，3D Realms公司也由此而成为引擎授权市场上的第一个“暴发户”。不过从整体来看，Build引擎并没有为3D引擎的发展带来任何突破性的变化，而为3D引擎带来突破性的任务就落在ID Software公司的“Quake”（雷神之锤）身上了。

“雷神之锤”的游戏是紧跟在“毁灭公爵”之后所发售的，这两者的优劣性，一时之间便成为玩家所热烈谈论的话题。从游戏画面的角度来看，“毁灭公爵”的确超过了“雷神之锤”不少，但是以技术层面的角度来看，“雷神之锤”又更胜“毁灭公爵”一筹。Quake引擎是当时第一款完全支持多边形模型、动画和粒子特效的3D引擎，它所支持的效果不像是Doom和Build那样只有2.5D的效果而已，此外Quake引擎还是联机游戏的始祖，尽管在几

年前的“毁灭战士”也可以通过调制解调器来进行联机对战，但是最终将网络游戏带入大众化的则是“雷神之锤”的引擎。在Quake推出的一年之后，ID Software公司又推出“Quake2”（雷神之锤2）的游戏，这一举确立了ID Software公司在3D引擎市场上霸主的地位。

“雷神之锤2”采用了一套全新的引擎，这个引擎可以更充分地利用3D的加速效果与OpenGL的技术，在图形成像和网络方面也与前作有了更佳的效果支持。在“雷神之锤2”推出之后，更有许多第一人称上的游戏大作都纷纷利用了Quake2的引擎技术来研发，例如Raven公司的“Heretic 2”（异教徒2）和“Soldier of Fortune”（佣兵战场）、Ritual公司的“Sin”（原罪）、Xatrix公司的“Kingpin：Life of Crime”（黑街太保），以及ION Storm公司（雷神之锤之父John Romero离开ID公司之后成立的新公司）的“Anachronox”（时空传说）等等，它们都是采用了Quake2的游戏引擎技术。

在这里，我们就可以很轻易地算出Quake2的引擎为ID公司赚进了多少腰包，例如Quake2的引擎向各家游戏厂商所收取的授权基本许可费大约是50万美元到100万美元不等，而版税金则可以视基本许可费的多少而定，例如每一家游戏厂商的基本许可费为50万美元的话，那么ID公司大约会提取10%以上的版税金，如果基本许可费为100万美元的话，那么版税金就会相对地减少一些，依照这样算起来，Quake2的引擎将可为ID公司赚进至少有一千万美元以上，尽管“雷神之锤”游戏本身的销售量比“毁灭战士”还要来的差（大约卖了110多万套，其收入在4500万美元左右），但是它在授权金的盈利上就显然要远远高出“毁灭战士”许多，从此之后，游戏引擎已经从一种单纯的工具而变成了一块令人垂涎欲滴的肥肉了。

正当Quake2在独霸整个游戏引擎市场的时候，Epic游戏公司的“Unreal”（虚幻）就在这个时候问世了。虽然当时的“虚幻”游戏只能在300×200的分辨率下运行，不过它却可以呈现出相当惊人的画面与效果。

在游戏中，除了可以看到精致的建筑物之外，游戏里也可以看到许多出色的特效，例如宽广的场景、美丽的天空、荡漾的水波、逼真的火焰、烟雾和物理力场等效果。以当时的第一人称的游戏角度来看，“虚幻”游戏是当之无愧的佼佼者，它的震撼力完全与之前的“德军总部”可以相比拟。

“虚幻”的游戏引擎可是当时被使用最广的一款引擎，在它推出以后的两年之内，它就有18款游戏与其Epic公司签订了授权的协议，这18款游戏中还不包括Epic公司自行开发的“虚幻”数据片——“Unreal Tournament”（魔域幻境之武林大会），它将可以全面的取代“雷神之锤2”在第一人称射击游戏中的霸主地位。其中比较新的几部作品如第三人称动作游戏“Rune”（维京战神）、角色扮演游戏“Deus Ex”（骇客任务），以及第一人称射击游戏“Duke Nukem Forever”（永远的毁灭公爵），这些游戏都曾经获得不少好评。

其实我们可以说Unreal引擎的应用范围不被限制在游戏的制作上，例如软件开发商Vito Miliano公司也采用过Unreal的引擎开发了一套名为Unrealty的建筑设计软件，其软件是用在房地产上的展示。另外还有Digital Design公司就曾经与联合国的文教组织合作采用Unreal

的引擎制作过巴黎圣母院的内部虚拟演示。还有Zen Tao公司也曾经使用过Unreal的引擎为空手道选手制作过武术训练软件，所以Unreal可称得上是包含娱乐、建筑、教育等其他专业领域的3D成像引擎。

不过因为受到Unreal引擎压力的影响，“雷神之锤2”经过不断的修改更新，所以直到现在它依然还活跃在游戏市场上，而且似乎没有明显老化的现象，实在非常难得。

14.2.5 游戏引擎上革命性的突破

从Unreal的引擎当中，我们看到游戏的画面可以发展到一个天花板的高度了，接下来，游戏引擎的发展方向明显地不能再朝着游戏画面的改良而进行下去了。如同前面所说的一样，游戏引擎的技术对于游戏而言，它的作用并不仅仅只被局限在游戏画面中，它还会直接影响到游戏的整体风格。例如，利用相同的引擎所制作出来的游戏，无论是在其内容或情节设定上，它们都有极为相似的地方，甚至其玩法都大同小异。慢慢的，玩家们开始对这种端着枪跑来跑去的单调模式感到厌倦了，所以游戏开发者就不得不从其他游戏的方面去寻求突破，因此第一人称射击游戏又将掀起另外一种新的游戏风潮。

在第一人称射击游戏突破先前单调的模式过程中，曾经获得无数大奖的Half Life（战栗时空）就在这时候诞生了，它是采用Quake和Quake2游戏引擎的混合体，而这个混合体是由Valve公司在这两部引擎的基础上加入了两个很重要的特性。

脚本序列的技术

这种脚本序列的技术可以让游戏以合理的故事来触动游戏整体架构上的变化，这种技术让玩家可以真实地体验到游戏情节的发展，这对于3D游戏引擎发展至当时之前，无疑是在第一人称射击游戏上的一个伟大突破。

人工智能的改进

混合体引擎的第2个最大的突破就在于人工智能演算上的改进，在游戏中，敌人的行动与以前相类似的游戏来看，他们明显有了更多狡滑的行为，而不再只是单纯地扑向主角的枪口。

这两个特点明显地突破了以往的引擎架构，其又有网络联机机制的扶持，这令“战栗时空”的游戏引擎成为日后第一人称游戏的基础。

不过，对于突破性的人工智能引擎而言，真正的代表作则是Looking Glass工作室的Thief：The Dark Project（盗王之王），它的游戏故事内容是发生在中古时期，而玩家是扮演一名盗贼的角色，而在游戏的任务中，主角可以进入到不同的场所中，并且在不引起敌人注意的情况下来窃取游戏中的物品。

“盗王之王”的游戏引擎就是采用Looking Glass工作室自行开发的Dark引擎，虽然Dark引擎在游戏画面上比不上“雷神之锤2”或“虚幻”的引擎画面，但是Dark引擎在人工智能的演算机制上，它的水平却远远地超过这两者许多。

在Dark引擎游戏中，敌人会懂得根据声音的远近来辨认主角的方位，而且他们还能够分辨出不同地面上的脚步声，还有敌人会在不同的光线环境下具有不同的视力，如果发现到同伴的尸体后，他们便会进入警戒的状态，而且还会针对主角的行动而做出各种合理的反应，所以主角就必须暗藏在不被敌人发现的角落，这才有可能完成关卡的任务。诸如此类的特性，这些都是在以往那些单纯的杀戮游戏中所看不到的。在现今的大部分第一人称射击游戏中，或多或少我们都可以看到如同上述这种风格的游戏，例如Medal of Honor：Allied Assault（荣誉勋章：反攻诺曼底）。

在3D引擎受到“战栗时空”与“盗王之王”两款游戏的启发后，现在有越来越多的游戏开发者开始将游戏中的重点由单纯的视觉效果慢慢转变成更具有丰富变化性的游戏内容，这也成功说明了故事内容与人工智能对于第一人称射击游戏的重要性，而是否能够支持更好的游戏故事内容与敌人的反应特性则成为衡量引擎优劣的另一项新标准。

14.2.6 引擎的发展趋势

从2000年开始，3D引擎就朝着两个不同的方向发展：

第1种是如“战栗时空”和“盗王之王”那样融入许多故事情节和角色扮演的成分，以及加强游戏的人工智能来提高游戏的可玩性。

第2种是朝着单纯的网络架构来发展。

在这个分化的演变中，ID Software公司意识到第一人称的游戏要能够利用人与人之间的互动才是其乐无穷，于是在Quake2引擎的架构上便加入了许多网络机制。这种破天荒的做法顺势推出了一款完全没有单人过关模式的网络游戏——Quake 3 Arena（雷神之锤3之竞技场），这款游戏与Epic公司所推出的Unreal Tournament（魔域幻境之武林大会）一同成为引擎发展史上的一个转折点。

随着ID Software公司在Quake3引擎上的成功，Raven公司又再次与ID Software公司合作，Raven公司采用了Quake3的引擎制作出一款以第一人称射击游戏为主的Star Trek Voyager：Elite Force（星际迷航：精英部队），在当时，这一款游戏也深受玩家的好评。虽然Epic公司的“虚幻竞技场”比“雷神之锤3之竞技场”推出的时间落后了一大步，但是仔细比较一下可以发现，“虚幻竞技场”的表现似乎略高“雷神之锤3之竞技场”一筹。虽然两者从画面的角度来看，差不多可以打成平手，但“虚幻竞技场”在网络机制上，它不仅提供“死亡竞赛”的模式，而且还提供团队合作等激烈火爆的“对战”模式，这让“虚幻竞技场”的引擎不仅可以应用在动作射击游戏当中，而且它还可以为多人联机游戏、实时战略游戏和角色扮演游戏提供强而有力的3D支持，所以一直到现在，Unreal Tournament

引擎还是能够在授权方面胜过Quake3许多。

14.2.7 LithTech 引擎

另外我们还要谈到的是，在1998年到2000年之间，有另一款游戏引擎迅速在引擎市场上崛起，它就是Monolith公司的LithTech引擎。LithTech引擎最初是被使用在机甲射击游戏上的，如Shogo（升刚）。虽然LithTech引擎的总开发时间整整花了5年，其耗资成本花掉了700万美元，但是皇天不负有心人，在1998年，LithTech引擎的第一个版本推出之后立即引起了游戏业界的注意，而且它为当时处于红透半边天的“雷神之锤2”与“虚幻”引擎在游戏市场上的战争泼了一盆冷水。采用LithTech第一代引擎来制作游戏的有众所皆知的Blood2（血祭2）和Sanity（异变）等等。而LithTech公司又在2000年推出了引擎的2.0版本和2.5版本，其中加入了骨骼动画和高级地形的系统，例如采用LithTech 2.5引擎的是Global Operations（全球行动），而在此时，LithTech引擎可以和 Quake3与Unreal Tournament引擎相提并论了。

一直到现在，Monolith 公司已经为 LithTech 引擎推出到 3.0 版本了，并且在其中衍生出了 4 种不同的系统，分别如下列所示：

- 木星（Jupiter）
- 鹰爪（Talon）
- 深蓝（Cobalt）
- 探索（Discovery）

其中“木星”系统被用在 No One Lives Forever（无人永生 2）的开发上；鹰爪系统则被用在 Alien Vs. Predator 2（异形 2）的开发上；“深蓝”系统则是被用在开发 PS2 版的“无人永生”上；“探索”系统则是被用来制作一款尚未公布的大型网络游戏上。 LithTech 引擎除了本身的强大性能外，其最大的卖点就是在于详尽的服务，换句话说，购买者除了可以获得 LithTech 引擎的程序代码和编辑器之外，购买者还可以获得免费的升级，以及电子邮件和电话的技术支持等等，其公司甚至还会将购买者请到公司进行专业性的培训，而且 LithTech 引擎的平均价格大约在 25 万美元左右，这与 Quake3 引擎的 70 万美元相比之下，LithTech 引擎就显得相当的物美价廉。

虽然游戏引擎的进化过程非常复杂，不过这对于游戏而言，它创造出游戏在品质上的提升，而以游戏的技术层面来说，游戏引擎也在不断地突破新的技术。接下来，让我们好好地来深入探论一下，游戏引擎在于未来所能发展的空间又是如何呢？

14.3 未来发展的憧憬

游戏引擎发展至今，有许多表现非常优秀的3D射击游戏陆续被推出，而这些游戏中，其中有一部分是采用Quake3和Unreal Tournament的现成引擎，例如“重返德军总部”和“荣誉勋章：盟军进攻”，但是有许多游戏公司还是采用自行开发的游戏引擎来制作游戏，例如比较具有代表性的作品有Tribes 2（银河生死斗2）、Red Faction（赤色战线）等等。

在开始讲述游戏引擎的未来憧憬之前，我们先来介绍一下这些自行开发的引擎与被授权的引擎有何不同。

14.3.1 V12 引擎

“银河生死斗2”是采用V12引擎来制作的，V12引擎虽然没有办法与Quake3和Unreal Tournament相提并论，但是其开发者为V12引擎所制定的授权模式却相当的新颖。简单的说，我们只要花上100美元就可以轻易获得引擎的使用权，虽然这种授权的价格是相当地吸引游戏开发商，不过天下没有免费的午餐，其授权随之而来的规定则相当地苛刻，例如授权条件是游戏开发者不得利用该引擎来为其他游戏发行商或其他商业游戏站等竞争对手制作游戏，而且在游戏发行之前，游戏必须先交给GarageGames公司（V12引擎的所有者），不能交给任何的第三方，而GarageGames公司将拥有这些游戏5年的独家发行权。尽管引擎的授权条件如此的刻薄，但是对于那些规模较小的独立开发者来说，V12引擎仍然具有非常大的吸引力。

14.3.2 MAX.FX 引擎

如果您是一位3D的玩家，那么一定听过3DMark这套软件吧！3DMark是一套3D测速软件，它所采用的成像引擎则是MAX.FX，而MAX.FX引擎是第一款支持辐射光影成像技术（Radiosity Lighting）的引擎，一般而言，这种技术只被使用在一些高级的建筑设计软件中。MAX.FX引擎能够结合物体表面所有的光源效果，根据材质的物理和几何的特性，准确地计算出每个端点的折射率与反射率，让光线可以以最自然的方式散播出去，为物体营造出一种十分真实的光影效果。

MAX.FX引擎的另一项特点就是所谓的“子弹效果”（Bullet Time），相信您看过“骇客任务”的电影吧！“子弹效果”就是一种慢动作镜头的表现，在这种慢动作的状态下甚至连子弹的飞行轨迹都可以看得一清二楚，而MAX.FX引擎在问世的时候，它便将这种在游戏中所呈现的新视觉效果推向了另一个崭新的高峰。

14.3.3 Geo.Mod 引擎

“赤色战线”所采用的是Geo.Mod引擎，它是第一款可任意改变几何形状的3D引擎，换句话说，它可以使用武器在墙壁、建筑物或任何坚固的物体上炸开1个缺口，并且穿墙而过，或者在平地上炸出一个坑洞后再躲进去，Geo.Mod引擎就具有这一项独一无二的特性。

Geo.Mod引擎的另一个引人入胜的特点就是它具有高超的人工智能演算，在游戏中，敌人不仅仅可以在看见同伴的尸体或听见爆炸声后才会做出反应，而且当敌人发现留在周围物体上的痕迹（如弹孔或血渍之类）时，它们也会警觉起来，并且懂得远离那些可能对自己造成伤害且又无法做出还击的特性，甚至于在受伤的时候，它们也会没命地逃跑，而不会冒着生命危险继续地作战下去。

14.3.4 Serious 引擎与 Krass 引擎

“重装武力”采用的是Serious引擎，这款引擎最大的特点在于它有相当强大的成像效果，当面对大批的敌人和一望无际的场景时，我们可以感觉到画面不会有停滞的现象，而且游戏画面的成像效果也相当出色。此外，值得一提的还有AquaNox（怒海潜将）所用的Krass引擎，这款引擎被当成GeForce 3的官方指定引擎，它是可以用来宣传或演示GeForce 3的显示效果，其视觉的表现方面也是无可替代的。

14.3.5 引擎未来的发展趋势

从游戏引擎的发展史可以看出，这几年所推出的几部引擎依旧是延续了两年多来的发展趋势，一方面不断地追求游戏中的真实效果，例如 MAX.FX 引擎与 Geo.Mod 引擎所追求游戏画面的真实感，另一方面则继续朝向网络机制的方向来探索，如“银河生死斗 2”，以及 Monolith 公司尚未公布的大型网络游戏。不过，由于受到各方面技术的限制，3D 引擎要将第一人称射击游戏放入大型网络环境中的构想至少在目前还是很难去实现的（目前只能在局域网络上联机）。一般而言，大型的网络游戏多半是流程节奏较慢的角色扮演游戏，而这些游戏所使用的引擎都无法支持一个可以提供数百名玩家同时在大型战斗的团队动态环境下执行，基于这样的考虑，ID Software 公司只有重新将引擎的重点放在单人模式上，如该公司在去年所公布的“雷神之锤 4”和“毁灭战士 3”将重新建构一个以单人游戏为主的引擎。在这个时候，ID Software 公司的老对手——Epic 游戏公司也在紧锣密鼓地开发新一代 Unreal 引擎，尽管目前这几款新一代引擎的具体数据并不是很多，但是从它们所展示的几段采用新引擎实时成像的动画片段来看，的确是超越了市面上的其他引擎，这也预言着一个崭新引擎时代的到来。

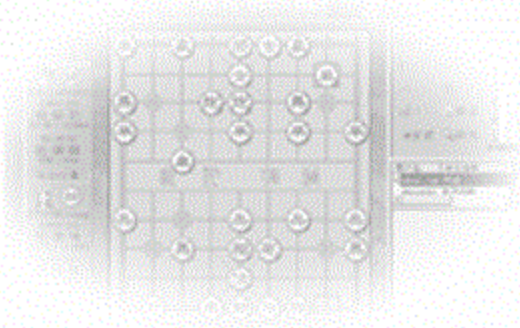
在这种游戏产业竞争激烈的市场中，许多优秀的游戏开发者正在退出游戏开发的市场，

转而进入引擎授权的时代。而且仅靠开发游戏引擎吃饭，这是个非常危险的信号，尽管游戏引擎不断地演进，而使得游戏的技术品质越来越高，但是我们相信决定一款游戏的优劣性是在于使用技术的人员而不是技术本身。如同前面所说的一样，游戏引擎相当于游戏的框架，在框架基础打好以后，我们只需在框架内设定其内容就可以了，但是游戏的精彩与否则取决于故事内容的丰富性而不是框架。

总结上述的讨论，我们归纳出一个结果，那就是“游戏引擎的优劣与否并不能决定游戏是否好玩。”所以我们只能说，游戏引擎只能带来游戏技术的提升，而不能使游戏更加地好玩。

第 15 章

编辑工具软件的制作



在游戏开发的同时，必须要有许多工具来配合，而这些工具是为了游戏中的某一些功能而编造的；如地图编辑器、剧情编辑器等。本章就是介绍在制作一套游戏时所有可能会用到的编辑工具软件。

15.1 地图编辑器

在一套游戏的制作过程中，不管是要开发 2D 或是 3D 的游戏，我们都需要使用地图编辑器来编制游戏中的场景。地图编辑器是策划人员将游戏中所需要的场景元素告诉程序设计师与美工人员，然后程序设计师要利用美工人员所绘制出来的图素，编写一个可以编辑此套游戏场景的应用程序，而这个应用程序即可提供策划人员编制游戏场景时所使用。接下来就来看看制作一套地图编辑器需要经过哪些过程和技巧。

15.1.1 地图

游戏是策划人员或策划团队搅尽脑汁且经过长时间修改所编制而成的，游戏中的最主要灵魂关键就是在于它的游戏背景，不管是过去、现在或者未来，它都有一定的时代背景所呈现画面模式，基于时代背景的合理化条件，一个策划人员就必须将游戏时空场景里所有的合理地形、建筑物及对象都一一归纳出来，并且配合美工人员进行图素的绘制。在一套大型的游戏开发过程中，美工人员不可能将一张张大型图片一一画出以提供程序使用，因为如果这样做的话，不但会累死美工人员，而且还会发生不可避免的问题，如大型图片可能不能用的情况，如此一来，这又是一件浩大的工程，因此建议不要使用这种非常不经济的做法。

在地图上的所有场景与对象，我们可以利用单一组件的表现方式来呈现全场景的观感，例如，将一棵树的图片组件放置于场景中，如图 15.1 所示。

图 15.1　将一棵树放入场景中

然后用相同的方法将这棵树复制成许多组件，最后再贴到背景中，如图 15.2 所示。

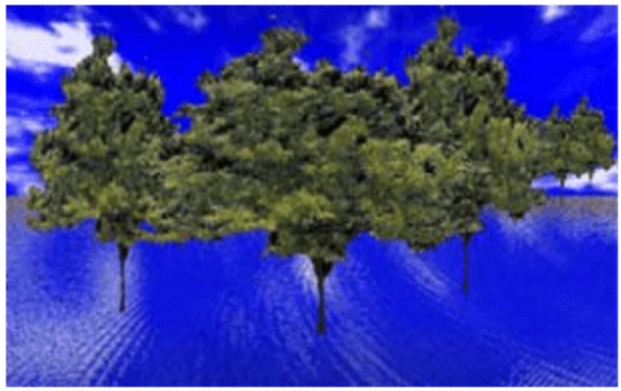


图 15.2　在场景中复制多棵树

在上述过程中，我们只使用了一张背景地图与一张树的元素图片，就可以完成一个游戏的场景了。如果再多增加几个地图上的元素，那么游戏中的地图就会显得华丽许多了，如图 15.3 所示。

图 15.3　在场景中添加多个背景图片

其实地图编辑器就如上述所说的这种原理，它的主要功能就是可以让使用者自行编辑游戏中的地图，以提供游戏时使用。

不管是哪一种类型的游戏，只要是牵涉到场景地图的部分，我们都可以利用这一系列的原则来制作地图编辑器。

制作一套地图编辑器，首要的条件就是在地图上所有元素都必须以等比例的方式来绘制，或许您会问：“什么是等比例的呈现方式呢？”答案很简单，那就是将地图上的图素以一定的比例来制作。举例来说，假设地图中人物占了 1 个方格单位，而树占了 6 个方格单

位，房子占了 15 个方格单位，如图 15.4 所示。

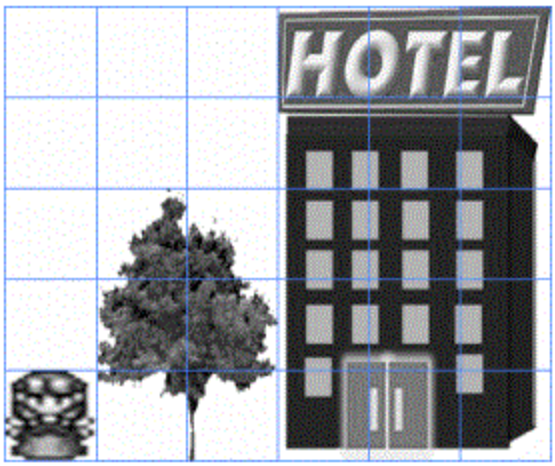


图 15.4　地图中人物所占的单位

如此一来，人物与其他地图上的对象不就是等比例的图示了吗？

人物:树＝1:6
树:房子＝6:15
人物:树:房子＝1:6:15

或许您又会问：“等比例的图示有何好处呢？”其实图素的等比例长宽是可以用来避免许多在游戏设定上不必要的麻烦，在解释等比例的好处之前，首先来介绍一下地图编辑器有哪些优点。如下所示：

- 地图数组
- 良好的扩充性
- 属性编列
- 碰撞侦测

地图数组

在编写游戏主程序时，地图上场景的贴图处理是必然要执行的，不过在进行游戏时，主程序又可能会使用到许多不同的演算公式（如路径搜寻），所以如果要在不浪费系统资源的情况下，我们只有在地图场景上下功夫。我们可以将地图上的各种图素编制成一系列的数字形态，并且提供游戏主程序来读取，换句话说，这种方法是将地图上的图素以一种特定的数字排列方法来表示该图素所在的位置，如表 15.1 所示。

表15.1　图素及代表数字

图素	代表数字
草地	1
泥沼	2
石地	3
高地	4
水洼	5

在地图编辑器中，我们可以看到如图 15.5 所示的地形。

1	3	1	1	1	3
1	4	1	2	1	1
3	1	2	1	1	1
2	1	1	1	4	4
1	2	5	5	2	1

图 15.5　地形数组

在游戏中，我们可以看到如图 15.6 所示的画面。

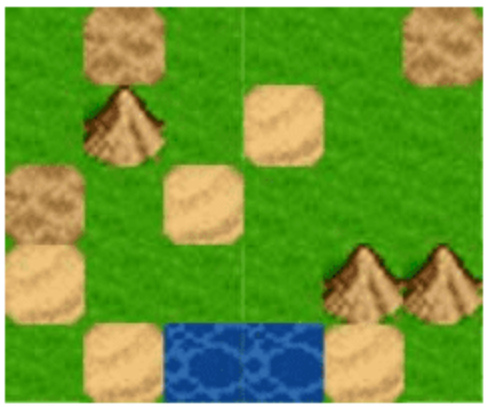


图 15.6　游戏场景

如此一来，当我们将地图编辑的结果储存起来的时候，便可以在该档案中将所有用到的图素加以筛选，并且在游戏主程序读取该地图数据的时候，它只要去读取需要的图素就可以了，您看！这不就节省了许多系统资源了吗？而地图上的数组又可以用来显示画面中

应该显示的图素（如第 8 章所谈到的一样），如此一来，又更加减少了系统资源的浪费，如图 15.7 所示。

图 15.7　地形数组

良好的扩充性

游戏中最难处理的是游戏场景，因为我们要考虑到游戏的执行速度（场景是消耗系统数据最大的因素）、未来场景的维护（使美工人员容易修改与换图），这些都是编写地图编辑器的主要目的。一套成熟的地图编辑器不仅仅可以帮助策划人员编辑他心目中理想的场景，而且它还可以提供美工人员作为修改图素的惟一依据。在地图场景上，如果有某一个部分不符合策划人员的想法时，策划人员只要将场景的这个错误的地方利用地图编辑器来修改，而不需要请美工人员重绘这张场景（修改大型的场景对于美工人员来说，它是一件相当辛苦的事），如果场景的图素不够用时，策划人员还可以请美工人员再绘制其他的小图素来补满场景不足的地方，而策划人员只要在新增的图素上另外再编列新的代码就可以了，这对于地图未来的扩充性有相当大的帮助。

属性编列

在场景图素中，我们也可以将这些小图素设定它们特有的属性，如“不可让人物走动”（墙壁）、“可让人物走动”（草地）、“让人物中毒”（沼泽）等等，这些属性都可以在地图编辑器中设置。

碰撞侦测

将场景中的图素以等比例的方式来绘制，主要原因是可以用来做碰撞侦测。我们可以

利用图素的属性与数组格数来侦测人物是否碰撞到了物体。以等比例的方式就是要将图素填满数组方格，当人物在踏进数组方格之前，我们可以先判断人物在下一个方格内的图素是否可以走，或会触发某些事件，如果下一个方格不能走的话，那么势必要在人物移动坐标上做递减或递增的动作，如图 15.8 所示。

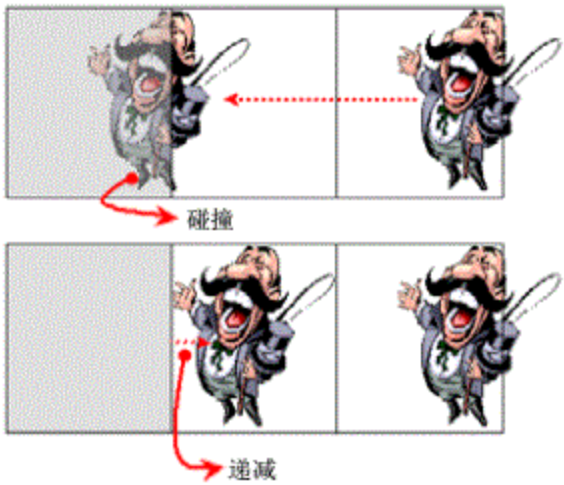


图 15.8　碰撞的递减动作

如果场景上的小图素没有以等比例的方式来显示的话，那么势必会遇到一种“似碰非碰”的情形，如图 15.9 所示。

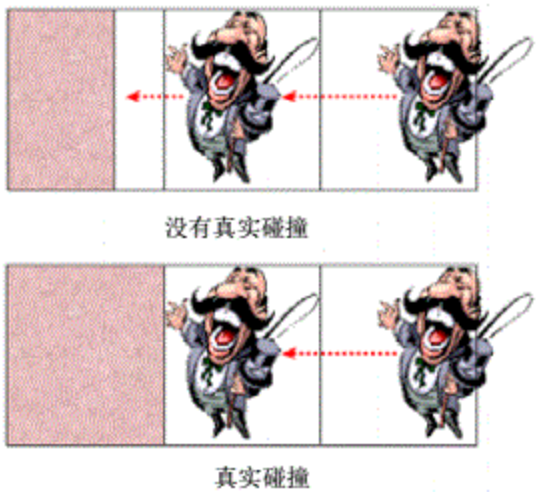


图 15.9　碰撞效果

所以美工人员在绘制场景小图素之前，我们都会先规范小图素的比例大小，一般而言，

我们会使用 2 的次方来表示，因为 2 的次方可以符合屏幕的长宽大小，例如场景小图素的长宽为 32×32（25×25）像素，那么在 640×480 的分辨率中便可以放下 20×15 块小图素了。

以目前的游戏来说，许多战略类型的游戏在产品中也会附上该公司所研发的地图编辑器，而它可以用来编辑该游戏中的战场地图，例如魔兽争霸系列、星海争霸系列、世纪帝国系列等等，而它们的运作方式都逃脱不了上面所说的原理。其实地图编辑器的制作过程并不是很困难，而最头疼的是要如何设定地图编辑器上所有对象的属性。接下来，我们来看看地图编辑器上这些对象的属性要如何设置。

15.1.2 属性

地图编辑器除了可以编定场景小图素之外，我们也必须利用它来设定小图素的属性值，其中小图素在程序内必须被使用到，所以更要针对程序所需要的属性来设置，其属性设定值如表 15.2 所示。

表15.2　属性设置值

元素	编号	长/宽	可让人物走过（1/0）	是否会失血（1/0）	行动是否缓慢（1/0）
草地	1	16/16	1	0	0
泥沼	2	16/16	1	1	1
石地	3	16/16	1	0	1
高地	4	16/16	0	0	0
水洼	5	16/16	0	0	0

这些属性值会直接影响到人物的移动变量，例如，人物在石地地形上移动的话，其行动会变得很缓慢，或者是人物在经过泥沼地形时会导致失血等等。在这里，我们只列举几项基本的属性设定值，其实在一套成功的游戏中，光是地图属性就可能有好几十种变化，而这些极具真实的地图属性则会让玩家在游戏中直呼过瘾。

15.1.3 地图编辑器（Map Editor）

从一个 3D 的地图编辑器来看，在地图编辑器上，使用者可以编辑 3D 地形的地表、全景长宽、地形凹凸变化、地表材质、天空材质，以及地形上所有存在的对象（如房子、物品、树木、杂草等），如图 15.10 所示。

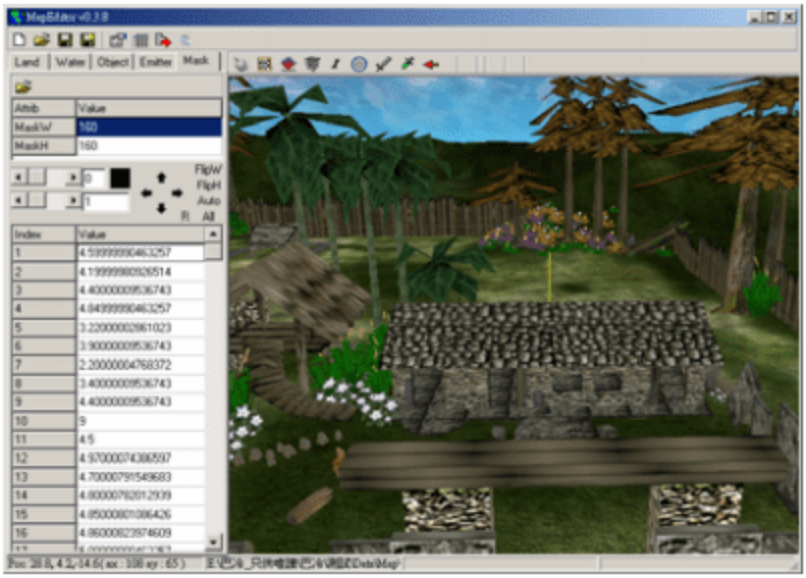


图 15.10　地图编辑器

甚至可以在地表上放置设定好的分子特效对象（稍后会谈到），就好像是在编辑器上盖房子一样，并且依据策划人员的喜好在特定的地形上放置任何对象，如此一来，在创造场景上不就更加方便许多了吗？如上面所说的一样，一套游戏是否能够满足玩家的胃口，完全要靠地图上任何对象的贴心设计，并且在各个对象上加注特殊的属性，如此一来，让玩家感觉到有如在现实生活中一样。

15.2 剧情编辑器

贯穿一套游戏的主要干道是游戏中的剧情演进，而游戏中的剧情通常是用来主导整个游戏的流程进行，一般而言，游戏中的剧情可以分为两大类，一种是主要的 NPC 剧情，另一种是旁支剧情，接下来就针对这两大剧情作详细介绍。

15.2.1 架构

在开始介绍游戏的两大类剧情之前，我们先来看看一套游戏的主要流程是如何进行的。一套游戏的主要剧情架构流程如图 15.11 所示。

在游戏中，为了让游戏的故事剧情发展的更加丰富，所以在主要的剧情上另外再编制一些较为无关紧要的人物对话，而这些另外加入的人物对话则以不影响整个游戏流程的进

行为主要原则。

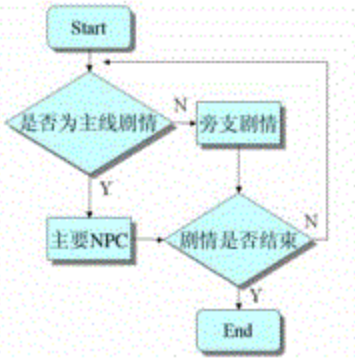


图 15.11　剧情架构流程

其实在规划游戏剧情的时候，我们可以将主要的 NPC 剧情由单线再扩充成多线的剧情，如图 15.12 所示。

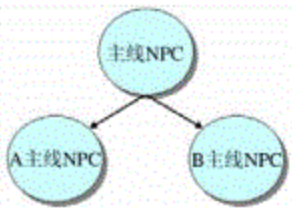


图 15.12　NPC 剧情

如果为了让故事剧情再增加一些复杂性的话，其实还可以继续的分类下去，如图 15.13 所示。

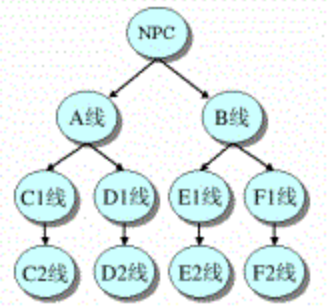


图 15.13　故事剧情分类

谈到这里，值得注意的是，不需要为了提升游戏中故事的丰富性而随便增加一些无意

义的剧情，因为这样会导致玩家在游戏中无所适从，而且对于剧情的架构而言，它会更加的难以维护。虽然这么说，不过，我们还是可以利用“多线式”的方式来进行游戏的故事发展，但是惟一的条件就是最后还是要让这些多线式的剧情再统一起来。其架构图如图15.14所示。

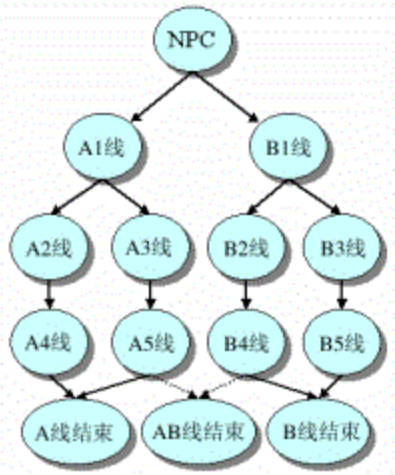


图15.14　统一多线式剧情

如此一来便可以大大地提升游戏故事剧情的复杂性，而且还可以增加游戏剧情的合理性。简单地说，我们可以将这种方式想象成在现实生活中，不单单只有自己一个人的行为模式而已，而是在同一个时间里，会有许多人的行为模式一起进行。

介绍完游戏故事剧情的流程之后，接下来便可以开始来讨论游戏的主要NPC流程要如何设计了。

15.2.2 NPC人物

在第2章，我们曾谈到，NPC（Non Player Character）指的是游戏中一些“非玩家角色”的意思。在一个游戏时间背景里，不可能只有一个主角存在于游戏世界中，而是会存在一些特定与非特定的人物，这些人物都可能为玩家带来一些剧情上的提示，或者为玩家主角提供武器与防护器具上的提升，这些人物是不能够主动去操作它们的，而是由策划人员所提供的AI（人工智能）、个性、行为模式等等相关的属性让程序设计师设计出这一些人物在游戏中的行为模式。

NPC人物不单只是玩家的朋友，它们也可能是玩家的敌人，为了让游戏的故事剧情能够延续发展下去，这些NPC人物的对话内容就显得非常重要。在这里先卖个关子，我们先来介绍一下什么是旁支剧情，再来看看主要NPC与旁支剧情要如何来设计。

15.2.3 旁支剧情

旁支剧情在游戏中具有陪衬角色的使用，如果游戏中少了旁支剧情故事的话，那么势必会让玩家觉得游戏少了几分乐趣，在游戏界竞争激烈的情况下，我们要尽量博取玩家所寻得的乐趣来增加游戏的可玩性。其实严格说来，旁支剧情并不足以取代游戏中主要流程的进行。一般而言，旁支剧情是用来让玩家可以取得一些特定且有用的物品，如道具、金钱或经验值等等，例如，假设玩家在游戏的某一个村庄里，而主角可以在路上遇到一些 NPC 人物，有些 NPC 人物可能会说出一些无关紧要的话，例如，“今天天气真好！”或“请给我钱好吗？”，甚至于有些 NPC 人物会有更惊人的话语，例如，“我有一个非常棒的道具，价格是‘99999999’，您要不要买？”笔者曾经玩过类似这样的游戏，当笔者好不容易存满这么多钱以后，决定去买那一个道具，结果竟然得到一个很普通的补血剂，当时笔者真是欲哭无泪，如此一大笔钱竟然被一个 NPC 人物给骗光了。虽然这是一个很简单的例子，但是对于一个玩家来说，我们已经成功的接近了玩家与游戏之间的互动，如此一来，玩家们会更加喜欢玩这一类游戏。

如同上面所说的一样，旁支剧情最主要的目的是用来增加游戏的娱乐性，虽然上面例子是一种令人发指的行为，不过，玩家还是可以从旁支的 NPC 剧情中得到游戏中不可能会出现的物品或道具，对于玩家而言，他们会更加喜爱游戏中的故事剧情。

15.2.4 脚本编辑器

在了解了主要 NPC 剧情与旁支剧情的原理后，接下来便可以开始来规划游戏中的剧情编辑器了。所谓剧情编辑器它就是让使用者依据自己的喜好，在特定的指令条件下，编写属于自己的故事剧情，我们称为“编写 Script 指令”，如图 15.15 所示。

```
,,[EVENT] = 555,,,,,,,,
,,,ID_TALK,IDS_NORMAL,MAN100,,是噢～,f100,1,00,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,我是想回送一些猎物给作糯米糕的老奶奶，至于・・・,f100,1,00,,,,勇士2说明
,,,ID_SYSTEM,IDS_SHOWICON,MAN100,,M800010,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,我也不太记得到底吃了几个 ？,f100,1,00,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,只知道我是第二个去吃的・・・,f100,1,00,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,桌子上剩下的糯米糕，被我吃了一半・・・,f100,1,00,,,,勇士2说明
,,,ID_SYSTEM,IDS_SHOWICON,MAN100,,M800016,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,但这～实在太好吃了・・・,f100,1,00,,,,勇士2说明
,,,ID_TALK,IDS_NORMAL,MAN100,,于是我～又多拿了一个・・・,f100,1,00,,,,勇士2说明
,,,ID_SYSTEM,IDS_SETFLAG,,FLAG_RICEEVENT = 3,,,,糯米糕事件起动
,,[/EVENT],,,,,,,,
```

图 15.15　编写 Script 事件

为了让使用者可以编写故事剧情，剧情编辑器就必须规划出一系列的“指令”以供使用者来输入，例如当使用者在编辑一个 NPC 人物的对话时，剧情编辑器就必须提供一个人物说话的指令，如下所示：

```
TALK MAN01, "你好吗？"
```

上面的 TALK 是剧情编辑器所提供给 NPC 人物说话的指令，MAN01 是定义 NPC 人物的编号（NPC 人物的编号等一下会谈到），“你好吗？”则是 NPC 人物所说的话，这就是剧情编辑器的主要指令用法。其实我们还可以将上述的 TALK 指令再扩充，如下所示：

```
TALK  NPC人物编号, "对话字符串",NPC人物动作,NPC人物示图,示图方向（L/R）
```

关于剧情编辑器的指令参数设定，我们就要靠策划人员来规划，将游戏中所有可能会发生的情况一一列出，以方便程序人员编写剧情编辑器指令所用，而程序人员可以将剧情编辑器的流程规划成如图 15.16 所示。

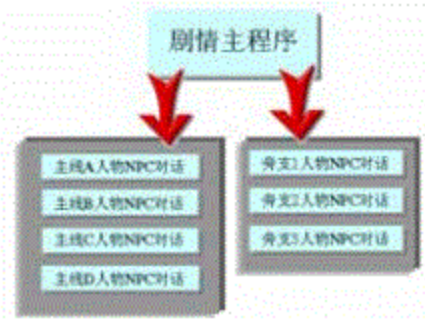


图 15.16　剧情人物对话

而策划人员所着手的 NPC 人物指令可以规划成如表 15.3 所示。

表15.3　MPC人物指令参数说明

指令	附加参数	说明
TALK	NPC 人物编号, "对话字符串",NPC 人物动作,NPC 人物示图,示图方向(L/R)	NPC 人物的对话
MOVE	NPC 人物编号, X/Y 坐标,移动速度,移动方向（1/2/3/4）	NPC 人物移动
ATT	NPC 人物编号，被攻击的 NPC 人物编号, NPC 人物动作	NPC 人物攻击某一个 NPC 人物(包括主角)
ADD	加数,被加数	指令内的加法运算（通常用来计算人物的血量）
DEL	减数,被减数	指令内的减法运算（通常用来计算人物的血量）

上面表格只是一个范例，一个成功的策划人员可以很轻易地规划出游戏中所有可能会

发生的事情，以提供程序设计人员编写剧情编辑器所使用。其实这一点都不难，在这里，笔者提供一个建议供大家参考，那就是您可以在纸上利用想象力将游戏从头到尾Run一遍，然后将所有可能会发生的事件与行为都记录下来，最后再将这些事件或行为的动作归纳成一连串的指令，如此一来便可以轻易地编写出剧情编辑器所要使用的指令了。

15.3 特效编辑器

“特效”是一个可以更加衬托出游戏品质的重要角色，一套成形的游戏对于玩家来说，除非它是继承以前经典或是当红的热门游戏，否则一套新型的游戏是很难被玩家所接受的，而我们就是要从游戏中华丽的画面来呈现而吸引玩家的目光，简单地说，“特效”即是游戏与玩家之间的第一扇门。

不过以一套大型的游戏来说，程序设计师必须要依照策划人员的规划，将所有特效程序代码一个一个编写出来，如果游戏的特效不多时，那么这种方法还可以接受，但如果游戏特效很多的时候，甚至超过1000多种的话，那么再让程序设计师一个一个撰写程序代码的做法实在是太不经济了。基于这一理由，我们就想到了一个解决的方法，就是请程序设计师编写一个可以符合游戏中特效的编辑器，以提供所有人使用。如果一个人可以利用特效编辑器做出200个特效的话，那么只需要5个人就可以编出1000多种特效了，那不是更加经济实惠吗？

15.3.1 特效

如同在第8章与第9章所谈论的一样，特效可以通过2D或3D的方式来呈现，不管怎么说，当策划人员在编写特效的时候，就必须将所有的属性都列举出来，以提供程序人员编写特效编辑器时的方便。

其实特效在游戏中也是一种对象，所以它可以被放置在游戏地表上，如前面所说的一样，我们可以利用地图编辑器将特效“种”在地表上（如烟、火光、水流），而游戏中便可以在地表上看到这些特效了。在进行游戏的时候，也可以利用特效来提升主角攻击的刺激感，并且表现出力量的美。

15.3.2 属性

特效本身也需要许多属性的配合，以一个3D分子特效而言，它就必须包含特效的原始坐标、粒子的坐标位置、材质、运动途径与方向等等，这都是策划人员必须详细归纳出的示意图，以提供程序设计人员编写特效的属性，如图15.17所示。

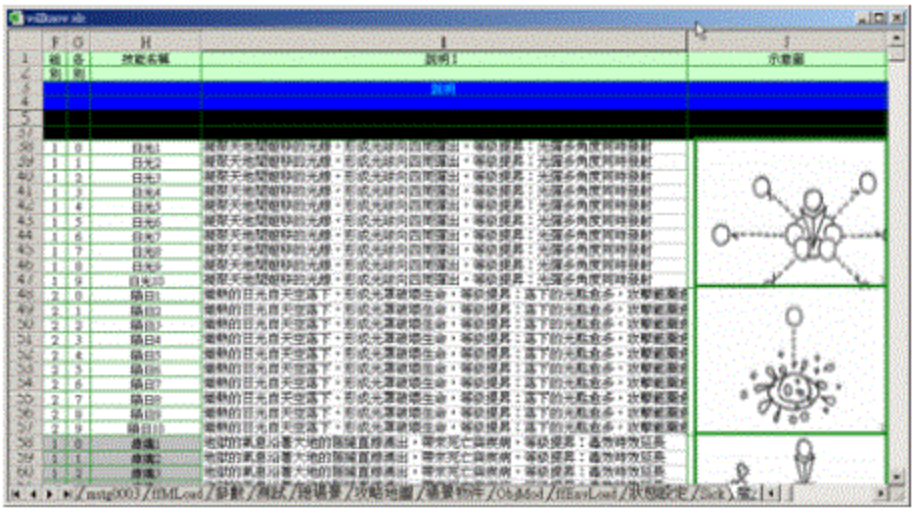


图 15.17　特效属性

有了这些特效的属性之后，接下来，便可以请程序设计人员开始编写特效编辑器的应用程序了。

15.3.3 粒子编辑器

在程序设计师接手策划人员的特效示意图之后，程序设计师便可以着手开始设计特效编辑器了。在上面例子中，我们是以 3D 的特效效果为主，所以策划人员所绘的示意图便可设定成三维坐标图，并且编写所有粒子应该拥有的属性，如表 15.4 所示。

表15.4　属性设置说明

属性设定值	说明
PosX/PosY/PosZ	粒子 X 坐标/Y 坐标/Z 坐标
TextureFile	粒子的材质
BlendMode	粒子的颜色值
ParticleNum	粒子的数量
Speed	粒子的移动速度
SpeedVar	粒子移动速度的变量
Life	粒子的生命值
LifeVar	粒子生命值的变量
DirAxis	运动角度

关于粒子的属性，可以参阅本书第 8 章与第 9 章的介绍。编写出粒子的所有属性后，程序设计师只要再配合 3D 成像技术便可轻易地编写出如图 15.18 所示的特效编辑器了。

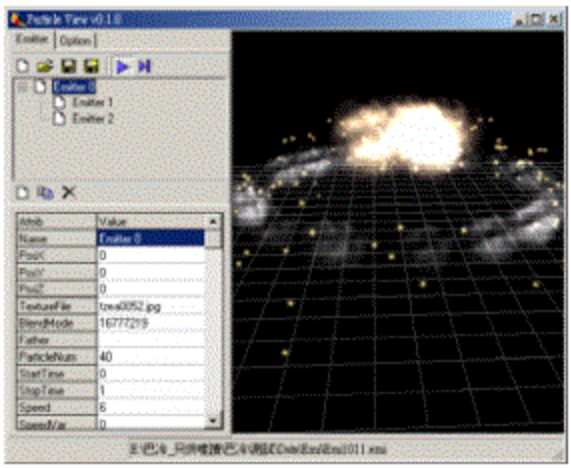


图 15.18　特效编辑器

介绍完特效编辑器之后，接下来，我们来介绍游戏最后的一种编辑器——人物道具编辑器。

15.4 人物道具编辑器

在一套游戏中，人物与道具是最难管理的数据，因为它们的数量是最多的。基于我们要有效地管理这些数据，则利用一种方式来解决这个问题，那就是使用 Microsoft Office 所提供的软件包 Excel。虽然 Excel 是一个电子表格软件，不过因为它的表格化字段明确可见，所以便可以用来管理游戏中所有的数值数据，而且容易维护这些庞大的数据库，更重要的是 Excel 的功能相当齐全，不管是排序数据或在数据库中寻找某些特定的数据等，其功能都非常的好用，而且程序设计师只要编写一个可以读取 Excel 文件的程序函数便可以解决上面所提到的所有问题了。

15.4.1 人物

接下来，我们来探讨游戏中的人物应该要如何来设定。其实人物的设定不外乎一个原则，那就是利用游戏中人物的个性与特性来设定所有的属性。举例来说，游戏中的某个角色是属于火爆型的个性，而且体格健壮，从一般的角度来看，我们可以将他归列于攻击力

强、魔法力（智力）弱、防御力一般的属性。简单的说，这种角色在游戏世界里是属于头脑简单且四肢发达的人；再举个例子来看，某种角色是属于较为年长的老人，他是以魔法攻击为主，所以我们便可将这种角色归列于攻击力弱、魔法力（智力）高、防御力弱的属性。简单地说，他就是游戏中的魔法师。

如同上面所说的一样，我们将这些属性归纳一下，如表 15.5 所示。

表15.5　属性说明

属性	说明
LV	人物的等级
EXP	人物的经验值
MAXHP	人物的最大血量
MAXMP	人物的最大魔法量
STR	人物的攻击力
INT	人物的魔法力（智力）

在 Excel 中编写成类似如图 15.19 所示的外观。

图 15.19　Excel 表格

其实我们可以利用同样的道理，将游戏中的怪物以 Excel 电子表格来设定其属性值，如图 15.20 所示。

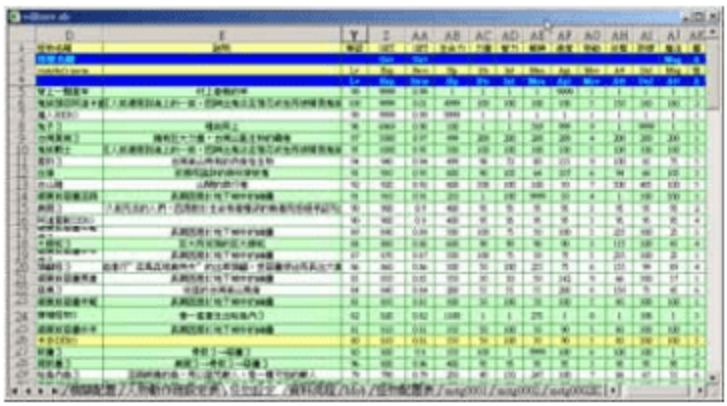


图 15.20　Excel 电子表格设置属性值

人物与怪物的属性配置是否恰当，这全然要看策划人员的能力了。以笔者看来，属性的配置可称得上是一门大学问，因为它是由游戏开发初期到游戏开发完成之后，惟一修改不完的任务，说到这里，或许您会问：“那么我们应该要如何来配置这些属性呢？”说实在的，如果光只靠上述例子的做法来评定一个主角或怪物属性的话，那么一定会遇到一些不合理的事件发生，例如主角的等级很高，但却轻易地被一只小怪物给打死了。这对玩家来说，做法非常不佳。其实我们可以利用一个很简单的方式来避免这种情况的发生，那就是在编写这些属性之前，我们可以建立一个合理的公式表，如图 15.21 所示。

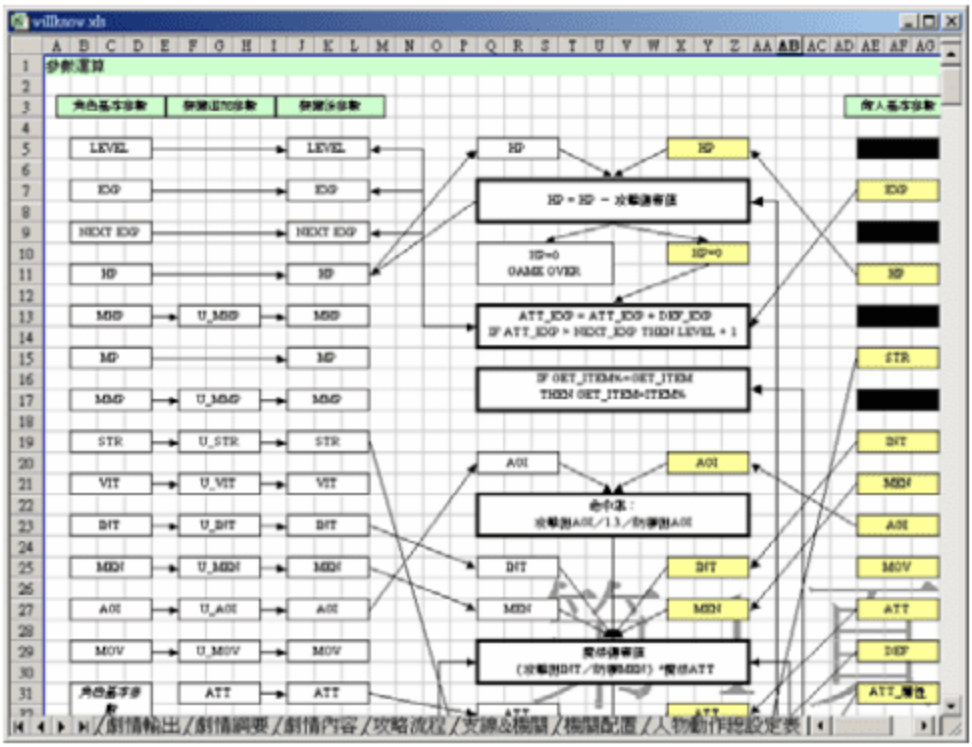


图 15.21　公式表

我们可以利用一些运算公式来求得属性的合理性，举例来说，如果是失血的话，我们可以将公式写成：

```
敌方防御力/（（人物STR+STR加值）*0.1）=失血量
200/（（100+50）*0.1）=13.3
```

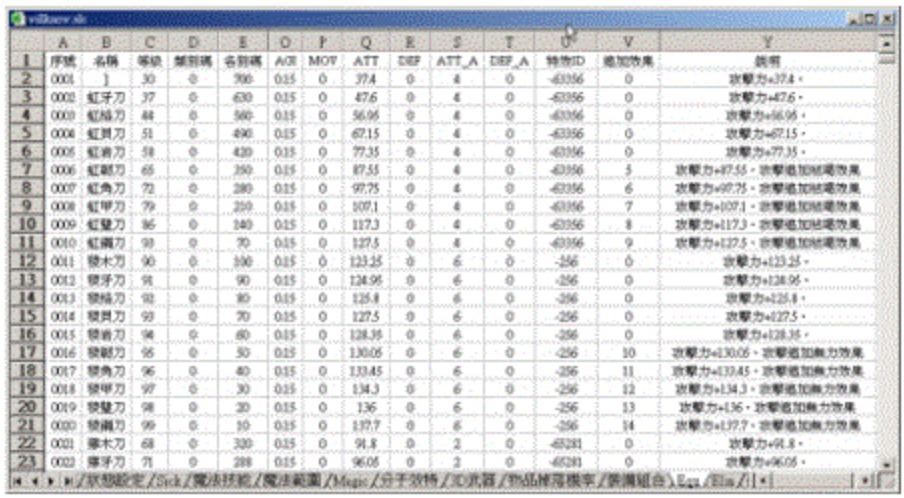
从上述公式来看，敌方人物的血就会失去 13 个单位（四舍五入）。

这是一个很简单的做法，虽然在求得运算公式时需要花上一点心思，不过这对于尔后编排人物属性时就非常方便，而且程序人员也可以利用这些公式来编写游戏中人物与敌人

之间的攻击和防御的运算，

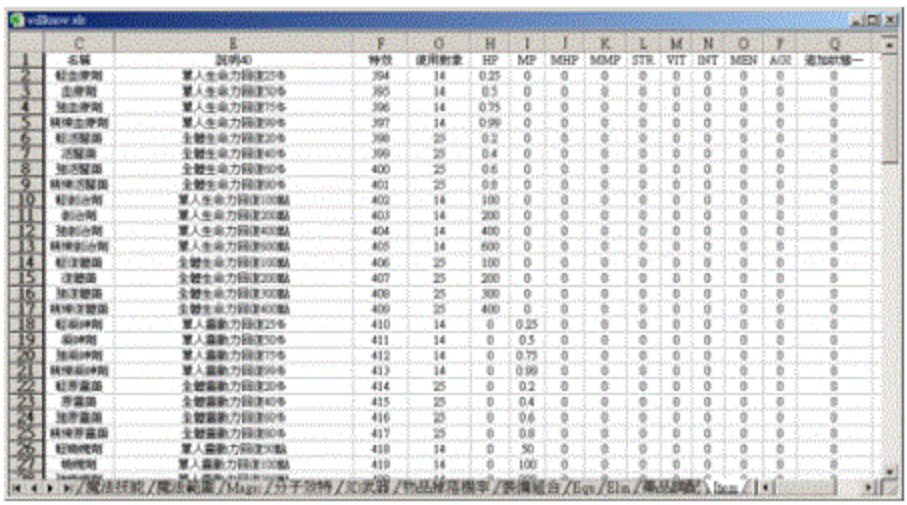
15.4.2 武器道具

在游戏中，武器道具的数据库也是最难管理的东西，甚至它比人物的数据还要多，所以必须小心谨慎地处理这些数据。在这里，我们还是依照上面所讲的方式，将武器道具的数据填写在 Excel 之中，便于我们管理与维护，如图 15.22 和图 15.23 所示。



	A	B	C	D	E	O	P	Q	R	S	T	U	V	Y
1	序號	名稱	等級	類別碼	名別碼	A/E	MOV	ATT	DEF	ATT_A	DEF_A	特物ID	追加效果	說明
2	0001	]	30	0	700	0.15	0	37.4	0	4	0	-63356	0	攻擊力+37.4 -
3	0002	虹牙刀	37	0	630	0.15	0	47.6	0	4	0	-63356	0	攻擊力+47.6 -
4	0003	虹格刀	44	0	560	0.15	0	56.95	0	4	0	-63356	0	攻擊力+56.95 -
5	0004	虹貝刀	51	0	490	0.15	0	67.15	0	4	0	-63356	0	攻擊力+67.15 -
6	0005	虹齒刀	58	0	420	0.15	0	77.35	0	4	0	-63356	0	攻擊力+77.35 -
7	0006	虹鄰刀	65	0	350	0.15	0	87.55	0	4	0	-63356	5	攻擊力+87.55 - 攻擊追加結果效果
8	0007	虹角刀	72	0	280	0.15	0	97.75	0	4	0	-63356	6	攻擊力+97.75 - 攻擊追加結果效果
9	0008	虹甲刀	79	0	210	0.15	0	107.1	0	4	0	-63356	7	攻擊力+107.1 - 攻擊追加結果效果
10	0009	虹雙刀	86	0	140	0.15	0	117.3	0	4	0	-63356	8	攻擊力+117.3 - 攻擊追加結果效果
11	0010	虹鋼刀	93	0	70	0.15	0	127.5	0	4	0	-63356	9	攻擊力+127.5 - 攻擊追加結果效果
12	0011	狼木刀	90	0	100	0.15	0	123.25	0	6	0	-256	0	攻擊力+123.25 -
13	0012	狼牙刀	91	0	90	0.15	0	124.95	0	6	0	-256	0	攻擊力+124.95 -
14	0013	狼格刀	92	0	80	0.15	0	125.8	0	6	0	-256	0	攻擊力+125.8 -
15	0014	狼貝刀	93	0	70	0.15	0	127.5	0	6	0	-256	0	攻擊力+127.5 -
16	0015	狼齒刀	94	0	60	0.15	0	128.35	0	6	0	-256	0	攻擊力+128.35 -
17	0016	狼鄰刀	95	0	50	0.15	0	130.05	0	6	0	-256	10	攻擊力+130.05 - 攻擊追加無力效果
18	0017	狼角刀	96	0	40	0.15	0	133.45	0	6	0	-256	11	攻擊力+133.45 - 攻擊追加無力效果
19	0018	狼甲刀	97	0	30	0.15	0	134.3	0	6	0	-256	12	攻擊力+134.3 - 攻擊追加無力效果
20	0019	狼雙刀	98	0	20	0.15	0	136	0	6	0	-256	13	攻擊力+136 - 攻擊追加無力效果
21	0020	狼鋼刀	99	0	10	0.15	0	137.7	0	6	0	-256	14	攻擊力+137.7 - 攻擊追加無力效果
22	0021	獠木刀	68	0	330	0.15	0	91.8	0	2	0	-65281	0	攻擊力+91.8 -
23	0022	獠牙刀	71	0	288	0.15	0	96.05	0	2	0	-65281	0	攻擊力+96.05 -

图 15.22　武器的 Excel 数据



	C	E	F	O	H	I	J	K	L	M	N	O	P	Q
1	名稱	說明ID	特效	使用對象	HP	MP	MHP	MMP	STR	VIT	INT	MEN	A/E	追加狀態一
2	輕生療劑	單人生命力回復25%	394	14	0.25	0	0	0	0	0	0	0	0	0
3	生療劑	單人生命力回復50%	395	14	0.5	0	0	0	0	0	0	0	0	0
4	強生療劑	單人生命力回復75%	396	14	0.75	0	0	0	0	0	0	0	0	0
5	精煉生療劑	單人生命力回復99%	397	14	0.99	0	0	0	0	0	0	0	0	0
6	輕活髮菌	全體生命力回復20%	398	25	0.2	0	0	0	0	0	0	0	0	0
7	活髮菌	全體生命力回復40%	399	25	0.4	0	0	0	0	0	0	0	0	0
8	強活髮菌	全體生命力回復60%	400	25	0.6	0	0	0	0	0	0	0	0	0
9	精煉活髮菌	全體生命力回復80%	401	25	0.8	0	0	0	0	0	0	0	0	0
10	輕創治劑	單人生命力回復100點	402	14	100	0	0	0	0	0	0	0	0	0
11	創治劑	單人生命力回復200點	403	14	200	0	0	0	0	0	0	0	0	0
12	強創治劑	單人生命力回復400點	404	14	400	0	0	0	0	0	0	0	0	0
13	精煉創治劑	單人生命力回復600點	405	14	600	0	0	0	0	0	0	0	0	0
14	輕活體菌	全體生命力回復100點	406	25	100	0	0	0	0	0	0	0	0	0
15	活體菌	全體生命力回復200點	407	25	200	0	0	0	0	0	0	0	0	0
16	強活體菌	全體生命力回復300點	408	25	300	0	0	0	0	0	0	0	0	0
17	精煉活體菌	全體生命力回復400點	409	25	400	0	0	0	0	0	0	0	0	0
18	輕超神劑	單人靈動力回復25%	410	14	0	0.25	0	0	0	0	0	0	0	0
19	超神劑	單人靈動力回復50%	411	14	0	0.5	0	0	0	0	0	0	0	0
20	強超神劑	單人靈動力回復75%	412	14	0	0.75	0	0	0	0	0	0	0	0
21	精煉超神劑	單人靈動力回復99%	413	14	0	0.99	0	0	0	0	0	0	0	0
22	輕原靈菌	全體靈動力回復20%	414	25	0	0.2	0	0	0	0	0	0	0	0
23	原靈菌	全體靈動力回復40%	415	25	0	0.4	0	0	0	0	0	0	0	0
24	強原靈菌	全體靈動力回復60%	416	25	0	0.6	0	0	0	0	0	0	0	0
25	精煉原靈菌	全體靈動力回復80%	417	25	0	0.8	0	0	0	0	0	0	0	0
26	輕補魔劑	單人靈動力回復50點	418	14	0	50	0	0	0	0	0	0	0	0
27	補魔劑	單人靈動力回復100點	419	14	0	100	0	0	0	0	0	0	0	0

图 15.23　道具的 Excel 数据

不过，武器和道具的属性设置显得比人物设定简单多了，我们只要在武器上编排一系统的等级，再以等级来区分攻击力的强弱，如果还要再细分武器属性的话，也可以加入武器加强值（除攻击力之外的附加值）、武器防御值（可提升人物防御力）等等，这些附加的属性只要是合理的话，您都可以在这个数据库里加入一些额外的属性。

15.4.3 对象编辑器

接下来，再来介绍一种比较特别的编辑器，称为人物动作编辑器。以 3D 效果为例，它是一种让使用者用来编辑 3D 人物动作所使用的，并将同一种物品或人物的动作数据全部储存在一个档案当中。简单地说，这一档案包含了物品或人物的所有行为动作的数据，而我们就必须使用人物动作编辑器来编排分离这些模型动作的数据，以提供游戏引擎使用，如图 15.24 所示。

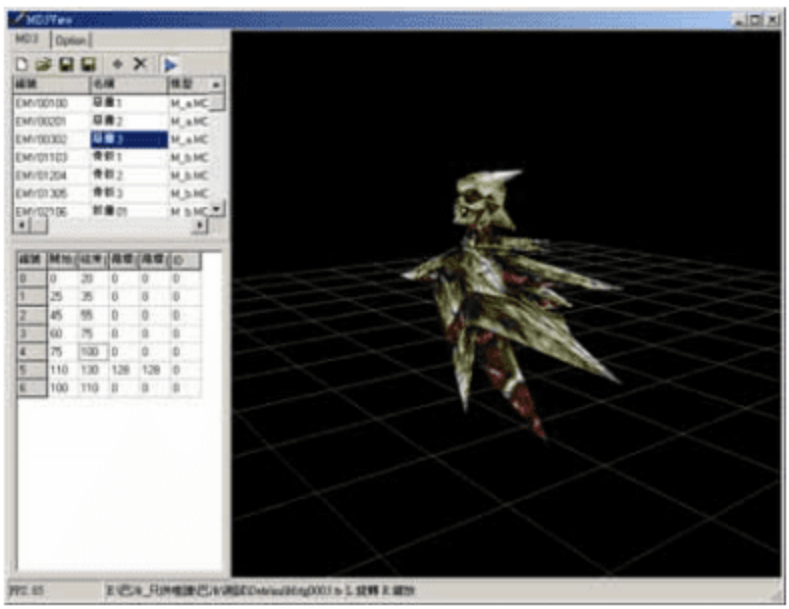


图 15.24　人物动作编辑器

其运作的方式如图 15.25 所示。

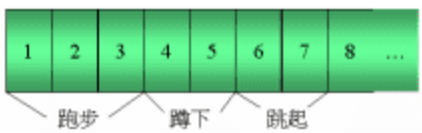


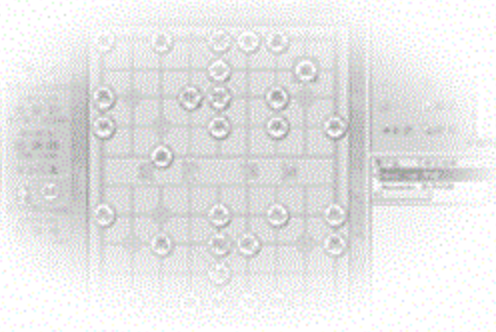
图 15.25　人物动作方式

应该如何分离这种文件中的数据呢？笔者在这里暂且不谈，等到您对 3D 的呈像、演算及运用能够得心应手时，再来研究它也不迟，在这里，您只要了解人物编辑器的原理与运作方式就足够了。

在介绍完游戏中所使用的编辑器后，我们接着来讨论一下研发游戏的成本问题与游戏未来市场的走向。

第 16 章

游戏市场与未来



从今天的游戏市场来看，游戏已经成为一种生财的工具。目前的市场需求与开发游戏的成本控制，似乎很难从这两者中找到一个平衡点，而本章将探讨游戏的现代市场走向与未来发展。

16.1 游戏开发前的思虑

在开发游戏之前，我们需要了解游戏市场的走向、游戏是要给谁玩的、游戏未来的拓展等等因素，这些理论虽然不属于游戏制作的一部分，不过它却是关系到游戏未来发展的关键点，接下来就让我们来介绍这些可以让游戏生存与未来发展的基础理论。

16.1.1 开发游戏的迷失

从现今市面上所发售的游戏来看，我们不难发现，不管是在游戏的画面、玩法或操作等机制上，它们都有着异曲同工之处，不过这些相类似的游戏却只有几套可以在游戏业界上大放异彩，或许您会发现，在这些相类似的游戏中，您可能只会想玩几套较为著名游戏开发公司所发行的游戏，而其他相类似的游戏，您可能只会在卖场上拿起来看看它的游戏盒子而已，除非这些较为不“红”的游戏价格可以低到让您觉得无所谓的话，不然相信您还是会考虑买这些游戏所包含的附加价值吧！

在国内的游戏业界，我们可以发现国产游戏都会存在两种独特的现象，一种是盲目的跟从；另一种是梦想与现实两者都无法兼顾的现象，这种不明确的现象导致国产游戏的销售业绩都一落千丈，以致于国产游戏的品质也越来越不能让国内玩家所接受了。接下来就以这两种特殊的现象来加入分析与评论。

盲目的跟随

国内的游戏业界当中，我们发现了一种很奇特的现象，那就是国产游戏会有一种“盲目追求”国外游戏的感觉，只要是国外某一些游戏在国内发展起来以后，过不了多久就会有许多相类似的国产游戏诞生，但在与国外游戏竞争起来，国产游戏的内容却不及国外的游戏，而这种情况演变到现在，使得玩家有一种特别的观念，那就是“国产游戏不能玩”，这就是所谓“盲目跟从”的后果。

当我们能够将游戏顺利开发完成之后，或许会开始担心，“这一套游戏到底要卖给谁呢？”、“谁会来买我们所制作的游戏呢？”、“我们的游戏是否能够让一般人所接受呢？”，或者实际一点的解释是“这一套游戏能不能为我们带来一点收益呢？”，其实说实在的，我

们都不能预知这些问题的未来，但是在现实生活中，这些问题的确困扰着一个游戏决策者或游戏开发团队。其实以一个成功的决策者或开发团队来看，这些问题早在游戏开发之前，就被这些人详加思考过了，不过以国产的游戏而言，我们在开发游戏的过程中，却很少有人会去注意到它们。

在游戏业界，许多游戏开发团队会出现一种常见的现象，那就是会被游戏市场上当红的游戏所迷惑，并且一头栽进去。以相同的模式，再制作出另一套与当红游戏相类似的游戏，等到游戏一推出市场之后，过不了多久就马上被打得溃不成军，这不仅仅在金钱上或精神上都赔了进去，就连游戏开发团队的士气都被彻底地打败了，最后只有以“收摊”为理由，将游戏开发团队就这么解散了。

以现在市场上的游戏来说，这些相类似的产品通常会被第一套的游戏产品给打败了，或许您会问：“怎么会这样呢？”其实答案非常简单，那是因为在跟进这套当红游戏产品的同时，我们只能依照它们的游戏行为模式来加以仿效，尽管再做出一些与当红游戏不同的地方，不过主架构还是会与这些游戏相似。当游戏制作完毕并且推出上市之后，这种游戏的市场可能已经被第一套游戏给独占光了，而且光看游戏的机制模式与第一套推出的游戏太类似了，可能在游戏上市的第一天就有可能会被一些玩家加以评论比较，假如我们还可以做到与当红游戏非常相似的时候，那么还有可能会被接受，但如果连当红游戏都不及的话，那么游戏在上市不久后一定很快就无疾而终了。

梦想与现实

另外一种在游戏业界中常见的现象，那就是游戏开发团队的迷失。游戏开发团队是一套游戏的主要灵魂，而我们经常都可以看见游戏开发团队想要将自己的游戏梦想推往现实，让自己的游戏理念带进游戏中，每一个人都希望自己的游戏可以去带动游戏产业的风潮，但却不想去完善地规划出游戏制作的流程，只有将自己零散的梦想一点一滴地结合起来，以至于整套游戏只能实现这些一点一滴的梦想，而不能让玩家可以完全地融入到其中，最后却惹来玩家的“满腹臭骂”。

还有另外一种情况，那就是有许多游戏开发团队为了实现自己的游戏梦想，因而不惜开发成本地制作一套游戏，在游戏开发的过程中，一开始可能这些人都抱着崇高的理想沉浸于游戏开发的乐趣之中，不过等到无法负荷现实生活的压力之后，崇高的理想就会慢慢地变成一股沉重的压力，为了不让之前所花费的成本沉于大海，最后只有草草结束当初美丽的梦想。

梦想不等于现实

虽然开发游戏原动力是基于游戏开发团队的梦想，不过在现实生活中，如果要将这种梦想当作一种职业的话，我们只有在梦想与现实之间找到一个“双赢”的平衡点。

16.1.2 游戏开发的关键

在实现制作游戏的梦想之前，首先要试图了解到一些致胜的关键。如果我们想要顺利地完成游戏的开发工作并且上市的话，那么就必要在游戏开发之前考虑其成品到底要卖给谁？换句话说，我们到底是为了哪一些人而制作这一款游戏的？其实答案的关键可以归纳出下列几项原则：

从性别上来划分游戏的类型

在游戏市场中，我们可以将类型划分出男性玩家与女性玩家。一般而言，男性玩家几乎可以涵盖全部游戏类型，这些游戏类型都集中于 SLG、RPG、FPS、RTS、SPG（请参考第 7 章的游戏类型）等方面。至于女性玩家所涵盖的游戏面积就会较男性玩家小，而能让女性玩家所接受的游戏类型大略都集中于 RPG、TAB 等方面。

从年龄上划分不同年龄层的玩家

针对游戏玩家而言，也可以从玩家的年龄上来划分，关于这一点，我们则是考虑到玩家到底能够接受哪一种游戏的类型。其年龄划分如表 16.1 所示。

表16.1 年龄划分

年龄	阶段	说明
14 岁以下	童年期	以小学生到初中二年级以下为主，其家长决策购买为主
14~18 岁	青春前期	以初中三年级到高中生为主，两方式并存
18~22 岁	青春后期	以大学生与就职青年为主，自主决策购买为主
22~25 岁	青年前期	以在职青年为主，自主决策购买为主
25~30 岁	青年中期	以在职青年为主，自主决策购买为主
30 岁以上	青年后期至终	职业形态不定，自主决策购买为主

从收入来源来划分

也可以从玩家的经济来源来简单地划分出“学生”与“上班族”两种玩家。其玩家的特性如下列所示：

- 学生玩家：没有自主收入来源，他们主要是靠家庭供给来决策是否购买游戏。
- 上班族玩家：有自主收入来源，自主决策购买。

从文化程度来划分

也可以从玩家的文化程度来划分游戏的族群。我们可以将玩家的文化程序划分成“初等学历”、“中等学历”、“高等学历及以上”3 种。其特点如表 16.2 所示。

表16.2　文化程序划分

文化学历	说明
初等学历	初中以下文化程度，消费认知力较低
中等学历	初中至高中（中专）文化程度，消费认知力中等
高等学历及以上	大专及以上文化程度，消费认知力较高

从消费心理来划分

基本上，从消费者的心态上可以简单地划分出“保守型”、“冲动型”及“理智型”3 种。其特点如表 16.3 所示。

表16.3　消费心理

消费心理	说明
保守型	消费欲望低，注重产品售后等外延
冲动型	消费欲望高，注重产品观感与现场服务
理智型	消费欲望中等，注重产品性价比

从硬件环境来划分

以玩家的硬设备而言，我们也可以简单地划分出“原主流机种”、“现主流机种”及“高配置机种”3 种。其各特点如表 16.4 所示。

表16.4　硬件环境

硬件环境	说明
原主流机种	PII 800 及以下机种，部分无多媒体配置
现主流机种	PIII 500~800 之间机种，多媒体配置
高配置机种	P4 1G 以上机种，多媒体配置

从软件环境来划分

我们也可以从玩家的软件环境简单地划分出“DOS 玩家”、“Windows 玩家”及“Linux 玩家”。其特点如表 16.5 所示。

表16.5 软件环境

软件环境	说明
DOS 玩家	硬件配置一般较低，多为文字输出平台
Windows 玩家	硬件配置不等，主流软件环境
Linux 玩家	硬件配置不等，目前尚未流行于一般用户上

从接触程度来划分

从玩家所认知的程度，可以简单地划分出“普通用户”、“进阶玩家”及“熟练玩家”3 种类型。其特点如表 16.6 所示。

表16.6 接触程度

接触程度	说明
普通用户	游戏与计算机的知识匮乏，尚未实际接触游戏的潜在玩家
进阶玩家	初步掌握游戏知识，对简单操控的游戏刚刚上手的现实玩家
熟练玩家	熟悉游戏知识，能够熟练操作游戏与计算机的玩家

从购买模式来划分

一般而言，我们也可以从玩家的购买模式简单地划分出“试用版玩家用户”、“正式版玩家用户”及“综合玩家用户”3 种类型。其各特点如表 16.7 所示。

表16.7 购买模式

购买模式	说明
试用版玩家用户	一般试用版消费为主体。如杂志中取得
正式版玩家用户	一般以正式版消费为主体
综合玩家用户	此类玩家会兼具正式版与盗版对等的消费

从娱乐环境来划分

从玩家的娱乐环境中也可以简单地划分出“家庭娱乐型”、“办公室娱乐型”及“网吧娱乐型”3 种。其各特点如表 16.8 所示。

表16.8　娱乐环境

娱乐环境	说明
家庭娱乐	自有主机，游戏类型不定
办公室娱乐	公用主机，以消遣娱乐性游戏类型为主
网吧娱乐	公用主机，以竞技对抗性游戏类型为主

从游戏形式来划分

游戏形式上也可以简单地划分出“单机型”、“局域网络型”及“因特网型”3 种。其各特点如表 16.9 所示。

表16.9　游戏形式

游戏形式	说明
单机型	主要在单机上进行游戏，涵盖各种游戏类型
局域网络型	主要在局域网络上进行游戏，竞技类游戏为主
互联网络型	主要在互联网络上进行游戏，竞技类游戏为主

从游戏目的来划分

针对游戏的目的而言，也可以简单地划分出“操控型”、“掌握型”、“剧情型”、“娱乐型”及“综合型”5 种。其特点如表 16.10 所示。

表16.10　游戏目的

游戏目的	说明
操控型	主要在游戏中热衷操作与控制的乐趣，如 ACT、SPG、FPS 等
掌握型	主要在游戏中热衷发展与征服的乐趣，如 SLG、RTS 等
剧情型	主要在游戏中热衷故事与养成的乐趣，如 RPG 等
娱乐型	主要在游戏中热衷消遣与放松的乐趣，如 TAB 等
综合型	以上两种或数种的综合类型

从游戏人格来划分

从游戏人格上可以简单地分出“焦躁型”、“冷静型”、“平衡型”及“多变型”4 种。其特点如表 16.11 所示。

表16.11 游戏人格划分

游戏人格	说明
焦躁型	在游戏中较容易冲动，喜欢探索新领域与自我表现，较注重游戏声光与操作
冷静型	游戏行为较为保守，喜欢研究系统与剧情，较注重游戏主题与系统
平衡型	处于焦躁型与冷静型的中间状态并保持持续稳定
多变型	不同时间、地点中游戏人格多变，难以作统一结论

综上所述，玩家群体可以从多方面来归类分析，上面是以简单的类型来区分游戏玩家，事实上，我们还可以更加详细地继续深入分析，例如以因特网的玩家也可以划分出“连接速率”、“连接形式”、“单位时间”及“游戏量”等类别。概括地讲，我们可以在玩家所界定的范围的建立起相互关联的关系。

上述游戏类型划分只供读者参考，其实只要能够确认玩家的喜好、消费能力及市场等等关键因素，并且结合游戏制作的理念，再加上开发成本的许可，便可以利用这几项原则与有效的资源来制作一款高品质、低成本、有利益的游戏。

在游戏业界，游戏未来的展望是许多人所追求的目标，为了达到这种目标，在游戏产业中，有许多专家和学者也正在分析这种相互性的关系。在下一节中，我们来介绍一些游戏未来的展望与游戏生态环境的行为。

16.2 游戏未来的展望

在科技日新月异的今天，不经意间，游戏产业已经步入一个群雄逐鹿的时代。在短短的 40 余年中，游戏已经融入了电影、音乐、文学等艺术形态的呈现方式，并且创造出一个新的繁荣世界。不过，值得我们去思考的是游戏又将面临着一个什么样的未来呢？而在游戏玩家的掌握中，它又将呈现哪一种色彩的梦幻世界呢？依照目前的游戏业界发展来看，游戏产业将展现 5 种大变化的趋势，这种趋势可以分成“游戏类型的突破”、“游戏网络化”、“游戏的多重触觉”、“互动性的突破”及“游戏的虚拟现实”5 种。

接下来就以游戏产业未来的这 5 大趋势来详细地分析与讨论。

16.2.1 游戏类型的突破

在游戏产业界日渐模糊的情况下，伴随着游戏产业在游戏类型领域上所能突破的局限。曾几何时，游戏玩家会为了电子游戏的优劣性争得你死我活，例如玩家讨论的究竟是“实时战略”（RTS）游戏为优呢？还是“第一人称射击”（FPS）游戏能够成为最具有代表性呢？这种议题同样在网络公布栏上经常被讨论得不亦乐乎。事实上，这都要归根于游戏尚未步入一个成熟阶段。

从去年最能牵动游戏产业神经的 Sony PS2 与微软 XBOX 计划显示看来，未来的游戏平台将打破 PC 与 TV 的界限，而成为另一种游戏功能、播放器、网络浏览与互动电视于一体的多媒体平台。我们之所以会如此断言，那是因为如同微软的 TV 平台游戏机 XBOX 所拥有的 PIII CPU、Windows 2000 的操作系统、DirectX 9.0 游戏开发工具以及网络接口，它们已经实现了与 PC 游戏资源互换的理念。在这里特别要提到的是，XBOX 的开发工具包——DirectX 接口，PC 游戏产业的龙头老大——微软在历经这么多年来，有许多的 PC 游戏上的大作都是通过 DirectX 所编写出来的。而 XBOX 的强劲对手——Sony 的 PS2 也同样决非等闲之辈，Sony 的 PS2 它本来就是一款具有 3C（Computer、Communication、Consumer Electronics，即为计算机、通讯和消费性电子）的产品。如今，尽管 PC 游戏还尚处于一个 CD-ROM 的时代，不过，我们却已经将 DVD 产业的优异性发挥得淋漓尽致了。

近几年来，PC 游戏与 TV 游戏的相互移植性也越来越成熟了，从玩家良好的反应与移植作品的销售量来看，它们不断地在说明双方对于彼此之间的技术基础已经是越来越成熟了，所以游戏产业的界限也就慢慢地消失于无形中了。虽然它们之间的兼容性还有一些值得商讨的地方，不过 PC 与 TV 游戏的统一化则是迟早的事。同时，我们也看到游戏类型之间的差异正在做互补，而且渐渐地在取代传统的游戏类型，如同 Blizzard 公司所出的“魔兽争霸 3”游戏，它就已经全然地突破 RTS 游戏类型的传统理念，而且导入了大量 RPG 类型的要素，例如以拥有特技能力的英雄来指挥队伍作战，并且取代原先建筑物补给的概念。

又如同 Playnet 公司所出的 World War II Online（二战在线），它虽然是一款以战棋为玩法的游戏，不过游戏玩家却几乎可以利用第一人称的表现方式来扮演任何的兵种角色，无论是步兵、坦克手还是飞行员等等，所以我们怎么又能够断定这究竟是策略模拟类型的游戏呢？还是第一人称射击（FPS）类型的游戏呢？又如同赛车类型（RAC）游戏也有角色扮演（RPG）化的倾向，例如赛车手的成长模式、购置设备来升级车辆，以及参加赛事来取得金钱等的理念，这让传统 RAC 游戏类型呈现出焕然一新的感觉。游戏类型领域的突破好像是万花筒一样，它们的相互组合变幻无穷，这也让玩家们感受到另外一种更为异彩缤纷的虚拟世界。

16.2.2 游戏网络化

1997年，由美商艺电所设计发行的“网络创世纪”（Ultima Online）来看，玩家在感到新鲜之余时，他们仍对网络RPG的游戏理念感到不以为然，而现在网络游戏的脚印早就已经踏在游戏业界的每一寸土地上，这不仅开辟了另一个热门游戏的讨论话题，而且就连著名的几家游戏开发商也开始陆陆续续地在跟进。网络化是游戏技术发展的趋势，就连XBOX与PS2等游戏平台也都在争夺这一块网络游戏的大肥肉，而PC上的游戏则更没有理由坐失网络化的机遇。

我们先暂且不谈网络RPG如日中天的力量，就其他类型网络化的游戏来看，几家著名的游戏开发商也已经在大张旗鼓地进行了。网络游戏是玩家的福音，也是游戏开发商利润的源泉，如果我们告诉您，一套网络游戏一天的收入是80万的话，那么您还会认为游戏还只是个游戏吗？

16.2.3 游戏的多重触觉

现今的游戏玩家已经不能再满足于键盘与鼠标的原始操作模式了，玩家追求的是视觉与听觉感受是否能够更上一层楼，然而基于这一种玩家高享受的因素驱使之下，越来越多的游戏正在朝向这些高感受领域迈进。例如触觉感受、运动感受，甚至尔后的味觉感受与嗅觉感受等等，这都是将来游戏所发展的趋势。如同玩家所熟悉的力回馈手把与方向盘和一些街机游戏，如跳舞机游戏，玩家只要在主机上的踏板上踩出一系列的节奏，游戏里的虚拟人物也会依照玩家所踏出的角度、力道与综合其他因素而跳出有规律的舞步了，这类游戏更是综合了触觉、运动与力感的多重触觉性感观。

在不久的将来，玩家有可能会面对更先进的VR设备，如数字神经系统，它可以将我们带进游戏的虚拟世界中，而玩家便能够在游戏中扮演起主角的角色，如此一来，如果到那时候再来玩“恶灵古堡”之类的游戏时，相信一定会被吓破胆不可。游戏产业就是离技术领域这么近，以至于它的发展潜力就像科技一样的灿烂。

16.2.4 互动性的突破

计算机游戏追根到底，它就是人与机器之间的互动，而游戏网络化其实也就是以因特网为媒介，并且构成人与人之间的互动。以一款游戏的互动性而言，它并非是一个具体的要素，而它是完全能够贯穿于游戏的整体架构上，它只是透过一些基本的要素（如图像、音乐、音效、人工智能等）为主体而渗透到玩家的每一个层面之中。由于游戏朝着网络化方向发展，它也就朝着人与人之间互动的大方向迈进，因而在不久的将来，这些游戏依赖了几十年的要素，如人物对话的剧本和NPC等就不会再成为构成游戏的必要条件了。

过去这些基本的游戏要素都是玩家借以获取信息的主要来源，不过在一个具备完全互动性的游戏中，可能还会在网络的另一端遇到一个“真人”，而这个“真人”也可以告诉我们一些游戏中的信息，而网络上的每一个人都是“玩家”，而且每一个人也都是 NPC。当我们在与“真人”对话交流的时候，还需要预设的剧本吗？而且玩家所扮演的角色都具有独自的思考、语言、善恶等方面的特性差异，这与前面具有特定剧本与结局的游戏模式来说，无疑地增加了“不可完成性”的机制，如果玩家要在这样一个互动环境中求得生存发展的话，他就非得做出与现实相同的一切，如人与人的沟通、自我魅力的提升与人格的发展等等，这也就赋予未来游戏近乎无限的表现空间，因此游戏也表现出真正的“生活”。

在未来一个完全交互式的游戏中，玩家可以扮演起任何类型的角色，体验任何一种角色的生活形态，而这就是网络游戏所要表现的一种精神，游戏制作者只要提供一个剧情的大环境、世界观与时代背景等方面广义的限制，玩家就可以在游戏中任意地驰骋在这一片天地之间。举例来说，在一款实时战略的游戏中，我们可以通过第三人称的视角来掌控全局，也可以通过第一人称的视角来扮演一名带领一支队伍直捣黄龙的角色，又可以扮演为一名弑杀者，在杀戮快感中感受另一番乐趣。现在的 RPG 游戏中也已经允许玩家自由决定主角的善恶和成长的历程，在互动性的召唤下，RPG 所诠释的空间仍然可以大大拓宽。

16.2.5 游戏的虚拟现实

在现在的游戏观念中，玩家都想在游戏中追求真实性，而真实性便成为游戏制作者最想要达到的目标，这是玩家所期待的，更是游戏产业得以向未来迈进的原动力。

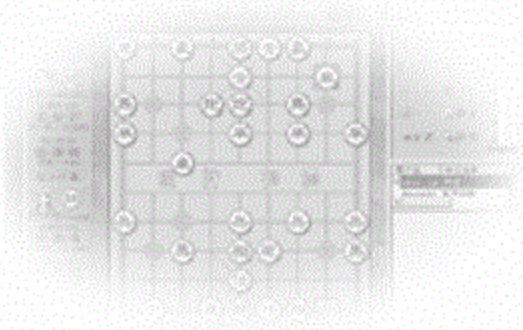
如果说过去真实性的评判标准是来自于游戏中 3D 模型网格数目的多少、画面色调的丰富性与否、阴影的真实变化、纹理贴图是否细腻等因素的话，那么未来游戏在于建构其真实的内涵上，无疑要比过去还要丰富许多。单单以一个角色的脸部来说，如 TECMO 公司所出的 DEAD OR ALIVE 3（生死格斗 3）游戏一样，它能够生动地呈现出 3D 伸缩技术效果的面部表情（Facial Animation），还有 Sony PS2 作为自己最大卖点之一的情感引擎（Emotion Engine，EE），更是能够赋予游戏中角色这种看不见、摸不着的情感表现，而游戏业界在过去却无法表现这些人物的真实感，然而这就是我们所要努力的目标。

在 E3 电玩展中获得“最有希望游戏”大奖的作品 Halo（星河奇兵）身上，我们更可以看到各种以 VR 为终极目的技术的完美结合，如倒转运动原理、多路纹理绘图、多面体比例缩放，等积光影、像素反射以及重力、风力风向等现实因素的模拟。可以肯定的是在未来的游戏中，VR 依然是一种理想的目标，一种可以接近现实的梦。

不管是在视觉上、听觉上，甚至在触觉上，现在的游戏都已经可以做到与现实生活非常类似的界限了，而随着科技日新月异的变化，游戏也顺势带领了人们走向虚拟的世界，它更展现了无穷魅力与希望。

第17章

地图编辑器



在此我们来介绍配套光盘中 MapEditor.exe 地图编辑器的基本操作。图 A.1 是执行地图编辑器时的初始画面。

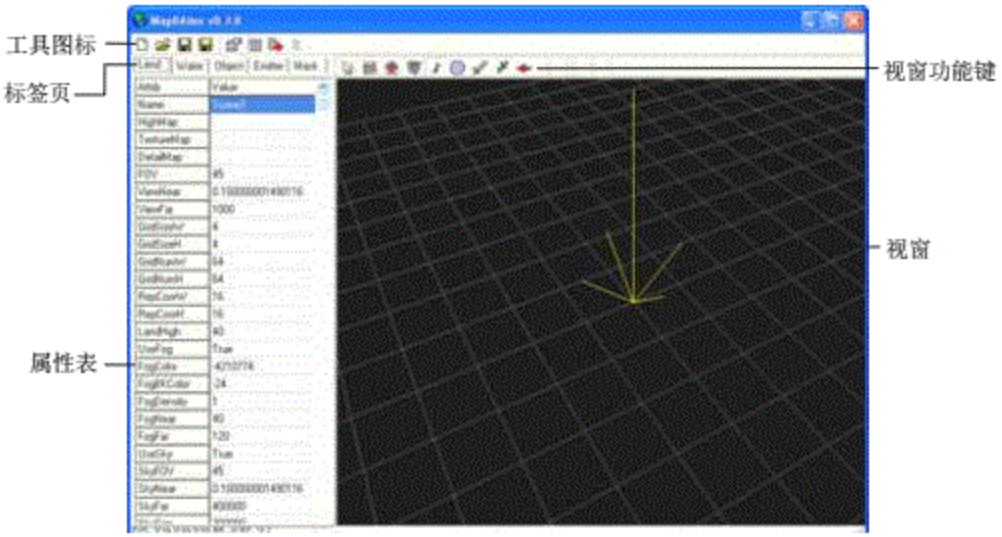


图 A.1　地图编辑器

接下来，我们来设计一个简单的 3D 地形。

（1）开始建立地形前，我们要先设定建立它的黑白图。在图 A.2 中双击 HighMap，打开“开启”对话框，如图 A.3 所示，选择地形图。此时视窗如图 A.4 所示。

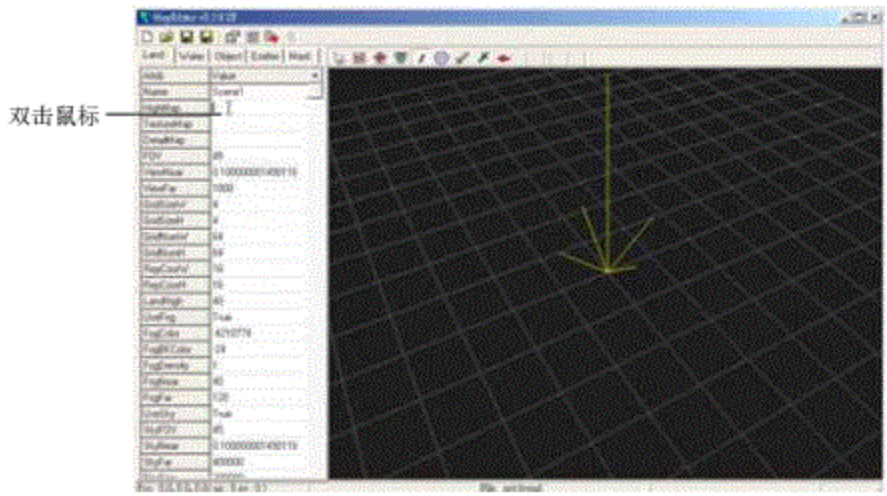


图 A.2　设置新建地形

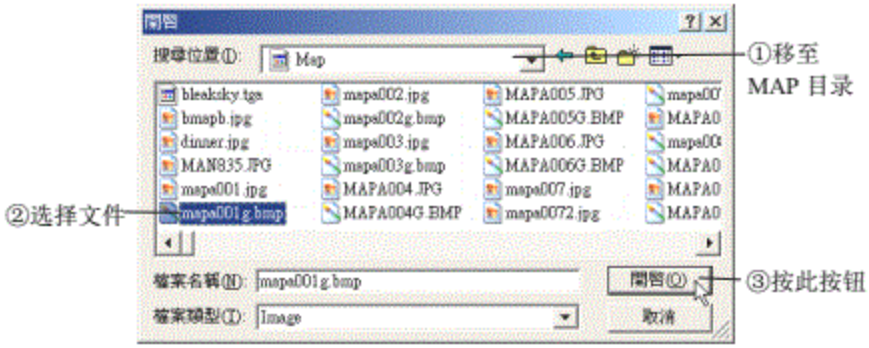


图 A.3　打开图片

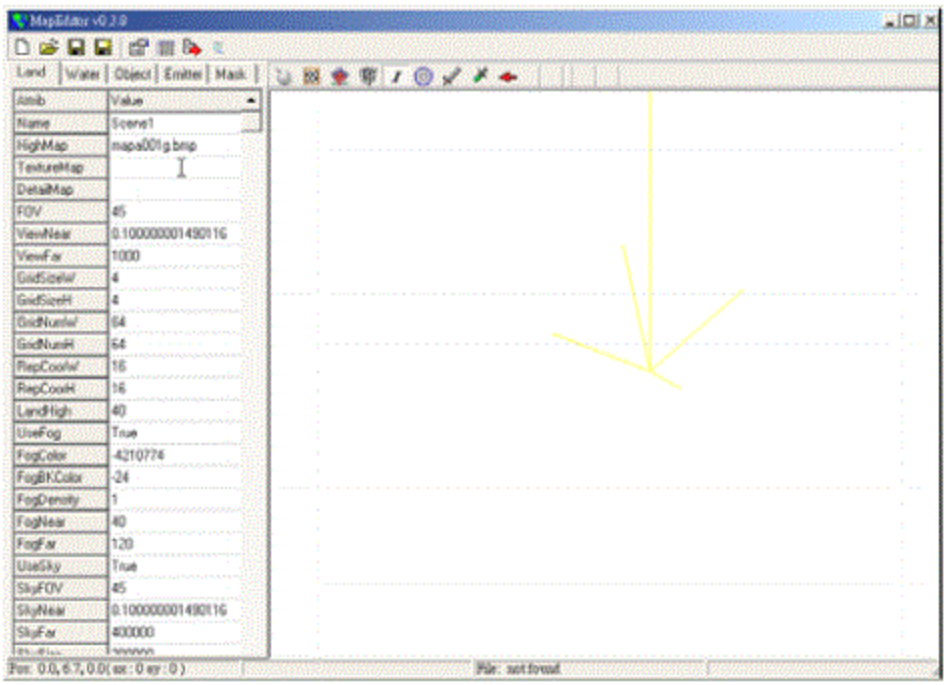


图 A.4　视窗变成白色的

（2）图 A.4 的窗口会变成白色，那是因为还没有加入地形的材质所引起的。接下来我们可以开始设置地形的材质。在图 A.5 中双击 TextureMap，则弹出图 A.6 所示对话框，选择图片，则在视窗出现地形，如图 A.7 所示。

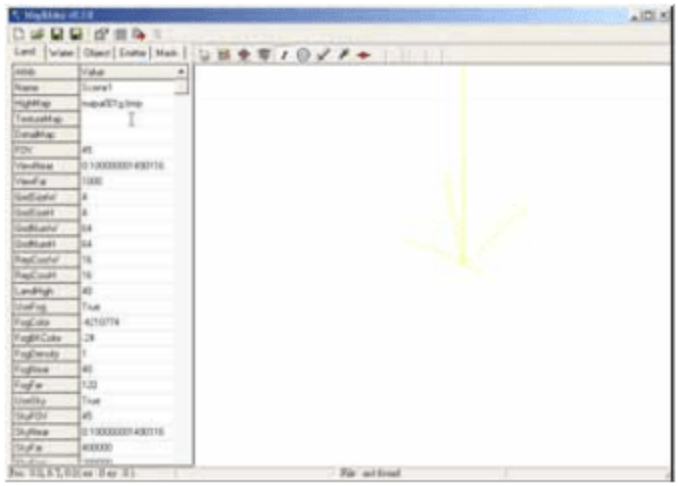


图 A.5　双击 TextureMap

图 A.6　“开启”对话框

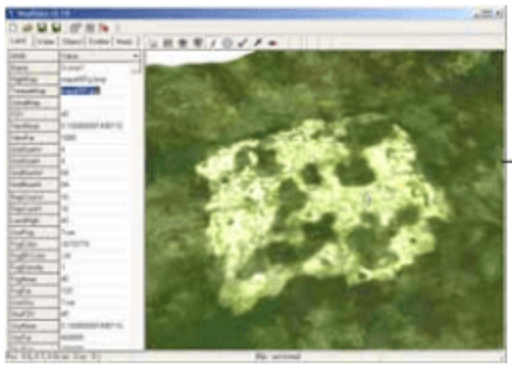


图 A.7　打开地形图

（3）下面我们开始在初始的地形上建立几个对象，例如房屋、石头、树木、花草等等。

调整地形位置如图 A.8 所示，然后单击 Object 选项，则弹出 Object 选项卡，如图 A.9 所示，单击“+”按钮，则弹出 A.10 所示对话框，选择文件，则在图 A.11 中的 File 栏出现打开文件的名称，此时出现一个房子对象，如图 A.12 所示。然后调整房屋视线，如图 A.13 所示。

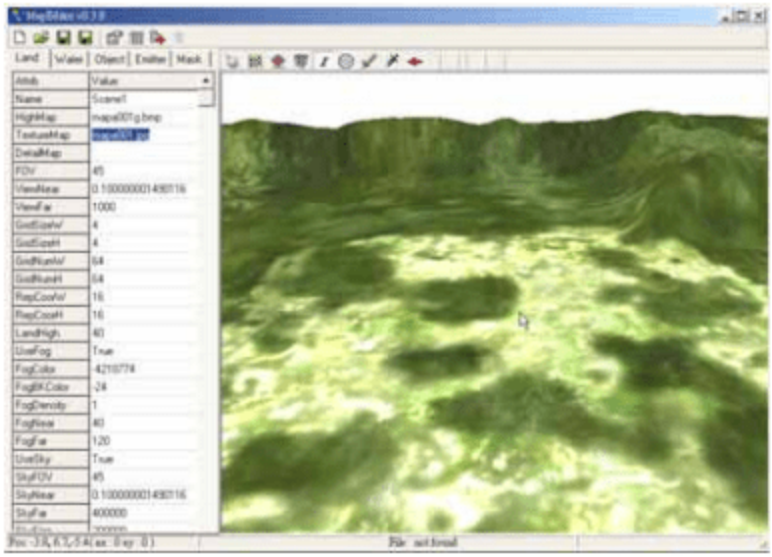


图 A.8　利用鼠标的左键（平移）、右键（旋转）、左右键一起按（远近）将地形拖至适当的位置

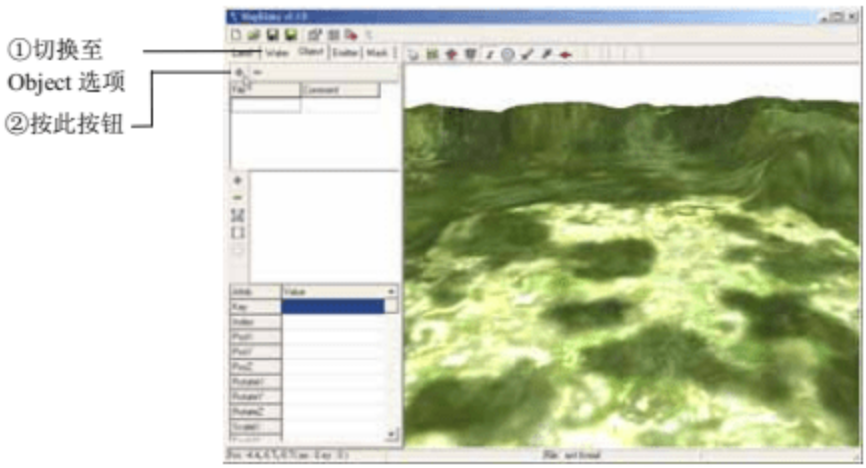


图 A.9　Object 选项卡

图 A.10　“开启”对话框

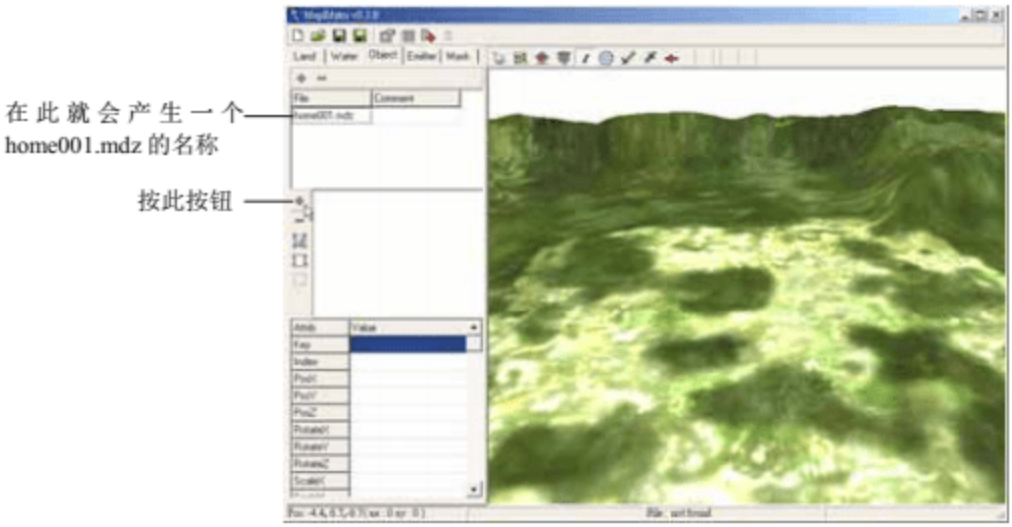


图 A.11　打开 home001.mdz 文件

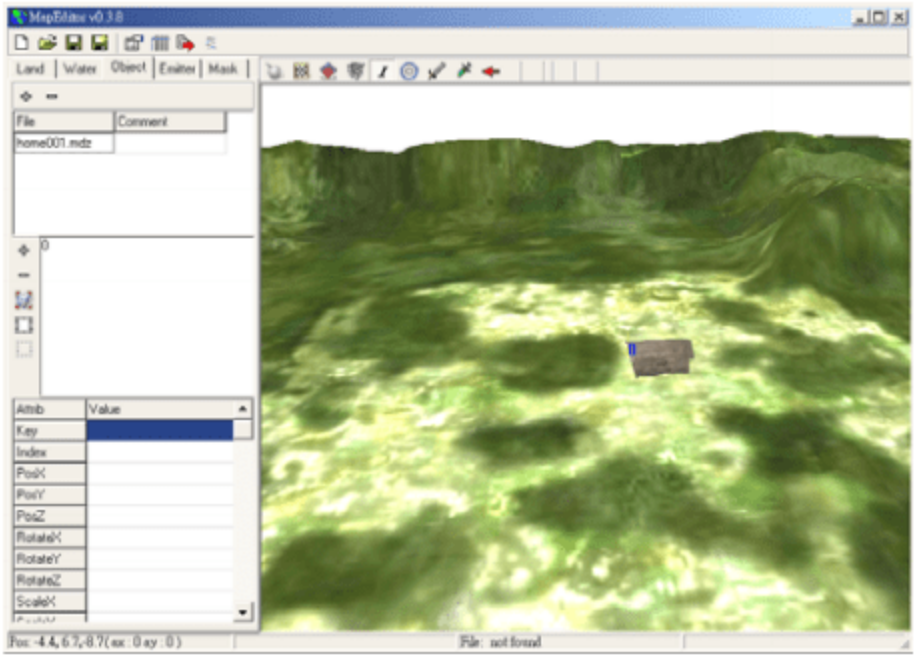
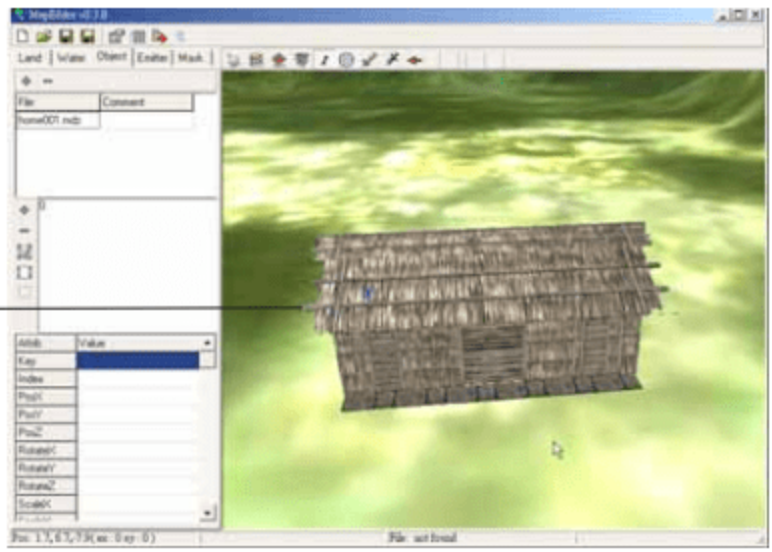


图 A.12　产生一个房子的对象



我们利用鼠标的左键、右键及左右键一起按，将视线拉至适当的距离及角度

图 A.13　我们利用鼠标的左键、右键、左右键一起按将视线拉至房屋处

（4）接下来，我们来调整一下房子的位置与方式。在图 A.14 中选择“+”下的“0”，则房屋会被选中。在图 A.15 中调整 RotateX，RotateY，RotateZ 的值旋转房子。

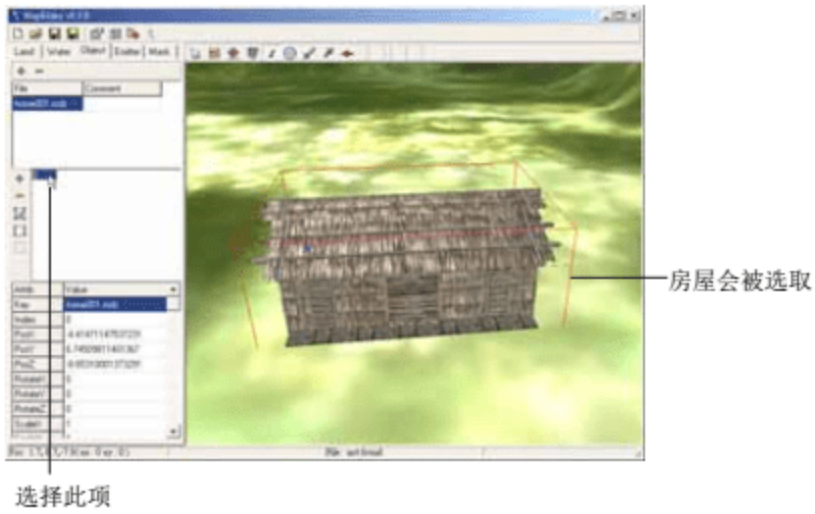


图 A.14　选中房屋

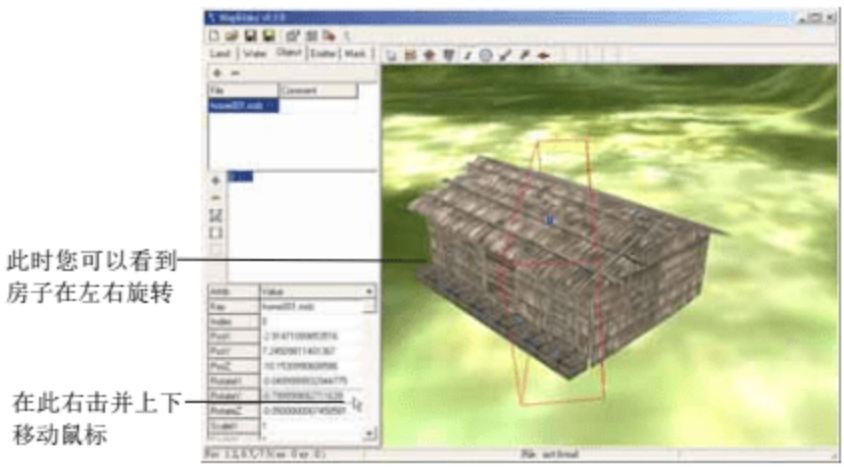


图 A.15　旋转房子

（5）在地面的对象配置完毕之后（您可以利用上面所介绍的操作步骤再增加其他对象），接下来便是将对象设定屏蔽（屏蔽是在游戏中人物可走与不可走的依据）。

在图 A.16 中单击　按钮，将视点移至视图；选择 Mask 选项，在图 A.17 中设置 MaskW，MaskH 的值，则视窗中出现方块屏蔽，如图 A.18 所示；用鼠标右键调整视点位置，并设置屏蔽颜色，如图 A.19 所示；然后按住 Ctrl 键移动鼠标将屏蔽清除，如图 A.20 所示。

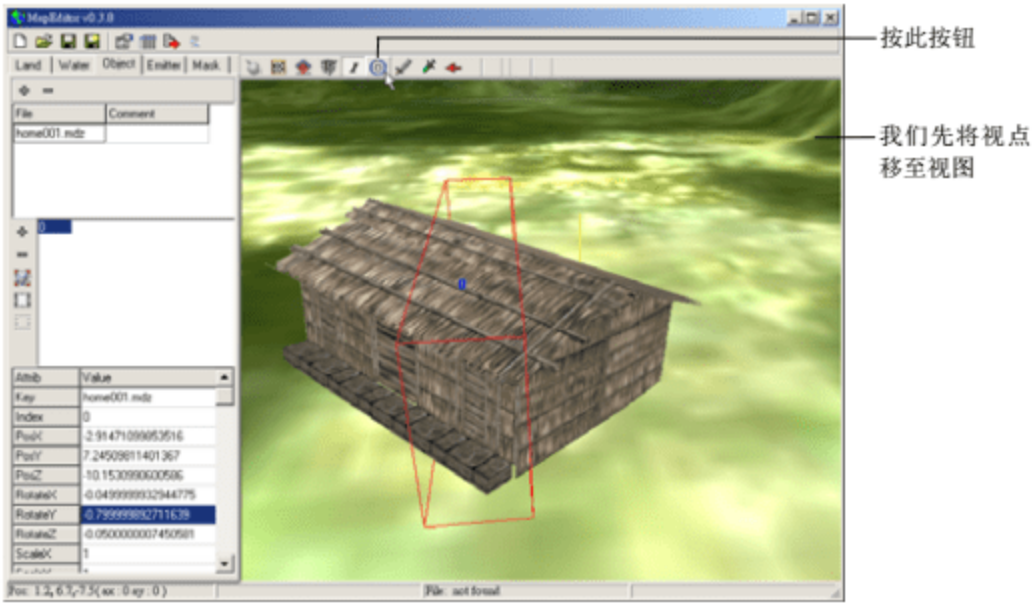


图 A.16　将视点移至视图

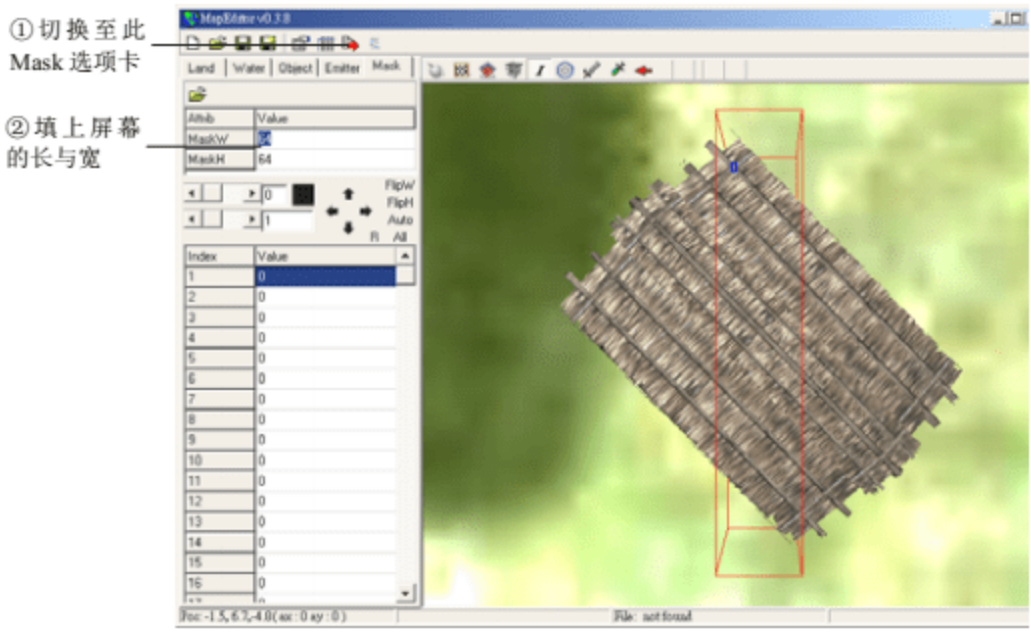


图 A.17　设置 MaskW，MaskH 的值

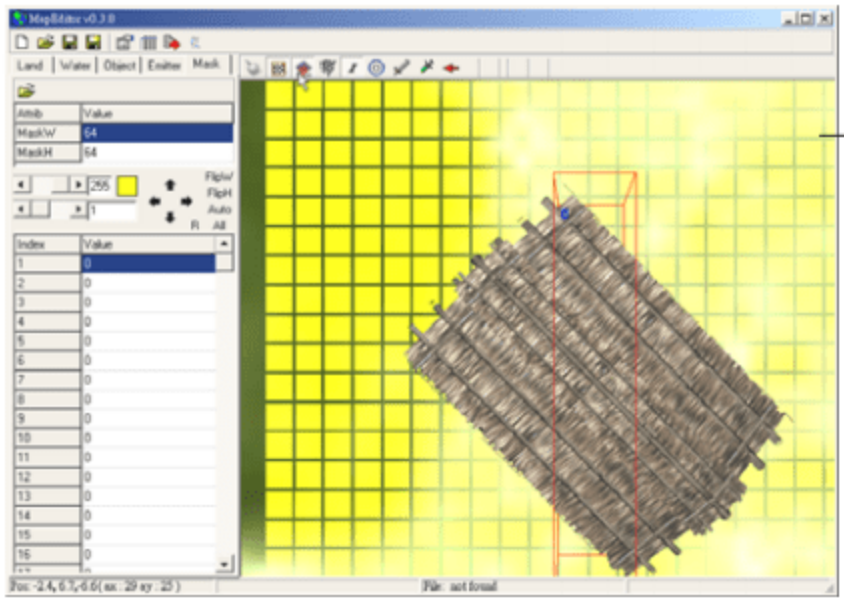


图 A.18　视窗中出现方块屏蔽

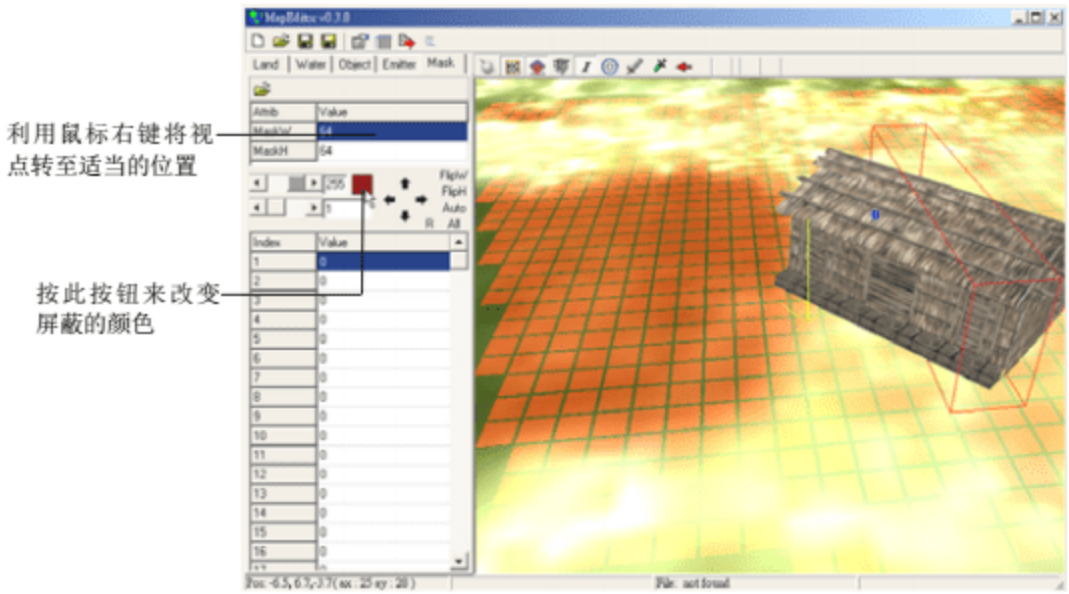


图 A.19　设置屏蔽颜色

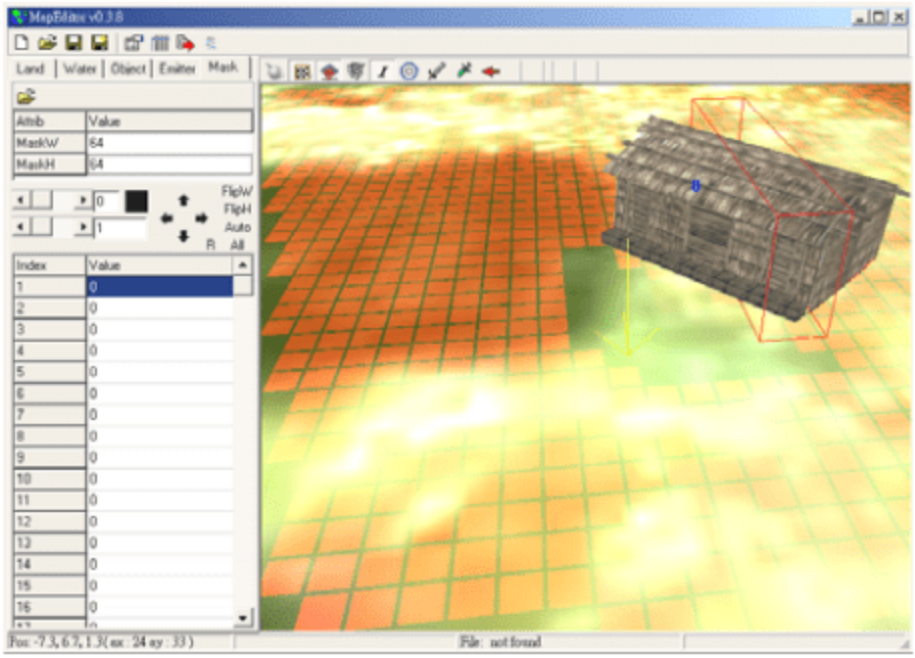
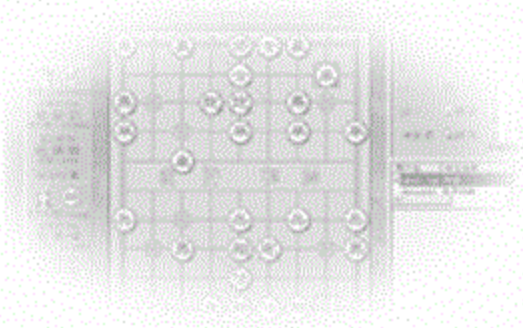


图 A.20　按住 Ctrl 键并移动鼠标（或按上、下、左、右键）将屏蔽清除

按上面步骤，您便可简单的在 3D 地形上建立一个房子的对象。接下来，您还可以利用配套光盘中，选择其他的对象来摆设在地形中。

第18章

粒子编辑器



粒子编辑器用来编辑游戏中的所有粒子特效。有了它，程序设计师便不用再大费周章编写一大堆的程序代码，来满足游戏的需求。接下来，我们利用粒子编辑器来简单地设计一个小型的粒子特效。

首先请您先执行配套光盘中的 PVIEW.exe 程序，接着按照下面的介绍来设计一个粒子特效。

（1）先创建子粒子树状表。双击图 B.1 中按钮，则出现图 B.2 中的粒子树状表。

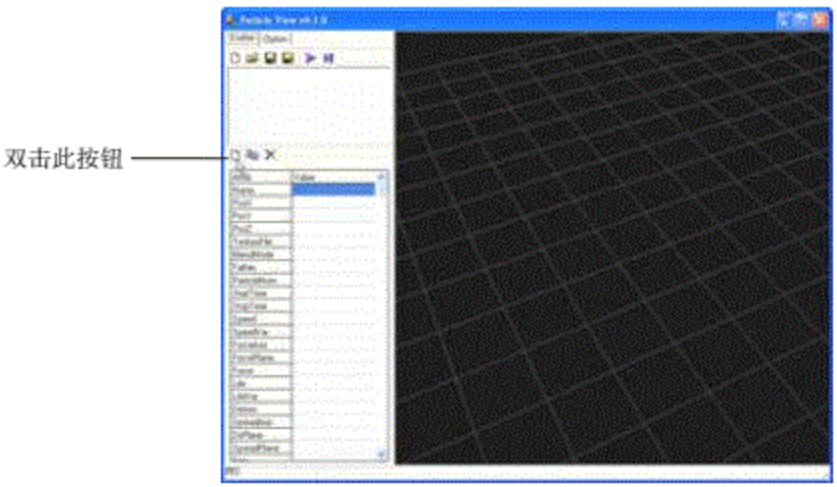


图 B.1　双击按钮

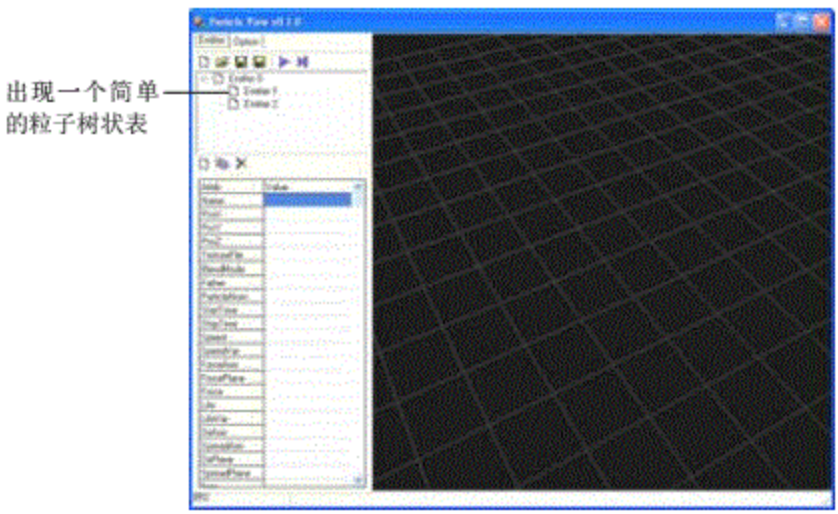


图 B.2　出现一个简单粒子树状表

（2）接下来，开始设定 Emitter0 与 Emitter1 的粒子属性。

选择 Emitter1 项，双击 BlendMode 项，如图 B.3 所示；则弹出如图 B.4 所示的对话框。

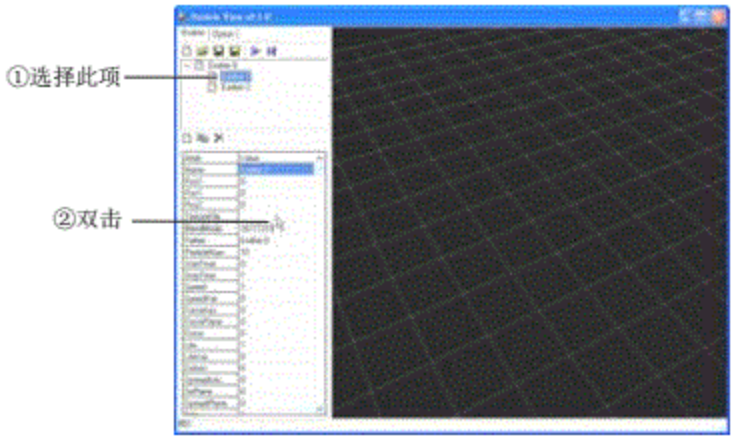


图 B.3　双击 BlendMode 项

图 B.4　“开启”对话框

（3）接着我们将 Emitter1 的属性值设定成如表 B.1 所示。

表B.1　Emitter1属性值

属性	参数值	说明
Speed	0.7	粒子速度
SpeedVar	0.5	粒子速度的变量
Life	1.0	粒子生命值
Life	0.5	粒子生命值的变量
DirAxis	90	移动方向

（续表）

属性	参数值	说明
	其他属性不变	

（4）然后按下粒子编辑器上的播放按钮，如图 B.5 所示。

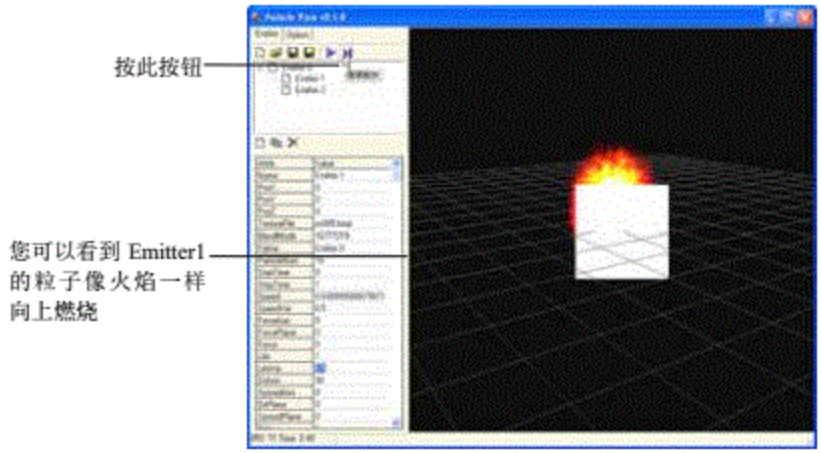


图 B.5　播放粒子效果

（5）在图 B.5 中，您会发现，图中有一个白色块状的物体在移动，其实那是 Emitter0 的粒子，只是我们还没设定，所以看不出什么所以然。在此您必须再设定 Emitter0 的属性，如图 B.6 和图 B.7 所示。

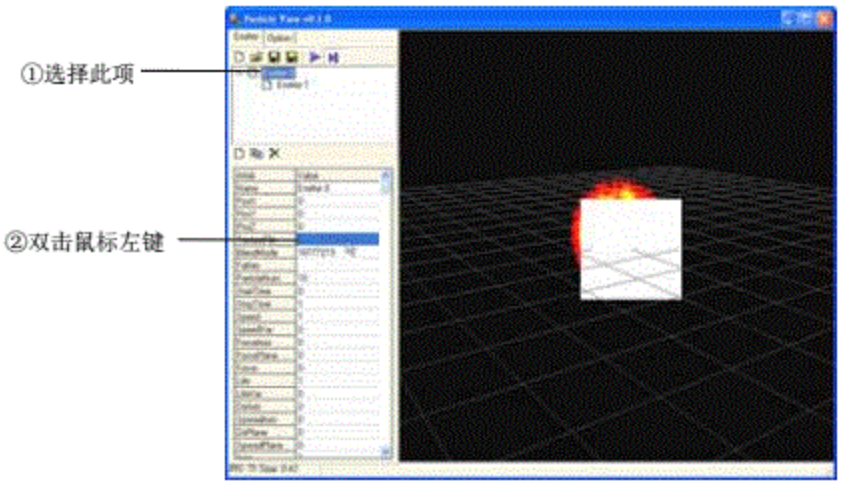


图 B.6　设置 Emitter0 的属性

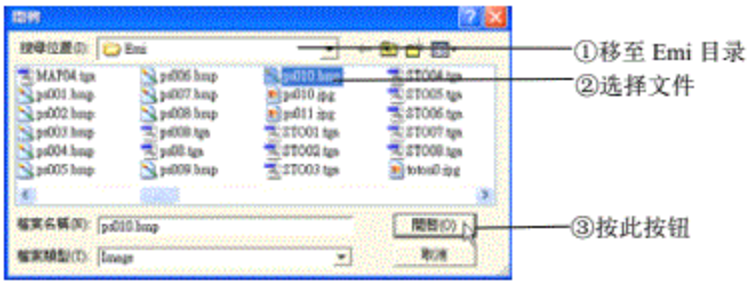


图 B.7 “开启”对话框

（6）将 Emitter0 的属性值设置成如表 B.2 所示。

表B.2 Emitter0属性值

属性	参数值	说明
ParticleNum	30	粒子数
Speed	3.0	粒子速度
SpeedVar	2.0	粒子速度的变量
Life	1.8	粒子生命值
Life	1.5	粒子生命值的变量
DirAxis	90	移动方向
SpeadAxis	180	移动范围
其他属性不变		

（7）如此一来，您将会看到如图 B.8 所示的粒子特效了。

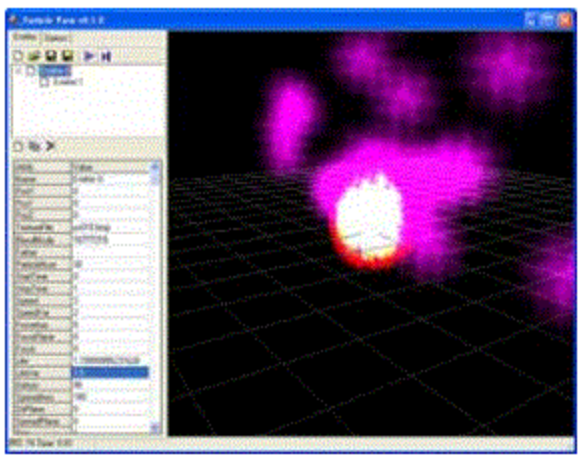
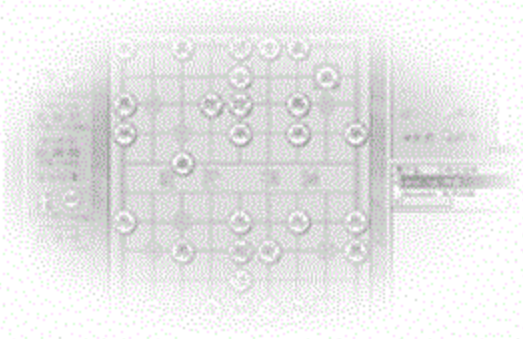


图 B.8 粒子效果

附　录

人物动作编辑器



人物动作编辑器主要是用来编辑 3D 人物的动作。在 MD3 格式中，它可以将人物的所有动作都保存在一个文件中（关于 MD3 格式请参阅其他相关书籍），而人物动作编辑器则是将这种动作加以分类。

接着来试试人物动作编辑器的基本功能。首先请执行配套光盘中的 MD3View.exe 程序，MD3View 初始界面如图 C.1 所示，单击“打开”工具图标，则弹出“开启”对话框。

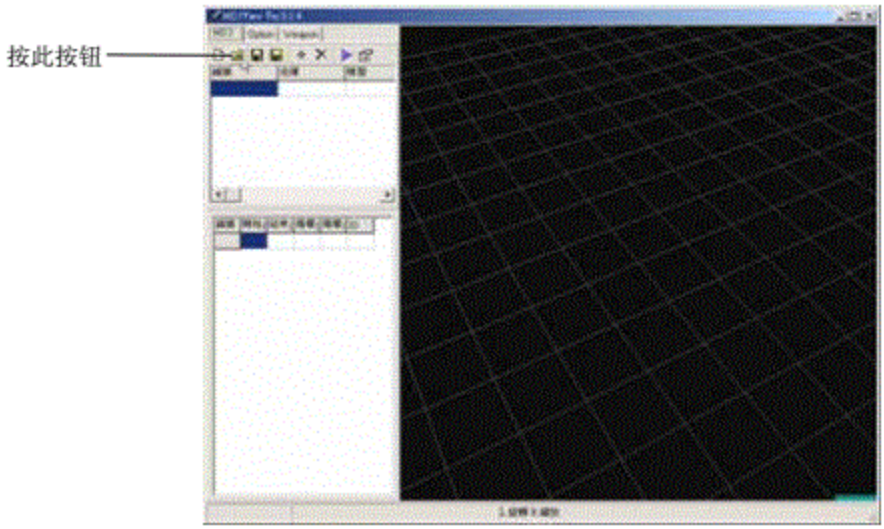


图 C.1　初始界面

图 C.2　“开启”对话框

如此一来，您便能看到人物的 3D 模型了。在图 C.3 中怎么没看见呢？其实刚读进来的 3D 人物模型，是因为摄影机拉的太近了，所以什么都看不见。

在此是读取人物属性设置，而不是直接打开 MD3 格式文件。

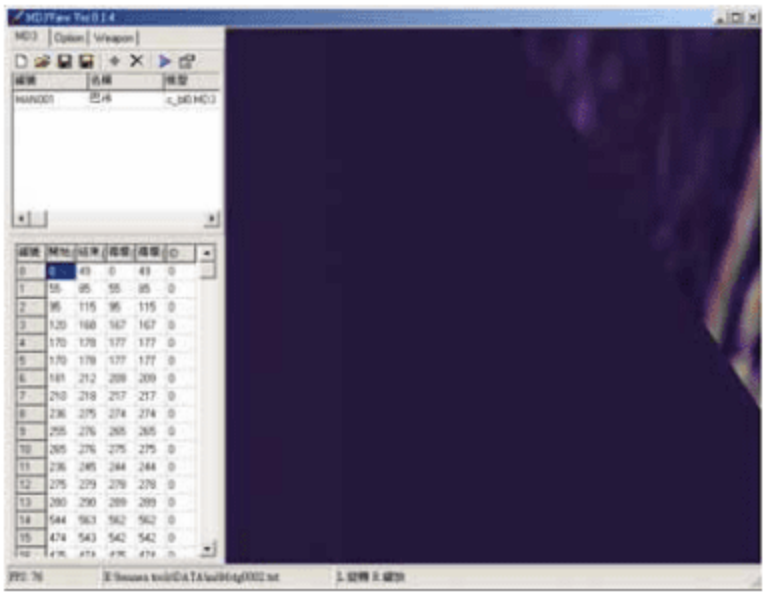


图 C.3　显示 3D 人物模型

注意，如果您还是看不到任何模型，那可能是模型文件与材质文件没有连接起来，请在模型与材质栏中上双击，并重新读取图 C.4 中的文件（文件在 md3 目录中）。

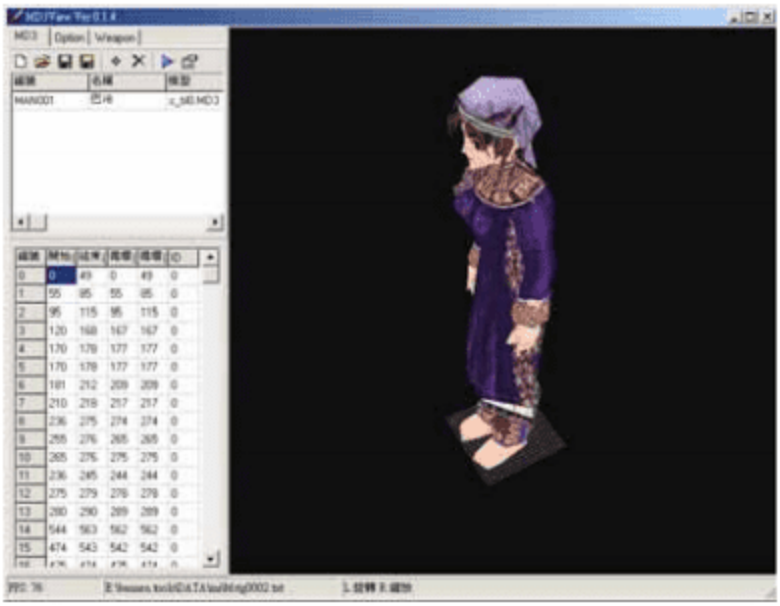


图 C.4　您可以按下鼠标左键或右键来调整摄影机的角度与位置

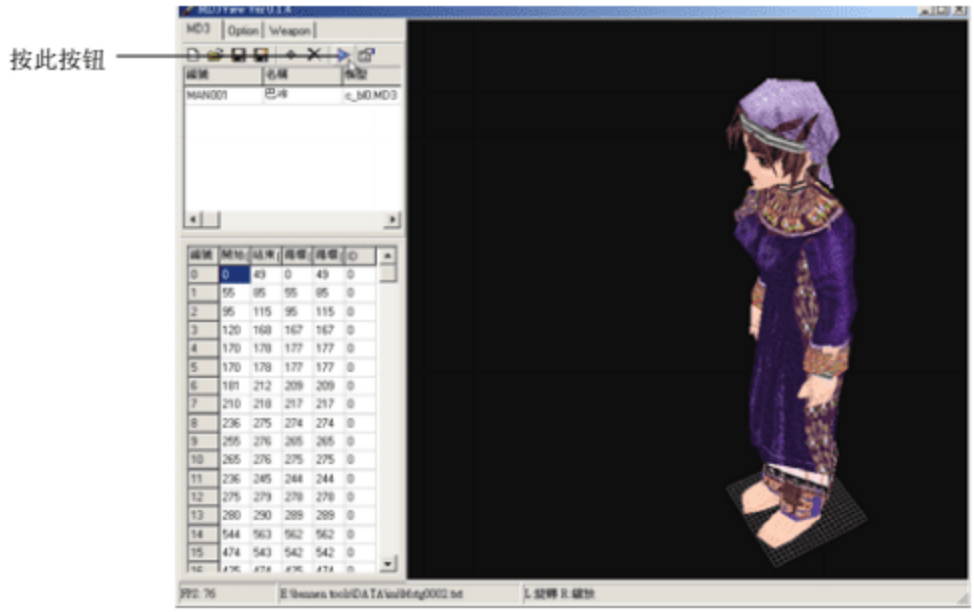


图 C.5　播放 3D 人物效果

在此我们按下人物播放按钮，如图 C.5 所示，3D 人物便开始动作。在人物动作编号栏中，可以选择要播放的动作编号，如图 C.6 所示。

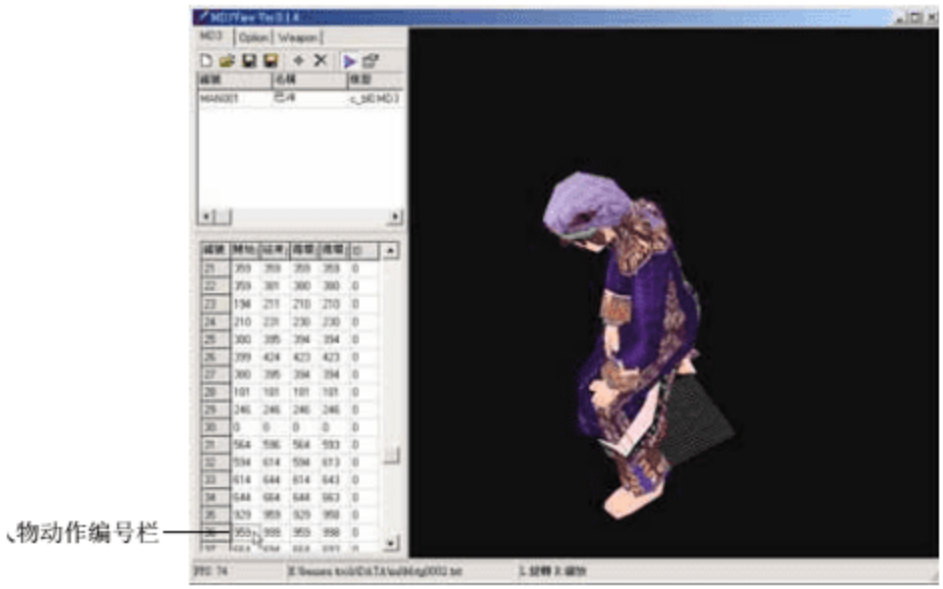
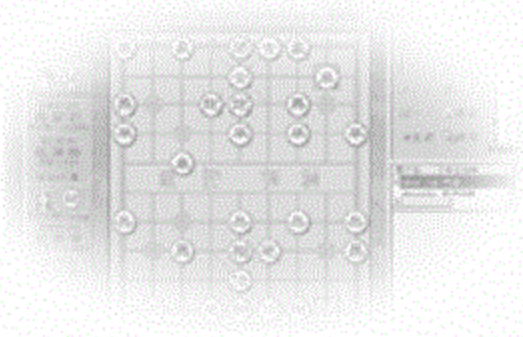


图 C.6　选择人物动作编号

附　录

MDZ 预览器



MDZ 是“巴冷公主”游戏中的所有静态 3D 对象的文件格式。与 MD3 格式不同的是，MDZ 文件只有一种状态，因此它不需要由人物动作编辑器来编辑。

下面来看看 MDZ 预览器要如何使用。首先请执行配套光盘中的 MeshView.exe 文件，然后按照下面步骤来预览 MDZ 3D 的静态对象，打开 MeshView，初始界面如图 D.1 所示。单击“打开”工具图标，则弹出“开启”对话框，如图 D.2 所示。选择文件，则显示静态物体如图 D.3 所示。

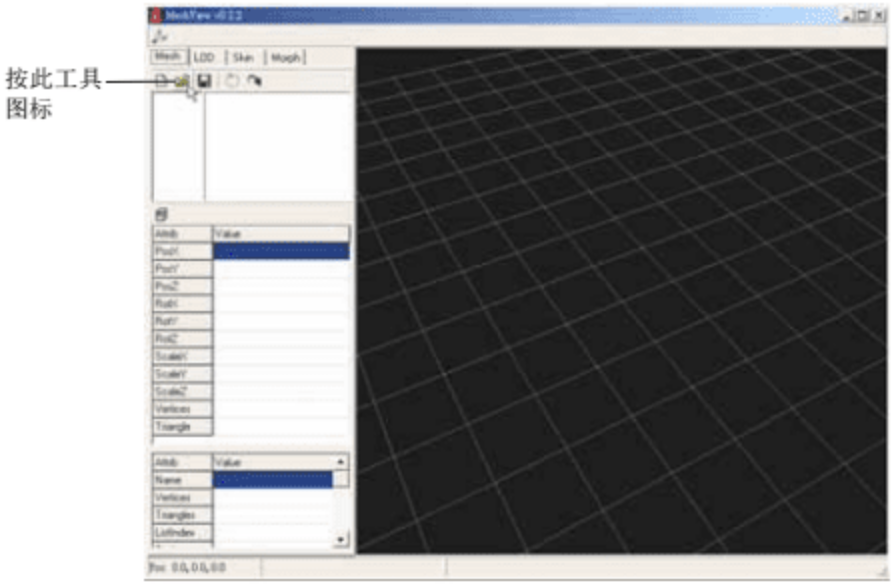


图 D.1　MeshView 初始界面

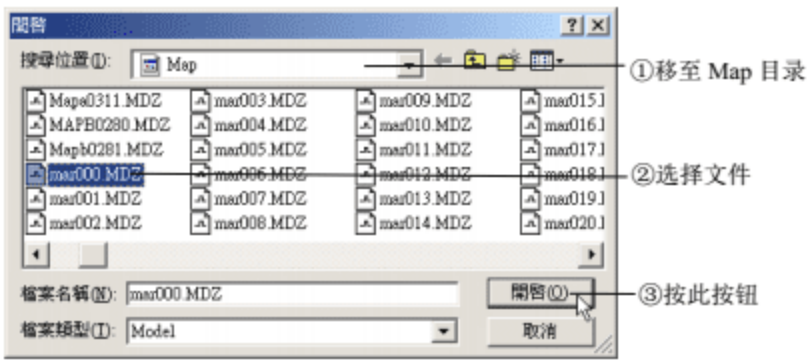


图 D.2　打开文件

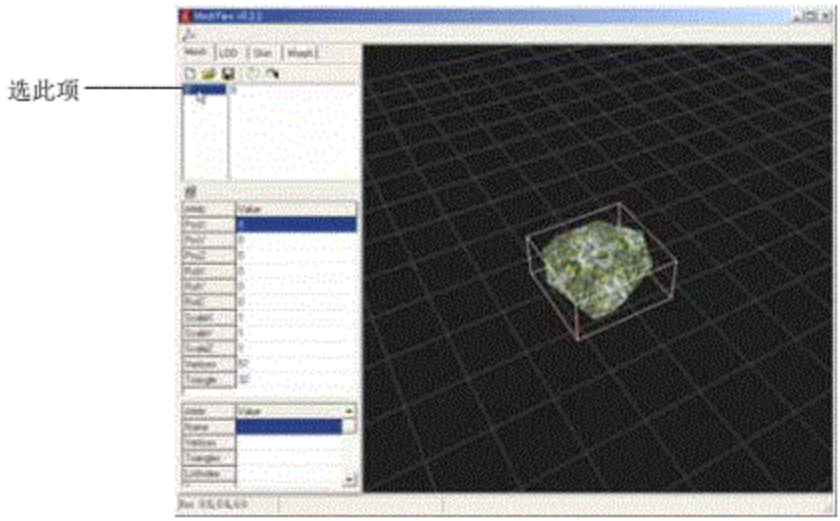


图 D.3　打开 MD2 静态对象

在图 D.3 中，您就能看见您所读取的 MDZ 静态对象。接着调整对象的实际位置与偏移位置，如图 D.4 所示。

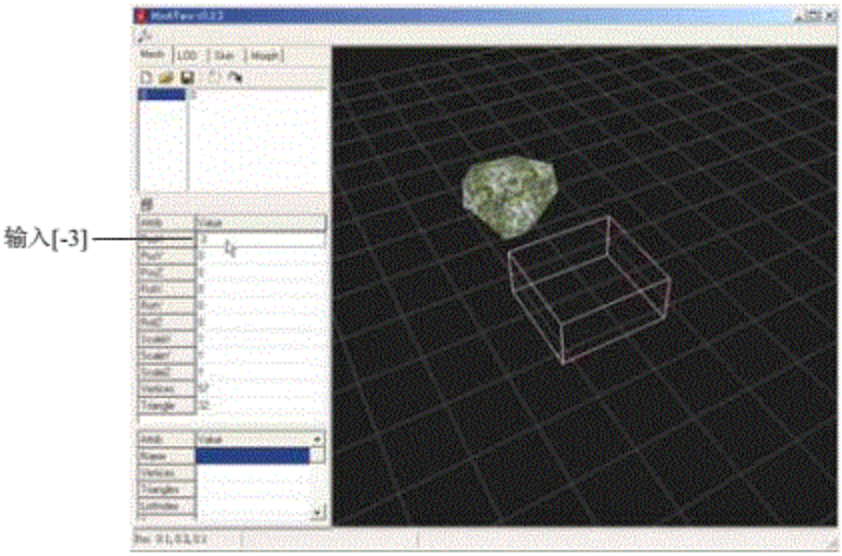
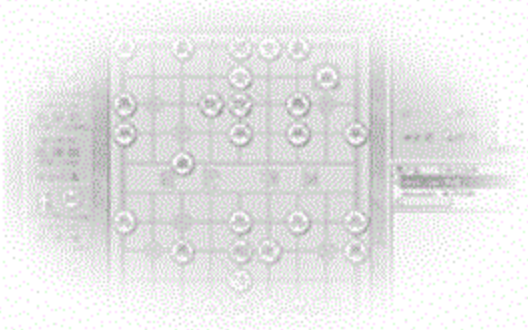


图 D.4　调整对象的实际位置与偏移位置

在此您会发现红色方框与石头的位置已经不同了。其实红色方框便是对象的实际位置，而石头则是对象的偏移位置。

第19章

人物动作编辑器



人物动作编辑器主要是用来编辑 3D 人物的动作。在 MD3 格式中，它可以将人物的所有动作都保存在一个文件中（关于 MD3 格式请参阅其他相关书籍），而人物动作编辑器则是将这种动作加以分类。

接着来试试人物动作编辑器的基本功能。首先请执行配套光盘中的 MD3View.exe 程序，MD3View 初始界面如图 C.1 所示，单击“打开”工具图标，则弹出“开启”对话框。

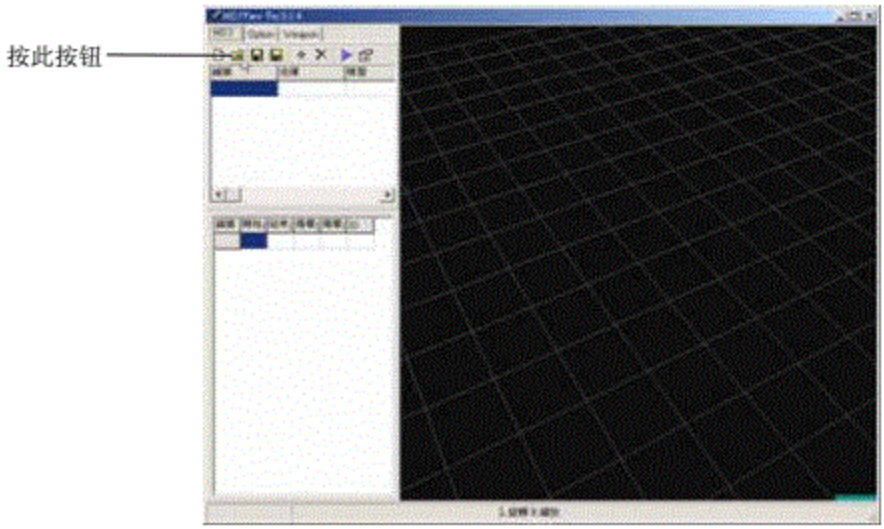


图 C.1　初始界面

图 C.2　“开启”对话框

如此一来，您便能看到人物的 3D 模型了。在图 C.3 中怎么没看见呢？其实刚读进来的 3D 人物模型，是因为摄影机拉的太近了，所以什么都看不见。

在此是读取人物属性设置，而不是直接打开 MD3 格式文件。

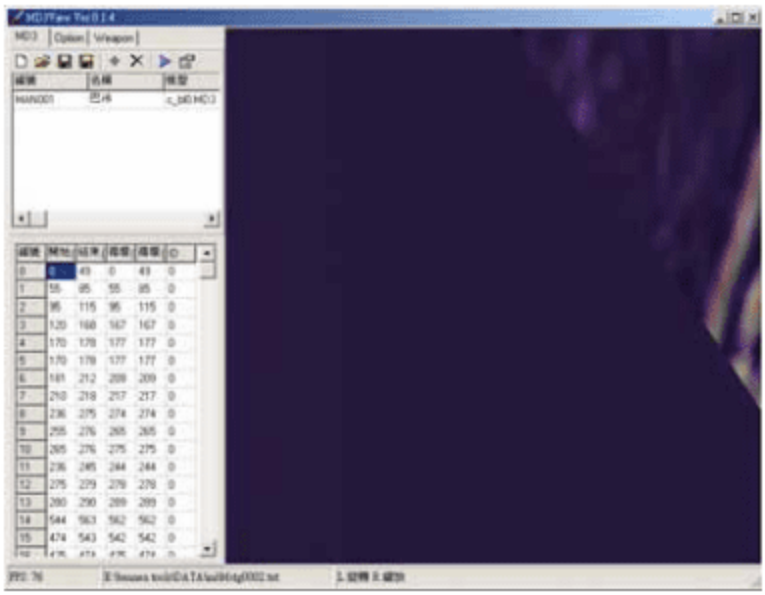


图 C.3　显示 3D 人物模型

注意，如果您还是看不到任何模型，那可能是模型文件与材质文件没有连接起来，请在模型与材质栏中上双击，并重新读取图 C.4 中的文件（文件在 md3 目录中）。

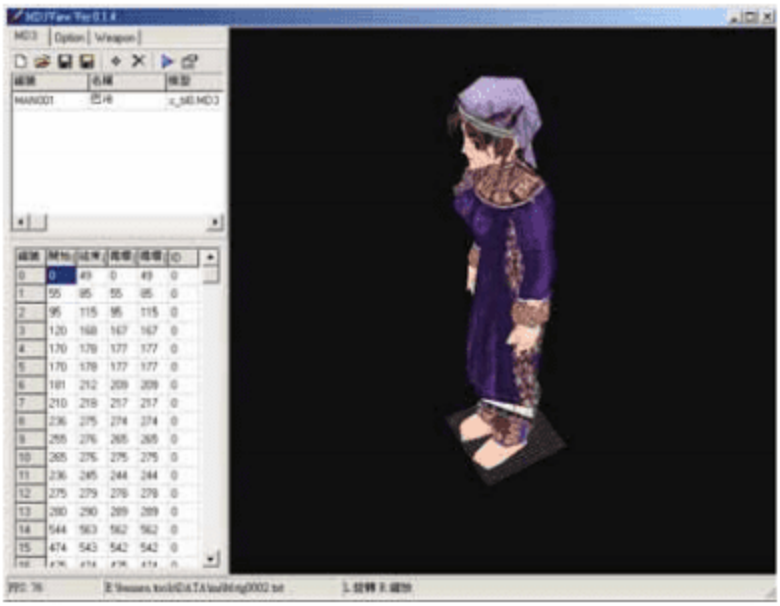


图 C.4　您可以按下鼠标左键或右键来调整摄影机的角度与位置

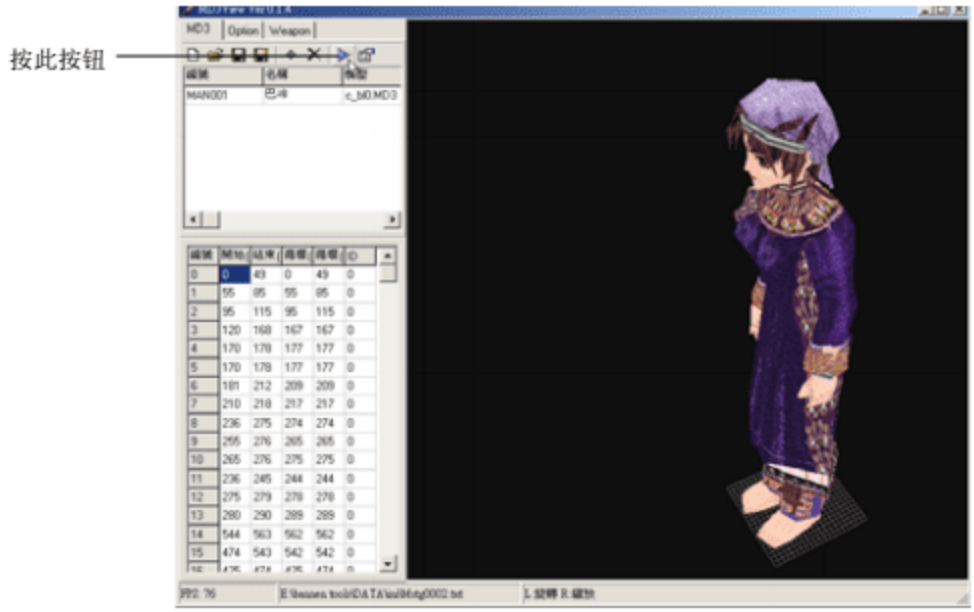


图 C.5　播放 3D 人物效果

在此我们按下人物播放按钮，如图 C.5 所示，3D 人物便开始动作。在人物动作编号栏中，可以选择要播放的动作编号，如图 C.6 所示。

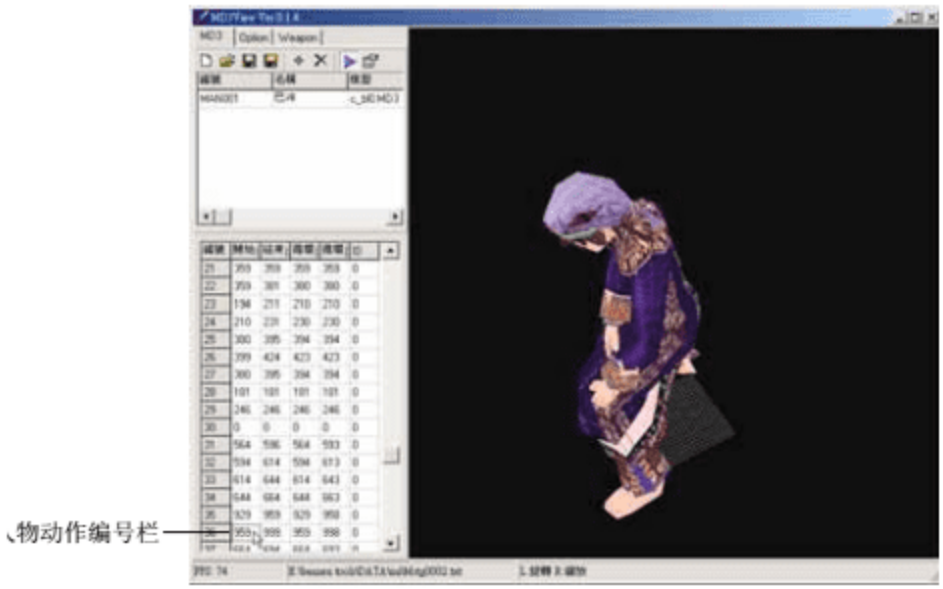
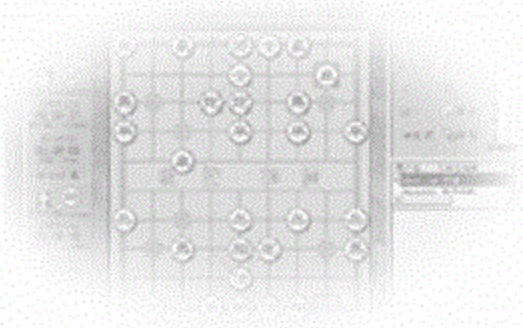


图 C.6　选择人物动作编号

第20章

MDZ 预览器



MDZ 是“巴冷公主”游戏中的所有静态 3D 对象的文件格式。与 MD3 格式不同的是，MDZ 文件只有一种状态，因此它不需要由人物动作编辑器来编辑。

下面来看看 MDZ 预览器要如何使用。首先请执行配套光盘中的 MeshView.exe 文件，然后按照下面步骤来预览 MDZ 3D 的静态对象，打开 MeshView，初始界面如图 D.1 所示。单击“打开”工具图标，则弹出“开启”对话框，如图 D.2 所示。选择文件，则显示静态物体如图 D.3 所示。

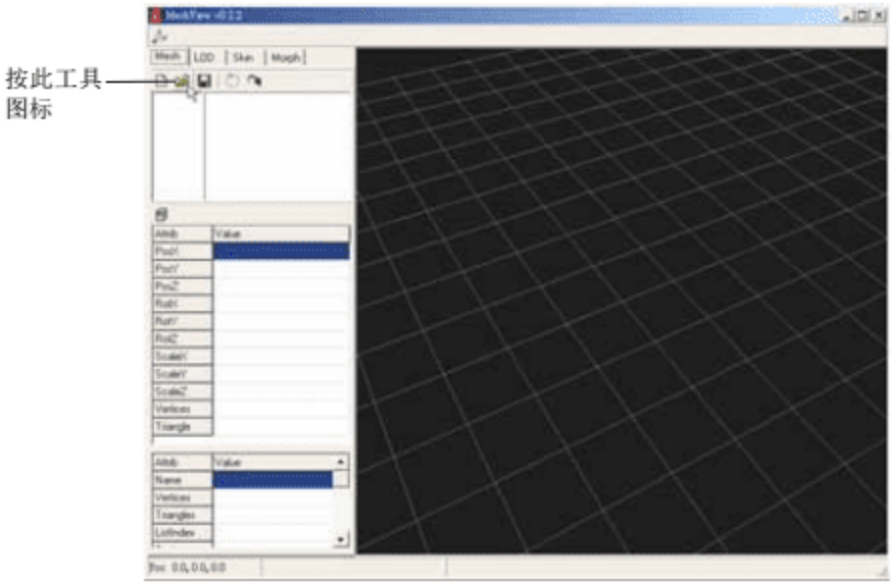


图 D.1　MeshView 初始界面

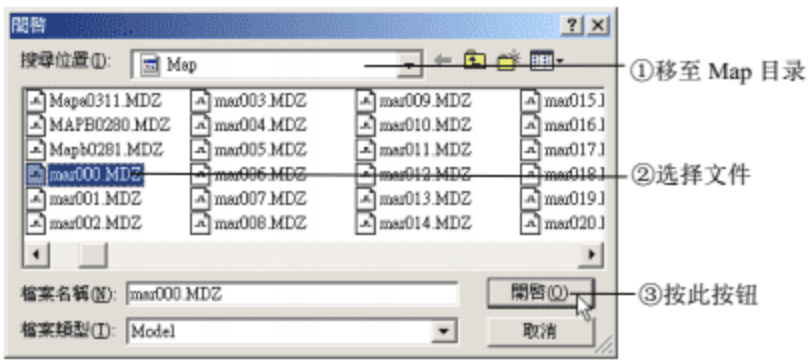


图 D.2　打开文件

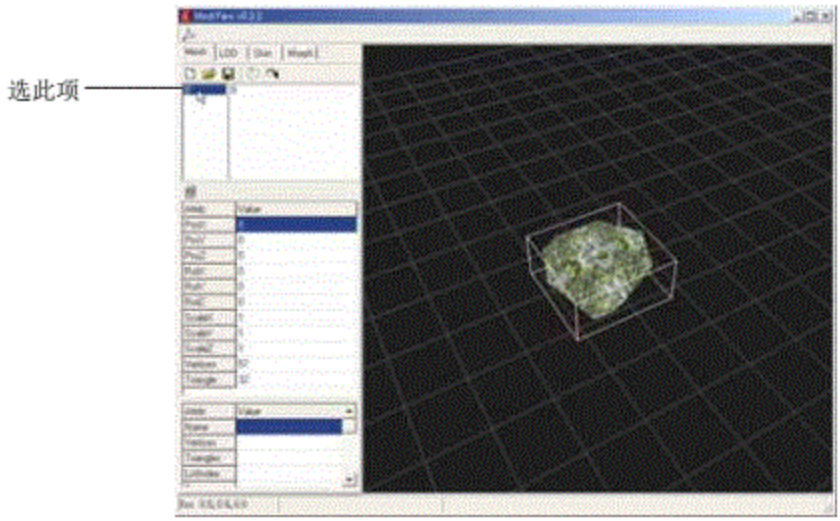


图 D.3　打开 MD2 静态对象

在图 D.3 中，您就能看见您所读取的 MDZ 静态对象。接着调整对象的实际位置与偏移位置，如图 D.4 所示。

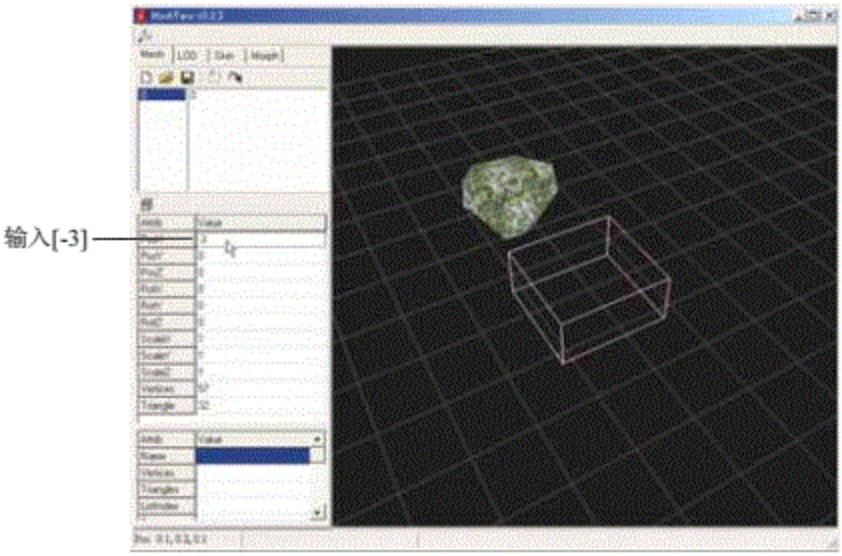


图 D.4　调整对象的实际位置与偏移位置

在此您会发现红色方框与石头的位置已经不同了。其实红色方框便是对象的实际位置，而石头则是对象的偏移位置。