Game Design and Development

Zhan Ye ◆ Ding Ye

游戏的设计与开发

——梦开始的地方

叶展 叶丁 编著

人民交通出版社
航空工业出版社

目　　录

基础篇

第一章　总论

本章为全书的第一章，起到总论的作用。所谓总论，其中所要阐述的，自非一枝一蔓的细节，而是提纲挈领，对一些属于大层面上的问题提出看法和心得。由于以后各章节都是讨论各个细节层面的问题，这一篇总论自然就成为一个极难得的机会，可以来讨论一些横亘整个游戏设计开发领域的重复出现的模式（pattern）和问题，并对这些模式和问题，提出一些思想方法方面的领会和思考。

本章分为四个部分。第一部分对游戏未来发展趋势做了一定的概括和预测，提出了三个重要的发展方向，也即目前游戏业界和学界着力力求突破的所在，并用三个以 i 打头的英文单词来表示。然后由未来回溯到历史，从历史角度对比游戏和动画电影发展的历程，探讨游戏作为一种不成熟的艺术形式所面临的困难和机遇。最后两部分更为重要，讨论的题目从游戏本身的发展转移到了游戏设计开发的实际问题，讨论了设计与开发这两个阶段的不同点，并提出了游戏设计开发的三大基石—vision、technology、和 process。

游戏的三个发展趋势

游戏发展至今，历数十寒暑。早期游戏脱胎于纸牌棋牌，其规则简单而变化无穷。后 ACT（动作游戏）兴起，以其手眼协调条件反射等要素，吸引孕育千万闯关族。而随着计算机处理能力日强，软件技术月新，更多类型（genre）的游戏涌现，其表现力的深度广度均有极大拓展。FPS（第一视角射击游戏）及 RTS（实时策略游戏）的 AI（人工智能）愈来愈复杂，RPG（角色扮演）所构建世界愈来愈宏大。近年网络游戏的兴起，更要全面改写单机游戏的设计法则和游戏规则。面对今日如此之纷繁复杂的游戏世界，如何探知其潮流所向，把握其未来发展的脉络，并做出前瞻性的努力呢？

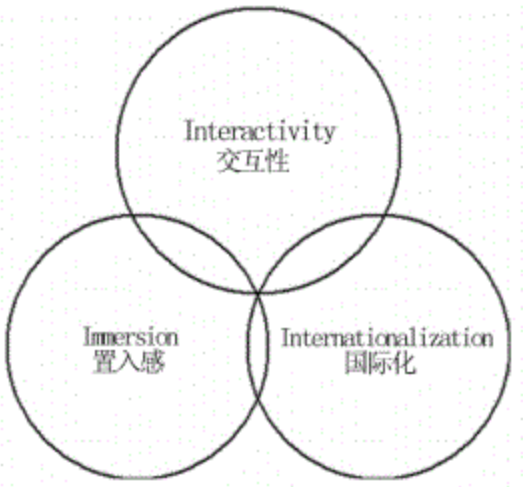


图 1-1 游戏的三个发展趋势

对游戏的未来发展，笔者认为，主要会体现在三个趋势（方向）上，以三个以 i 打头的英文单词来代表，见图 1-1。

下面就这三个发展趋势和代表它们的单词一一细说。

游戏发展趋势之一：置入感的深化

什么是置入感?所谓置入感，就是指深入其中，忘乎所以。《聊斋志异》中《画壁》一篇，说一个书生在寺院里观赏壁画的时候，看到画中有个美丽的散花天女。于是他“神摇意夺，恍然凝想”，突然就进入了画中的世界，和散花天女短暂地相会。这个美丽的故事，实际上就是置人感的体现——壁画的表现力和观赏者的欲望，共同地造成观赏者的迷失，导致了最强烈的置入感（观赏者进入了壁画中的世界）。在故事的末尾，作者又假借“异史氏”为名发表了一番评论，说“幻由人生，此言类有道者。人有淫心，是生亵境；人有亵心，是生怖境。”他的这番议论中，点出一个很重要的道理，就是人类特别容易就被假象所迷惑，进入物我两忘的境界。人类更以自己的想象去配合实际的假象，从而到达更深的迷恋。当小孩子手里拿着玩具飞机在草地上奔跑，嘴里嘟嘟嘟地叫的时候，他想象自己是战斗机飞行员，在驾驶舱中驾驶着战鹰飞翔。而当他在桌上摆上十几个玩具兵，边移动他们边嘴中念念有词的时候，他想象自己是指挥千军万马的将军。从某种意义上说，所有的小孩子都是出色的电影导演——他们自导自演自己沉浸其中。而大人们听戏曲，为才子佳人的故事落泪；看电影，在特殊效果面前目瞪口呆。所有这些深度的感动，都是观赏者主动以想象来配合的结果。

所以说，人真是一种很容易就被骗并骗自己的动物。而这种被骗（被简单的虚幻假象所迷惑）和骗自己（通过自己的想象和感官的调整去和假象协调一致以达到深度置入感）的行为，可以带来极大的愉悦感和刺激感。

对所有的艺术形式来说，其引人入胜乃至使人如醉如痴的不二法门，主要就是置入感，为了达到置入感，各种艺术形式采用了诸多技术手段。

首先第一步就是要构建一个虚拟的世界，无论是绘画、电影、还是小说，实际上都可看作是给观赏者创造一个虚拟的世界。而要构建一个虚拟的世界，就必须在一定程度上模拟现实世界。这种努力实际上横亘人类整个艺术发展的历史。比如说西方绘画中发展出来的透视法和对外光的研究，就是为了在二维的平面上虚拟三维的空间，动画技术实际上也是一种虚拟现实技术，只不过它虚拟的是动态而非静物，它通过连续播放静态的画面造成运动的假象。而近些年发展迅猛的游戏三维图像技术，多边形数越来越多，材质越来越细致，光影效果越来越复杂，做得真假难辨，也是虚拟现实的一种努力，目的就是要让玩游戏的玩家看到一个比较“真实”的虚拟世界，产生置入感。而近 20 年来计算机领域比较热门的虚拟现实技术（virtual reality，缩写为 VR），其中很多

图 1-2 文艺复兴时期意大利佛罗伦萨的建筑师兼工程师 Filippo Brunelleschi 通过光学试验发展了线性透视的概念。这一“科学”发现马上被应用到了绘画上。图为其浮雕

东西实际上并不是什么新概念。

第二步是要尽量使虚拟的世界占据人的大部分感官。电视屏幕越来越大，环绕立体声音响都是起到这个作用。最著名的例子，如莫奈的《睡莲》。这幅巨作陈列在巴黎奥朗热利博物馆一间圆形厅内的四壁，使得画中的池水和莲叶环绕在观赏者周围，使人产生梦幻般的置入感。

图 1-4 莫奈的《睡莲》（局部）

第三步则是尽量减少观赏者所处的现实世界的干扰。电影院放电影时要熄灯就是这个道理。在一片黑暗中，只有银幕上的影像，观众的注意力完全集中到它上面，为银幕上的故事所吸引，忘记他们现在身在影剧院里，忘记他们和陌生人坐在一起。而老戏迷们听戏的时候闭上眼睛摇头晃脑，也是要尽量从感官上和所处的现实世界隔绝，投入到戏文所讲述的才子佳人的悲欢离合故事中。

图 1-3 文艺复兴时期的一幅壁画，从附加的线段我们可以看出画家对线性透视的运用极为严谨。西方的画家们得到透视法后如虎添翼，在画布上开始了模拟真实世界的不懈努力。直到印象派画家们对外光的研究和表现登峰造极，才走向变形抽象之路。在此之前，可以说西方绘画就是一种虚拟现实的技术。（图片来源：http://www.crs4.it/Ars/arshtml/arstitle.html）

图 1-6 三维立体眼镜（HMD）越来越小巧轻便

我们现在来考察游戏中的置入感的实现。游戏和我们前面所介绍的其他艺术形式一样，也是构建一个虚拟世界（游戏世界），把玩家拉进这个虚拟世界中，达到深度置入感。在游戏中虚拟现实世界由两个步骤组成：软件处理和硬件处理。软件处理是指使用软件技术和媒体文件构建一个虚拟世界——一个由数学模型所构成的在现实中并不存在的世界。而硬件处理主要是通过输出设备把这个数学模型转换成阴极射线管所显示的图像——可以乱真的图像，并将其展现在玩家的面前。

图 1-5 拉菲尔的油画《教皇利奥和两位红衣主教》的局部，西方传统油画中关于衣服细节的处理（材质纹理）的威力可见一斑

目前游戏虚拟现实世界的水平，实际上和传统油画差不多。计算机图像由二维进化到三维，多边形越来越多，模型越来越复杂，越来越像真实的事物，相当于文艺复兴时通过人体解剖和各种观察，使得造型能力越来越强；而材质（texture）的采用，使得游戏中三维物体表面具有自然的形态，则和西方

传统油画的细节处理的细腻程度差可比拟；而游戏中实时光源和光照模型的普遍采用，则相当于印象派对外光的研究。

而硬件方面，游戏的输出设备在近二十年变化不大，基本还是以电视机和计算机显示器为主。它们的缺陷显而易见——显示平面过小，视野过窄，不利于玩家真正融入游戏世界。而三维图像技术的改进虽然能提高置入感，但当多边形达到一定数量，三维模型和真人无异的时候，游戏设计师必然面临一个棘手的问题：如何进一步提高置入感？在过去二十多年里不断发展的虚拟现实技术，有可能在近期被游戏业界所采用，以硬件升级更新的形式，给游戏世界带来更强烈的置入感。

近 20 年间各大学和公司开发出了各种各样的虚拟现实系统。我们就简单考察一下这些系统。它们基本可以分为两类：HMD 和 CAVE。HMD（head mounted display），即头盔式显示器，是现在最常见的 VR 设备。一提起虚拟现实，公众马上就会联想到一个人头上带着一个头盔、手上戴着数据手套、手舞足蹈的样子。确实，HMD 是目前所比较流行的简单的 VR 设备。它易于携带安装，价格也较便宜。

而 CAVE（Cave Automatic Virtual Environment，洞穴状自动虚拟系统）系统则是由伊利诺伊大学芝加哥分校（UIC）开发出来的一种更大型的虚拟现实系统。它可以有一间屋子那么大，里面可以有多个人同时参与，“屋子”由三面墙体和一块天花板组成，状如洞穴。通过使用三个背面投影仪，图像被投射到三面墙体上。顶部的天花板则使用反射式投影仪。用户进入“屋子”后，戴上立体眼镜，身上带着方位传感器，他眼前的是一个充斥其前向视野的虚拟世界，这个世界根据他的位置和动作做出实时的反应，这样就达到了置入感。

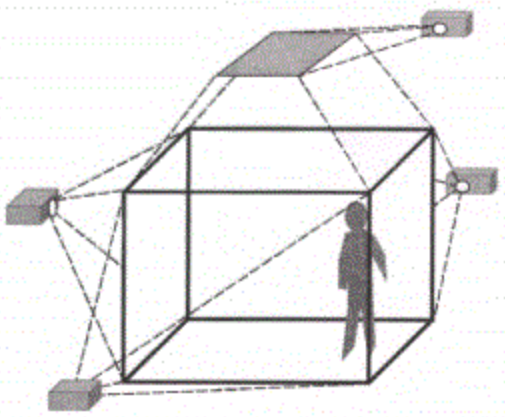


图 1-7 CAVE 系统示意图

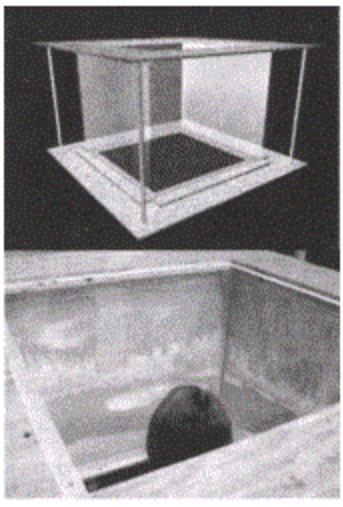


图 1-8 CUBE 系统

由卡内基美隆大学娱乐技术中心（ETC）开发的 CUBE 系统（Computer-driven Upper Body Environment，计算机驱动上半身显示环境），则是 CAVE 的一种缩小版本。它是一个顶部悬挂的玻璃箱体，使用时箱体降下来罩在人的四周，这样人就可以看到 360 度视野的实时画面。

我们考察这三种有代表性的 VR 系统，不难发现它们所使用的方法不外乎我们前面说的三条。戴上 HMD，视野完全封闭，不受外界干扰。CAVE 系统是把人的全部前向视野用

来显示虚拟世界。而 CUBE 系统则直接借鉴了莫奈的《睡莲》的展示方法。所以说它们所使用的技术是新的，但方法和思路却都无甚新鲜。

除了上面所介绍的几种 VR 技术外，全息图像也是一种可能的替代技术。SQUARE 的某些人士在数年前曾经提及全息图像在未来游戏中应用的可能，但目前从事这方面工作的并不是很多。

总之，置入感是人类一切娱乐艺术形式的重要组成要素之一。一切娱乐艺术形式都以置入感为其第一个层次的目标，在达成置入感的基础上才能实现更高层次的功能。纵观游戏发展历史，技术层面的很多进步，其目的都是要提高游戏所能达到的置入感。而今后游戏技术的发展，也必然会有很大的努力是投入到置入感的深化上。

特别需要指出的是，近些年国外提出了所谓“体验经济”的概念，并说体验经济将代替信息经济，成为社会的驱动。其中所提及的“体验”，实际就是置入感。而游戏业实际上就是体验经济最雄辩的代表。关于体验经济的具体内容，请参考托夫勒的一系列著作。

游戏发展趋势之二：交互性的增强

游戏作为一种娱乐形式，第一要实现置入感，构建一个虚拟的游戏世界；第二就是要提供交互性，使得玩家可以和这个虚拟的游戏世界进行交流。那么什么是交互性呢?交互性体现在两方面：交互手段、交互的可能兼反应。先举一个简单的例子来说明这两个概念。假定在一个游戏中，游戏人物需要从山崖上摘一朵花献给自己的女友。首先我们遇到的问题是如何让玩家操纵游戏人物去摘花。可以有以下多种设计方案。

● 最简单的一种设计：当玩家操纵游戏人物走到距花朵的一段距离内时，弹出一个窗口，显示提示“花朵在附近，是否摘下？”，提示下方有“是”和“否”两个按钮。玩家可以在“是”的按钮上点一下以摘下花朵。

● 更复杂一点的设计，可以让玩家操纵游戏人物移动到花的旁边，然后改变屏幕光标的形状，改成手的形状，同时显示一个信息条：“在花朵上按鼠标按钮即可将其摘下！”。这时候玩家按一下鼠标，花朵被摘下来了。

● 更复杂的设计，是类似于《黑与白》中的姿势识别与控制系统（gesture recognition and control）：玩家所控制的人物走到花朵的旁边，玩家移动鼠标使得屏幕光标位于花朵上方，这时屏幕光标改变成一只张开的手的形状，当玩家在花朵上按下鼠标按键后，屏幕光标显示一只收拢的手的形状，玩家必须按住鼠标按键不松开，拖动光标到主角的身上，然后松开。这样屏幕上才会显示信息：“花朵被摘下。”同时屏幕光标的形状恢复到张开手的形状。

● 最复杂的设计，从玩家角度讲又是最简单的设计，是采用 VR 系统：玩家头戴头盔式

显示器，手戴触感手套，眼前是一个全三维的虚拟世界，当他“走”到花朵旁边时，他向花朵伸出了手，握住了花枝，将其拔起。

我们上面所列举的四种设计方案，实际上是四种不同的交互手段。第一种是标准的菜单选择式的交互，简单而又乏味；第二种是使用图标和对图标的直接操纵（direct manipulation）进行交互，有了一定的变化（时态性）；第三种是在第二种基础上引入连续时间的概念，使得操作更复杂，含义更丰富；而第四种则是使用 VR 系统来最大限度地模拟我们在真实世界中的交互，对玩家来说最直观简单。

交互的可能兼反应，指的是当玩家在使用一定的交互手段进行输入之前有什么限制，输入后又能体验到什么效果。限制越少，效果愈真，则交互性越强。还以摘花为例：在玩家摘花之前，对可摘之花有何限制?是否只有那样特定的一朵花是可以摘的?还是游戏中所有的花都可以摘?显然后者给玩家的自由度更大，更近似于真实世界（当然在真实世界中有些花也是不能随便乱摘的），需要构建的游戏世界也更宏大，游戏世界的驱动规则也更复杂。当玩家摘花之后，以何种形态去反映摘花这个事件的效果呢?早期二维 RPG 游戏中，屏幕上不能做出复杂的变化，顶多在物品菜单里面显示一下。三维游戏中，可以用三维动画显示花朵被摘下的过程；而使用 VR 系统，则可以从玩家的视角显示花朵被摘下的过程，更为真切；而更登峰造极的，是可以利用电子香味发生器，当玩家把花摘下移近鼻子的时候，散发出更浓烈的香味。显然上面所列举的几种效果一种比一种更真实生动。

图 1-9　适用于 PC、PS2 和 XBOX 的三维数据手套。可以取代游戏手柄

通过前面的介绍，我们对交互性的两个方面——交互手段和交互的可能兼反应有了一定的了解。更强的交互性，是游戏业所不懈追求的目标。下面就对近年来游戏业在增强交互性方面的进展和趋势简单介绍。

从交互手段上看，传统的键盘鼠标和游戏手柄已经使用多年，渐呈老态。更多更新的交互手段开始涌现。这些新的交互手段使得人和机器（计算机或者游戏机）能够进行更自然的交互，交互手段所能包涵的“语义”更丰富。比如说姿势识别与控制技术是利用连续时间的鼠标输入来进行控制，使得鼠标可以表达更复杂的指令，PC 游戏《黑与白》为其代表。语音识别与控制技术是利用玩家的语音来进行输入、计算机合成的语音作为输出，这是使人机交互更轻松自然的努力，世嘉的《人面鱼》为其代表。而 VR 系统中的数据触感手套，既是输出设备（触感）又是输入设备（代替键盘光标），可以提供六个自由度和更为自然的三维交互——有什么比使用自己的双手更方便自然的呢？

这些技术在各大学实验室里已经经过了许多年的精琢细磨，逐渐成熟，在未来游戏中

的应用已露端倪。比如 SONY 在 2002 年的 GDC 上就展示了 PS3 上所使用的姿势识别与控制系统，令与会者大开眼界。

图 1-10　SONY 在 2002 年 GDC 上展示的姿势识别与控制系统。玩家手拿一个绿色的小球站在一台 PS2 前面。PS2 上装了小摄像头。玩家在镜头前舞动小球，摄像头将人物的动作捕捉下来，计算机分析处理后，判定其含义，便可发出相应指令。比如进行魔法攻击，玩家只需凌空画个三角形，屏幕上就会随着绿球移动出现了一个粉红色三角形轨迹，然后一霎那间屏幕上阴云密布雷霆电闪——发出魔法攻击了！使用小球更可以在三维空间中自如地转换视角和进行移动

在本书的人机界面设计一章，将会着重讨论人机交互的问题，并介绍一些新近涌现的交互手段。

从交互的可能兼反应来看，它实际上是游戏世界的驱动规则，即游戏世界作为一种虚拟世界自己能够有什么组合变化，可以有几种状态，状态之间如何转换。显然组合变化越多，可转换的状态越多，玩家的可能性越多，交互性就越强。自然世界的组合变化无穷，未来状态更不可预测。游戏世界显然永远不能成功模拟自然世界，也就是说永远无法如自然世界般宏大，充满无数交互的可能和生动的反应。但在一定限度内，新技术的采用可以使得现有的游戏世界的复杂度得到某种程度的飞跃。游戏业界希望在今后几年起码可以做到以下几点。

●通过使用人工智能和认知模型（cognitive model）等技术，使得游戏世界中的人物和生物具有了解周围环境、适应周围环境、根据周围事物决定行动对策的能力。

● 使用物理编程，使得游戏世界中的物体的运动符合自然世界的物理规律，比如说重力和加速度。

● 研究交互式故事和故事的自组织结构，使得游戏世界摆脱纯线性剧本的局限。

在本书的人工智能一章中，将谈及认知模型和人工生命。在故事性和交互性一章，对交互性和故事性这一对“冤家”的恩恩怨怨有一定的阐述。

游戏发展趋势之三：国际化设计

前面两条趋势讲的是技术方面的趋势——游戏将不懈地追求更深的置入感和更强的交互性。而这第三条趋势讲的是设计文化方面。近几年来，人们所明显感受到的一个的趋势是游戏设计更趋国际化了。实际上在所有的领域，国际化全球化都是一个无法阻挡的潮流。有人痛心疾首，有人螳臂当车试图诋毁并阻挡这种潮流，实在是可笑而又可怜。全球

化不就是人类大同思想的体现吗?之所以有全球化趋势，正是因为世界上有一些人们所共同认同共同追求的理念。比如说每个人都希望能够自由独立尊严地活着。游戏也是这样，一个日本人开发的游戏，可以让全世界的玩家神魂颠倒，就是因为全世界的人们的审美情趣有一定的共性。

图 1-11 美版和日版古惑狼从形体到神态都有一定差别

虽然说全世界的人们的审美情趣有共性，这是国际化设计的基础，但在设计中抓住并反映共性并不是一件容易的事情。有很多的棘手的问题要处理，有时候也必须做出一定的妥协。目前人们所做的，还是着眼几个大市场，实际主要是美日两个市场。针对这两个市场的消费者的取向，来调整设计。《古惑狼》就是一个很好的例子。

《古惑狼》是美国公司开发的游戏，其人物设计风格自然是美式的。为了面向国际市场，特别是日本市场，“海外版”古惑狼的形象设计有细微的改动。我们看到在日本发售时的海报和杂志封面上，古惑狼更单纯天真一些。而美国本土的古惑狼，更有美国街头坏小子的狂放劲。两个版本的古惑狼笑起来就不一样，日本的古惑狼笑起来是孩童般阳光灿烂的微笑，而美版的古惑狼笑起来是捣蛋鬼似的狡鲠的笑，更成人化一些。由于调整了人物设计，更照顾到日本人的审美观点，古惑狼在日本获得了美国公司以前无法想象的极大成功。

图 1-12　FF8 中的角色，原型为美国明星，发型是美国近年来的流行式样

同样地，一个日本公司若想把游戏卖到美国并且卖得好的话，自然就要顾及一下美国玩家的审美观点和取向。比如 SQUARE 的 FF 系列人物设计自从三维化后，受到美国的影响越来越大。有两个原因，一是 20 世纪 90 年代末期以来 RPG 在北美市场的潜力被真正发掘出来，市场份额之巨大使得游戏设计者无法忽视这部分玩家的要求；二是三维技术方面 SQUARE 公司雇佣了很多美国三维图形和动画人员，他们自然要带来一定的美式影响。我们看到 FF8、FF 电影的人物设计，很多都是以美国影星为模特，并采用了时下流行的发型和服饰，也就不足为奇了。

美国动画和游戏界，也受到了日本风格的影响。这体现在他们的人物设计更加秀气了，在具体的技法上则更加注重了双色阴影。（美国传统的动画片，特别是电视动画片，一般是单色平涂。）

前面说的是调整风格去适应不同的市场。与之相对应的另一种思路是找出一种通用的国际化的风格。国际化的风格必然是抽象化的、中性化的，也就是人物设计不具有鲜明的民族特征，而以某种简单抽象的形态来表达。

在 2000 年的 GDC 上，《啪啦啪啦啪》（Parappa the Rapper）的设计者松浦雅也曾做了一个报告，探讨如何克服不同文化之间的差别局限，设计出适应全球市场的游戏。从《啪啦啪啦啪》的人物设计，我们也可以看出他所提出的设计中性化和国际化的某些特点。

图 1-13　由美国设计师设计的日本游戏《Parappa the Rapper》的主人公

我们可以看出《Parappa the Rapper》的人物设计，有一种独特的简约风格。这种风格和美国动画片《南方公园》的人物设计有些类似。这种风格和建筑方面的国际主义有异曲同工之妙。

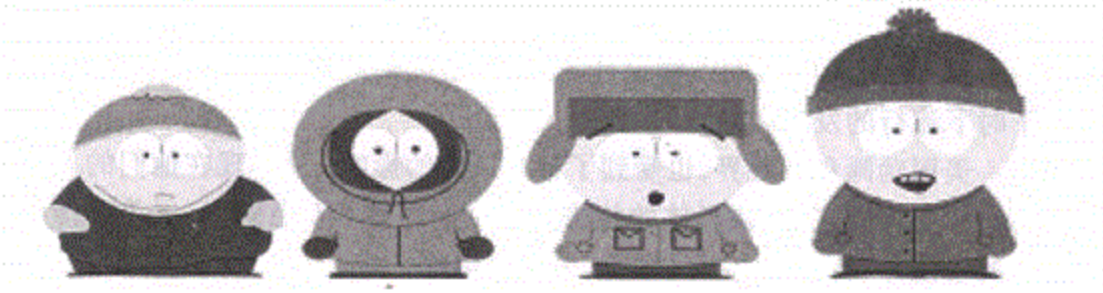


图-14 美国动画片《南方公园》的主人公们

游戏设计的国际化，除了风格的相互影响和抽象化中性化外，更重要的一种思路是把游戏层和文化层在一定程度上隔离，从而使得来自不同国度的玩家可以克服文化上的障碍，这种思路就是在一个通用的游戏系统外面披上一层文化的外衣，文化层实际上成了“毛”，而游戏系统是其所附的“皮”。关于这种思想，将在游戏类型一章有所阐述。

游戏是一种成熟的艺术形式吗

一个成熟的艺术形式，应该有两个特征：（1）拥有比较稳定的表现媒介和技术；（2）建立了比较规范的艺术法则来指导创作。从这两个方面来看，游戏都没有达到。游戏还不是一种成熟的艺术形式。

艺术形式的三个阶段

一般来说，一个新的艺术形式从被发掘出来到成熟，要经历三个阶段（如表 1-1）。

处于第二阶段的游戏，它的第三阶段是怎样呢

游戏目前应该属于第二阶段，也就是说游戏是一个崭新的艺术形式，虽然经过 20 年左右的发展，但其内在规律和法则都没有很好的归纳和总结，也没有能够用来指导开发的范式。更通俗的比喻：大学里有电影系，系里的学生经过系统的学习电影理论后，如果能够基本掌握并执行那些法则和范式，那么他们拍的电影大概不会太差。（当然真正的高手和平庸之辈的区别是高手并不是完全遵循范式，而是知道在什么地方什么程度上打破常规。）而游戏显然没有这样的理论指导，生产的“产品”质量也毫无保障。

如果我们再看看和游戏关系密切的动画片的历史，我们可以看到：20 世纪 20 年代左右的动画片，由于制片人对动画片独特的规律认识肤浅或者根本没有，因此动画片极为粗糙和随意。对现实观察不够，又没有理论指导，于是出现以下情况：为了使动画人物的手从一处移到另一处，胳膊只能做得像橡皮筋一样“伸缩自如”。即使那时迪斯尼所做的著名的《汽船威利》，也还不能超越时代的局限。而到了三四十年代，迪斯尼公司众多高手在十几年艰难探索之后，终于集大成地发展出所谓美国经典动画理论，包括八条左右黄金法则，如挤压与拉抻、辅助动作、期待、慢进慢出。这几条黄金法则，成为了全世界动画片制作人的金科玉律，放之四海而皆准。从此，动画片进入了崭新的时代，成为了有正确理论指导的成熟的艺术形式。半个世纪过去了，几乎没有一部美国动画片不遵循这些准则。即使是最新的全三维动画片《玩具总动员》和《玩具总动员 2》，虽然使用三维技术和图像，但动画方面还是遵循经典动画法则，没有越雷池一步。反观游戏，迄今为止没有任何理论来指导其开发，对其本身规律认识也极为不足。

游戏诞生也差不多 20 年了，这应该说也不算短了。人们不禁要问：游戏是否能出现类似于 40 年代动画片的理论大成熟呢?从目前来看，这种理论上的成熟还缺乏必要的条件，其原因将在下面阐述。

表 1-1 艺术形式的三个阶段

艺术形式的三个阶段	电影	游戏
第一阶段：新奇阶段 新的艺术形式被发掘出来，人们不知道能用它来	最早的电影《火车进站》，《工厂大门》。发明家们扛着电影摄像机满大街跑，拍摄各种片	“发明家”的时代，各种各样的新奇的想法，游戏开发处于乱战和蛮荒状态，探索虽然很

干什么，因此用它来试各种东西	段，自己也不知道要干什么	多，但大部分经不起时间的考验
第二阶段：模仿阶段 新的艺术形式被发现可以帮助老的艺术形式解决问题，或者可以模仿老的艺术形式	电影导演们发现可以把歌剧什么的拍下来，使得没有去歌剧院的人也能看了。但对电影本身的艺术规律和特性还不了解，没有剪辑，没有镜头技术，观众看电影就跟坐在剧场看歌剧一样，视点从来不变	在前一个阶段的基础上，游戏类型（genre）开始归纳形成，如 RPG、SLG 等。所有的类型都基于对现实世界的模仿，既模仿表象（通过三维技术，产生可乱真的模型），也模拟现实中的行为，如射击、策略、甚至人生过程
第三阶段：成熟阶段 新的艺术形式发展出自己独特的艺术法则，体现了自己独特的优点	电影导演们终于“发现’了电影的独特规律和能力，发现了电影可以通过剪辑产生时间和空间的跳跃，（关于如何偶然发现电影剪辑技术就流传着各种各样的小故事）可以运用镜头技术，观众再也不像坐在一个固定的地方看歌剧了，而是可以远景 / 近景 / 特写 / 甚至 360 度环绕一周。电影终于成	?

	为了一种独立的成熟的艺术形式了	

阻碍游戏成熟的重要因素

游戏的技术层面太不稳定，是艺术理论和法则无法确定的一个重要原因。如果一门艺术形式的技术层面不断剧烈变化的话，产生稳定的指导理论是不可能的。我们看其他艺术形式，如绘画音乐等，其工具和媒介都相对稳定，几百年前人们使的画笔画布透视原理，用的钢琴五线谱对位法，到现在改变不大。但游戏就不一样，从最早街机上的黑白矢量图像技术，到 8 位机的字符型活动块，到 PC 早期的 CGA / EGA 色彩表，到 32 位机的三维实时图像。几乎每隔几年就大变样一次，而前一阶段积累的技术基本被完全推翻，经验基本不适用于新的条件。这一切都使得总结游戏制作理论难上加难。

虽然前面所说的难度那么大，但目前人们普遍认为出现了一丝曙光。人们现在基本同意游戏将向两个极端发展，也就是有两个潮流：一个是搞所谓纯粹的游戏，以 TETRIS 为代表；另一类是走复杂化的拟真化的道路。后一类为现在的主流。

游戏发展至今，很多东西都已经被抛弃了，像横卷轴 ACT 等。但有一些小游戏却显示出顽强的生命力，长盛不衰，像 TETRIS（俄罗斯方块）和 PACMAN（吃豆）。其特点是图像简单、规则明了、变化多端。进入 32 位机时代后，人们也曾一厢情愿地将其复杂化，将其 3D 化，引入更复杂的规则等。但这些尝试无一成功，即使是任天堂自己做的 TETRIS 变种也是毫无例外地遭到惨败。这一切促使人们思索这种游戏得以成功的关键不是在外表，而是在游戏规则上。这种类型的游戏可看做与传统棋牌同类，它们都是规则简单易懂，但同时玩起来变化无穷。而游戏的另一个极端是采用三维图像，做得像真人一样，并模拟物理世界构造出复杂的虚拟世界和交互的法则。新一代的 PS2 和 DC 上的大多数游戏就体现这种趋势。

这两种极端，像前一种，是比较容易研究其规律的。而后一种，由于和技术发展息息相关，变化莫测，难于把握。但其发展总会走到一种饱和状态。分辨率高到一定程度，人眼也就分辨不出来了；POLYGON 数多到一定程度，再多无益。模拟物理世界总是有个尽

头。就像传统油画，对透视原理、色彩关系、外光的研究达到完全成熟，油画画得跟真的一样，也就达到饱和了。也就是说，将来会有一个相对稳定的时期，技术层面相对稳定，不像现在这么动荡不安。而那个相对稳定的时期，则是总结创作规律和法则的黄金时间。游戏的成熟，可能要等到那个时期之后。

设计与开发——是分是合

前面我们讲的是游戏自身的发展趋势和规律，下面来讨论游戏的设计与开发。本书名为“游戏的设计与开发”，显然对游戏设计和游戏开发有所区别对待。首先对比一下这两个概念。

就笔者看来，游戏设计（game design）是指游戏设计师头脑中产生出来有关游戏的最初的粗浅的想法，然后在这些想法的物质化过程中，逐渐进行筛选细化和改进，到最后完成详细的设计文档。所谓物质化，就是指把头脑中的东西表达出来，固化到各种媒介上，并利用各种媒介和其他人进行交流。游戏设计可以看成游戏从一个虚无缥缈的状态到第一次固化的过程。固化的结果是详细的设计文档。设计文档相当于建筑蓝图，它对游戏最终的形态要有详细的各种形式（使用各种媒体文件）的描述。

而游戏开发（game development），是指使用设计文档，把游戏从一种纸上的形态，最终转换为可执行的程序，并完成各种辅助文件。游戏开发的结果，就是最终形态的可以运行的游戏。这可以看做是第二次固化。

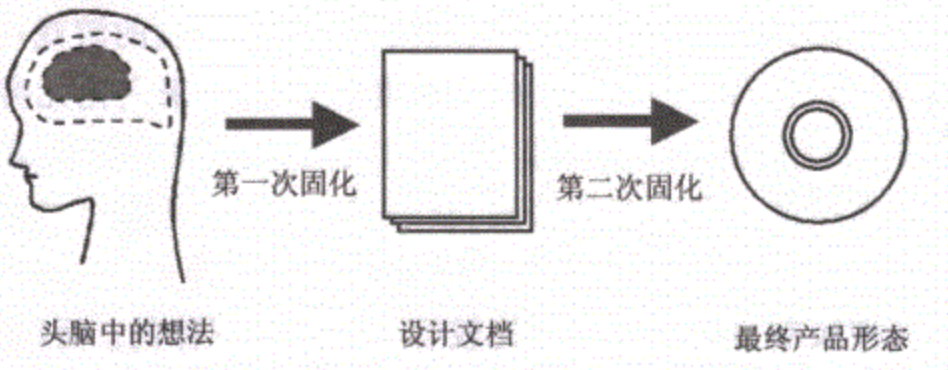


图 1-15 设计与开发实际上是两次固化过程

游戏设计是一个设计过程，而设计过程实际上就是决策的过程——对几种可能的设计方案进行取舍。设计过程同时又是一个思维游戏的过程，具有比较高的自由度。而游戏开发是一个执行过程，因为蓝图已经大致有了，在开发阶段只需根据蓝图来进行过程化生产，自由度比较小，比较严谨。游戏开发也是一个技术的过程，通过各种技术手段将蓝图中的各项指标实现。

目前对于游戏设计和游戏开发两者的关系，业界有很多不同的意见。一批人倾向于不

做区分，认为人为地把游戏的产生过程分割成两个阶段并无益处。而且游戏开发过程中肯定要经过多次反复修改，在开发过程中必然会返回去修改设计。但如果我们考察商用软件领域或者建筑领域的诸多实例，我们就会发现凡是设计和开发分得比较开的，过程规划就比较简单，质量控制比较容易实现，这一切都有利于最终产品质量的提高。笔者认为设计和开发这两个阶段客观存在，具有各自不同的特性，如若区别对待，将有助于提高游戏和游戏产生过程的质量。

在软件工程一章中，有各种过程模型的介绍，其中有一种新的 Triptych 模型，就体现了这种将设计和开发分离的思想。

游戏设计开发的三大基石

一个游戏的设计开发要成功，其基本的必要条件有三。分别为 vision（设计）、technology（技术）、和 process（过程）。三个条件，缺一不可。如图 1-16 所示。

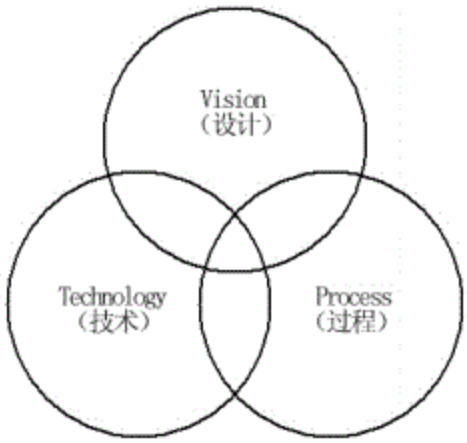


图 1-16 游戏开发的三大基石：vision、technology 和 process

最上面的一个圆圈里是 vision。这个词很难用中文解释，因为中文中没有对应的能够恰当解释它的词汇。翻译成“设计”，只是权益之计。这里简单地尝试性地解释一下什么是 vision，游戏一开始肯定是在某个人，或者某几个人脑子里想出来的——关于这个游戏是什么样的，如何玩等等。也就是说最后展现在玩家面前的游戏将是什么样的，游戏的设计者们肯定是先在自己脑子里形成想法的。然后他们要做的，就是把这个虚无飘渺的想法，表达出来，并通过自己和其他人的工作，把这个东西物质化实体化。在具体执行的过程中，必然会由于交流的原因或者技术的局限，使得想法和实际效果出现偏差。在这时候游戏设计者就必须有那种从总体上掌握的能力，知道什么要变通修改，什么要强调坚持，什么要进一步解释说明。同时，他还需要考虑到策略性的问题，就是开发的过程的大框架和协调问题。因此 vision 可以解释为对一个现在还未具体实现的事物的总体上的掌握、前瞻性的理解和策略性的考量。

有了 vision，如果没有技术（technology）的话，则各种美妙的想法就只能停留在虚无缥缈的阶段。技术使得 vision 固化（多次固化），成为可以看得见摸得着的实体。这里的技术，并不局限于计算机编程技术，如三维图形和人工智能等，而是广义地指各种技术性手段。技术是 vision 实现的保证。同时技术也是 vision 实现的最大障碍。技术的局限，使得 vision 中很多东西无法实现或者被迫做出妥协。

有了 vision 作指导，有了技术为手段，也还不一定能够把好想法转换成高质量的游戏。要创造出高质量的游戏产品，尚缺重要的最后一环，即过程（process）。我们知道游戏本身的最终产品形态是静态的（一张 CD 或者卡带），而制造产生这个游戏是一个长时间的动态的过程。静态产品的静态质量，要靠动态过程的动态质量来保证。过程由很多关系复杂相互牵制的环节和部件组成，如果任意环节或者部件出了问题，都会对最终产品的质量产生影响。因此对这个动态过程，一定要有规划和控制，以保证按部就班，按质按时完成工作。

通过上面对这三个条件的介绍可知，要开发出一个成功的游戏，这三个条件缺一不可。如果缺了 vision，做出来的游戏只是炫耀各种技术的演示而已。而缺了技术，则 vision 根本无法实现。如果缺了 process，则可能游戏质量不稳定、BUG 无数、项目拖期、成本超过预算。这三个条件，实在是游戏设计与开发的三大基石。缺了一个，游戏就成了站不稳的瘸腿猫。

我们更可以用这三个方面的指标去衡量一个公司的游戏设计开发的实力，图 1-17 显示了一种叫做 VTP 图的方法。

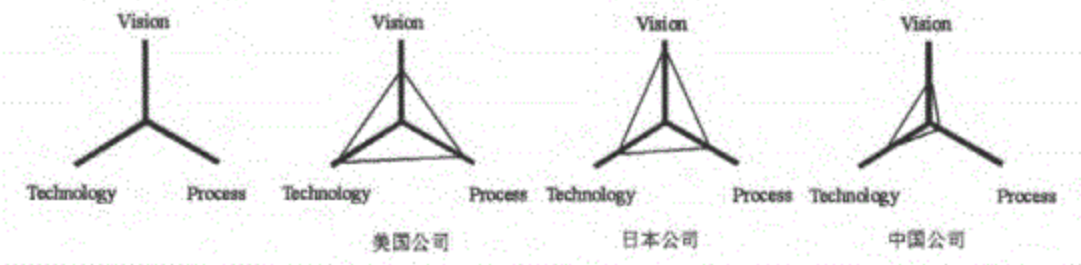


图 1-17　VTP 示意图。用三个坐标来表示 Vision, Technology, Process。在三个坐标上根据一个公司在此方面的实力做出标识，然后把三个点连接起来，形成一个三角形。三角形面积越大，说明这个公司游戏设计开发能力越强。上面显示了一个一般美国公司和日本公司的 VTP。可以看到美国公司和日本公司相比，技术上和过程控制上占一定优势，而 Vision 略逊。日本公司在 Vision 上更突出，技术上比美国公司稍微差一点，过程控制差得更多一点。最右边显示的是一个中国公司的 VTP,其三角形的面积比其他两个公司小多了。我们更可以看到中国公司和外国公司比，技术上的差距反而是最小的，Vision 上差一截，最迫切需要改进的则是过程控制。VTP 图不仅可以作为公司自我评估的工具，也是制定公司长期发展策略的有力武器

第二章　游戏理论

成功的游戏设计者们应该能够而且必须超越直觉判断和草率行事，他们必然在设计中或有意或无意地遵循着某些准则，正是对这些准则的正确理解和灵活运用保证了一部游戏作品在商业上和艺术上的成功，而这些准则是以下列形式出现的：

1．底层游戏理论及模型
2．专门技术及艺术表达手段
3．具体实践及反馈信息

构成了一种三层金字塔结构，如图 2-1。

图 2-1 游戏设计的三层金字塔结构

其中游戏理论及模型构成了金字塔结构的底层。游戏之所以为游戏，不同于其他艺术形式或娱乐形式，必然有其自身内在的规律。游戏理论，顾名思义是有关游戏的理论。它应该能超越具体单个游戏的纷繁复杂的外部特性，对有关游戏最本质最共性的问题进行理性的思考，最终提炼出游戏的一般模型，确立游戏设计的多项准则，对游戏的设计开发工作起根本的指导作用。游戏理论涉及艺术理论、心理学、计算机学科等诸多领域，从多角度探讨游戏与游戏设计者、游戏与游戏者之间的复杂关系。分层次研究游戏所包含的科学技术层、艺术审美层、心理情感层等问题。对上述问题的思考和阐述，不仅对有志于成为游戏设计者的人们，而且对广大对游戏抱有无比热忱的游戏者都是有益的。

建立游戏理论的目的，不仅在于针对那个被称为游戏的对象去考察和阐述有关其性质的永恒真理，而更重要的是针对游戏设计者在设计实践出现的某些问题，通过思考找出某些解决办法来。本章所涉及的一切都直接地或间接地与当今游戏界的状况有着实际的关联。

游戏模型

无论 RPG、SLG 还是 ACT，透过游戏千差万别的外部特性，考察游戏本身和游戏者构成的统一的游戏系统，可以发现这是一个动态的多层模型系统，如图 2-2 所示。游戏本体包含游戏内核（内层）和交互层（外层）两层。通过交互层，游戏可以有效地向游戏者展示内层的某些信息，又能接受游戏者的输入，交互层是游戏者眼中所能见到的游戏。而内层对游戏者来说相当于一个黑箱，游戏者通过交互输入一定的行为，内层根据自己的内部机制产生一定的反应，又通过交互层输出。这种根据一定输入决定产生什么样输出的内部机制对游戏者来说是不可见的。一个设计出色的游戏必须要细心地隐藏内层的运行机制，因为内层的运行机制一旦泄露，游戏者完全掌握了其规律，游戏在游戏者眼中将失去一切挑战性和趣味性，则游戏的生命周期也就至此结束了。

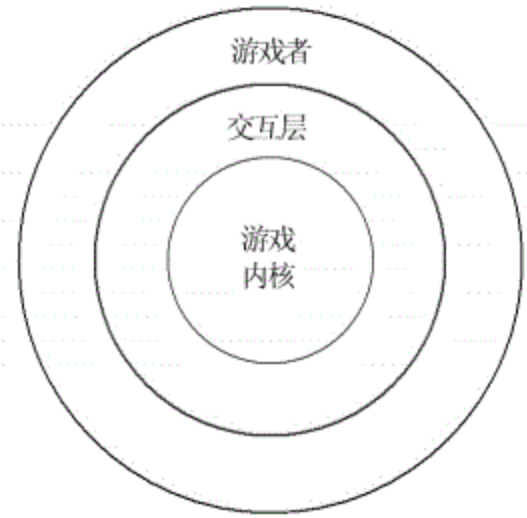


图 2-2 游戏模型

目前在游戏界存在的一种不良倾向，即忽略了游戏内核的设计，只注重游戏的交互层，甚至只注重属于交互层的一部分的外部效果，在新一代游戏机上许多游戏使用三维动画技术产生了令人瞠目结舌的视听效果，但被广大游戏者认为徒具外表，操作性游戏性则十分缺乏。因为游戏的交互层细分起来包括游戏的外部效果和操作性两部分。外部效果指展现在游戏者面前的画面、动画、音乐、音效和文字等。游戏者对外部效果是处在被动欣赏的位置。而操作性才是游戏所特有的使游戏者有一定主动性的关键内容。显然，游戏的操作性的重要性应引起设计者重视。而更关键的则是游戏内核，因为游戏的操作性只是决定了如何进行输入/输出行为，而并不决定输入/输出什么，决定输入/输出集及输入/输出对应关系的是游戏内核，它才是游戏真正的核心，才是游戏最深层次的灵魂，才是吸引游戏者为之废寝忘食的魔力所在，应把盲目投入外部效果的努力投入到对游戏内核的严格设计审核上，拥有了优秀的游戏内核，才有发挥外部效果的可能性，否则游戏的外部效果将成为无源之水、无本之木。这一点迄今为止都没有引起大多数游戏设计者的重视，他们大多数都被外部效果这一叶所障目，不见真正游戏实质之泰山。于是赶潮流、追风头，正如《DOOM》系列成功后，产生了一大

批模仿的游戏；取得显著成功的《MYST》也产生了大量的模仿者。它们和原型作品同样具有精细的三维动画、动人的音响效果、具有神韵的对象、光线颜色俱佳的画面，总而言之在可视媒体素材上它们一样的丰富，但模仿者并未享受到与原著相同的欢迎度。而开发出著名的《MYST》的 Broderbund 公司在开发新游戏时，先做出游戏的工作原型，再利用 70 人花费数个月对其测试。这种原型是游戏的一个纯文本版本，而与媒体的开发无关。调试人员考察此纯文本版本，摆脱具体媒体效果的影响，以便游戏内部的运行机制和交互功能得到彻底的测试。测试的目的主要是看用户对游戏机制的反应，最终决定游戏的内核是否真正吸引游戏者，值得去为它加上媒体的绚丽的外壳，进行实际的开发工作。

最后着重强调由游戏分层模型引出的第一条游戏设计准则：

决定游戏成功与否的永远都是游戏的内核而非游戏的外部效果。在游戏的设计中，特别是前期设计中，应给予游戏的内核足够重视，不要贪多求快、盲目地过早进入具体的外部效果设计。

游戏的情感世界

倘若一部游戏不能使游戏者获得某种深层的情感体验，那么它所受到的欢迎程度将是有限的。在确定了具有竞争性的游戏内部机制后，下一步需要考虑的就是游戏的情感世界，实际上是特定游戏者群的情感世界。下面要讨论的是游戏采用何种手段使游戏者获得情感体验。我们将会看到游戏独有的虚拟情境，以及普遍存在的焦虑产生及释放过程，还要附带提及期待及悬念问题。

虚拟情境

如果一件制造品的设计意在激起一种情感，并且不想使这种情感释放在日常生活的事务之中，而要作为本身有价值的某种东西加以享受，那么这种制造品的功能就在于娱乐。娱乐并不实用而只能享受，因为在娱乐世界和日常事务之间存在着一堵滴水不漏的挡壁，娱乐所产生的情感就在这间不漏水的隔离室里自行其道。游戏作为一种娱乐形式，也存在着自己的情感隔离室，称为虚拟情境。

游戏是以不干预实际生活的方式释放情感的一种方法，为了使情感可以不影响生活地释放出来，必须创造一种虚拟情境。所谓“虚拟”情境被理解为情感会因为被释放而“接地”，它不会涉及到那些在实际生活条件下会涉及到的种种后果。在现实生活中，如果一个人要表示对另一个人的忿恨，朝他挥舞拳头进行威胁等等，通常他会被认为是一个危险人物，而对被他威胁的那个人来说是尤其危险的；于是那个人会采用种种步骤来保护自己：或平息前者怒气，或申请警察的保护。如果人们认识到不会出这类事情，生活将照样进行，那么，在其中表现愤怒的那种情境就被称为虚拟情境。

为了在游戏和实际生活之间比较．我们可以把情感分成两部分，显然在游戏者游戏过程中，情感本身被当做目的加以对待；而在现实生活中，情感本身不是目的而是后果。现实生活中的情感也许会渗入游戏的虚拟情境的情感中，而游戏的虚拟情境中的情感不会影响到现实生活中来，因为在游戏中它们已经被“接地”释放了。

焦虑及其释放

从动态观点来考虑，任何情感在其存在过程中都有两个阶段：负荷即兴奋阶段，以及释放阶段。一种情感的释放，是在那种情感的推动下完成的。一定的情感产生一定的动作，借助这一动作我们又会最终消除那种情感，也就使我们自己从情感释放以前加在我们身上的紧张中解脱出来了。与此对应，虚拟情境的主要任务有两个：

1．唤起游戏者某种情感。
2．在那种情感的推动下完成某些动作，借助这些动作最终消除那种情感。

这正是游戏的独特性。我们发现很有趣的现象：在游戏中我们获得的愉悦和兴奋，其实是在一个高度负荷的情感释放过程中获得的，游戏也为这种释放过程提供了虚拟情境(场所)和游戏行为系统（手段），而产生这一高度负荷的情感及其所带来的焦虑、紧张等不适感的恰恰正是游戏本身。游戏本身在扮演一个“双簧”的角色，它实际上在一定程度上“玩弄”了游戏者。这一点与音乐带给我们的情感体验是一样的，在交响乐作品中，作曲家通过反复重现一个旋律片段，使我们进入某种情感体验，但随着旋律重复的继续，我们开始期待着它的变化和完成，产生疑惑、焦虑的情绪，随着时间的流逝，听众的紧张度越来越大，迫切需要从这种精神状态中解脱出来，这时作曲家等待听众的紧张度达到承受的极限后，马上使用与上一个旋律截然对立的另一个旋律来打破上一个旋律，从而使听众从某一个感情的高点跌落下来，获得强烈的解脱感。在 RPG 游戏中，为了让游戏者最终获得打败大魔头的快感，游戏往往通过无休止的三四流小妖反复进攻游戏者，在游戏者长时间的鏖战中增加焦虑和紧张感，而游戏设计者也适当掌握着度数，达到一定阶段后，游戏者最终通过艰苦战斗获得胜利，产生无以名状的快乐。

因此我们看到，游戏的目的在于产生确定的、预期的效果，即在某种类型的游戏者身上唤起某种情感，并在虚拟情境内释放这种情感，情感释放使游戏者获得快乐。游戏设计者把通过唤起某些情感来取悦游戏者作为自己的任务。整个游戏过程中，游戏者将体验许多个这

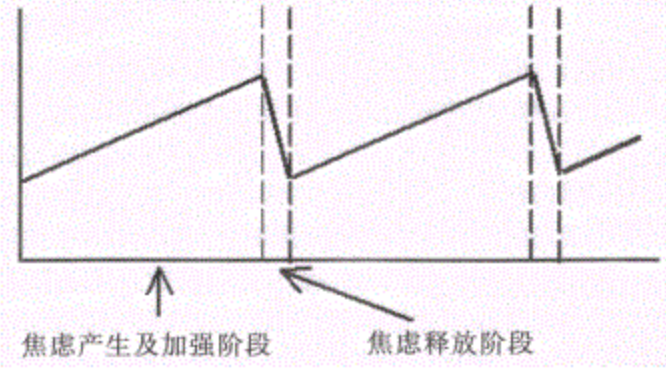


图 2-3 焦虑及释放示意图

样的焦虑——释放过程，他的情绪也处于波动之中，被游戏设计者灵活调动、层层推进，见图 2-3。

期待、悬念

游戏的一个重要组成部分是不可预见性，由此产生期待与悬念。游戏者在游戏前对游戏的最后目标将是什么有一个大概的感觉，但是关于当前这个过程将怎样带他到达那里，在途中将遇到什么曲折和障碍，他是不能断定的。游戏者在游戏中由于并不知道游戏内核的运行机制，因此对于自己动作的结果有一种忐忑不安的期待。在所有的游戏中，游戏者总是通过经验实现对不可预测性的抗争。从不可预测性上看，游戏可分为两种，一种称为技能游戏，另一种称为机会游戏。前一种游戏的内部运行机制是确定的，不可预测性产生的原因是由于游戏设计者故意隐藏了运行机制，游戏者可以最终通过对游戏运行机制的理解和控制（即某种技能）解除这种不可预测性。而后一种游戏中游戏本身的运行机制具有模糊性，具有随机因素，不能完全通过对游戏机制的解码消除不可预测性，游戏动作产生的结果是机会的。

期待是在与特殊的游戏规律相联系中发展起来的习惯反应。游戏者不断根据自己的期待决定下一步所要采取的动作，再根据动作结果修正期待，如果期待长期偏离实际效果则产生紧张、焦虑感。在游戏中，期待和对期待的控制很有意义。不能使游戏者的期待完全落空，这将使游戏者产生严重的挫折感，也不能使游戏者的期待完全应验，否则游戏将失去不可预测性。应该时而使游戏者的期待变成精确的结果，使其增强信心，获得欣喜；时而抑制游戏者的期待，使其产生疑惑，疑惑的时间持续越长，悬念的情绪就愈强烈，建立起来的悬念紧张度越大，由解决引起的情感上的解脱感就越强。悬念产生的价值不在其本身，而在于随之而来的解脱。期待、悬念及其解除过程实际上与焦虑、释放过程是相对应的。只不过一个更偏重于经验方面，另一个更偏重于情感方面。

最后着重强调由游戏的情感世界引出的第二条游戏设计准则：

在虚拟情境中要故意制造某种情感的负荷，使游戏者产生焦虑、紧张情绪，然后巧妙地调动引导游戏者，最终使其解除焦虑状态，产生高度的解脱感和兴奋感。同时要针对游戏者的期待，适度产生悬念对抗游戏者不断增长的经验，使其能感到游戏处于一种动态的变化中。

游戏的行为系统

游戏的行为系统，实际上是游戏内部运行机制决定的游戏的输入/输出集，它决定了游戏者在特定的游戏系统中可以做什么，不可以做什么。游戏行为系统的功能就是作为情感释放手段，它也是游戏交互性的重要组成部分。

封闭系统

任何游戏的行为系统，都是一个封闭系统。游戏者所具有的选择能力和处理能力都被严格限制在这一封闭系统中。这个封闭系统具有自己特有的反应机制，对应一定的输入产生一定的输出。

一个封闭的行为系统有两个组成要素：交互手段（输入/输出手段），交互法则（输入/输出对应关系）。

交互手段

目前我们所熟知的与计算机交互的手段，如：菜单、窗口、鼠标操作等 GUI（图形用户界面）的诸要素，都是建立在计算机科学与技术前一阶段研究的基础上，都是先在大学的实验室中得到试验与应用，后来成为工业界的实际标准。而作为软件中对市场和底层技术反应最快的游戏软件，往往最早应用这些研究成果。当然目前的交互手段有很大局限性，带有太大的计算机色彩，太拘泥于键盘和鼠标。VR 战士们是不应该通过一连串乱七八遭的按键（按键的设定就是一个稳定的封闭的行为系统）出拳的。目前很热门的研究领域如人机交互（Human-Computer Interaction）和虚拟现实（Virtual Reality），如能突破技术难关，则比之现在的交互手段将会有很大进步直至飞跃，将使我们置身于梦幻般的虚拟游戏世界，带来强烈的临场感受。这样一个虚拟的物理世界和一个虚幻的情感世界结合在一起，将产生一门最具震撼力的娱乐形式，其表现力将使电影电视相形见拙。当然这大概是很远的将来的事情了，因为技术上的难度还是比较大。（虚拟现实系统的实时处理能力和游戏故事情节多线拓扑结构及其数据组织是为游戏增添更大自由度的两个最大的瓶颈，此不赘述）

交互法则

作为行为系统第二个要素的交互法则（输入/输出对应关系），应该具有一定的动态性，也就是说在游戏者游戏过程中，游戏行为系统的反应机制从来都不是一成不变的。游戏者在游戏过程中通过学习、运用、获得反馈，从而逐渐了解游戏的内部机制。这时游戏应该改进反应机制，迫使游戏者调整自己的学习—反馈—掌握—运用曲线，基本保持游戏的全程新鲜感。当然反应机制应该采用渐改的方法，采用向下兼容、渐进发展的行为模式。现在的经验是将来可用的、有用的，但将来的情况又不是现在的经验完全对付得了的，即现在是将来的真子集。

由游戏的行为系统我们可以引出第三条游戏设计准则：

游戏的行为系统是一个封闭系统，但不是一个静态系统，应采取向下兼容、渐进发展的行为模式，使游戏尽可能不被游戏者“琢磨”透，使其尽可能长地具有挑战性。

第三章　游戏类型

游戏类型（genre）的提出是游戏作为一种娱乐形式发展到一定成熟程度的标志。正如电影中的相似概念“类型电影”一样，游戏类型（game genre）和类型游戏（genre game）的概念的提出，代表着游戏设计理论的提高，同样体现着游戏在社会和玩家中的影响的扩大。

对游戏类型的分析，目前还处于比较初级的阶段。因此，本章从类型电影的分析理论中借鉴了一些思想和方法，目的是探讨游戏类型的历史和由来，未来可能的发展，以及为什么产生了这么多种类型的游戏，游戏类型的功能是什么，社会影响是什么等问题。

游戏类型的由来和功能

游戏类型的定义

什么是游戏类型，或者类型游戏?这个问题很难回答。虽然大多数玩家都可以如数家珍地举出一大堆游戏类型和它们的代表作，如 RPG（角色扮演类）和《最终幻想》系列，FPS（第一视角射击游戏）和《UNRE-AL》系列。但要仔细地说明类型这个概念却是很难的事情。

简而言之，游戏类型是一种分类法，是某种技术上的妥协。如果我们把游戏简单地看作是对人生和世界的模拟的话，由于技术上的局限性，游戏设计者不可能把握这么广阔的主题。他们必须将世界、社会、人生的各种错综复杂的主题分门别类，在一个较小的范围内，利用现有的可行的技术来予以表现，这样各种游戏类型就诞生了。

有的类型起源于旧有的手工或者纸牌游戏，比如说美式 RPG 起源于他们的纸上 RPG 游戏。有的类型则是由一部开山之作确立，比如《DOOM》之于 FPS。还有的类型是几个类型的融合，比如《黑与白》是宠物类游戏和上帝类游戏的中和。

游戏类型的演变过程

所有的类型都有一个循环往复的演变过程，一般被称为类型环（genre cycle），如图 3-1 所示。

图 3-1 类型环

这个环所表达的，就是类型的演变过程。整个环分三个阶段。一开始是类型初始阶段，这时候类型还处于雏形阶段。它的特征有可能在各种游戏中显现，这个游戏体现一点，那个游戏体现一点。但总的来看，各种特

征还没有得到总结和确认，零散而不系统，设计者对这个类型的游戏的规律没有理性的认识和把握，玩家也还没有把这种游戏类型和其他游戏类型区分开来。第二个阶段是类型确立阶段。在此阶段，这个游戏类型的独立性被承认，其特有的规则被总结归纳，成为所谓的“范式”，并在玩家中得到广泛认同，并且拥有了稳定的特定玩家群体。这个玩家群体是这种游戏类型的铁杆支持者，或者说是所谓的“专业”玩家。第三个阶段是类型融合阶段，一个类型在进入其成熟期后，必然要有所变革发展。新的要素会被引入，或者和其他类型融合，这意味着前一个阶段形成的关于这个类型的范式被打破。类型重新又回到一定的混沌状态，等待新的规则和范式的确立。这样就成了一个周而复始的循环。

举一个简单的例子来说明这个类型环。作为 FPS 的开山之作《DOOM》，并不是从石头里蹦出来的。它也是从很多比它更早的游戏中采取有用的素材，比如早期的俯视视角的迷宫类游戏和第一视角的射击游戏（如《打鸭子》）。《DOOM》推出后，把迷宫、敌人、第一视角射击组合到一起，确立了 FPS 这个新的游戏类型。而《UNREAL》系列，《HALF-LIFE》系列则体现了这个类型的发展和成熟。而随着技术的发展，更大胆的想法又诞生了——把策略游戏和 FPS 合在一起，设计出虚拟的战场。你可以扮演步兵，去进行 FPS 游戏。也可以扮演坦克兵，去进行模拟类型的游戏。当你战功卓著后，可以得到提升，去扮演将军，去进行策略型的游戏。这就到了类型融合的阶段。目前《WWII Online》（网上模拟第二次世界大战）这个游戏，就是向这方面努力的第一个作品，虽然并不成功。

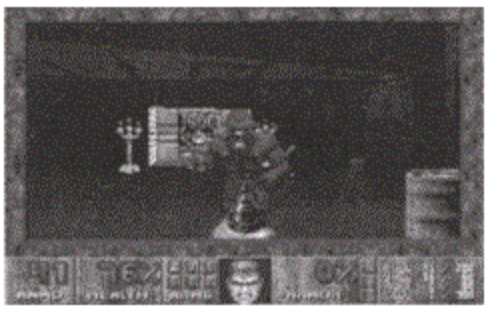


图 3-2　FPS 开山之作《DOOM》

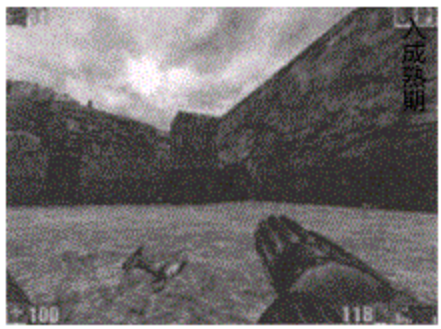


图 3-3《UNREAL》标步前进

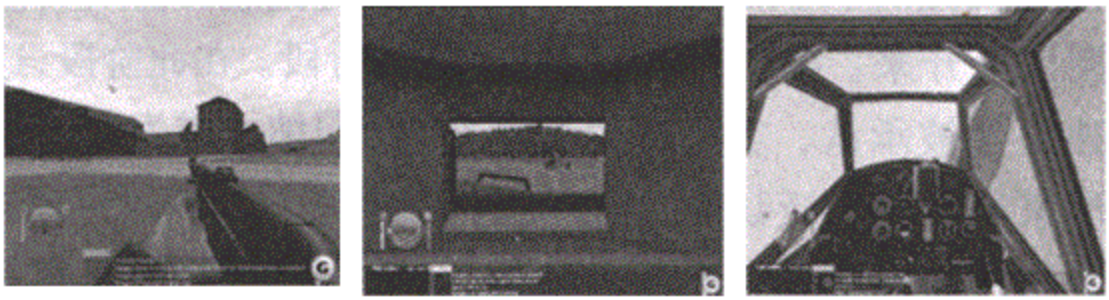


图 3-4《WWII Online》试图整合多种类型的游戏，你可以作为步兵玩 FPS，也可以开坦克和飞机

类型范式

谈到类型，就必须要谈到类型范式（genre conventions）。所谓类型范式，是指同属于一个类型的游戏和游戏之间，必定有某些东西是共通的，是被不断重复的。之所以这个类型的游戏和其他类型的游戏不一样，就是因为类型范式上的不同。当玩家们一开始游戏，就知道这个游戏是 RTS，那个游戏是 RPG，也是因为玩家们对游戏类型范式都有一定的认识。

类型范式由许多部分组成。其中最主要的有四项：主题（theme）、故事情节（plot）、视觉风格（visual style）和游戏规则（game mechanism）。其中前三项是游戏和电影所共有的。

1．主题：不同的类型有不同的主题。西部片中的复仇主题，香港功夫片中宣扬的江湖义气，游戏中 FTG（格斗游戏）的争取最强和超越自我的主题，等等。

2．故事情节：不同类型有不同的故事情节。早期 ACT 游戏的英雄救美，RPG 游戏的英雄之旅等。

3．视觉风格：不同类型有不同的视觉风格，包括色彩、构图、光影、特殊的物品和场景。比如 film noir（黑色电影）中阴影的应用；音乐片华丽的服装和布景；一辆老式福特汽车里一个人手持冲锋枪探出身来，则是早期警匪片的特征。对游戏类型来说不同类型的视觉风格尤其明显：一个三维的屋子内部，屏幕正中一件武器，这肯定是 FPS。一个俯视的地图，底下一个菜单，上面各种选项，这八成是 RTS。AVG 游戏一般比较阴暗，RPG 游戏一般屏幕上零碎很少，等等。下面的三个图例分别来自《帝国时代 2》、《傲世三国》和《哥萨克：欧洲战争》三个 RTS 游戏，我们可以很清楚地看出它们的视觉风格和屏幕布置基本相同，也就是构成了一定的范式。

图 3-5（a）图为《帝国时代 2》，（b）图为《傲世三国》，（c）图为《哥萨克：欧洲战争》。可以看出相似的视觉风格和屏幕布置

4．游戏规则：对游戏类型范式来说，又比电影类型范式多了一项重要内容——游戏规则。也就是说，不同类型的游戏之间最大的不同点是游戏规则的不同。RTS 之所以是 RTS，是因为其特殊的游戏规则。包括建筑、生产、微观管理（micromanagement）、战争。FPS

之所以是 FPS，是因为关卡（level）、探索（exploration）、射击（shooting）三要素。而从更具体的方面来看，游戏的基本操作方法也是范式的一部分。比如 FPS 类型，一般是键盘控制行走，鼠标控制方向和射击，鼠标左右移动对应左顾右盼，上下移动对应仰俯。这种操作方式是经过无数次考验，得到玩家的普遍认同，已经定型，被证明是最符合这种游戏类型的配置。

前面介绍了游戏类型范式的四个主要方面。当然，范式并不是一成不变的。玩家们在玩一种游戏的时候，除了期待着相同的熟悉的范式，也期待着新奇和革新。因此，游戏实际上是范式和革新、熟悉和新奇这两对矛盾的复合体。

类型的作用

类型的产生和流行，最主要的原因是类型可以起到以下几点作用：

1. 技术上的局限性：这一点在前面已经谈及。由于技术上不可能实现过于宏大的游戏主题和游戏世界，设计者们必然要做出妥协，找到实际可行的主题和边界。而随着技术的发展，边界又将会被不断打破，游戏世界也将不断扩大。类型这一概念的提出，使得这种扩大和发展在一种有序的情势下进行。

2. 划分玩家群，锁定特定玩家群体：观众定位的概念，在电影界很早就出现了。因为一部电影很难让所有人都叫好，必然首先要吸引一小部分人，也就是基本观众群。在牢牢地抓住这部分基本观众后，才能进一步考虑边缘观众群。类型电影的出现，使得观众群的划分更明确合理。在拍摄电影之前，制作者们就可以确定自己这部片子的主要观众是哪些人。青春片是给中学生看的，家庭伦理片吸引的主要是成年人，枪战片的男性观众居多，而浪漫爱情片主要是哄女性流泪。同样，游戏类型也有这种分类的功能。比如说《MYST》这种类型的游戏就吸引了很多中年人，因为《MYST》节奏慢，要动脑子。让这些中年人去玩年轻人中流行的 FPS，则绝无可能。一分钟下来，他们肯定会头晕眼花。

3. 使玩家更容易掌握新游戏，游戏更容易上手：由于类型范式的存在，使得玩家们对同一种类型的游戏的操作比较熟悉，容易掌握。研究表明，虽然购买游戏时会附带说明书，但玩家很少阅读。玩家是通过摸索和对以前同类游戏的经验来玩新游戏的。

4. 社会的影响和影响社会：类型在某一历史阶段可能非常流行，然后衰落，然后再流行起来。这种波动起伏的过程是和社会现实生活有关的。也就是说在一定的时期内，流行的类型和当时社会形态有密切的关系。一方面，流行的类型反映了当时社会生活状态。比如说大萧条时期的歌舞片，是处于绝望状态的社会的自我麻醉剂。二战后的 50 年代，在原子弹的阴影下，怪兽影片（受核辐射而变态的巨大生物，如 Godzilla）的出现和盛行。另一方面，类型也在影响着社会上的思潮。在目前来说，游戏对社会的影响和社会对游戏的影响

还不像电影那么显著，因此还不能清楚地看到流行的游戏类型和社会思潮之间的互动关系。但是随着游戏的影响越来越大，这种关系是迟早要显现的。

类型和文化

不同国家中诞生了体现自己独特文化风格的不同的电影类型，比如说香港的武打片，德国的 Heimatfilm（描写小城市生活的伦理片），墨西哥的 cabaretera（描写风尘女郎的歌舞片）。游戏也是一样。在不同国家里和不同的历史文化环境下，会发展出不同类型的游戏。同样地，同一个类型的游戏，在不同文化和国家中所受到的待遇不一样。中华文化地区所流行的武侠 RPG，是其他地区和文化所不可能产生的。而日本的恋爱模拟游戏，对美国玩家们也是不可想象的。

虽然不同文化的鸿沟使得不同类型的游戏在各国推广起来会有难度，但大的趋势是融合和交流。正如本书总论一章中提出的未来游戏的 3 个 i 中，最后一个是 internationalization，也就是全球化的游戏，能够被不同种族和文化的玩家所普遍接受的游戏。大的跨国游戏公司，已经在这方面做出了一定的工作。远一些的例子如 FF7 把日式 RPG 引入美国，使得美国人对 RPG 的定义完全改写。近一点的例子如《星际争霸》在韩国的盛行。

目前很多游戏设计师都试图突破文化的界限。他们所采用的方法就是把游戏层和文化层分离，把文化作为一个表层，而不是核心。这样的话，基本的游戏层，包括游戏性和游戏机制，是全世界玩家都能接受和理解的。文化层则根据各地各人理解不同，即使不能完全理解也不会影响对游戏层的掌握，如图 3-6 所示。

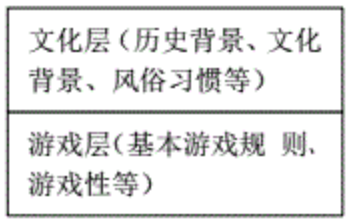


图 3-6 把游戏层和文化层在某种程度上分离

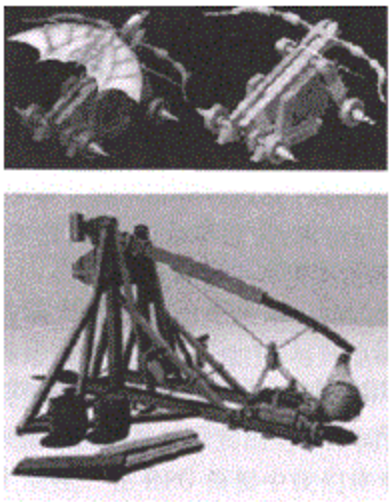


图 3-7 《傲世三国》中的火神鸢和《帝国时代》中的 trebuchet。虽然文化来源不同，实际功能是相似的

下面举一个游戏层和文化层分离的最简单的例子。同为 RTS 游戏，微软的《帝国时代》中，有西方中世纪的各种武器系统。而《傲世三国》中，则有由中国传统文化中衍生出来霹雳车和天神鸢等新奇的东西。但所有的武器系统，虽然名字不一样，有英文也有中文，但各国玩家似乎并不受影响，都能够顺利玩下去。中国人玩《帝国时代》没有问题，《傲世三国》在美国也卖了一些（霹雳车、天神鸢这些东西是根本不可能翻译明白的）。为什么呢?原因就是：虽然那些武器系统名字千差万别，无法翻译，外形千奇百怪，但不外乎以下几种：速度快但威力弱的，速度慢但威力大的，生产快速便宜但威力弱的，生产费时费钱但威力强的。也就是说名字

虽然不一样，但基本的游戏规则和武器系统平衡原理是一样的。玩家们只要对游戏层有了足够的认识，即使背景文化不清楚也能玩下去并享受到乐趣。

游戏类型简介

以下简单介绍已经成为业界和玩家共识的各种游戏类型。

RPC（角色扮演类）

RPG（Role-Playing Game）游戏无疑是最受欢迎的游戏类型之一。但很难对其进行确切定义。下面采取用其性质或者说其构成要素来定义其本身的方法，在阐述了下述问题之后，对 RPG 游戏的定义问题也就得到了解决。

对人生的模拟

如果说飞行模拟类、体育类、动作类等类型的游戏都是对现有的某项人类活动的再现与模拟的话，那么 RPG 游戏体现的则是对整个人生的再现与模拟。正因为如此，RPG 游戏所构造的情感世界是所有类型的游戏中最为强大的，能带给我们深刻的体验感。这种体验感来源于每个人内心深处对人生的感悟和迷茫、无奈与苛求、失意与希望，所有这些都可以在 RPG 游戏所构造的虚拟人生的情感世界中得到共鸣。

RPG 游戏的三维空间

可以用一个三维坐标系统来定位 RPG 游戏，所有类型的 RPG 游戏都位于这个坐标系所界定的三维空间中。

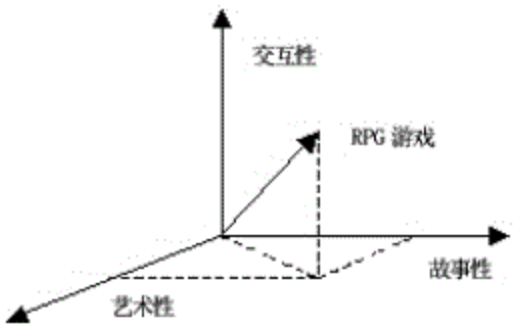


图 3-8　RPG 游戏的三维空间

图 3-8 中的三个坐标轴所表示的内容分别为构成 RPG 游戏的三大特性，即：

- 艺术性（Z 轴）
- 故事性（Y 轴）
- 交互性（X 轴）

倘若我们把每个坐标轴的最大坐标值定为 1，那么坐标点（0，0，1）代表纯粹的艺术

作品，如视觉艺术和音乐作品等；（1，0，0）点代表完全的操作性活动，如体育运动；（0，1，0）点则代表故事情节及其纯线性的展现和播放，如电影剧本、VCD 和录像带。而 RPG 游戏则位于点（x，y，z）。其中 0<x<1，0<y<1，0<z<1。而不同类型的 RPG 游戏，在这个三维空间所处坐标不同。偏重交互性的，其 x 值较大；偏重故事性的，其 y 值较大。需要指出的是：x，y，z 的值都不能为 0，因为构成 RPG 的三大特性缺一不可。

在艺术性上，RPG 游戏和其他类型的游戏一样，借助于多媒体视听的强大能力，综合了美术、动画、音乐、音效、文学、戏剧等多种艺术娱乐表达形式。在故事性上，与其他游戏类型相比，RPG 游戏和电影的关系更为密切。因为它们的“情节”都是由“剧本”严格限定的，也就是单线发展的。但与被动欣赏的电影不同的是 RPG 游戏给游戏者提供了虚假的主动性。在 RPG 模型中我们将要看到这种虚假的主动性是如何达成的。这种虚假的主动性和被动设定的故事情节相结合而构成了 RPG 游戏的交互性。

RPG 模型

剥去各种 RPG 游戏的外部特性，我们可以看到 RPG 游戏的普适模型，如图 3-9。这也是所有单线发展的 RPG 游戏的拓扑结构。它由两部分构成：一部分是主控部分，也就是交互部分。当主控部分起作用时，游戏的操纵权被授予了游戏者，游戏者可以充分利用游戏所赋予的交互手段进行输入；另一部分是设定的被动的剧情，由线性排列的一连串事件组成。所谓事件，就是在一定时间内从游戏者手中剥夺游戏的操纵权，从而使游戏按设定的轨道向下发展，比较普遍的是被动地显示一段动画。在游戏过程中，游戏者获得操纵权后，进行输入。一旦引发某个事件（显然单线 RPG 游戏同一时刻只可能引发一个惟一的事件），游戏者操纵权被剥夺。当事件完成后，操纵权又被赋与游戏者，用来引发下一个事件。游戏者就是这样不停地交替地被赋予和剥夺游戏操纵权，事件也就这样按设定的轨道发展下去。所以我们发现：RPG 游戏中游戏者只是虚假地拥有主动性，游戏者实际上只拥有决定何时引发事件的权利（玩 RPG 游戏的能力高低就在于是否能很快找到引发事件的“点”，能力低者会淹没于 RPG 游戏中各种信息的海洋中，不知哪个信息是决定事件发展的关键），而不具有任何决定事件发展顺序或事件本身的权利。

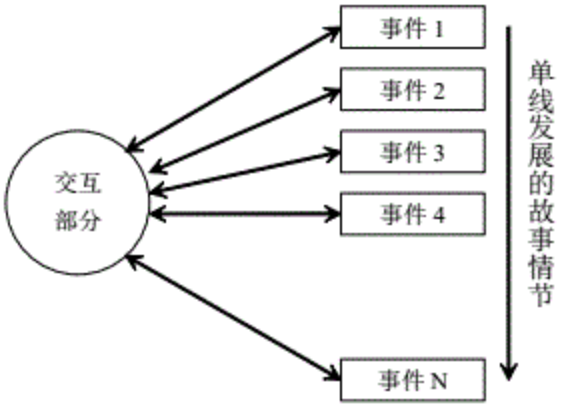


图 3-9　单线 RPG 模型

RPG 游戏的代表作

RPG 一般又分为美式 RPG 和日式 RPG。日式 RPG 的代表作是《最终幻想》（Final Fantasy）系列和《勇者斗恶龙》（Dragon Quest）系列。美式 RPG 的大作如《柏德之门》和《Diablo》系列。日式 RPG 是 RPG 的鼻祖，也可以说是比较“纯”的 RPG，而美式 RPG 中就加入了很多新的要素，比如说实时战斗和人工智能等。可这么以说：日式 RPG 向故事性倾斜得比较多，而美式 RPG 向交互性倾斜得更多一些。

ARPG（Action RPG，动作 RPG）是 RPG 的一个子类型。它引入了动作游戏的要素和实时性，也就是说把交互性发挥到了很高的程度。我们所说的美式 RPG 一般都是 ARPC。

图 3-10 《最终幻想 8》和《柏德之门》是日式和美式 RPG 的两大代表

近几年随着网络游戏的兴起，一种新的 RPG 子类型出现了。这种新类型叫做大规模多用户网上 RPG（缩写为 MMORPG，Massively Multiplayer Online Role-Playing Game）。这种 RPG 游戏的故事性已经被极度弱化，因为在网上那么多人玩基本上无法控制故事情节，而交互性则得到了提高，包括玩家与玩家之间的交互。例如我们天晴数码公司出品的奇幻武侠类型的《征服》和奇幻宠物类型的《幻灵游侠》，见图 3-11 和 3-12：

图 3-11《EverQuest》是目前最成功的 MMORPG

图 3-12 大中华文化地区所流行的武侠 RPG 的代表作《仙剑奇侠传》

另外值得一提的是在中华文化地区，武侠 RPC 是一种独特的类型。但这种类型的特点更多地是在文化层上，而非游戏层。

图 3-13 显示了 RPG 游戏的子类型们在 RPG 三维坐标中的相对位置。

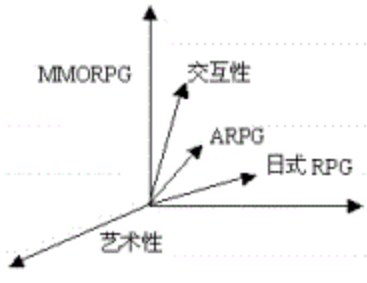


图 3-13 RPG 子类型在 RPG 三维坐标中的位置

ACT（动作类游戏）

ACT（Action Game）原来是家用游戏机上最流行的游戏类型。在任天堂 8 位机的时代，《魂斗罗》《双截龙》等横卷轴 ACT 曾经风靡一时。最早的玩家们被冠以“闯关族”一名，就是因为在横卷轴 ACT 游戏中，玩家要从左向右一关一关地闯过去，最终打倒大魔头。

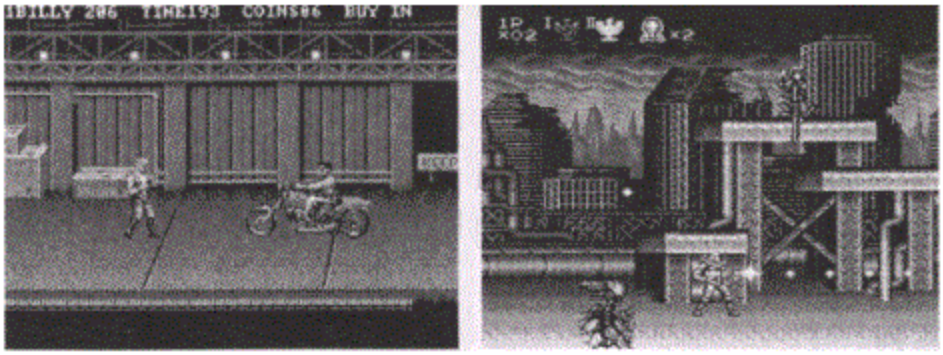


图 3-14 在 20 世纪 80 年代末风行一时的《魂斗罗》和《双截龙》系列

以现在的眼光来看早期的 ACT 游戏，我们会发现这种游戏才是真正传统意义上的“游戏”——它侧重于手眼协调和条件反射。每一关的敌人都是从固定的地方跳出来，按固定的轨迹运动，可以说没有任何智能可言。因此，玩家如果玩了足够多次之后，对整个游戏可以说是成竹在胸。ACT 游戏的乐趣，就在于通过不断的训练达到某种技巧上的娴熟，并培养出一定的条件反射，然后在玩游戏时达到下意识或者无意识的高超水平——一种行云流水似的流畅感觉。

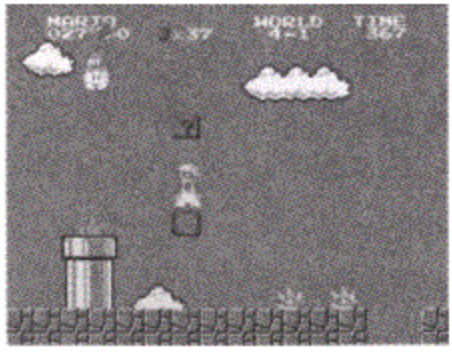


图 3-15 FC 上的《超级马里奥》（旧译《超级玛丽》）是不少玩家的启蒙之作

但随着 32 位游戏机的问世和三维图形的广泛使用，ACT 游戏反而势微了。在很长的一段时间内没有推出重量级的 ACT 作品。其主要原因是三维图形使得原有的二维 ACT 游戏的游戏规则受到了破坏。在三维世界中，复杂地形和多种路径的产生，使得二维 ACT 中直线前进的固定模式被打破（在最早的三维 ACT 中，游戏世界是三维的，但路线只有一条，是一维的。但这种模式很快就被抛弃了。）在三维世界中，视角和镜头的处理更为复杂，为玩家的观察和控制带来了困难。在三维世界中，敌人必须具有更高的智能和自主性，才能够在复杂的地形里找到并攻击玩家。这三点使得玩家在玩三维 ACT 游戏时，永远也不可能建立玩二维横卷轴 ACT 时那样的简单的条件反射了。

既然三维 ACT 已经把旧有的二维 ACT 的包括游戏规则在内的类型范式打破，从某种意义上说，新的三维 ACT 和旧的二维 ACT 应该属于不同类型。目前的三维 ACT 可以看作是 FPS 和冒险类游戏的中和。FPS 对三维 ACT 的影响体现在三维迷宫和 NPC 上，所不同的是 ACT 游戏一般采用第三视角，而 FPS 采用第一视角。冒险类游戏对 ACT 的影响主要体现在解谜（puzzle）上。目前三维 ACT 游戏最雄辩的例子就是《古墓丽影》系列。

图 3-16《古墓丽影》系列可以看作是 ACT 游戏和冒险类游戏的融合 ACT 游戏一般采用第三视角

图 3-17《鬼武者》(Onimusha) 和 (合金装备) (Metal Gear Solid) 是目前有代表性的 ACT。它们可以说是以卷轴 ACT 闻名的老牌日本厂商对三维 ACT 的新规则体系的尝试和思考

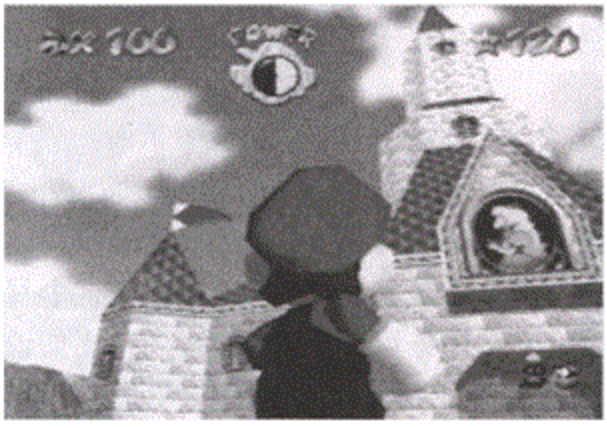


图 3-18 硕果仅存的《马里奥 64》。作为惟一能够在二维和三维世界里同样成功的 ACT 系列，有必要仔细研究马里奥的演变过程和规则改变

FPS（第一视角射击游戏）

FPS（First-Person Shooter）是从美国流行起来的一种游戏类型。它起源于早期苹果机上的迷宫游戏和游戏机上的 ACT 游戏。在融合了两类的特点后，通过引入第一视角和三维图形使得游戏的表现力得到极大提高。首先是置入感（immersion）的提高：三维世界和第一视角的应用使得玩家第一次能够感到他们“面对”着一个真实的三维世界。其次是交互性（interactivity）的提高：三维地图使得玩家们摆脱了 ACT 游戏由一个路线前进的限制，玩家可以沿多种路径到达终点，更增加了搜索前行的乐趣和不确定性。NPC（Non-Player Character）概念的提出使得玩家所要面对的对手有了一定的智能和适应性，不再是像 ACT 游戏中的敌人那样定点定时出现。

图 3-19 《QUAKE》系列和《UNREAL》系列是 FPS 的代表作

FPS 游戏的要素有三：

●三维关卡（3D level）：关卡这个要素是从 ACT 游戏中继承的。整个游戏由一系列关卡组成。每个关卡有自己独特的三维场景。

●任务（mission）：任务就是玩家在一个关卡里要完成的使命和要达到的目的。不管是夺取旗子也好，还是安置炸弹也好，达到这个目的才能通过这一关，进入下一关。任务可以是相互嵌套的，比如说一个主任务的完成需要先完成几个子任务。

●NPC：NPC 是具有一定智能和适应性的，不是由玩家来控制的敌方或者友方。游戏业人工智能技术的应用最初就是从 NPC 上发展起来的。NPC 要阻碍（敌方）或者帮助（友方）玩家完成任务。

图 3-20《Half-Life》被认为是 FPS 的突破性作品。在《QUAKE》和《UNREAL》系列的阴影下仍能脱颖而出，受到空前好评

目前 FPS 的发展趋势是：第一，提高 NPC 的人工智能，引入小组机制，使得 FPS 不仅仅是疯狂扫射和冲锋，而有了战术配合。第二，被 FPS 长期忽视的故事性也越来越得到强化。最后，联网对战使得 FPS 更加刺激。受到空前好评的《Half-Life》就代表了这三种趋势。

FTG（格斗游戏）

格斗游戏也是一个长盛不衰的游戏类型。其基本特征是在一个狭小的场景里，通过复杂的按键序列来控制双方角色进行一对一的打斗。二维格斗游戏一般采取平视镜头。三维格斗游戏一般使用第三视角。比之 ACT，FTG 的背景固定，玩家的注意力完全在对手身上。

目前 FTG 的发展趋势是三维场景变得更大更复杂，可以使用场景里的道具，以及使用物理编程来使得打斗真正符合现实世界的力学原理。

图 3-21《街霸》（Street Fighter）和《格斗之王》（King of Fighters）系列是 2 维 FTG 的代表作

图 3-22《VR 战士》（Virtua Fighter）和《铁拳》（Tekken）系列是 3 维 FTG 的代表作

RTS（实时策略游戏）

和 FPS 一样，作为一种游戏类型，RTS（Real-Time Strategy Game）也是由一两个开山大作所确立的。1992 年的《沙丘 2》（Dune 2）和 1995 年的《命令与征服》（Command & Conquer）是早期 RTS 的代表作。从 90 年代中后期开始，RTS 的发展十分迅猛。很多最新的技术都是从 RTS 游戏提出来的，比如说寻径算法、人工智能、指令序列等。这些技术后来又被应用到了其他类型的游戏上。

所谓策略，包括两个含义。从广义的角度来说，策略是指运用政治、经济、军事手段来实现国家意志和维护国家利益。对游戏来说，一般是通过资源采集、生产、后勤、开拓和战争来振兴本种族或国家，战胜其他种族或国家。从狭义角度讲，策略是指用来击败敌人的各种军事指挥手段。作为 RTS 游戏，既包括广义的策略也包括狭义的策略。

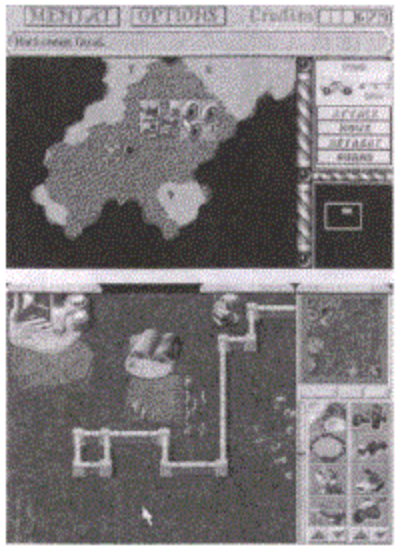


图 3-23 《沙丘 2》（Dune 2）是 RTS 的开山之作。《命令与征服》（Command & Conquer）是早期 RTS 的代表

所谓实时，是指玩家在紧张地派兵布阵的时候，敌方也在进行各种操作。当玩家停下来时，敌方不会停下来等他。这样双方都在和时间赛跑，使得游戏更加紧张刺激。

大多数 RTS 的游戏规则都遵循“采集一生产一进攻”的三部曲原则。即通过对几种资源的采集和利用，来构建基地或城市，生产武器，组建军队，然后向敌方发起进攻。RTS 最重要的两个要素是资源管理和狭义的战争策略。有的游戏侧重于战争策略，基本省略了资源管理，比如像《MYTH》这款游戏。也有的游戏资源管理的任务很重，甚至到了繁琐的地步，比如在《傲世三国》中，引入了后勤的概念。

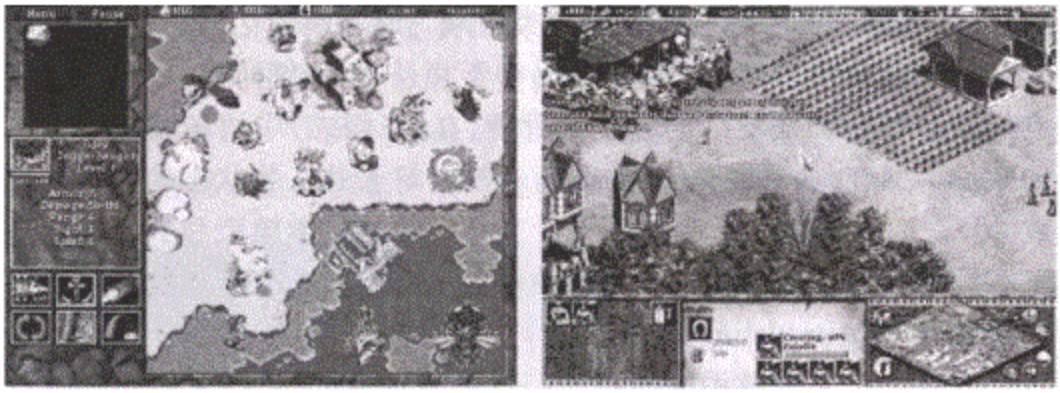


图 3-24 《魔兽争霸》（Warcraft）系列和《帝国时代》（Age of Empires）系列是 RTS 游戏成熟期的代表。《帝国时代》一开始设计时本来要模仿《文明》（Civilization），开发到一半时《魔兽争霸》问世了。被其魅力所震慑的设计师们于是改变了《帝国时代》的设计思想，把它由一个模拟游戏改成了 RTS

TBS（回合制策略游戏）

TBS（Turn-Based Strategy Game）是很古老的一种游戏类型。早期计算机的运算能力捉襟见肘，无法在游戏中实现真正的实时对抗，于是回合制策略游戏就作为一种妥协方案被推出。在回合制的游戏中，己方和敌方交替采取行动。只有当玩家完成己方的操作后，敌方才能开始考虑对策并实施行动。这样交替进行的话，计算机就可以集中全部资源处理一方的计算要求了。

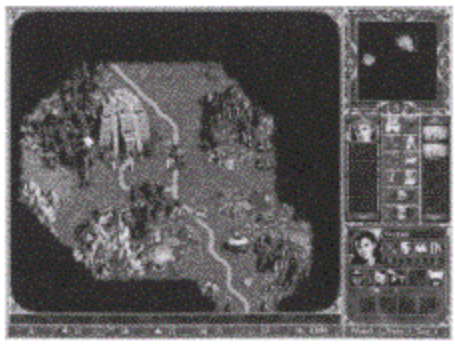


图 3-25 《魔法门英雄传说》系列是硕果仅存的 TBS 游戏

TBS 节奏比较缓慢，游戏中一半时间是在等待。RTS 游戏问世后，TBS 已呈衰落之势。目前只有几个系列还在靠惯性支撑着。如《魔法门英雄传说》（Heroes of Might and Magic）系列。

SLG（日式模拟游戏）

SLG（Simulation Game）是定义最混乱的一种游戏类型了。这个类型是日本人提出的（东方民族在分类归纳方面一向不太擅长）。美国游戏业的字典里没有这个类型的定义。它实际包括了几种差别很大的游戏。既包括策略模拟类游戏，如《三国志》系列；又包括恋爱模拟类游戏，如《心跳回忆》系列。表面看起来策略模拟和恋爱模拟真是风马牛不相及，但仔细研究一下，则发现它们共有的特征是复杂的数字式管理，在《三国志》中的各城市各武将的数值。而在《心跳回忆》中也是有各种各样表达人物状态的数字。因此，SLG 中的模拟，实际上是用一个非常粗糙的数学模型和数字式管理来实现的。这些日本游戏并没有使用任何高级的人工智能技术和认知模型（cognitive modeling）。

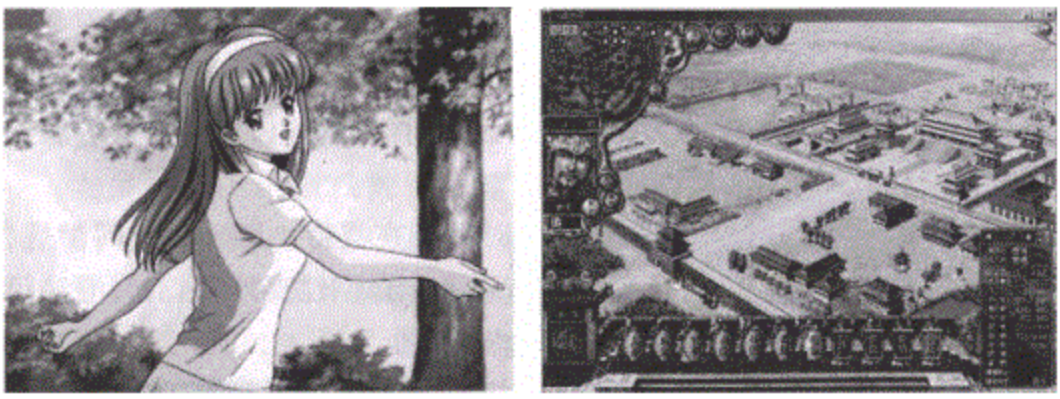


图 3-26 《心跳回忆》是恋爱模拟类游戏，经久不衰的《三国志》系列是策略模拟类的代表作

另外一个非常棘手的问题是《三国志》这样的游戏，它既有回合制战斗的要素，又有数字式管理，究竟应该归在 SLG 类还是 TBS 类。笔者这样认为：如果是数字式管理为主，回合制战斗为辅，数字式管理的重要性远远大于回合制战斗中策略的使用，则这个游戏是

SLG。如果一个游戏一开始是围绕着回合制战斗设计的，每次战斗胜负的影响远远大于数字式管理的作用，则应该归入 TBS 类。所以《三国志》系列要算到 SLG 类。

SIM（美式模拟游戏）

和 SLG 一样，SIM（Simulation Game）也是一个定义混乱的类型。这个类型是美国人定义的。一般来说，它包括了《模拟城市》《模拟医院》《模拟人生》等一系列以“模拟”为招牌的游戏。这些游戏也是对现实世界人类生活的一部分的模拟。比之日式模拟游戏，它们对数字的依赖少些，交互性更强，使用了更复杂的人工智能技术。也就是说美式模拟类游戏中所构建的世界模型比日式模拟类游戏更复杂，更具模糊性。

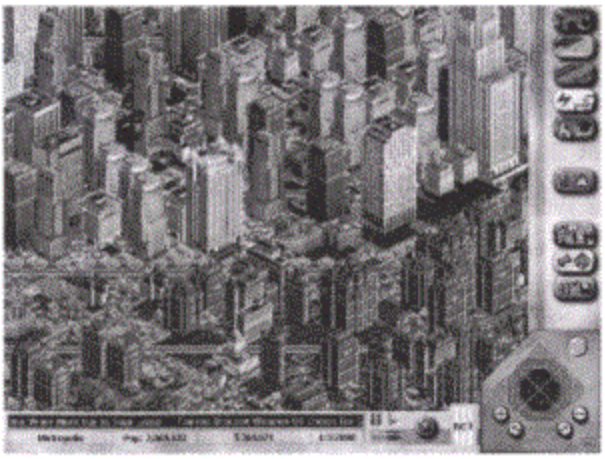


图 3-27 《模拟城市》（Simcity）系列是美式模拟游戏的典型

这些游戏的一个共同点是玩家在游戏中处于造物主或者领导者的地位，以俯视众生的姿态来看待他所创造的世界和世界中芸芸众生的活动。游戏的乐趣来自于领导者的领袖欲望和管理发展的成就感。游戏没有明确的目的，一般是开放式结局。

美国又有人把《黑与白》这样的游戏称为上帝模拟游戏（God Game），算作 SIM 的一个子类。

有人把飞行模拟游戏也算在 SIM 类里，当作它的一个子类。这样的分法比较尴尬。其实我们可以定义一个新名词——机能模拟类游戏。在《模拟人生》和《模拟城市》等游戏中体现的是对人类社会或其某个功能单位的模拟，玩家是以造物主或者管理者的身份出场。而机能模拟类游戏，无论是飞行模拟还是坦克模拟，都是模拟现实世界中的某种机器的机能。玩家只是一个飞行员或坦克驾驶员。

机能模拟类游戏的目的是真实地再现现实世界的机器原型——飞行模拟游戏必须严谨地在游戏世界中再造一架虚拟的飞机，而且从控制面板的细节到飞行性能都要和现实世界中的原型相似。游戏的目的是培养玩家掌握控制这种机器的技能。

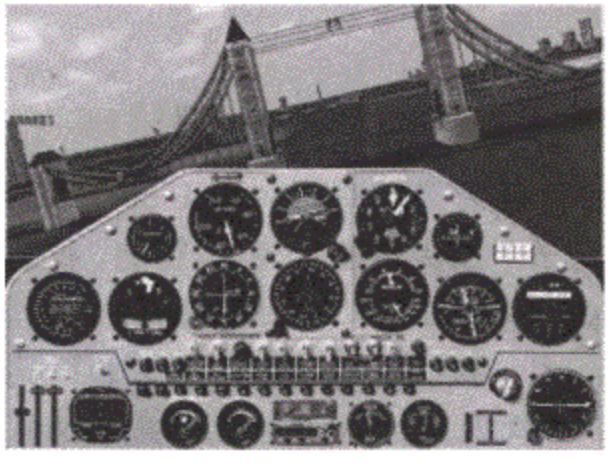


图 3-28《微软飞行模拟》（Microsoft FlightSimulator）

AVG（冒险类游戏）

AVG（Adventure Game）游戏以故事（story）、冒险（exploration）和解谜（puzzle-solving）为三大要素。玩家扮演一个角色，在充满了悬念的故事情节的指引下，一步步探索游戏中的未知世界，在探索过程中合理地使用道具，解开各种谜题，最终破解了整个故事的秘密。和同样注重故事情节的 RPG 比起来，AVG 更注重故事的流畅性和悬念。在画面的构成上更注重借用电影镜头和剪辑技术，以求达到刺激、恐怖、和悬念效果。

图 3-29 《MYST》是最著名的 AVG 游戏

AVG 节奏慢，故事吸引人，需要动脑子，和电影相似，是中高年龄段的玩家中比较流行的一个游戏类型。

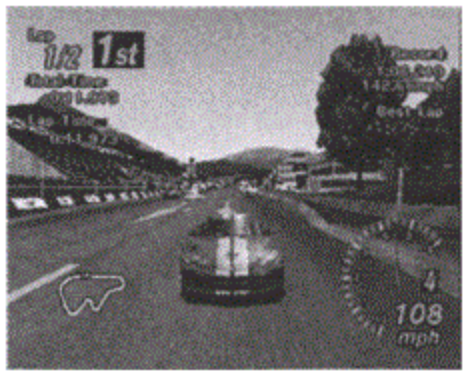


图 3-30《Gran Turismo》赛车系列

RAC（赛车游戏）

赛车游戏（Racing Game）是机能模拟类游戏的一种，但由于其十分流行，所以被单分出来了。

SPT（体育类游戏）

体育类游戏（Sports Game）可以涵盖三个层次：管理、战术、技能。技能方面，单纯的模拟某项运动，比如滑雪。战术，比如足球的团队配合和排兵布阵。管理，包括俱乐部的管理和球员的培训。

图 3-31 《FIFA》系列足球游戏

前面介绍的几种游戏类型，它们的定义并不是很严谨，它们相互之间也不是界限分明。特别是当文化的差异出现时，有可能一个游戏可以被同时划到两个类型里。比如宠物蛋（Tamagochi）这个游戏，可以算宠物游戏，也可以按美国人的分法算作 SIM 类的。另外上面列出的类型也并不能涵盖所有目前的游戏，有很多游戏属于两种或多种类型的交叉和融合。比如说《黑与白》是上帝模拟类游戏和宠物类游戏的融合。《古墓丽影》由于其中的探索和解谜要素，又有人称其为 AAVG（Action Adventure Game，动作冒险类）。而其他的例子还有《超时空英雄传说》等，是 RSLG，即 RPG 和 SLG 的融合，使用 RPG 的方式发

展故事，使用 SLG 的方式进行战斗。

总之，游戏类型的分类是一种习惯和约定俗成，不是 100%科学的，其实用性大于理论性。正如任何行业都有自己的行话一样，游戏类型及其各种古怪的缩写的主要作用是有利于业界人士之间和业界和玩家之间的沟通。

第四章　游戏性

在游戏业，没有比游戏性（gameplay）更重要、更混乱、更容易引起争议的名词了。人们普遍认为游戏性是游戏的最重要的属性，是决定游戏成功的关键。什么是游戏性？游戏性包含哪些要素?则众说纷纭，没有定论。这就出现了一个尴尬的局面——游戏性对游戏的成败这么重要，但大家又说不清楚游戏性是什么，也没有很好的方法来保证游戏性的成功实现。当然有人会争辩说游戏性就和艺术家的直觉一样，是只可意会而不能言传的。你去问一个著名画家怎么才能画出不朽的杰作来，或者什么才能够称得上是不朽的杰作，他肯定也说不清楚。的确，游戏性和游戏设计在很大程度上决定于直觉和经验。作为工业化协作制造出来的软件“产品”，游戏和游戏性又不能完全停留在直觉阶段。只有在明确了自身规律之后才能保证其产品质量，可持续地制造出高质量的产品，并保障开发投资获得回报。我们知道软件工程有一个著名的论断：对于软件或者软件过程的一个属性，如果不能明确定义的话，就不能有效测量；如果不能测量的话，就无法改进。因此，整个软件工程的思想基础就是为软件和软件过程定义一些属性，以及测量它们的方法，然后再通过系统化的工作去改进这些属性。同样地，对游戏性来说，如果不能确切定义它，就无法改进它。本章所要探讨的，就是如何定义游戏性，如何从多方面多角度了解它，从而为进一步采用更成熟和更正规化的方法去改进游戏性铺路。

可用性

在探讨游戏性的问题之前，我们先从另一个角度介绍一下可用性（usability）的概念。可用性和游戏性是一对孪生兄弟——可用性是商用软件的重要属性，而游戏性则是娱乐软件的重要属性。两者之间有很多

共性和千丝万缕的联系。了解了可用性的概念，有助于我们开拓视野，去更好地了解并定义游戏性。

从历史角度看可用性和游戏性

非常幸运的是，可用性的概念的定义和发展走了一条和游戏性完全不同的道路。从历史的角度来看，游戏性这个概念很早就被提出来了。从 20 世纪 80 年代中期开始 gameplay 这个词就被造了出来（这个词在英文字典里是没有的，是游戏业自己造的词）。然后所有的游戏设计师都接受了这个词，并成为业界的标准用语。从那时候起，所有的游戏设计师都同意游戏性是游戏最重要的属性。但是，大家都谈游戏性，却都是泛泛而谈和漫而谈之，没有采用系统的方法去严谨定义，没有理论上的深化，也没有具体的分析。也就是说游戏设计师们对游戏性的认识还是处于很粗浅很原始的阶段，还只是感性经验，没有理性的量化分析。而反观可用性，这个词在商用软件领域，特别是网站方面的流行，是在 90 年代中期，也就是

落后了游戏性 10 年左右时间。但可用性一开始就有强大的理论基础，提出可用性概念的不是业界，而是大学研究机构。可用性的理论基础是综合了计算机科学、设计、心理学和人机交互学（Human—Computer Interaction，缩写为 HCI）。在理论的基础上，实际工作者们在软件设计开发过程中，对可用性概念进行了更深的分析综合，甚至对某些方面采用了量化分析的方法，从而使得他们对可用性的认识超越了一般的感性经验。更重要的是，通过对可用性的分析，开发出一系列方法、流程、规则和标准来帮助软件设计开发人员去设计更可用更好用的软件。这点需要格外强调：因为分析可用性本身是没有意义的，比事后分析一个已知软件的可用性更重要的是基于分析去总结经验和规律，并以此事先指导软件的设计，设计出好用的软件。

所以从历史的角度看，游戏业在游戏性方面的工作虽然起步得很早，但其发展应用则完全落后于可用性。商用软件业在可用性方面的工作对游戏性可以有很大的借鉴意义。

定义—评估—设计

人机交互研究领域和商用软件业在可用性方面的工作基本上可以看成有三个步骤。首先是定义一个概念，用这个概念代表一个软件系统的某项属性。然后根据这个定义去设计一些评估方法，用来评估现有系统的这个属性。最后在评估的基础上，搞出一套设计方法、工具、流程，来帮助软件设计人员更好地进行设计工作。定义的目的是为了评估，评估的目的是为了更好地设计。如图 4-1 所示。

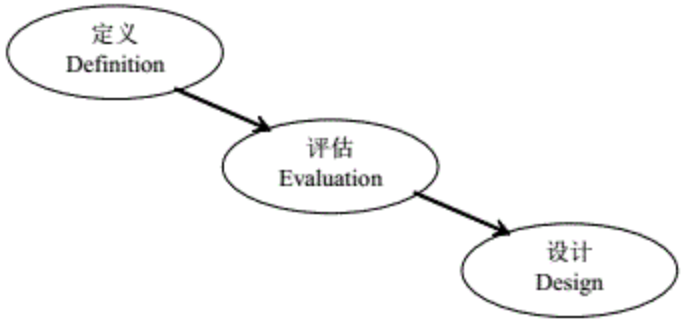


图 4-1 定义—评估—设计

可用性的定义

首先采看可用性的定义。初看上去，可用性和游戏性最大的共同点就是其模糊性和主观不确定性。什么是可用？什么是好用？一个软件，如微软的 WORD 字处理软件，可以帮助用户完成一定的工作任务。软件开发人员去采访用户的时候，问他们：“这个软件好用吗？”用户们会给出各种回答，“好用！所见即所得（WYSIWYG）的排版功能太棒了！”，“不好用！这个标题的字体怎么改不了？”，“太过分了！怎么才能去掉那个讨厌的曲别针？”这些情态各异有时甚至相互矛盾的回答是用户从个人的角度对一个软件的可用性做出的评估。这些评估是模糊的，因为用户很多时候并不能给出确切的答案。这些评估又是主观的，因为它们是用户根据自己的体验总结的。人和人之间计算机操作技能不同，年龄不同，眼力不同，必然导致对同一个软件做出不同的结论。

对软件开发人员来说，显然不能满足于从用户那里得到这么多乱七八糟自相矛盾的评

估。他们设计软件的目的，不是给一个两个用户使用，而是给成百上千人使用。这就带来一个难题：如果软件只是给一两个人用，那很好办，我们去和他们坐在一起，详细地了解他们的需要，他们的脾气秉性，他们的习惯和技能，然后根据这些信息去设计软件。这样设计出的软件，肯定符合他们的要求，对于这一两个用户来说肯定是“可用”的。但问题是，现代的商业软件是给成百上千人用的，他们之间的个体差异太大了，如何设计软件使得他们都满意呢？也就是说，如何设计软件使得软件对于成百上千人都是“可用”和“好用”的呢？如何综合成百上千人的意见，找出其共性，并基于共性去设计软件呢？

另一方面，可用性是一个很复杂的概念。当用户说：“怎么也搞不清楚如何设置文本？”时，他所遇到的问题和“上次用了数学公式，这次怎么也找不到那个选项了”是不一样的。

通过上面的分析，我们看到可用性是一个很复杂的有极大主观因素的概念。我们再来看看 HCI 研究工作者是如何解决它的定义问题的。

在西方思想库中威力最强大的武器之一就是分类法了。在古希腊时就发展出的动植物分类法是西方现代科学的基础。可以说没有分类法就不会发展出进化论。分类法的基本思想就是把纷繁复杂的世界分门别类，分而击之（divide and conquer）。

我们看看分类法是如何在可用性定义上发挥作用的。分类就是首先承认这个概念十分复杂，不能简单地定义或者简单地分析，必须把它分成各方面，从各角度来分析。当用户拿到一个新的软件时，他要打开包装，安装然后开始使用。由于是第一次接触这个软件，必须有一个学习的过程。可学习性（learnability）的概念就被分离出来了，作为可用性的第一个子概念。当学习期结束，用户完全掌握了软件的主要功能的使用后，他必须使用软件完成日常的事务性操作。在进行日常事务性操作的时候，由于用户已经对软件的主要功能烂熟于心，他基本上是靠条件反射，非常迅速地完成一系列操作。也就是说，理解和记忆让位于视觉匹配和肌肉反应。这样效率（efficiency）的概念就被提出来了。效率的概念是针对一个软件的主要功能的。除了那些用户每天都在使用的主要功能外，一个软件还有其他并不常用的辅助功能。这些辅助功能可能一个月被用一次（比如月度统计功能），也有可能一年才被用一次（比如被黑客攻击后的数据恢复功能）。这就带来了另一个问题，当这个月费九牛二虎之力把这个冷僻的功能学会后，过几个月再用时，还记得起来如何使用吗？可记忆性（memorability）的概念又被提出来了。

所以我们看到，通过分类法，我们已经把可用性分离成了三个子概念。每个子概念都比较清楚，相互之间界限比较明确。我们可以针对不同子概念，进行深入一步的工作了。

在完成了分类后，第二项工作就是把可用性的主观因素和客观因素分离，模糊的部分和确定的部分分离。这也是西方思想界的传统——科学有科学的领域，宗教有宗教的领域。科学解决不了的东西，属于上帝。理性与神性分开。这么做对科学的发展同样是非常重要的。

对可用性来说，就是要分清楚哪些部分是客观的，是共性的，是能够精确分析的；哪些部分是主观的，因人而异的，模糊的，无法使用精确的量化分析处理的。

我们发现前面介绍的三个子概念——可学习性、效率、可记忆性，都是共性大于个性（如果学习没有共性的话，学校也就不可能存在了），客观大于主观。也就是说，它们都属于客观因素，经过这样分析后的可用性的定义，是一个多维的概念既有主观因素，又有客观因素，同时含有多个子概念。如图 4-2 所示。

图 4-2 可用性的定义

可用性的评估

在可用性定义的基础上，我们可以开始进行评估工作。

首先看看客观部分。既然我们已经分离出一些子概念了，那么可以对各个子概念对症下药，设计出各种评估方法来测量之。表 4-1 列出了评估可学习性和效率的部分方法。这些方法是基于认知科学（Cognitive Science）和心理学研究的成果发展出来的。有的是定量分析方法，有的是定性分析方法。（由于本书并不是关于可用性和商用软件设计的，所以各种评估方法的具体细节就不做介绍了）

子概念	评估方法
可学习性	CW（Cognitive Walkthrough）方法（定性分析）
效率	Model Human Processor（定量分析） Fitt's Law（定量分析） GOMS 模型（定量分析）

表 4-1 子概念和与之对应的诸多评估方法

为什么我们可以针对客观部分的子概念设计出这些评估方法呢？因为客观部分的子概念共性大于特性，是确定的而非模糊的。而心理学和认知科学的研究就是基于人类共性的，所以其基本理论和思想可以被 HCI 研究工作者用来测量这些子概念。

对于主观的部分，由于其模糊性，其特性大于共性，无法采用前述各种评估方法，必须采用实证和考察来评估。做个比喻：比如说一部电影，在使用了包括编剧、镜头、剪辑等通用的电影技术后，最后观众是否喜欢，还是得等他们到电影院里坐下来看一遍后才能确定。有可能出现这种情况：一部电影，从技术角度看天衣无缝，完全符合电影学院里所教授的理论和法则，但却不卖座。对于电影来说，由于不能事先放映并从观众处获得反馈并修改，所以其票房成败很有赌博的性质。软件设计人员们就想到是否可以在软件开发过程中，不断地请“观众”来事先观摩并提出意见。这样的话，最后推向市场的软件产品的可用性就可以得

到一定的保证。

基于这种思想，HCI 研究人员提出了各种用户测试方法（user test）。用户测试方法就是针对可用性的模糊性的主观性的一面而设计的。使用这些方法，先对用户群采样，然后实地观察用户在使用软件时的行为，并以观测结果为根据来评估可用性。有代表性的用户测试方法，如 Think-Aloud Protocol 和 Contextual Inquiry。

通过上面的介绍，我们看到对可用性的客观部分和主观部分，我们都有不同的评估方法来处理了。对客观部分，用偏理论的，偏量化的方法来处理。对主观部分，用实证和观测来处理。

基于用户的设计

前面说过，测试的目的是为了指导设计。我们现在有了许多评估方法，它们如何能够帮助我们进行设计呢？显然，评估方法是不能直接指导设计的。评估方法是在一个设计方案提出后，用来对这个已经成形的设计进行评价的。也就是说先有设计，后有评估。于是人们很自然地想到，可以先进行快速而简要的设计，然后评估，然后根据评估的结果去修改并丰富设计，然后再评估，如此循环往复。这就是快速原型法的基本思想。因此，评估方法对设计的显式指导作用，主要体现在搞出一个大的架构或者说模型，这种架构就叫做设计流程（design process）。设计流程把设计和评估的各种方法有机地集成起来，成为软件设计时遵循的指导性纲领和策略。目前 HCI 领域经常提到的基于用户的设计（User-Centered Design，缩写为 UCD），就是这种类型的架构。图 4-3 展示了各种方法是如何被组合到一个设计流程中的。

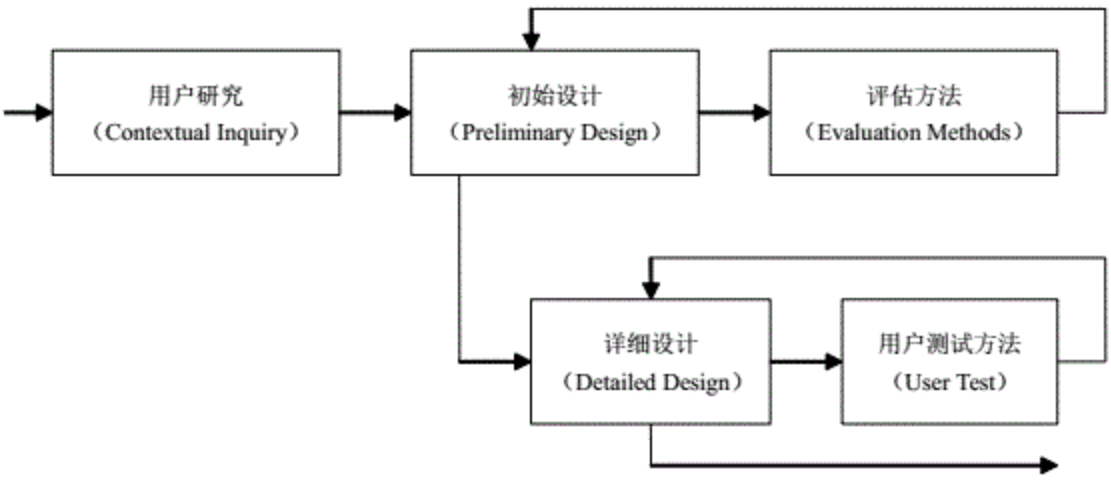


图 4-3 一个基于用户设计（UCD）的设计流程

通过前面的介绍，我们看到可用性这个概念是如何被定义的，又是如何被评估的，如何在设计中发挥作用的。我们介绍的各种评估方法和用户测试方法，以及设计流程，在目前的

商业软件开发领域得到了广泛的应用。尤其是在中大规模的公司中，都建立了相应的部门负责可用性测试和设计。其效果已经得到了体现——现在的商用软件的可用性越来越得到重视，越来越得到提高。

游戏性

在对可用性的定义、评估、设计有了一定了解之后，再来考察游戏性。

对游戏性的思考

迄今为止，对游戏性还没有一个很好的定义。很重要的原因，就是大家只是看到了游戏性的某些方面，而没有以一种多维的多层次的观点去看待它。换言之，没有用分类法把这个复杂的概念系统地解剖开来，一一分析之。下面就举出几个观点，作为定义游戏性的准备工作。

●游戏性的基础是可用性

前面我们介绍的可用性，是针对商用软件说的。实际上可用性的概念应该扩展到所有类型的软件。对于游戏软件来说，如果其不可用，就不能发挥娱乐的功能。比如说，如果游戏软件没有很好的可学习性，玩家就不能很快地上手，大部分玩家就会知难而退，那么游戏性就根本谈不上了。因此，游戏性的基础应该是可用性。

●游戏性不仅仅是操作性

早期的游戏，如任天堂 8 位、16 位机上的 ACT 游戏。它们的游戏性很大程度上就是操作性——手眼协调、条件反射、难度递阶。但随着游戏的发展，游戏类型的增多，游戏性就远远不仅仅是操作性了，策略和 AI 等要素对游戏性也越来越重要了。操作性只是游戏性一个层面上的内容。

●不同的游戏类型，其游戏性有共性

游戏之所以为游戏，必然有独特的特征。各种类型的游戏，必然有其共性。比如说，置入感（immersion）就是所有游戏的共性。游戏最重要的就是使得玩家可以感到自己置身于虚拟的游戏世界，暂时忘记自己所处的现实世界。这种置入感可以采用多种手段获得，比如 FPS 中使用三维迷宫和第一视角，使得玩家和游戏主人公融为一体。而 RTS 使用俯视视角，使得玩家有掌控全局的感觉。关于游戏性的共性，本章后面附录的一篇文章中有详尽的解释。

●不同的游戏类型，有不同的游戏性要素

不同类型的游戏，其游戏性有不同的要素，或者说侧重不同的要素。当我们研究 RTS 的游戏性时，游戏性更侧重于策略和微观管理。而对 FPS 游戏，游戏性则侧重于任务、迷宫、战术协调。

对于同样的一个概念，不同类型的游戏的侧重也有可能不同。比如说可用性中的效率概念，对 FPS 和 RTS 来说就不一样。对于 RTS 来说，如微软的《帝国时代》系列，使用一个鼠标就可以完成所有操作。因此其效率主要体现在鼠标的移动和鼠标按键的交替进行上。对于这种情况，可以用费茨定理（Fitt's Law）来评估其效率。而 FPS，其操作是由键盘控制移动，鼠标控制瞄准和射击。对于这种情况，可以用 GOMS 模型来评估其效率（关于费茨定理和 GOMS 模型的详细内容，请阅读书后所列的参考书目）。

游戏性的多维模型

既然游戏性中有共性也有特性，不同的游戏类型又有不同的侧重，那么游戏性的定义就应该体现这种多层次的多样统一。在以上分析的基础上，笔者提出游戏性的多维模型，如图 4-4 所示。

图 4-4 游戏性的多维模型

这个模型可以比较完整地表达游戏性的复杂性和多样性。它是一个多维的模型，分为四层，每层中又包含不同的模块。底层是可用性层，因为可用性是游戏性的基础。可用性层中包含的模块就是我们前面介绍的可用性中客观部分的各个子概念，如可学习性和可记忆性等。第二层是游戏层，包含了游戏性中的共性，也就是说不管什么类型都拥有的共性，如置入感等。第三层是类型层，也就是各个游戏类型所特有的具体的游戏性要素。对 RPG 来说，可以是魔法和战斗系统参数设置。对 RTS 来说，可以是武器系统平衡性和微观管理等。在类型层的上面，是情感层。它涵盖了游戏性中所有主观性的、模糊性的因素。

这个模型由底层到顶层，体现的是由共性到特性（可用性层代表所有软件的属性，游戏层代表游戏所特有的属性，类型层则是某种游戏类型的属性），理性到模糊，由底向上逐渐细化，由抽象到具体。

这个多维模型又是像积木一样可以自由组合的。各层中的模块可以根据需要进行取舍，对不同类型的游戏，肯定有不同的组合方法。游戏设计师们所要做的，是根据自己对广义的娱乐的理解（包括心理学的基本知识和对其他娱乐形式的了解），和对自己所专攻的某一游戏领域的知识（domain knowledge），去确定自己所要设计的游戏的游戏性模型。这个多维模型可以帮助游戏设计师更好地理解游戏性的复杂多样性，从而抓住自己所要设计的游戏的游戏性中最重要的因素，同时又不忽略其他可能有影响的因素。当然这个模型并没有提供具体的针对某个类型游戏的游戏性的定义，这项工作应该是由这种游戏类型的专家来完成的。

用多维模型对游戏性进行定义的主要目的，就在于在开始设计游戏时确定设计目标和设计策略。我们知道任何设计行为，比如工业设计和人机界面设计，都要首先确定设计的目标

（objectives）和策略（strategy）。目标是设计所要达到的最终效果（可以包括很多条目标），策略是为了达到目标所应采取的过程和方法。如果在开始设计时没有一个明确的目标，最后设计出来的产品必然不能达到用户的期待。有了设计目标，如果没有很好的设计策略去指导设计步骤，设计出的产品也不能保证质量。对游戏来说，显然其设计目标是使游戏好玩，而使游戏好玩这个目标过于笼统和空泛，并不能成为设计目标。多维模型可以帮助我们确定游戏设计的目标，并进一步确定一定的设计策略。

游戏性的评估

我们现在有了一个游戏性的多维模型。考察这个模型，我们可以看到，表面上难以捉摸的游戏性，还是有很多部分是可以把握的，是可以仔细研究并精确评估的。比如说可用性层的内容，既是游戏性的基础，又是最容易评估的。提出这个模型的基本思想之一就是先分离出来可以理性评估的东西，让游戏设计师们对这些东西有足够的重视。游戏性和游戏设计绝对需要直觉和天才的梦想，但基础的理性的东西同样重要，必须先把最基础的东西做好，然后再在理性的基础上发挥感性的直觉和梦想，才能更好地完成游戏设计的任务。对照这个模型来说，就是自底向上，一层层地考察和评估。

对于类型层的内容，同样可以评估。因为类型层中的内容是非常具体的针对某一类型的，所以可以使用 Heuristic Evaluation 的方法，去总结其规律，制定标准，然后对照标准去评估新的设计（关于 Heuristic Evaluation 的详细情况，请见书后所列参考书目和论文）。

对情感层中主观性的因素，使用用户测试的方法来评估比较合适。游戏业也有一个自造的名词，叫“playtest”，即游戏性的测试。但 playtest 主要是在游戏的开发已近完成后进行的，其目的是微调各种参数，对游戏设计的影响不大。而用户测试方法的要旨是尽量早地在设计过程中邀请玩家来对设计进行评估，并用反馈指导设计。因此，游戏设计师们应该从 HCI 的用户测试方法中汲取有用的东西，改进他们现在所使用的 playtest 方法。

上面介绍了一个游戏性的模型，和对其各层面所可以使用的评估方法和测试方法。我们说过，比定义和评估更重要的是设计，游戏性的设计就是游戏设计。我们同样需要一个好的设计流程作为框架结构，然后把游戏性的各层次和各要素的考量集成到这个设计流程里，把各种评估方法和用户测试方法集成到这个设计流程里。这样才能从根本上保证游戏的质量。在以后的关于游戏设计的章节中，我们将会看到这种思想的具体操作和实现。

附录：娱乐的 14 个要素

对于游戏性的共性，Pierre-Alexandre Garneau 在一篇文章中列举以下 14 种娱乐要素：

- 美（beauty）
- 置入感（immersion）

●解决问题（intellectual problem solving）
●竞争（competition）
●社交（social interaction）
●喜剧（comedy）
●惊险刺激（thrill of danger）
●体育运动（physical activity）
●爱（love）
●创造（creation）
●权力（power）
●探索和发现（discovery）
●进展和完成（advancement and completion）
●使用技能（application of an ability）

这 14 个要素基本上把游戏能够带给玩家什么乐趣总结得很清楚了。下面简单介绍一下其内容。

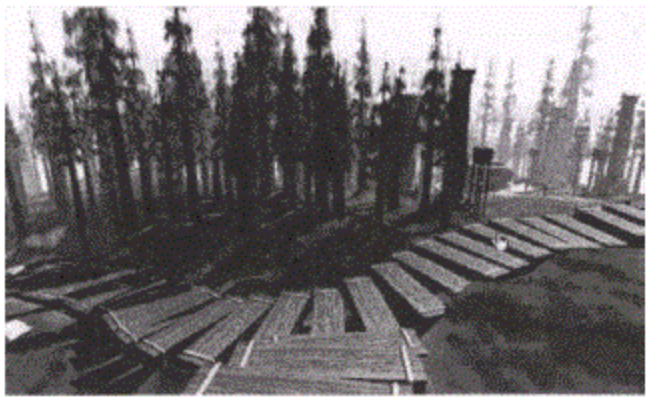


图 4-5　作为最有名的 PC 游戏之一，《MYST》以其美仑美奂的静态图像吸引了无数的玩家。其中一个玩家这样写到："我在无数的繁星闪闪的夜晚的窗前打开计算机，启动了《MYST》。我的眼前是一个如梦如幻般美丽的小岛。这个小岛有如仙乡幻境，使我流连忘返。这时候，我对自己说，'《MYST》是游戏之王！"美是《MYST》成功的最重要因素，其三维静态画面为当时最高水平

●美

人类在欣赏自然的或者人工的美的东西的时候会感到愉悦。对美的追求是人类的共性。

●置入感

当一个小孩子躺在阳光下的草地上，手里拿着一个玩具飞机，边舞动边嘴里发出嘟嘟嘟的声音时，他已经进入了自己所制造的幻想世界——幻想着自己开着飞机在天空中翱翔。这种人类暂时离开现实世界，进入虚拟世界体验到快乐，就是置入感。置入感一半源于人类的易于幻想和遐思，另一半要借助于人工的虚拟环境。比如电影院就是一个很好的例子。在放映电影的时候，灯光全灭，观众沉入黑暗之中，只有屏幕上的人和事，以极大的超乎自然的亮度展现在观众面前。似乎茫茫宇宙，只有观众和屏幕两个存在，这样观众和屏幕的距离拉近了，更有助于观众体验甚至融入电影故事情节。对游戏来说，更有多种方

图 4-6 使用虚拟现实技术可以达成最强的置入感。它将成为未来游戏的主要交互手段

法来达成置入感。而更登峰造极的方法，要数虚拟现实（Virtual Reality）系统了。通过使用全封闭式眼镜和感应手套，使得玩家可以真正投入虚拟世界中，忘乎所以。

●解决问题

人类从解决问题中获得乐趣。解决问题包括对问题的诊断、动脑筋提出解决方案、并利用技能执行这一方案。它体现了脑力和体力的结合。在美国家庭里，男主人以拥有各种各样门类齐全的电工、木工工具为自豪——更让他们自豪的是在周末去修修补补，通过自己的努力，去排除汽车的故障，修补房子的渗漏，或者造出新的玩意儿。在游戏中，比如说 AVG 游戏的解谜。而最雄辩的例子就是俄罗斯方块（TETRIS）了。

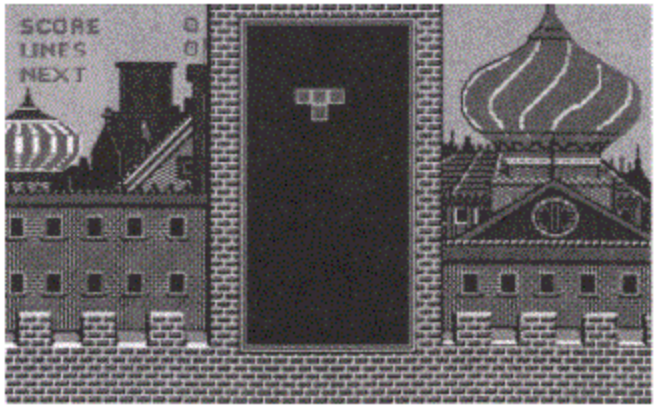


图 4-7　TETRIS 长盛不衰，是因为其解决问题的机制简单而又充满挑战性

图 4-8　FPS 游戏的网上对战功能的盛行，体现了竞争在游戏中的重要性

●竞争

人类通过竞争来体现自己比别人更强，证明自己的价值。各种体育比赛的目的就是要争第一，争出谁是最强。游戏中的街机格斗游戏是最明显的例子。玩家通过无数次的历练，力求比别人更强。FPS 游戏的网上对战功能，更把人类这种基本要求发挥得淋漓尽致。

●社交

社交是人类基本需求。ICQ 和网上聊天室的盛行就说明了这一点。在单机游戏的时代，社交对游戏的影响不太明显。但网上游戏出现后，社交成为网上游戏成功的重要因素之一。

●喜剧

喜剧效果在游戏中的应用还不太普遍。在所有娱乐形式中却是大行其道，尤其是电影。即使很严肃的题材，也可以黑色幽默一把。像 David Lynch 的《Blue Velvet》这么严酷的电影中，都有一段充满了喜剧性的片段。可以说，对严肃题材的喜剧化处理，目前在游戏业还是一项空白。

●惊险刺激

从危险中获得的刺激和快感。这点在美国流行文化中极为重要。从千旋百转的过山车，到高耸入云的自由落体机，到蹦极，美国人是什么悬玩什么，并从中获得极大的刺激和快感，值得指出的是：这种惊险刺激是在当事人知道没有危险的情况下获得的——玩蹦极时大家知道不会出事，否则谁敢去玩。游戏也一样，玩家在玩 FPS 的时候，知道自己不会真被子弹射中，但当他面对敌人的时候还是同样地紧张而激动。

●体育运动

在现实生活中，成千上万的人们怀着极大的热情投入各种体育健身运动中，在付出汗水的同时也获得了欢畅。体育运动和游戏的结合还不普遍。曾经风靡一时的跳舞机（DDR）和跳舞毯显示了把体育运动和游戏融合的巨大能量。

●爱

爱情和性爱是其他娱乐形式永恒的主题之一。但在游戏中还不很普遍，只有日本的恋爱模拟游戏尝试了这一主题，此外 RPG 游戏中一般都有爱情戏。对其他类型的游戏，爱情现在处于可有可无的地位。

●创造

上帝从乌有中创造了人类，而人类也对创造乐而不疲。从沙滩上的城堡，到小孩子的积木和 LEGO 玩具，都是从乌有中，使用一定的原材料，创造出全新的事物，并感到成就感和愉悦。

图 4-9　偏重于创造和建设的《Simcity》系列和偏重于破坏和杀戮的 FPS 类型都很流行——这个世界上就是有些人去建设，有些人去破坏

●权力

对权力的追求是驱动人类社会生活的重要因素。公司职员都希望被提升——拥有更大的权力。权力体现在对资源的支配权利。资源包括金钱、人力、政策等。权力对游戏来说是最重要的一个要素，对策略游戏尤甚。在 RTS 游戏中，玩家拥有无上的权力，决定己方所有单元去做什么，从而可以满足人们的领袖欲望。

●探索和发现

对未知世界的探索和发现是人类与生俱来的欲望。从几百年前的海上环球探险，到 19 世纪的西部淘金，到 20 世纪的阿波罗登月，都是这种欲望的体现。在 ACT 和 AVG 游戏中，探索和发现是最重要的属性。

●进展和完成

人类从循序渐近接近目标的过程中得到鼓舞，以达成目标为最终乐趣。对游戏来说，也

必然有这么一个过程——递阶前进，最终达成目标是很多游戏的游戏性的基础。

●使用技能

人类对于某项技能的迷恋，通过不断的锻炼，使得这项技能更趋完美的境界，并从娴熟地使用这项技能中获得某种满足和成就感。比如街机格斗游戏的高手们，为了技能的提高衣带渐宽终不悔。

设计篇

第五章　电影叙事结构与游戏

随着新一代的游戏主机 PS2 XBOX 和 Game Cube 的登台，电子游戏已经不是 10 年前的形态了。技术逐渐进步，图像声音等各方面功能不断增强。大家可以明显地感觉到游戏正在与电影相互融合。目前游戏业的主流趋势，是从电影中借鉴吸收各种有用的要素，使得游戏更丰富更复杂，并向综合娱乐的方面发展。

Square 就是最好的例证。自进入 PS 时代以来，且不论 Square 的游戏制作水平是提高了还是降低了，　它始终站在游戏界的最前端向电影化的方向努力着。Square 游戏的电影化，最显著的就是大家看得到的穿插游戏之中的电影片断和在游戏过程中的镜头运用。本章所要介绍的，不是这些表面的现象，而是电影的故事内容，亦即电影叙事理论对游戏的影响。其实电影从电子游戏出生之日起就潜移默化地影响着游戏，只不过到了现在，这种影响越来越显著，也越来越重要。因此，现在的游戏设计师或编剧绝对是必须了解这些电影叙事理论的。

在西方电影界影响深远的“英雄之旅理论”

Joseph Campbell 的《The Hero with a Thousand Faces》（中文译名：千面英雄）是对 20 世纪西方电影发展影响很大的一本书。书中所提出的一整套理论集传统的叙事方法（storytelling）之大成。通过这本书，很多作家都开始认识到叙事理论的重要性，并且在写作实践中有意识地应用这些理论。特别是在好莱坞电影界，无论是编剧还是导演都把 Campbell 书中的理论作为处理叙事的一个很好的指导性工具。从很多著名电影中都可以明显地看到他的理论的痕迹。

Campbell 书中提出的理论被称为“英雄之旅理论”（Hero's Journey）。他实际上是将自古以来的叙事手法进行了总结，进行了命名、组织、规范化、结构化，从而形成了系统化的理论。而且这个理论是不受地域性限制的，它可以用来解释流传在各个地方各个民族的神话故事、民间笑话、甚至正统文学作品。

先简单说一下英雄之旅理论吧。

英雄之旅的 12 个组成部分

尽管存在无数种可能性，英雄的故事总是一段旅程。　一个英雄离开他舒适平常的家园，进入一个充满冒险的不寻常世界，然后……

英雄之旅的 12 个组成部分：

1．普通的世界（Ordinary World）
2，冒险的召唤（Call to Adventure）
3．对冒险的拒绝，或者说是抵触（Refusal of the Call）
4．与智者的相遇（Meeting with the Mentor）
5．穿越第一个极限（Crossing the First Threshold）
6．测试、盟友、敌人（Tests，Allies，Enemies）
7．接近深层的洞穴（Approach to the Inmost Cave）
8．严峻的考验（Ordeal）
9．得到嘉奖（Reward）
10．回去的路（The Road Back）
11．复活（Resurrection）
12．满载而归（Return with the Elixir）

可能有些一头雾水，下面来详细说说每一部分。

●普通的世界（Ordinary World）

大多数的故事，都会把我们的英雄（Hero）拽出正常普通的世界，扔进特殊的不正常的充满异类的世界。

注意：在这里虽然用了英雄、普通世界和不正常世界这样比较抽象的词。希望大家不要把他们实体化！因为在以后的介绍里，如果把所有的词都实体化，就会把英雄之旅理论具体化为狭窄得多的神话理论了。

举个例子先说清楚吧!比如说英雄开始生活在普通世界，然后因为某个原因进入非常世界开始英雄之旅。可能大家马上就会联想起许多日本动画的情节吧？现实中平凡高中生受到异世界使命的召唤，由命中注定的卫士带到异世界开展冒险之旅。哈哈!而实际上呢，英雄不见得是真正意义上的英雄。异世界也并不见得是充满魔怪的。一个现实中的平凡高中生，一天在街上遇到了喜欢的美眉，开始了坎坷的追求之旅。这个高中生也是英雄啦!遇到美眉之前的生活是普通世界，遇到了喜欢的美眉是原因事件，之后充满了坎坷追求的生活便是新的非常世界啦。明白了？

好的！回到普通的世界。在这里，为什么要先展示普通世界呢？就是要体现对比——普通世界和非常世界的对比！在英雄进入非常世界之前，你必须要展示英雄在普通世界的生活。这样在进入非常世界的时候，才可以通过生动的对比体现出非常世界的不同。对比越强烈，效果一般来说越好。比如说电影《星球大战》中的天行者（Skywalker）在卷入命运之前是个农场男孩。普通世界是平和的农场生活，而非常世界则是广阔激荡的宇宙，在这里英雄开始了真正的冒险生活。

●冒险的召唤（Call to Adventure）

英雄是如何从普通世界进入非常世界开始他的冒险之旅呢?需要有一个特殊的事件，来作为英雄命运的转折点。这就是冒险的召唤（Call to Adventure）！这个部分要体现的，是英雄要面对问题、挑战和冒险。他已经无法再继续普通世界的平淡舒适的生活了，将面对一个明确目标。

在侦探小说中，侦探接受了一个新的案件；在戏剧里，王子的父亲被杀；在动画片里面，一次命运的邂逅和从 99.5 层台阶上飘来的红色草帽；在游戏中，一次祭祀前，被命运选出来的孩子……这些都是所谓冒险的召唤，都是对英雄的挑战！

●对冒险的拒绝，或者说抵触（Refusal of the Call）

这一节是关于畏惧心理的。英雄总是会在进入非常世界之前犹豫不决！一定要展示这个步骤的。英雄面对未知的恐惧，他还没有决定是否真的要开始这段冒险，是否要真的进入非常世界，他在思考是不是要回去。在这个时候，将会有很多外界因素，比如环境的变化、普通世界的需要、智者的鼓励等等，来帮助他度过这种恐惧。

在爱情题材里面，因为上一段恋情的阴影，考虑是否这次还要投入；在幻想题材里面，命中注定的孩子不想放弃现在的家园和小恋人，不想去远方冒险…

●与智者的相遇（Meeting with the Mentor）

当英雄因为畏惧心理对是否要进入非常世界开展冒险犹豫不决的时候，大多数情况下智者（Mentor）就要登场了。智者和英雄这种搭配是最常见于各种神话故事的。神话故事中智者一般是聪明的老人。他和英雄的关系嘛，比较常见的是父母和孩子，老师和学生，医生和病人，神和普通人……

智者的作用是为英雄进入冒险做准备，给英雄建议、知识或者装备等等。但是注意，智者所作的只是辅助英雄。英雄之旅始终要英雄一个人面对！所以智者不可以陪英雄很远很久。在一些情况下，可能智者需要给犹豫不决的英雄一脚，把他踹进非常世界，强迫他开始英雄之旅！哈哈！

一个家喻户晓的例子，就是《星球大战》里面的尖耳朵异形老师啦。他教会了天行者很多冒险所需要的东西。建议大家就用《星球大战》来理解英雄之旅理论，实在是贴切得很呢。还有《绿野仙踪》（The Wizard of OZ）也是经典范例呢！

●穿越第一个极限（Crossing the First Threshold）

现在英雄终于做出了决定，进入非常世界开始英雄之旅，决心面对一系列未知的困难挑战等等。从这里开始，这个故事的重点才真正的开始。英雄之旅开始了，英雄就再也无法回头了。

一般我们可以把整个电影看成有三个阶段：英雄决定去做什么，他是如何做的，然后是做的结果。如果按这个方法划分的话，穿越第一个极限就是第一和第二阶段的过渡部分，英雄之旅前 5 个组成部分都是描述英雄决定去做；而之后的部分，则是英雄如何去做。

●测试、盟友、敌人（Tests, Allies, Enemies）

一旦进入了非常世界，英雄自然而然的要面对未知的挑战和测验，认识新的朋友和敌人，要开始学会遵守非常世界的法则。

说到这里，需要指出的是：在西方的很多电影中，各种沙龙、聚会和酒吧是完成这些挑战和测试的常用场所（流汗……）。在这些地方，英雄很自然的遇到各种人，包括朋友和敌人，并受到各种类型的测试，得到有用的消息，学会非常世界的法则，等等。

当然，题材不同，场所是各有不同的。在《绿野仙踪》里面，这些挑战、测试和结盟发生在黄色的砖路上。主角 Dorothy 美眉遇到了很多困难，从而认识了稻草人、铁皮人还有狮子，他们成为了朋友。在森林里遇到女巫的手下，会说话的树，他们是敌人。她成功地通过了测试，包括把稻草人从钉子上解下来，给铁皮入注油和帮助狮子对付它的恐惧。

●接近深层的洞穴（Approach to the Inmost Cave）

最后，英雄开始接近最最危险的地带了——某个有龙深藏的地底洞穴，或者是暴力集团控制的岛屿，或者是梦中情人的另一个追求者等等。当英雄进入最后的充满恐惧的地方的时候，他需要穿越第二个极限（Crossing the second threshold）。这里有个有趣的法则，叫做接近的法则：英雄应该会在进入之前稍微停歇，准备装备，计划行程，用智慧骗过守卫。

在西方神话里面，这部分就是有恶龙守卫在迷宫的入口。英雄准备好剑和迷宫的大致地图，然后智取比英雄强大的恶龙，进入了死亡的迷宫。在《绿野仙踪》中，Dorothy 被坏女巫绑架进她的城堡。稻草人、铁皮人和狮子骗过众多守卫，混进城堡里去解救 Dorothy。

●严峻的考验（Ordeal）

在这里，英雄进入最底层的迷宫，面对最强烈的恐惧和可能的死亡，与敌对力量战斗！对于观众来说这是个阴暗的时刻：观众会焦虑不安，不知道英雄是否会幸存还是死亡!这是一个危急的关头，在这里英雄一定要死或者显得要死了，从而后面给他机会让他可以再复活。

在《绿野仙踪》中，Dorothy 和她的朋友们陷入邪恶女巫的陷阱。这时候看上去他们已经无路可逃了。这种绝望的时刻，就是典型的 Ordeal。另外比如在爱情题材里面，严峻的考验就是英雄与恋人关系出现问题和波折。这里有句经典的英文句子来形容如何描述爱情故事：Boy meets girl, boy loses girls, boy gets girl（男生遇到美眉，然后失去美眉，最后

又重新得到美眉）。几乎所有的爱情片都是这个模式！

●得到嘉奖（Reward）

好了！从死亡中复活，击败了恶龙。现在英雄和观众可以开始庆祝了。英雄终于可以得到他一直寻找的财宝了。这是对他的嘉奖。可能是特殊的武器或者是魔法剑，或者是可以治愈伤痛的良药。在一些情况下，嘉奖可能会是知识或经验，对于敌对势力的更细致了解等等。在这里，英雄可能会与一些人和解，比如父母、恋人等。很多故事里，恋人对于英雄来说是重要的嘉奖。在这个阶段经常会安排有爱情戏。经过严酷的考验之后，对于英雄的嘉奖也许还可能是他获得了更多的魅力吸引力。而且因为在冒险中的表现，他获得了真正的英雄的称号。

在《绿野仙踪》里面，Dorothy 成功地从邪恶女巫的城堡里逃出，并且得到了女巫的魔法扫帚和红宝石鞋，这是她回家的必须宝物，也是她得到的嘉奖。

●回去的路（The Road Back）

请注意：英雄只是得到了嘉奖，他(她)现在还没有走出非常世界呢！英雄要开始面对反抗非常世界黑暗力量的最终后果了。很多最精彩的追逐场景就是在这里展开的。在回到普通世界的路上，英雄被疯狂的一心报复英雄的敌对力量所追逐。一般是因为被夺走的财富，也即英雄所得到的嘉奖。

在这个阶段，英雄决定了要回到普通的世界中了。虽然还有很多的危险、考验、以及诱惑在面前，但是他认识到最后是离开非常世界的时候了。

●复活（Resurrection）

在远古时代，猎人或者战士在回到他们的部落之前，要先洗干净他们的手，因为他们的手上沾有血迹。从非常世界回来的英雄也一样。在真正地回到普通世界之前，已经经历过死亡或者接近死亡的他还需要在最后一次考验中经历真正的死亡，然后再次复活。所以说，不愧是英雄呀！怎么折腾也没关系！一般来说，这已经是英雄所经历的第二次复活了，死亡和黑暗再次被打倒。可以说这是对英雄的最后考验了，就像结业考试，哈哈。通过了结业考试，英雄获得了重生，终于可以回到原来的普通世界了。这时候英雄已经脱胎换骨，已经不是原来什么都不会的英雄了。

在《星球大战》里面，复活阶段总是使用同样的场景。在最后的战斗中，天行者总是显得要被打败，要被杀掉了。然后却又奇迹般地复活，获得新的经验。

●满载而归（Return with the Elixir）

终于英雄回到了普通世界。但是如果英雄没有从非常世界带回特殊的东西，他的英雄之旅就变得没有意义了。在这里称这些东西为 Elixir（万能药）。万能药可以是用于治疗

干涸大地的魔法药，或者是非常有用的知识和经验，或者是赢得了一场艰苦的比赛，或者是得到了忠贞不渝的爱情，或者是自由，等等。

在《绿野仙踪》里面，Dorothy 终于回到了在堪萨斯的家。她带回来的是爱和“没有一个地方像家一样好”的感觉。这就是她所得到的 Elixir。

好了，解释完了英雄之旅的 12 个阶段。Campbell 的英雄之旅理论是一个骨架的工具。他很好地适用于各种形式的剧本，不论是神话还是现实，古代还是现在，爱情剧还是动作片。我们可以以这个骨架为基础，然后用细节和独立的小段情节来丰富之。在描述故事的时候，不应该拘泥于这个理论。所有的这些阶段，是可以被删除、修改、省略和浓缩的。我们也可加入新的阶段，或者重新排列它们，这样才可以得到更多的变化和惊喜。

英雄之旅中的人物原型

在整个叙事过程中充斥的人物们，也是有章可循的。把他们分门别类，我们就会发现有着那么多的相似。这些相似点总结抽象起来就是所谓的人物原型(Archetypes)。下面就简单地看看在英雄之旅中会遇到哪些基本的人物原型。

在神话世界，你会注意到各种各样的角色类型和他们之间的联系——冒险的英雄们，在冒险之门冲着路人们召唤的传令官，提供知识和装备的聪明老人，凶猛的迷宫守卫者，总在英雄附近的麻烦制造者，时刻准备毁灭什么的阴险恶人，在路边的狡猾骗子，等等。在描述这些典型的人物类型的时候，Carl G. Jung 引用了 Archetypes（原型）这个词。原型这个概念对于理解在故事中各种角色的目的和功能是一个不可缺少的工具。如果你可以真正地掌握各种原型的功能作用，它绝对会帮助你来更好地确定每个角色的设定和份量。

在使用原型概念的时候要注意：通过原型来确定故事中角色的类型和功能，应该是灵活的角色功能，而不是死板的套用原型的某一类。可以把各种原型当作是不同的面具。这些面具被各种角色使用，当这个角色需要体现什么功能的时候就换上某种面具。这样塑造出来的角色才不是死板的，才是真正复杂的现实的角色。

下面是最基本最常用的几种人物原型：

1. 英雄（Hero）
2. 智者（Mentor）
3. 守卫者（Threshold Guardian）
4. 传令官（Herald）
5. 善变的人（Shapeshifter）
6. 阴暗面（Shadow）

7．狡诈者（Trickster）

当然还有很多很多可以当作原型的角色：童话故事中的狼、猎人、好妈妈、坏继母、善良仙女、邪恶女巫、英俊王子、美丽公主、贪吃的旅店老板等等等等。不过上面这 7 种是被总结出来的比较具有代表性的分类。

思考一下以下两个问题对于如何确定一个角色的原型有很大的帮助：
这个角色要展示哪些心理上的功能或者说哪一种的人物个性？
这个人物在整个故事里面要起到什么样的戏剧性的功能？

下面就这 7 种人物原型详细地说说，你在看的同时要想想上面的两个问题哦。

●英雄（Hero）

英雄这个词来自希腊语，意思是“去保护，去服务”。英雄是一个肯为其他人牺牲的人物角色。英雄这种人物原型所展示的是自我，是人性中独立的与他人不同的部分。在每个故事中，英雄所要做的是能够超越自我。英雄之旅也可以被看作是英雄在心理上的寻找自我超越自我的旅程。

对于戏剧结构来说，英雄这个人物原型的作用是给了观众了解故事的窗口。观众通过英雄的眼睛来看这个世界，体会英雄的感受，并融入故事当中。他需要一些令人羡慕的能力，这样观众才会想代替“他”，在故事中成为“他”。尽管有着很多常人所不及的优秀能力，英雄毕竟还是人。所以不是只有一个特点的典型，而是综合了很多优点缺点。这些优缺点有时候甚至是矛盾的。不过越多矛盾冲突越好，才会构造出真正的人物。英雄是各种复杂人类特性的综合体。英雄需要学习和成长，这是他的另一个重要的戏剧性功能。在有些剧本里，有时候很难说哪个角色是真正的主角。最好的回答可能是：那个在故事中学到最多，成长最快的家伙就是主角。英雄在超越障碍和完成目标的同时，也学到了新的知识，许多故事的主题就是学习——从智者那里，从恋人那里，甚至从敌人那里学习。在整个故事里面还需要展示英雄做了什么，即英雄的动作。一般来说，英雄是整个故事里面最辛苦的人啦，所做的最多。而且不光辛苦，英雄还要做出牺牲。观众总是认为英雄是勇敢强壮的。其实对于牺牲来说，这些只不过第二位的。牺牲才是英雄的真正标志。牺牲意味着一种升华。英雄总要面对死亡或者类似死亡的危机，他要展示给观众他是如何对待死亡的，他可能会幸存，来证明死亡不可怕；或者他会死，不过会复生，证明死亡可以超越；或者他们为了理想家园而死，来说明他们生命的价值。

每种原型都是要有丰富的变化的。各种各样的英雄类型：情愿和不情愿的英雄、成群结伙和独来独往的英雄、另类英雄、悲剧英雄、辅助英雄等等。情愿的英雄是那种充满热情，活跃的，敢于探索，没有怀疑，永远走在最前面的热血英雄。不情愿的英雄总是被动，不情愿，充满了怀疑和犹豫，需要被激发甚至被踹进冒险之旅。成群结伙的英雄和独来独

往的英雄是正好相反的类型。成群结伙的英雄经常是从团队中分离出来独自在荒蛮之地冒险，经历了很长时间的冒险，结识到了新的同伴组成了新的团队。而独来独往的英雄是从荒蛮之地孤独地出来，来到繁华的都市，但是最后不适应，又回到原来的地方，继续他的孤独。这两种英雄在最后都面对类似的抉择：是回到原来的普通世界？还是待在新的非常世界？有个有趣的现象：在西方的神话里面，很少有英雄选择待在非常世界而不回去的，而在很多东方神话里面，选择待在非常世界倒是很普遍。另类英雄，又称反英雄（Anti-Hero），是和正统英雄所不同的一个类型。可能以社会的观点来看，另类英雄是个混混、反叛者、犯法的人等等。他们所作的很多不被社会认同。他们有些正统英雄所具有的特点能力，但是他们还具有很多另类的性格，或者说是不被社会认同的阴暗面。可能他们会在故事的结尾取得胜利，得到观众的同情。我们喜欢这样的英雄因为他们是社会的叛逆者，讽刺戏弄这个社会，就像有时候我们也想做的。悲剧英雄也可以说是一种另类英雄，悲剧英雄也会具有很多令人羡慕的能力，但是他们始终没能克服超越自己心中的恶魔，最后被自己毁灭。悲剧英雄一般不会被人钦佩被人向往，但是观众还是会充满好奇地看着英雄一步一步地堕落。辅助英雄也很特殊。一般英雄在这个英雄之旅中会自我成长，改变很多，但是催化剂英雄却不会。他们很少改变，但是他们会促使周围的人改变，促使他们成长。这种类型的英雄经常被利用在长篇连续剧中，尤其是各自成章节的电视剧。

●智者（Mentor）

智者这种人物原型是除英雄外最重要的人物。他帮助英雄，训练英雄，保护英雄，给英雄礼物，引导英雄开始英雄之旅。无论是在神话还是童谣里，智者这个角色总是象征着英雄的最高志向。智者经常是前一代幸存的英雄。现在他把他的智慧、经验和装备传给新一代的英雄。智者这种角色来源于父母。就像父母教会孩子知识，智者同样教导英雄。

在戏剧结构中智者的作用有很多。第一个就是教授。正如学习是英雄的一个重要部分，教授或者说是训练也是智者的一个重要功能。赐予重要的礼物也是智者的一个重要功能。不过直接由智者赐予礼物，不如让英雄从智者那里赢得礼物。英雄通过在智者这里的学习训练，通过测试而得到礼物更有价值。如果智者提供给英雄装备，智者一般情况会说明如何使用，就像 007 里面的武器专家 Q。智者的另一个重要功能是激发英雄的斗志。有时候礼物就可以完成这一功能。有时候面对不情愿的英雄，智者干脆就给英雄一脚把他踹进非常世界，强迫他开始英雄之旅。

就像英雄一样，智者也有很多的类型，像什么黑暗的智者、堕落的智者、持续的智者、多个智者同时并存等等。智者有时候在故事中会把英雄引导向错误的方向，引诱他们犯罪或者毁灭，这种智者就是黑暗智者。在英雄的故事中，有些智者还在进行他们自己的英雄之旅，他们也经历过同样的困难和痛苦，但是他们堕落了。这些智者就是堕落的智者，虽然英雄仍然需要他们的帮助，可是他们是否可以胜任值得怀疑。持续的智者比较常出现在电视连续剧中，他会经常的出现，给英雄以帮助和提示等等。英雄是可以同时被很多智者

帮助的，不过这些智者要有明确的分工，负责不同方面的教学，提供不同的礼物，就像 007 当中的间谍头 M 和武器专家 Q。在浪漫爱情戏当中，有一种特殊的智者，一般是英雄的朋友或者同事，与英雄同性别。他总是给英雄很多关于爱的建议，而这些建议总是把英雄带进倒霉中，不过到了最后，这些建议都是正确的。还有一种存在于英雄内心而没有实体的智者，称之为内在的智者。这里的英雄是有经验的，很厉害的角色。他不需要什么智者或者教授。他有存在于自己内心的一种行为规范，也就是智者的另一种形式。

●入口的守卫者（Threshold guardian）

所有英雄在他们的旅程上都会遇到阻碍。在每个人口的地方都有强有力的门卫，即入口的守卫者。他们在这里阻拦不够资格的人进入。他们总是显得一副凶神恶煞的样子来恐吓英雄。但是当他们理解了英雄或者被理解，英雄就很容易克服他们，通过入口。有时候守卫者甚至会变成朋友、盟友等等。一般来说，入口的守卫者不是故事中主要的敌对力量，他们只是 boss 的走卒，也许只是暂时雇来守卫大门的，甚至是神秘的盟友来测验英雄的能力。

入口的守卫者可以为我们日常所见的普通障碍：坏天气、坏运气、某些人的偏见等等，也可以是英雄内心的一种障碍。当英雄每次想对事情有所改变的时候，一些内心障碍影响着英雄，不见得一定要阻止英雄，但是会来测试英雄是否真的做出了明确的决定。

在戏剧结构中，入口的守卫者的主要功能就是测试。当英雄面对入口的守卫者的时候，他就要解决迷题或者通过测验。如何对付守卫者呢？有很多种手段：英雄可以转身就跑，或者面对面地战斗，或者招盟友来解决这个麻烦。最有效也是最常用的方法就是智取，即 get into the skin of the opponent（装扮成敌人的样子）。在《绿野仙踪》里，当 Dorothy 被女巫抓进城堡，稻草人、铁皮人和狮子并没有和人数远远超过他们的守卫正面冲突，而是假扮成守卫的样子混进城堡。聪明的英雄不光要通过有门卫的入口，还要从他们的身上学到东西。对于英雄来说，这也是一次锻炼，而且守卫者不是被打倒，而是被智取。所以，英雄应该会从守卫者身上学到他们的技巧，这种技巧可能是英雄所不了解的新力量。

●传令官（Herald）

这个角色原型一般多在冒险的召唤（Call to Adventure）这个阶段出现。传令官的出现，意味着改变将开始。传令官为英雄带来了一个挑战的信息，他为英雄描述需要的冒险和提醒英雄注意将要到来的新讯息，英雄需要做出决定。

传令官可以起到激励英雄的作用，给英雄一个挑战，让冒险故事开始运转。这个角色可简可繁，可正可邪或者中立，可能是敌方势力的直接挑战，也可能是智者的另一个功能，传令官可以鼓励英雄接受这个挑战，也可以只是简单地把消息传送，甚至可能就是一个简

短的电话。

●善变的人（Shapeshifter）

用善变的人来形容这个角色不是非常的准确。这个角色总是改变着他的外貌或者心理状态，英雄和观众都很难很准确定他是什么样的人。他们可能会误导英雄或者总是让英雄猜不透搞不明白。一个盟友、朋友、有吸引力的异性在冒险中都可能成 Shapeshifter 类的角色。在神话故事中，各种男巫、女巫，还有善于变化的魔鬼都是传统类型的 Shapeshifter。在爱情剧里，犹豫不决或者捉摸不定的美眉也是典型的 Shapeshifter。

Shapeshifter 在戏剧结构中带来了怀疑和悬念。这种角色的出现，往往会导致英雄问自己：他是否真的可靠?她是否会出卖我？她是不是真的爱上我了？他到底是敌人还是盟友？Shapeshifter 可以说是最富于变化的人物原型类型了。最常见的是出现在爱情情节中的男女关系上。有时候，英雄也不得不戴上面具，暂时地成为 Shapeshifter 来逃出陷阱或者骗过入口的守卫者。而敌方势力或者他们的盟友有时也会成为 Shapeshifter 来阻碍或者扰乱英雄。也可能 Shapeshifter 附属在其他人物原型之上，成为一种属性。

●阴暗面（Shadow）

也可以用邪恶势力这个词来说，总之是故事中的反面，体现了黑暗的一面，与英雄对立。可以是各种有形的怪兽，可以是任何我们不喜欢的事物，包括在英雄内心的阴暗面。一般来说有几种阴暗面的类型：真正的坏人、敌手或对手、敌人。真正的坏人和敌人往往是想毁灭击败英雄的，而对手并不一定是完全敌对的，他们也许是与英雄意见不同但有一样目的的盟友。

在戏剧结构上来讲，阴暗面给了英雄一个真正的挑战，一个真正水平相当的对手。Shadow 有实力将英雄置于危险的境地。来自阴暗面的挑战可以来自一个单独的角色，也可以来自任何一个戴上了 Shadow 面具的其他人物原型，可以是智者也可以是 Shapeshifter 等等。

塑造阴暗面，有个非常重要的问题，就是要人性化。阴暗面的角色不会也不可能是个完完全全的恶魔。如果让阴暗面的角色有些优点或者独特的技能，再给阴暗面的角色加上些弱点，会让角色丰富真实化得多。还有，重要的一点：大多数阴暗面的角色不认为自己是恶魔，一般会认为英雄是他们的敌人，在他们眼里英雄就是恶魔。还有一种特殊的恶魔，就是英雄自己内心的恶魔。在这种故事中，英雄实际上是在和自己的内心战斗。

●狡诈者（Trickster）

狡诈者这个人物原型，也可以称为调皮鬼。在故事中是个弱者，但是总是使用自己的坏点子、小聪明和一些欺瞒的手段。在神话中一般是以小矮人或者捣蛋鬼的形式出现。还有一种特殊的 Trickster，被称为狡诈的英雄，也经常出现在神话故事中。当英雄显得太严肃的时候，狡诈者就会出现来进行调剂。

狡诈者可以是英雄或者恶魔的仆人或者盟友，也可以是独立的角色。在很多故事中，狡诈者通常起着催化剂的作用，影响着周围的角色们而自己却很少改变。狡诈的英雄这个特殊的英雄类型来引很多童话故事，最有代表性的就是狡猾的兔子。在各国的童话中，弱小的兔子总是用它的狡猾来对付比它强大得多的猎人、狼或者老虎等等。

虽然说这个英雄之旅的理论是西方总结出来的，并影响着好莱坞的主流电影。但是这个理论中的很多内容也频繁地出现在很多非西方神话传说中，同样适用于非西方的主流电影，而且的确也影响着很多非西方电影。可以说是世界通用的电影叙事工具哦。

电影中的英雄之旅

下面用经典影片《绿野仙踪》和 Disney 的动画片《木兰》来当作例子，给大家一个更实在的感性认识。

Campbell 的英雄之旅	绿野仙踪	木兰
普通的世界	Dorothy 平静地生活在 Kansas 的牧场里	幸福平和地生活在小镇上。木兰已经到了谈婚论嫁的年龄。相亲的情节显示木兰的性格
冒险的召唤	她的小狗 Toto 被带走，不过还好 Toto 自己跑回来了。暗示着更大的危险在后面	匈奴来袭，国家总动员。生活的宁静被打破。父亲准备带病上阵
对冒险的拒绝	Dorothy 从家里出走，遇到了一个聪明的老人。老人告诉她冒险是多么的危险和亲人会很惦记她。之后 Dorothy 决定回家，但是龙卷风强迫她开始了真正的冒险	在做了很多思想斗争之后，木兰鼓起勇气决定代父从军
与智者的相遇	在《绿野仙踪》里面，Dorothy 几乎从每个她遇到的人（Glinda 女巫，三个伙伴，甚至小狗 Toto）那里都学到了东西	小龙木须在这里可以称为第一个捣乱的智者。它帮助木兰克服困难，虽然总是在打闹之间
穿越第一个极限	实际上龙卷风把 Dorothy 带过了第一个极限，直接进入了冒险之旅	木兰终于成功地克服了第一个挑战——拔下了在高杆上的箭
测试，盟友，敌人	经过考验，遇到了稻草人，铁皮人和狮子，成为了冒险的伙伴；遇到了坏女巫的阻挠	在军营里，木兰结识了几个新朋友（打了一架），还有不太友善的大臣，并接受智者校卫的帮助
接近深层的洞穴	Dorothy，稻草人铁皮人和狮子一步一步接近女巫的城堡	训练完毕后，大家开赴战场
严酷的考验	经过重重困难，坏女巫被水溶解掉了	雪山上遭到了匈奴大军的埋伏. 千钧一发

		之际，木兰发炮触发了雪崩，埋葬了匈奴大军
得到嘉奖	带着战利品回到OZ却发现Wizard是个大骗子。Dorothy还没有办法回到Kansas	死里逃生，胜利回京，但大家识破了木兰是女子
回去的路	两人决定乘热气球回家。	木兰本欲回家，但在山上发现了单于的阴谋。
复活	Dorothy临走时被热气球拉下了．回家的希望可以说是死了，不过因为善良女巫的帮助，回家的希望再次复活	在紫禁城里的庆典上单于再次出现并绑架了皇帝，形势万分紧急！木兰重新担当起重任，机智勇敢地打败了单于
满载而归	这回Dorothy是真的回到了Kansas的家中。她认识到周围的人对于她是多么的重要，而世界上任何地方都没有家里好。这就是她从自己的英雄之旅中所得到的最重要的收获	木兰得到了单于的大刀，和皇帝的礼物，回到了家，而且得到了一份爱情。（真是满载而归啊！）

表 5-1 英雄之旅在两部电影中的应用

图 5-1 好莱坞经典影片《绿野仙踪》（The Wizard of OZ），其叙事结构极为规范，是英雄之旅理论应用的典型，不少电影教科书中都把它作为例子

图 5-2 Disney公司从20世纪三四十年代起到现在拍了近20部动画电影。无论其故事背景如何，无论技术多么发展，其基本叙事结构丝毫未变。这点近来也逐渐为人所垢病。在电影《木兰》中Disney第一次尝试采用东方题材，但其整个故事还是遵循Disney的旧有的叙事结构。这个几十年一贯制的叙事结构和英雄之旅理论有诸多重合之处

好了，关于Campbell的英雄之旅理论和其在电影中的应用就说这么多了。如果再细致的话，还是去找他的书看吧。那么这个理论对于游戏来说，到底有什么关联呢？请继续看。

游戏中的英雄之旅

最开始在游戏中突出表现情节并且能够有效表现情节的游戏类型就是RPG了。不知道日式RPG的鼻祖《勇者斗恶龙》在最开始设计的时候，有没有考虑到Campbell的英雄之旅理论。但看看标准的勇者型RPG的故事大纲，你会发现游戏的情节与英雄之旅理论是如

此的匹配。

通用 RPG 故事大纲：在安静平和的的小山村里面，有个孩子和单亲生活着。突然有一天，因为某件突发事情，孩子要离开家园去面对外面陌生的世界。在无奈中，孩子离开了小村庄。充满对未知的恐惧，开始了大冒险。这时候，命中注定的老者会出现。他告诉孩子该去找什么，或许给孩子些装备和药品，而且在孩子危机的时候会不定时的出现。在不断的探索中，孩子认识了新的伙伴，使用魔法的漂亮美眉和使用蛮力的壮士等等，大家一起冒险。在冒险过程中，总有些对手和敌人出现。孩子，哦，现在应该成为少年勇者了，和他的伙伴们克服重重困难，打败很多敌人，装备也得到了提升，也积攒了好多金钱，在世界上也越来越出名了。终于经过重重困难，到达了大魔王的住处，通过超级迷宫，击败大魔王，得到了宝物。不过这时候大魔王再复活加变身，再次击败大魔王，让世界恢复和平。大家回到勇者最开始的小村子，然后是庆祝。

以上是一个大纲，粗略得不得了的大纲。因为经典 RPG 好多，个个游戏的变化又是很丰富，想都用具体例子来说太复杂了。不过玩过些传统日式 RPG 的玩家看了这个大纲应该可以把它套入自己最喜欢的 RPG 中吧。

下面就用 Campbell 的英雄之旅理论来一一注解一下传统 RPG 的情节。

●普通的世界

普通世界就是一开始平静的小村庄，孩子的生活是普通平淡的。一般传统 RPG 会用所谓的序章来表现普通世界和冒险的召唤这两部分，展示孩子和小朋友们在平静山村是如何快乐的生活呀，还有小猫小狗之类的在村子里快乐的跑，就像《勇者斗恶龙 6》的开场一样。有些游戏这里是可以操作一段时间的，比如帮助村民做些小事情，到那里找些东西或者到邻镇送些货物之类的。有些游戏这部分就是简单的强制动画加文字，介绍一下游戏的世界是怎样的，多少年前的大战之后的平静生活啦。

●冒险的召唤

这里就会出现特殊事件来打破小村庄的平和生活，来促使孩子开始他的旅程。比如孩子被教会或祭祀中选出来背负村子留下来的历史使命，国家的征兵选中村里的孩子，这是比较平和的方式。如果要强调冲突性刺激一些，很常用的方式就是灭门惨案的类型，整个村子被不明力量彻底毁灭，孩子成了惟一的幸存者，被迫开始离家流浪。

●对冒险的拒绝（或者说是抵触）

如果是平和这种方式，就会表现孩子对亲人朋友的依依不舍，村里人的送行之类的。如果是灭门惨案的类型，孩子没什么可选择的，在离家的过程中，他可能会知道自己的使命，但是不想去接受，并漫无目的地流浪。这方面一般传统 RPG 这方面的描写比较少，因为冒险才是勇者 RPG 的重点。

●与智者的相遇

这里孩子接受到训练并且会得到些装备。比如学习到了初级魔法，得到了短刀、草鞋和布衣。主要的作用是知道了一些关于使命的残缺信息，并且有了冒险的目的。在有些传统 RPG 里面这部分可以是长期的培训，在老师的住所里面，孩子开始学习各种技能，从七八岁开始一直到十六七岁。然后某一天，老师把孩子叫来，告诉他该开始他的命运之旅了……

在传统 RPG 游戏中，一般的 Mentor 会时常地出现在旅途之中或者会有多个 Mentor 的存在。

●穿越第一个极限

这部分在传统 RPG 中不是很明显，因为在 RPG 游戏中有太多的迷宫，太多的守卫者，太多的相似的 Boss 了。这部分一般都融合于下面一个阶段（测试、盟友、敌人）了。

●测试、盟友、敌人

这部分是游戏的主要部分了。少年勇者的冒险之旅，不断地探索、练级、认识新的伙伴、打败不同的敌人。这部分实际上是游戏中最错综复杂的部分，由在不同地方发生的不同人物参与的很多小情节组成，有时候会显得很乱。不过不管怎么复杂，主线主要目的就是探索真相，逐渐明白冒险的目的。随着真相一层一层地被剥开，勇者知道了自己最终的目的地。

●接近深层的洞穴

从这个阶段开始，一直到复活阶段，基本就是和最终 Boss 之战了。传统 RPG 游戏在这几个部分没有那么多绕来绕去，一般就是进入最终的迷宫，依次击败最终恶魔的手下们，在迷宫里找到最强装备，最后面对最终恶魔。尽管他多次变身，不过还是被勇者们击败了。

●满载而归

这部分基本是以动画的形式来表现了。击败最终恶魔后传统 RPG 的操作部分一般就结束了。最后用动画来表现一下世界恢复了和平，经常有重归故里的场面。

看来不管是借鉴了英雄理论，还是与英雄理论的不谋之合，传统 RPG 的情节很大程度符合英雄之旅理论。不过因为 RPG 是游戏不是电影，所以很多情节的处理上与电影大有不同。电影一般在 1 到 2 个小时，集中地将一个结构清晰明白的故事展现在观众面前。而对于游戏，尤其是 RPG 游戏，游戏时间起码在 10 个小时以上。一般来讲 RPG 中出场的人物，所到的地域，情节的数量要比电影多得多，复杂得多。所以在很多阶段的处理上要比电影复杂得多。

传统 RPG 中一般是有确定的占主导地位的主角的，也就是勇者。整个 RPG 的进程完

全围绕这个主角展开。所以，占主导地位的主角的“普通的世界”，“冒险的召唤”和“对冒险的拒绝”三个舞台，就会作为整个 RPG 的前三个阶段。随着勇者冒险的逐渐展开，他会遇到伙伴，他们会伴随勇者一起冒险。而对于这些伙伴来说，属于他们自己的“普通世界”，“冒险的召唤”和“对冒险的拒绝”这三个舞台，就出现在整体 RPG 的第 6 个阶段“测试、盟友、敌人”之中。他们自己的经历也和勇者一样，有着同样的英雄之旅，不过因为他们不是占主导地位的主角，所以在游戏中的地位不如勇者，在这些部分费的笔墨也许会比勇者要少些。

智者，在传统 RPG 中一般不会只有一个。包括引导勇者面对现实和命运的智者，传授技能的智者，赐予装备的智者等等。因为 RPG 游戏是一个长时间的旅程，不像电影是一锤子买卖，所以勇者在旅程当中，会遇到很多次挫折，学会不同层次的技能，得到不同 level（级别）和属性的装备。如果所有这些都是由一个智者所赐予，未免也显得太过单调了。所以在不同的地方，不同的智者会出现来帮助勇者。关于智者的各种类型之丰富多变，前面有所阐述。而且，对于传统 RPG 来说，勇者水平的提高很大部分是来自自己不断的锻炼，也就是练级，在升级之后自己总结出新的更强的技能。这部分是传统 RPG 游戏部分的主体，所以所谓传授技能，一般也就会体现在游戏初期或遇到特殊人物的时候。

对于第 5 个阶段（穿越第一个极限）和第 6 个阶段（测试、盟友、敌人），因为传统 RPG 游戏的复杂性，结构不会像电影那样清楚明显。勇者要旅行全世界，要到达很多新的地方和面对很多和主线没有关系的情节，也就是分支情节。不要认为分支情节真的和主线无关！分支情节的最重要作用就是塑造一个有血有肉的情感丰富真实的主角（包括勇者和他的伙伴们）。勇者在分支情节中认识新的同伴，在分支情节中逐步得到线索，在分支情节中逐渐成长找到自我，在分支情节中被蒙蔽欺骗，在分支情节中学会分辨善恶，等等。但是如果分支情节处理得不好，会冲乱整个故事的主线，使整个故事显得支离破碎，让玩家在旅程中搞不清楚勇者冒险的真正目的。

每一个分支情节，实际上也会遵循英雄之旅理论。为了使故事结构丰富多变，每个分支情节都不会完全 copy 这 12 个舞台，而都会用不同的叙事方式，有着丰富的变化。比如勇者们刚进村子，就遇到了突发事件，村子被怪事笼罩。随着一步一步的调查，在村子的枯井中打败怪兽，村子又恢复了平静。一个标准的分支情节，简略了 12 个舞台，其中可能只用了同伴的对话来表现勇者对这个任务的犹豫，然后在同伴的鼓励下接受了任务。路上的某个人说：“村中的枯井前几天有神秘的蓝色光芒！”就是智者对你的引导。直接进入枯井的迷宫，打败了妖怪，也没有什么再复活之说，就回到了村子。不过，显而易见，这个分支情节平淡得要命，是 10 年前勇者 RPG 的叙事方式了。所以说，即使是分支情节，也不可以在情节设计和叙事方式上面过分缩水。

基本上来说，从英雄之旅的第 7 个阶段开始就是目的明确向最终恶魔的进攻了。这部分和前面的对比不像在电影里面那样强烈，只是作为冒险历程逐渐变难的最后部分，最复

杂的迷宫，最难对付的敌人，勇者的最强装备等等。一般在击败若干次变身的最终恶魔之后，就是游戏操作部分的结束了。很少可以操作第 10 个阶段“回去的路”的。满载而归就更不会要求玩家操作了，也没有必要。在最激动人心的最终战之后，玩家需要的是放松和过关之后的欣赏，如果回归故里需要玩家操作的话……

上面讲的基本上是传统 RPG 游戏的叙事结构。这种 RPG 的重头是勇者的冒险，勇者的自我成长。从超任时代开始，《最终幻想》为代表的非勇者 RPG 越来越注重戏剧性的表现，编剧开始更为主动的控制情节。这时的 RPG 情节可以说丰富多变，再加上多个主角同时演绎，让故事结构错综复杂。这种注重情节的 RPG，经常采用倒叙、回忆等手段，各种分支情节穿插在一起，纵横交错。再加上这里的多主角系统不再是像传统 RPG 中以勇者为中心的模式。每个主角的份量都不轻，都不是属于从属位置，还有着各自的故事和目的等等。比如最经典的《最终幻想 6》，多达 12（14）位的主要角色，可以称为核心的也有四五位了。在《最终幻想 6》里面，经常会通过操作不同的主角，来表现发生在同样时间不同地域的故事。这种交换主角的做法在传统 RPG 里是没有的，也导致叙事的困难。更有甚者，有的时候可以进入主角的梦里或者主角的回忆中进行操作。所以说《最终幻想 6》可能是经典中无法超越之作了。从叙事角度来说，能够将 12 位主角的故事有机地结合而没有感到混乱，每个人物的故事又真正的将角色塑造的有血有肉，实在是编剧的实力呀。

这种注重情节的 RPG 和传统的勇者型 RPG 在叙事上有几点重要的不同。一个就是分支剧情的穿插。在传统 RPG 情节中，一般要勇者到达某地，解决事件，才可以进入下个场景。所以说虽然分支情节众多，但是基本来说是按线性排列，比较有序，比较清楚。不过也比较单调。而对于注重情节的 RPG 来说，并不是、为了冒险为了展示世界不同的地域来安排新的分支情节，而是为了情节而安排冒险和新的地域。这就表现在需要经常来往于世界各地。这可能也是当年飞艇出现的一个原因吧，哈哈。情节为主之后，不必要按线性的方式排列分支剧情了，几件事情同时发生在不同的地方，彼此影响和交错。的确，在传统 RPG 中偶尔也会有同时发生的情节，不过都是通过某角色讲述或者强制的动画演示来表现的。而对于注重情节的 RPG 来说，这种情节要频繁得多，而且是可以控制的，由不同的主角来完成同时发生的事件并且相互影响。不过说回来，每个分支情节还是遵循了英雄之旅理论，只不过打乱了叙事顺序和删减了部分阶段。

另一个不同就是多角色的性质不同了。在传统 RPG 中，勇者是玩家，勇者是所有的核心，其他伙伴虽然也有自己的故事和性格，但是他们只是辅助主角冒险的。虽然他们也有自己独立的冒险之旅的开始 3 个阶段，但是当他们遇到了勇者和勇者一起开始冒险之后，后面的 9 个阶段是和勇者一起度过的，可以说基本是一样的。而注重情节的 RPG 中，玩家不会总扮演一个主角。随着故事的展开，玩家扮演多个角色来体会他们各自的英雄之旅，有着不同的多个舞台，虽然到最后会在一起面对同样的最终恶魔。所以说，每个主角的英雄之旅都是充实的，可以独立成章的。独立成章不是非常地难，但是如果将多个主角的英

雄之旅穿插在一起，成为游戏的英雄们之旅，就是非常难的工作了。这种富于情节变化和多角色控制的 RPG 可以说是在游戏编剧中最难的类型了。

随着硬件的进步，AVG（以生化危机系列为代表）这个类型终于可以从次世代开始风行了。这个类型与 RPG 相比，从镜头和叙事等方面来说，更为接近真正的电影。一般来说，AVG 都会发生在一个较小地域，一个小镇或者一个岛。主角一般就有一个，而且是故事的核心，不会出现多个主角平分秋色的情况。不会存在过多的分支情节，整个故事主线明确，如果除去游戏者操作的问题，游戏时间会在两个小时左右。可以看到和电影是多么地相似，可以说 AVG 是最接近电影的游戏类型了。

和 RPG 游戏相同的是，AVG 主要的游戏时间集中在英雄之旅的第 6 到第 11 个阶段。第 6 个阶段“测试、盟友、敌人”也是最丰富的舞台。不过与 RPG 不同，这部分不会被众多的分支情节所丰富，而是一直紧密追随着主线，所有的小情节都紧密围绕着主线，为阐述主线而服务，就像电影的这部分一样。虽然已经极其地接近电影，但是游戏毕竟是游戏，不光要从观赏的角度来考虑剧情，还要从游戏本身来考虑剧情的设置。

本章中主要从最强调情节的两类游戏：RPG 和 AVG 来说了说在游戏叙事设计方面，英雄之旅理论的可用之处。其他类型的游戏，像对战、实时策略、FPS 等等，在目前来讲，它们的主体不是剧情，不像 RPG 和 AVG 那么强调情节。总之，如果可以真正理解英雄之旅理论，灵活地运用，加以丰富的变化，就一定可以写出出色的游戏故事。

第六章　故事性和交互性

在魂斗罗和马里奥大行于 FC 之时，故事性似乎是不那么重要的。那时的玩家被称为闯关族，顾名思义，就是一个游戏关卡一个游戏关卡地闯过。对动作游戏（ACT）来说，故事情节也许是可有可无的。试问有哪个玩家在乎《魂斗罗》的故事情节呢?跳跃、开火、打倒 BOSS、过关就足够了。那时候交互性显然在游戏中占主导地位。但随着 RPG 的出现，故事性开始被游戏设计师们重视起来了。新一代游戏中，图像、人物、故事成为三个要素（虽然有关于游戏性越来越差的争议）。我们可以注意到连《半条命》（Half-Life）这样的射击游戏（FPS）也开始有复杂的故事情节了。《半条命》的故事情节是由专业作家撰写，受到了玩家和各游戏杂志的一致好评。但是我们注意到虽然游戏业近来重视了故事性，但对于故事性和交互性之间的矛盾则还是没有很好地解决。

不可调和的矛盾

自游戏问世之日起，故事性与交互性两者之间不可调和的矛盾就一直困扰着游戏设计师们。故事情节是由作者精心构思的对一个网状信息集合的结构严谨的惟一线性表述，而交互性则要求玩家对周围的环境和其他人物的反应具有一定的自由度和控制能力，也就是说是对一个网状信息集合的动态改变和组合。

举个例子：一部电影，我们看的时候所看到的是单线的故事情节，我们是单向接受信息的，但电影本身所包含的信息不是单线的，而是网状的。因此，电影是其所表达的信息集合

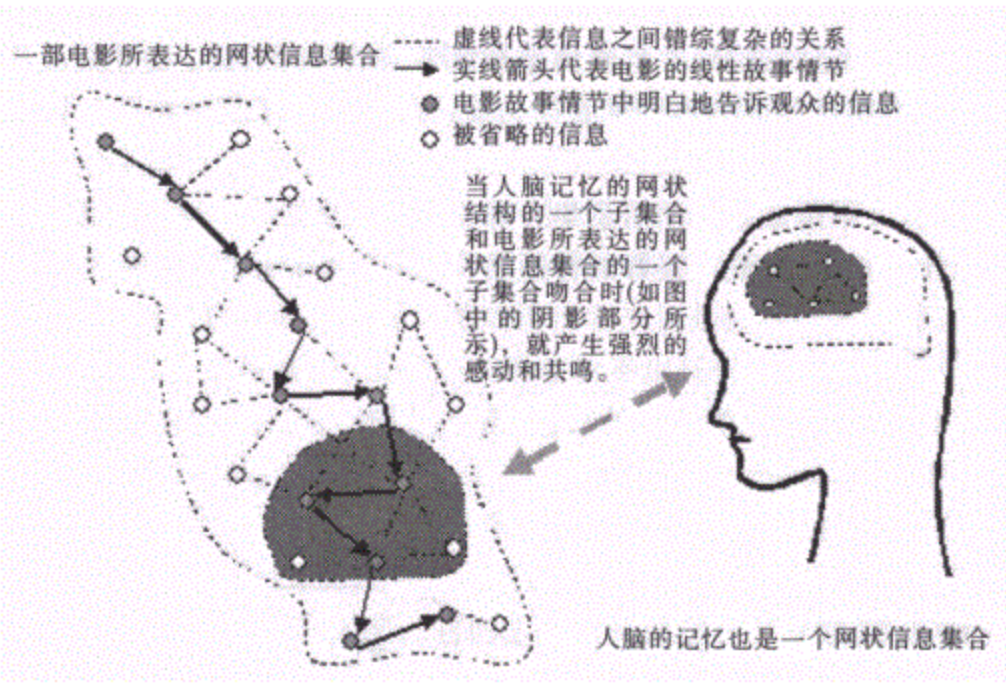


图 6-1 网状信息集合

的一个视图（view）。这个信息集合中，有些信息被省略了（因为观众能够推理出），有些实际发生顺序被颠倒了（即电影技术中的闪回）。因此，观众在观赏一部电影时，所接受的不仅仅是单线发展的故事情节，而是在故事情节背后的庞大的网状信息集合，而人的记忆也是一种错综复杂的网状结构。观众被电影感动了的原因，就是电影的网状信息集合中的一部分（一个子集），和人的记忆的网状结构中的一个子集吻合了，由是产生感动和共鸣。这种共鸣越多，作品就越感人。电影剧本的作者所做的，就是把他脑子里的一团网状信息集合，截取部分信息，连缀起来，形成故事情节，并且保证大部分观众能够根据故事情节，明白整个信息集合的内容，并且产生一定程度的共鸣。

交互性则是说，作为游戏主人公的玩家，面对构成一个虚拟世界的网状信息集合，没有事先规定好的线性表述的限制，而是通过自己的行为，去动态地改变这个信息集合的结构组合关系，去改变这个集合的状态。玩家的每一个选择，都是在已经形成的线性故事情节上又多加了一段，随着故事情节的增长，这个网状结构所可能转化的状态也越来越少，最后达到某种结局。

两者之间的差异和矛盾是显而易见的。一个是由线性到网状（故事性），一个是由网状到线性（交互性）。

什么是交互式故事

目前游戏业讨论的焦点是所谓的“交互式故事”（interactive storytelling）。这是一个很模糊的概念，业界对其确切定义并没有一致的意见。以笔者看来，交互式故事就是一种介于线性和网状之间的状态，也是故事性和交互性之间的某种妥协和调和。其特点就是由游戏设计师对网状的信息集合做某种程度的结构简化和调整，使得玩家在游戏玩时能够做出一定的选择，并且看到选择所带来的实际后果。交互式故事和传统剧本相比最大的不同就在于传统剧本是经过了高度简化的惟一的线性结构，而交互式故事没有简化到那样的程度，仍然保持一定的网状分支，但又不是像原始的信息集合那么繁琐庞大。但其结构具体来看应该是什么样的，有没有很好的普适结构，则还没有现成的答案。

RPG 的特殊性

RPG 是所有游戏类型中故事性比较强的了。因此，RPG 对故事性和交互性矛盾问题的处理也更值得我们研究。传统的单线故事情节的 RPG，有十几年的历史了，其基本结构变化不大，如图 6-2。

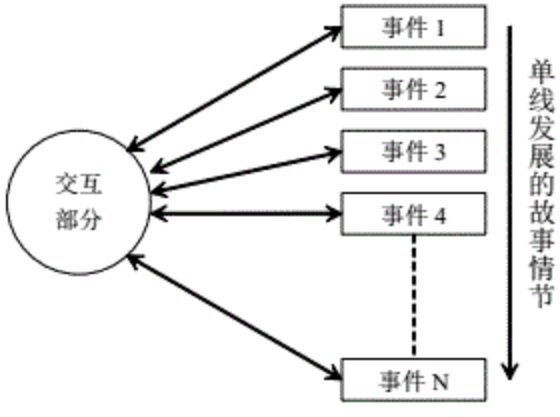


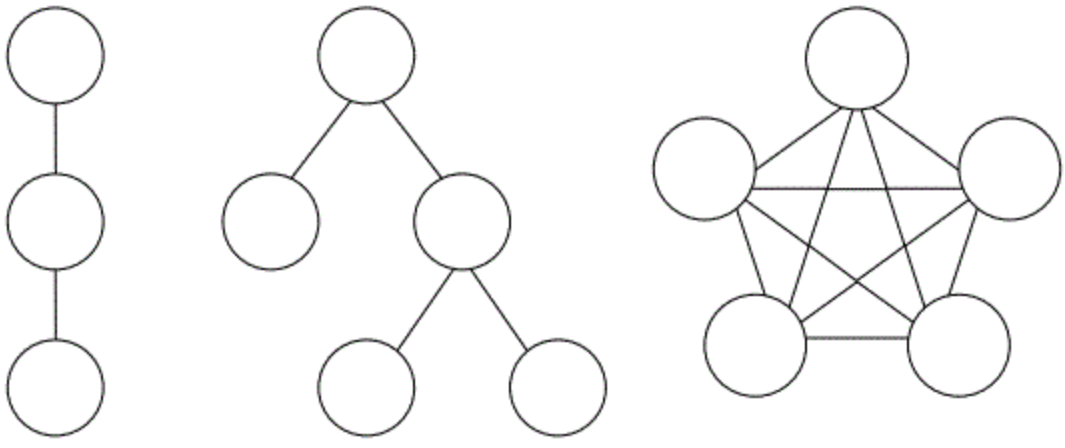
图 6-2　RPG 基本结构

单线故事情节的 RPG 的特点是，虽然

给了玩家很多选择和自由度，但这些选择和自由对故事情节的发展是毫无影响的。玩家在战斗和使用魔法时才是真正享有自由，而对故事情节则无能为力。玩家只有严格按照要求去触发下一个事件，才能推进故事情节的发展。因此 RPG 设计的真谛，就是给玩家一个假象，让它们感觉到好像有对自己命运的支配能力，让它们感觉到好像自己可以随心所欲地到处走来走去，让它们感觉到好像自己有选择权，而实际上他们没有任何真正的选择权，也没有真正的自由，游戏在巧妙地引导它们按步就班地根据预定剧本去完成一个个特殊事件。RPG 就是这样一个假象而已。

实际上上面所述的单线 RPG 结构在十几年前就很成熟并定型了，直至今日未有太大发展。现今的一些所谓 RPG 大作，如 PC 上的《仙剑奇侠传》，SFC 上《Chrono Trigger》等从本质上说比之八十年代 FC 上处于胚胎期的 RPG 游戏没有什么不同，只不过画面更精致了，有了花里胡哨的动画，高保真度的音效音乐等等。那么使 RPG 游戏真正具有一定的主动性，这一梦想无疑对游戏设计者来说是极具挑战性的。但遗撼得很，技术上难度相当大，起码在现在看不出任何曙光。因为这里面的问题早已超越了 RPG 游戏的本身。

我们知道，超媒体（hypermedia）或超文本（hypertext）技术出现之前，人们读一本书，一般是按一定章节顺序读的，也就是线性的。书的组织也是按一定的线性顺序组织的。超媒体（文本）技术出现以后，在多媒体出版领域成为事实的标准，它们实际上是通过在文章中设定关键字跳转，使线性的书形成一种网状结构。这样从同一个起点浏览，碰到关键字后跳转的可能性成 N 次方级激增。正如回字有几种写法一样，同一本多媒体读物，包含同样的素材，但可以有无数种读的方法。但这种多媒体读物一般适用于百科类图书，相当于资料汇编。对于有故事情节的文学性著作就无能为力了。具有情节性的 RPG 游戏现在也是线性结构，若采取多线结构，首先面临的是选择一个合适的拓扑结构。从数据结构角度看，可以有树状结构，网状结构等。如图 6-3 所示。



线性结构　　树状结构　　网状结构

图 6-3 三种拓扑结构

如图 6-3 可见，在多线 RPG 中，事件的含义与单线 RPG 有很大不同。在单线 RPG 中事件是一个无交互的叙事性段落。在多线 RPG 中，事件不仅是触发后的一段叙事段落，更重要的是一个选择点，它决定了事件流的导向。正如人生中遇到的许多选择一样，不同的选择将导致截然不同的结果。这样游戏者不仅拥有决定事件发生时刻的权利，同时也拥有了决定发生什么事件的权利。尽管选择的范围也是被游戏设计者设定的。

树状结构与网状结构不同的是，在一次游戏过程中，树状结构遵循因果律。事件是分级的，不同级的事件发生的先后顺序是确定的，有因有果，并且同一级只能有一个事件发生，则这一级其他事件（节点）及其以后的事件（子树）在以后的游戏过程中将不起任何作用。而网状结构则提供了在事件集中任意漫游的可能。无分级的概念，无因果的约束，任何事件都可能被触发。显然网状结构不符合我们日常生活实际，但网状结构的研究对多线 RPG 还是很有意义的。

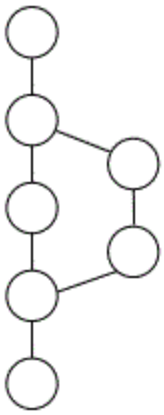


图 6-4 带支路的单线结构

考察树状结构，我们发现其实现的最大障碍不是技术上的，而是其数据（资源）费效比太低，数据（资源）冗余度太大，以至于完全不能按理想的树状结构去设计一个游戏，倘一个 RPG 游戏有 10 级，形成完全二叉树（每一个事件点上游戏者面临两个选择项），游戏者在一次游戏中只可能经历 10 个事件。但游戏设计者为了实现二叉选择这一功能，将不得不准备 2 的 45 次方个事件的有关数据，这简直是个天文数字（当然这和真正的人生有些相似，人生中每一天每一小时每一分钟每一秒的可能性都是无穷的，在每一天每一小时每一分钟每一秒做出的选择导致的一连串因果相循的后果的可能性也是无穷的）。

目前几种标榜多线的游戏，多采取图 6-4 的拓扑结构。

这种简化的结构是使单线结构出现了几个小支路，最后还归并到主线的不同地方。这实际上只是一种改进了的单线结构，实际意义令人怀疑。从心理上分析，游戏者在费很长时间很大精力玩完一遍游戏后一般不会为了几个无关紧要的支路再重新将主线遍历一遍。一般来说，第二次玩 RPG 游戏时，只有当游戏的 2/3 主线具有新内容才能被游戏者接受。但这样大规模对主线附加支路，将使 RPG 游戏的拓扑结构变得像图 6-5 所示。

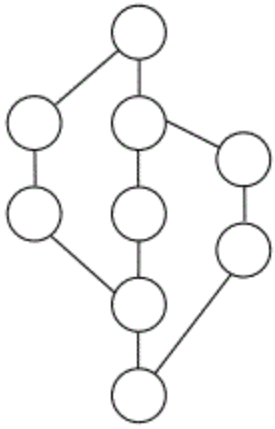


图 6-5 带多个支路的单线结构

可以看到，冗余度仍然相当大，接近 200%。这是目前技术所能实现的，有一定多线性的 RPG 结构。

在网状结构中，每一个事件都是可能发生的、平等的，这是一种有效的数据（资源）组

织形式，不存在资源浪费（冗余）问题。可以设想是否可将因果性的树状结构和非因果的网状结构结合，从而解决真正对人生选择的模拟和海量数据冗余之间难以调和的矛盾问题。当然这有待于进一步研究，而且也并非一朝一夕就可以解决的。

显然，传统 RPG 对故事性和交互性矛盾的处理方法是简单的。它实际上是让玩家被动地欣赏故事的同时，主动地参与一些无关紧要的小活动。可以说 RPG 的解决方法是有效的，但其弊端也是明显的。因为玩家无法改变故事情节。那么有什么方法可以让玩家影响并改变故事情节呢？是否能根据玩家的选择，动态地改变网状信息集合呢？这就引出了下面的题目，目前人们在故事自动生成方面的研究和传统的故事结构理论。

交互式故事生成系统的研究

目前有人在所谓“故事自动生成系统”方面作研究。就是研究是否能够输入一些人物设定和基本故事成分，由程序自动组合成复杂的故事情节和人物之间的交互。最有名的就是 Chris Crawford 所搞的 Erasmatron 交互式故事生成系统。使用这套系统，你先设定几个人物角色，然后设定他们的性格，如情绪激动型、沉闷型等，然后设定故事发生的一些场景，然后设定各人的初始地点和活动范围，设定各人之间的人际关系，如密友、夫妇、情敌等。在此之后，最主要的任务是定义一系列动词，比如：吃饭、睡觉、吵架、打斗、跟踪等等。然后这套系统就可以自己运行起来，形成一定的故事情节。

但有两个棘手的问题是 Erasmatron 和其类似的系统很难解决的：一是各人物角色之间的交互，如悄悄话、语言等功能。虽然 Erasmatron 系统包含了悄悄话等功能，但复杂的人际关系实现起来还是十分困难。二是用这种方法可以生成基本的故事情节（可以表达你吃饭、睡觉、闲逛等常规活动），但却不能形成戏剧冲突，特别是逻辑严谨令人信服的戏剧冲突！最后还得进行人工干预。因为没有这种戏剧冲突，故事就平淡无味。

虽然 Chris Crawford 的系统已经开发了数年之久，其人研究交互式故事系统也有很长时间了，但就业界目前的技术水平，把这一研究成果转化为实用还是遥遥无期的事。但这项研究无疑是有前瞻性的，因为随着网络游戏的兴起，新出现了大规模多用户网上 RPG（Massively-multiplayer Online Role-Playing Game，简称 MMORPG）。这种类型的游戏，

同时有成百上千的玩家登录加入，原有的单线发展的 RPG 故事结构显然不再适用了。人物之间的交互性显然比较容易实现，但如何在这么大的系统里保持一定的故事情节，产生戏剧冲突，则是难上加难。目前可能看到的解决方案还是由一定的故事生成系统外加人工干预。当然，我们可以预言这方面的研究将越来越多的。

有关故事结构和构成的研究

实际上，学者们在很久以前就开始了对故事结构和构成的研究。其中最具代表性的，就是苏联人 Vladimir Propp 对俄国民间传说所做的结构化分析。他在 1928 年写出了《民间传说的形态学》（Morphology of the Folktale)一书，对俄国民间传说的叙事结构作了深入的研究。由于显而易见的原因，他的这种“形式主义”的观点受到前苏联政府和御用学者们的压制。但在世界范围内，这本书的影响极其深远。

图 6-6 由得克萨斯大学出版社出版的《民间传说的形态学》的英译本

Vladimir Propp 在研究俄国民间传说时，发现各种故事传说，虽然情节人物各不相同，但从故事结构或者说故事构成来看，有极大的相似性。他发现俄国民间传说中的主人公们，虽然性别年龄职业各异，但他们在故事发展过程中所做的事情几乎都是一样的。

例如，三个民间传说的开始部分：

- 国王命令伊万去寻找公主，伊万出发了。
- 姐姐让弟弟去找药，弟弟出发了。
- 后母让孤女去采集木柴，孤女出发了。

前面三句结构是相同的，都可归纳为 A 命令 B 去找 C，B 出发了。其中 A、B、C 都是名词，或者是主语，或者是宾语。它们都是变量，随不同故事情节而不同。而谓语动词“命令”“找”“出发”是不变的。

其他的例子如：

- 恶龙绑架了国王的女儿。
- 女巫绑架了一对年老夫妇的独子。

结构也完全一样，Vladimir Propp 在广泛观察归纳的基础上提出了他的四点假设。

1．功能模块是民间传说中固定的、独立的元素，与其执行者（主人公和其他出场人物）无关，是构成一个民间传说的基础。

2．功能模块的数量有限。

3．功能模块的组合顺序有一定规律。

4．所有民间传说在结构上相似，都是由相同的功能模块按相同的次序组合而成。

他进一步定义了各种功能模块和其表达符号。下面举几个例子。

一个最常见的功能模块是：家庭成员离开了家，用β来表示，可以有几种变化。比如家庭中的老年成员外出了，只剩下了年轻一代，用$\beta 1$来表示。更极限的状况，比如老年成员死亡，用$\beta 2$表示。也可以是年轻一代出走，用$\beta 3$来表示。

当一个家庭成员离开的时候，他对年轻一代要留下一条禁令。比如：“千万不要打开那个壁橱！”或者“照看好你的弟弟，不要走出院子！”

留下禁令这个功能模块用γ来表示。

在民间传说中，禁令是注定要被违反的。禁令违反之时就是主人公冒险的开始。违反禁令这个功能模块用δ来表示。

Vladimir Propp 最后归纳出了 31 个基本的功能模块，使用这些功能模块就可以把一个故事抽象成符号的组合了，以下面的故事为例。

很久以前有一对老夫妇。他们有一对儿女，姐姐和小弟弟	初始（a）
有一天，老夫妇要外出。母亲对女儿说：“女儿，女儿，我们要出去。回来的时候我们会给你带来一块小圆面包，给你织一件小裙子，再给你买一块方头巾。你在家好好照顾弟弟，不要走出院子！”	留下禁令（r） 家庭成员离开（B1）
老夫妇离开了家	禁令违反（d）
过了一会儿，姐姐忘记了父母的嘱托，他把小弟弟放到窗前的草地上，自己跑出去到街上玩去了	
一对天鹅飞过，它们把小弟弟衔起来，放到翅膀上，带着他飞走了	恶魔绑架（A^1）
……	

最后整个故事用符号来表达就是：

$$\gamma^1 \beta^1 \delta^1 A^1 C \uparrow \left\{ \frac{DE^5 neg.Fneg}{d^5 E^5 F^9} \right\} G^4 K^1 \downarrow [\,\mathrm{Pr}^1 D^1 E^1 F^9 = Rs^4]^3$$

Vladimir Propp 他把成百上千个民间传说打碎开来，使用语法/句法结构分析，把它们分解到最小的功能模块，然后再组合成叙事结构的一系列拓扑模型。在他书中列举了 8 大类的模型。今天的学者们曾经用这些模型去套在现代流行电影电视作品的叙事结构上，如 20 世纪 70 年代卢卡斯的《星球大战》和近年来很有名的电视剧《x 档案》等，据说十分吻合。这种结果是意料之中的：电影镜头及特效技术的发展一日千里，但人类说故事和理解故事的

方法是不变的。

20 世纪 80 年代以来，有一些研究人员和游戏设计师们曾经抱着极大的希望去研究 Vladimir Propp 的理论，希望能够把他的理论用计算机实现。这样一来，由人工输入名词和动词集，程序把它们填入基本结构，似乎这样就可以自动生成故事了，但研究的结果令他们失望。首先，Propp 的工作只是归纳总结。从一百个具体的故事中总结归纳它们的符号表达比较容易，但要从符号表达衍生一个具体故事则完全是另外一个问题了，他的理论根本没有涉及后者。另外，他的理论是在 20 世纪初发展出来的，自然没有考虑到用计算机程序实现的问题。而要发展其理论，目前游戏设计师们的水平显然又不够。总而言之，现在的技术水平还没有达到应用他的理论的程度。在人工智能研究领域，倒有不少类似的系统问世，但也只是在实验室中而已。总之，他的理论虽然不一定能够达到实用化，但对今后的研究工作，仍旧是有一定的启发作用的。

第七章　关卡设计

关卡设计这个名词（level design）和关卡设计师（level designer）这个职务，是20世纪90年代中后期，随着三维射击游戏的流行而应运而生的。因此严格地说，应该称之为三维关卡设计。在DOOM类型的三维射击游戏出现之前，是没有这个称呼的，在游戏公司里也没有专门的被称为关卡设计师的人。在8位、16位游戏机上的动作游戏（ACT）中也是有多重关卡的，而且当时的玩家被称为“闯关族”。但二维游戏的关卡总体来说还是比较简单，因为是横卷轴或者纵卷轴，场景固定，敌人也是事先地在一定时间从一定的地方出来。而到了三维时代，关卡的复杂度极大地增加了，敌人（NPC）的智能也增强了。游戏对玩家的要求提高了。对二维ACT游戏来说，玩家知道自己在什么地方，知道自己在向什么方向进发，知道下一个敌人将会在什么地方出现。而三维射击游戏，你可以向四面八方走，还有不同的高度层，玩家的方位感尽失。比之二维ACT游戏的单向卷轴（一维自由度），三维游戏，也难怪很多老一辈的闯关族找不着北了。既然三维游戏对玩家的要求提高了，当然对设计者的要求也是提高了。当关卡设计的工作量和复杂度大到一定程度，关卡设计的工作就独立出来了，需要专人负责，关卡设计师这项职务就诞生了。

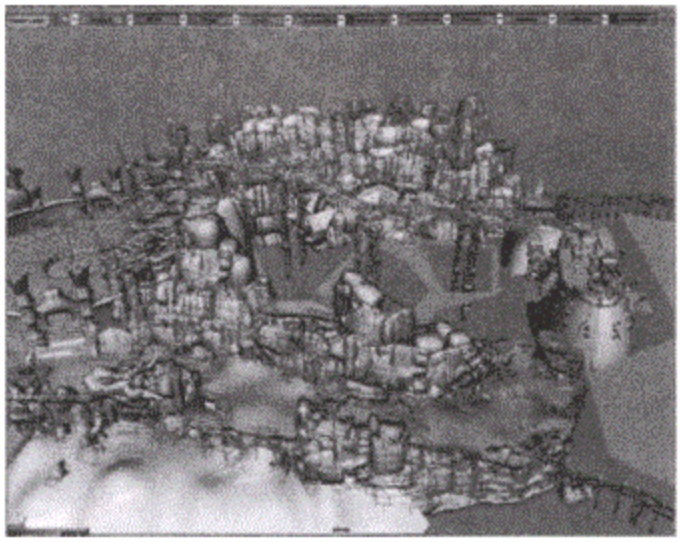


图7-1《Jack & Daxter》的三维关卡。由此，三维游戏的复杂程度可见一斑

什么是关卡设计

简单地说，关卡设计就是设计好场景和物品，确定好目标和任务，提供给玩家（游戏人物）一个活动的舞台。在这个舞台上，玩家表面上拥有有限的自由，而实际上关卡设计师通过精心布置，来把握玩家和游戏的节奏和导向，最终达成一定的目的。

关卡设计的重要性，在于它是游戏性的重要组成部分。游戏的节奏、难度递阶等方面很大程度上要依靠关卡来控制。

关卡设计要素

关卡由以下几个要素组成。

地形

地形是关卡最重要的组成部分。地形是指室内或者室外的建筑和地貌，抽象出来就是多边形拼接在一起形成的一个中空的空间，玩家就在这个空间里面漫游。空间之内可以又再分为几个相互连接的子空间。

关卡设计实际上就是对空间的规划，特别是建筑物内部空间的规划。除了几何形体外，还要考虑内部装饰、灯光效果和人在一个三维空间内的感觉和行为模式，这些东西显然和建筑学的很多方面有重合之处。目前的关卡设计师们已经开始有意识地借鉴研究一些建筑学方面的经验和理论了。

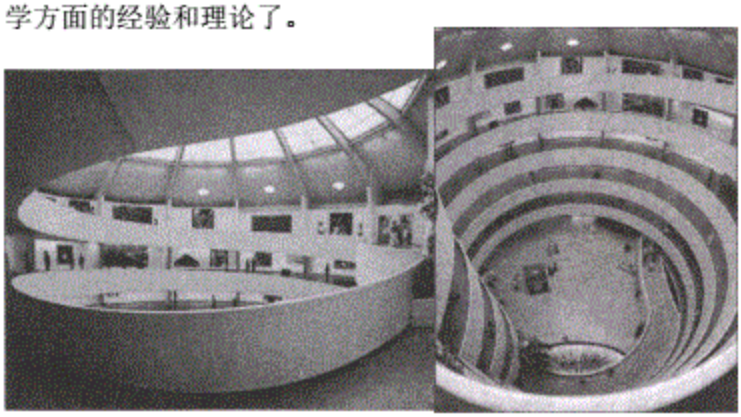


图 7-2 赖特（FrankLlovd Wright）所设计的纽约古根海姆（Guggenheim）艺术博物馆，在内部空间的设计和利用上有独到之处。其内部为开放式布局，中空，台阶沿螺旋线旋转上升，四周墙壁用来挂画。这样当参观者对着一幅画欣赏时，他所注意的是局部。而当他看完转过身来时，他所面对的是全局。这个设计巧妙地平衡了局部和整体，细节和大面，并照顾到了浏览的需要

图 7-3 米斯（Ludwig Mies Van der Rohe）为伊利诺伊理工学院所设计的建筑学系 Crown Hall，使用外部钢梁悬挂结构，建筑内部没有一根柱子，最大程度地提供了可利用的内部空间

在三维游戏刚刚兴起时，由于计算机处理能力的局限，大多数关卡都是在建筑物内部的狭窄空间内。随着计算机处理能力的增强和各种算法的优化，在新一代三维游戏中室外场景和自然环境变得更常见了。

边界

边界是一个关卡必须的组成部分。关卡不可能无限大，必然要有边界。关卡的大小和完成关卡需要的时间有直接关系。一般来说，关卡之间是不连通的，只有完成了既定的任务才能进入下一关。部分边界可以是关卡之间相连的纽带。

物品

各种物品，包括武器、加力、补血等功用。在关卡中，各种物品的安排和布置可以对游戏的节奏和难度起很重要的平衡作用。这些物品的安置完全是靠经验通过不断调整才能获得最佳效果。

敌人

同物品一样，各种敌人在关卡中出现的位置、次序、频率、时间，决定了游戏的节奏和玩家的手感。早期动作类型的游戏中，敌人不具有智能，其行为被预先设定得死死的，每次在同样地点或者在同样的时段出现。游戏设计师则具有完全的控制能力，通过细心调节，可以完全设定各种敌人出现的位置、次序、频率、时间，力求达到最优。那时候游戏的游戏性令人怀念，很大部分就是这种控制和调节的结果。

在三维射击游戏问世后，NPC 的概念得到发展，人工智能越来越得到增强。敌人出现的时机和行为，不再是事先规定得死死的，而是在一个大的行为系统和人工智能的指导下，有一定的变化和灵活性。这给传统动作游戏的游戏性反而带来了一些麻烦。游戏设计师这时候已经失去了对关卡中的敌人行为的完全的控制力。如何利用有限的控制力去实现最优效果，是摆在新一代游戏设计师——关卡设计师面前的难题。关卡设计师这时必须和人工智能程序员合作，使得游戏既富于惊奇变化，又具有一定的平衡性。

目标

一个关卡，要有一个目标，即希望玩家通过此关卡而达成的任务。目标也可以有一些子目标，子目标相互之间成为串联或者并联关系。目标应该明确简单，毫不含糊。

前后连缀的关卡之间，目标要有一定关联，并且和整个游戏的总目标形成渐近从属关系。

情节

情节和关卡之间的关系可以多种多样。两者之间可以没有什么太大的联系，比如说早期的动作游戏。也可以通过过场动画交代情节背景，特别是通过过场动画使玩家明确下一关卡的任务。更可以在关卡进行中加入故事要素，使得玩家在游戏过程中得到某种惊喜或

者意外。

大小

讲到关卡的大小，不仅仅指在玩家眼中关卡的大小和复杂度，更重要的是实际文件大小，比如材质文件大小。关卡设计师在设计关卡时对各种文件大小的问题注意得很多，因为这涉及到是否关卡可以被最终实现，特别是游戏的实时性能。

视觉风格

关卡的视觉风格，体现在地形设计、材质绘制、光影效果，色彩配置的组合。

关卡设计流程

正如一切设计活动一样，关卡设计需要一个流程（process）。设计流程的作用是保证每个关卡按时完成，使其质量具有连贯性，并且利于协作交流。

目标确定

关卡设计的第一步是确定目标。目标基于任务，也就是前面所介绍的一个关卡所要玩家达成的任务。目标是从设计者角度看问题，而任务是从玩家角度看问题。目标可以有多角度，多方面，比如“此关卡一般水平玩家将费时 10 分钟”，“此关卡将使得玩家得到 xx 宝物”。

除了确定目标外，还要初步了解技术上的限制，比如材质文件的大小、多边形数量的限制等等。除了技术上的限制外，还有其他非技术的限制，比如进度要求。

目标和限制相互作用。设计者要动用一切手段达成设计目标，但各种技术上和非技术上的限制使得设计者必须做出判断和一定的牺牲。所有的设计活动都是两者牵制作用的结果。

集体讨论

在明确了关卡的总体目标和具体限制后，就进入集体讨论阶段。一般是由所有组员（包括关卡设计师、美工、和程序

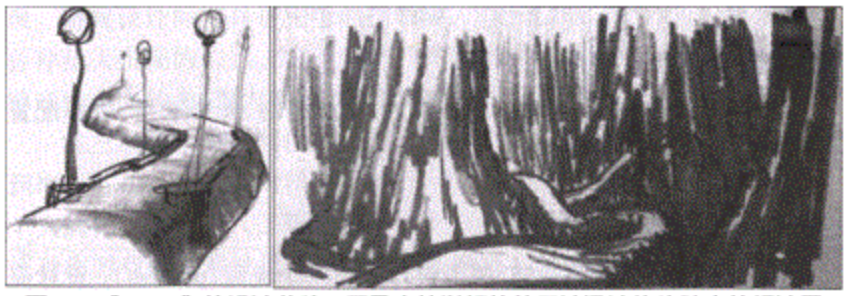


图 7-4 《Raven》的设计草稿。用马克笔以粗练的于法迅速将头脑中的想法展现到纸上。无论是在集体讨论中，还是在其他场合，一幅画的效用要胜过千句话。这样的草图有助于组员们在设计初始阶段相互沟通

员）聚集在一起，就关卡的地貌、标志性建筑物、关卡中的各种物品、敌人的特性等进行讨论，在白板或者纸上迅速地进行勾画。在集体讨论阶段，鼓励各种奇形怪状的想法和点子，所有的想法都可以提出。对这些想法，不马上做出取舍和判断，而是记录在案，留到下一阶段。

概念设计

在集体讨论后，关卡设计师得到很多好的想法和启发。他把那些想法进行初步的取舍和综合。概念设计即是把设计师头脑里的设想具体化、可视化，在纸上或者其他媒介上表达出来。如果关卡设计师自己就具有很好的速写能力，他可以自己动手。如果设计师本身没有美术技能，他需要和美工紧密合作，互相交流，共同把设计师头脑中的想法描绘出来。

这阶段关卡设计师和美工可以使用概念速写、二维平面图、关键地段的不同角度的速写、整体效果渲染图等来完成可视化（visualization）。下面是几个例子。

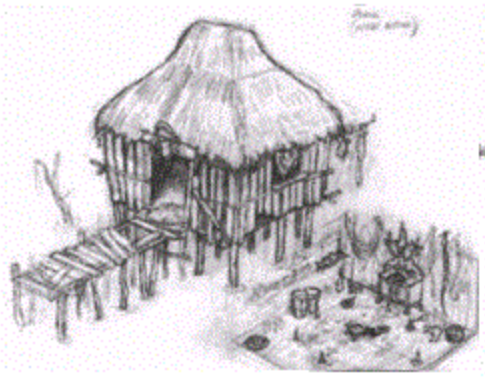


图 7-5《Diablo Ⅱ》的概念速写

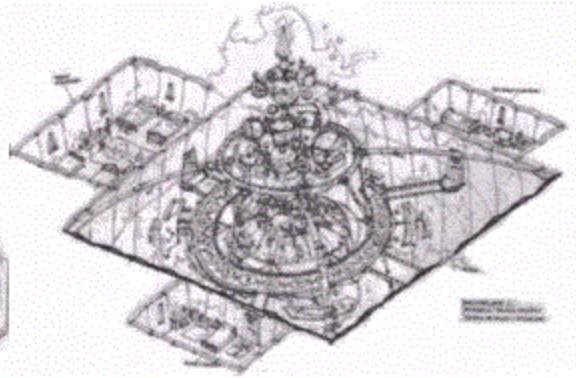


图 7-6 《柏德之门Ⅱ》的概念速写

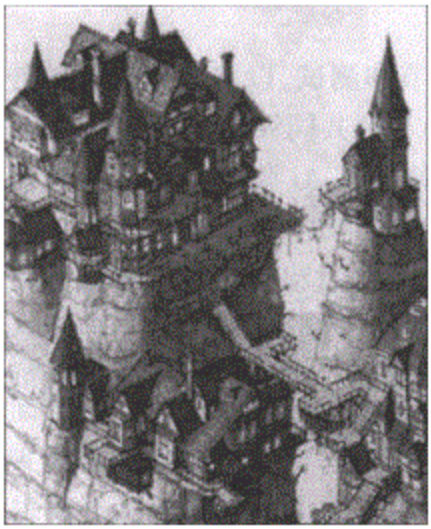


图 7-7《柏德之门Ⅱ》的粗略渲染图

图 7-8 《Jack & Daxter》的渲染图。更细致的渲染图，有助于三维（负责构建模型）及二维美工（负责绘制材质）把握整体感觉，使得最终成果和初始设计协调一致

概念评估

在各种概念速写完成后。整个小组可以进行初步的评估。全体组员坐在一起，利用各

种图片，在关卡设计师的讲解下，把关卡整个过一遍，看看其整体感觉对不对，发现一些明显的问题和疏漏。

使用关卡编辑器

经过反复几次概念设计和概念评估后，关卡设计师可以开始在计算机里使用关卡编辑器构建关卡了。一般来说每个公司都有自己的美工制作流程。取决于制作流程的规定，关卡设计师和三维美工（制作三维模型）和二维美工（绘制材质）必须搞好协调，前后衔接，流水作业。

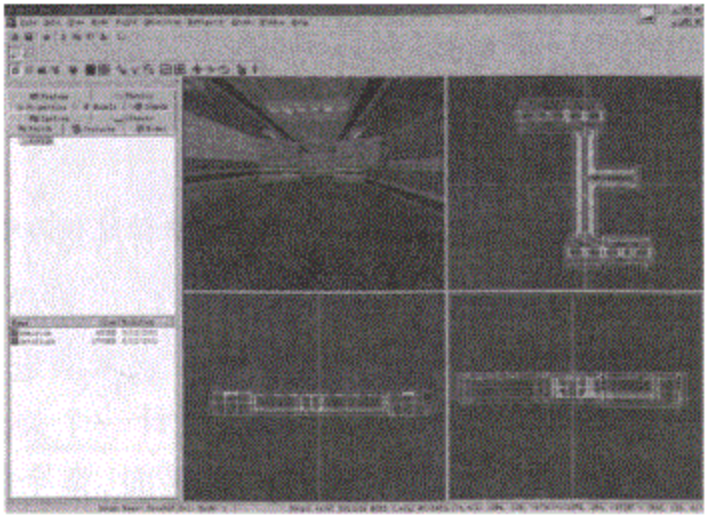


图 7-9 Lithtech 引擎的关卡编辑器 DEdit

测试

关卡设计出来后，必须经过不断的调节和测试，以求达到最好的效果。三维关卡基本成形后可以进行多人参与的共同评估(walkthrough)，将关卡浏览一遍，看看基本感觉。关卡、怪物和其 AI 脚本(script)集成后更可以进行更复杂的可玩性测试(playtest)。

通过上面的介绍，我们可以看出整个设计过程可以用一种螺旋线示意图来表示。每次循环由设计→细化→评估→测试组成。前一阶段的结果作为下一阶段修正的指南。经过多次循环，不断细化，不断发现问题并修正优化，这个设计不断充实和提高，最终达到最优。

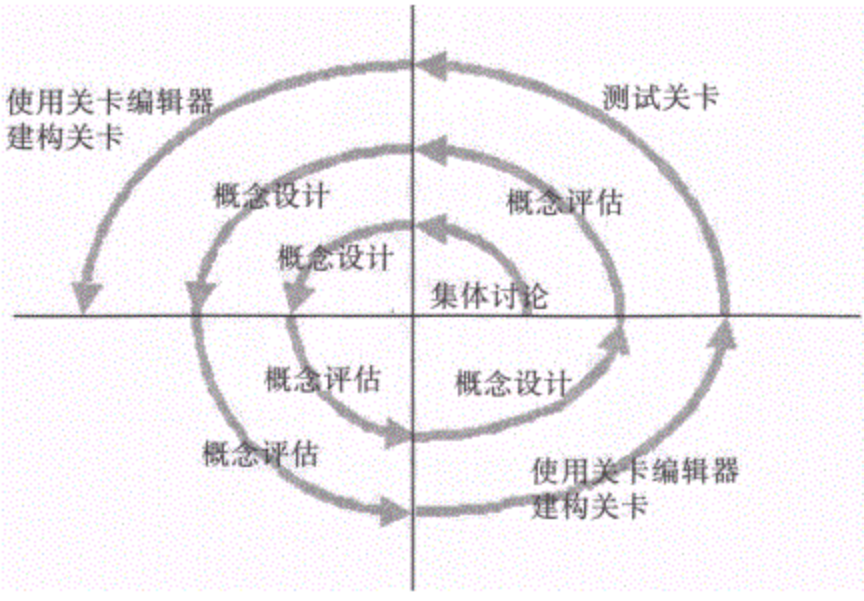


图 7-10 关卡设计过程的螺旋线示意图

这个螺旋线的示意图，将在以后的章节中不断出现。实际上，人类所有的设计行为，都是以这种方式来演进的。从小的方面来看，设计一个小小的图标是以这种方式，设计一个关卡也是用这种方式。而从大的方面看，设计整个游戏的人机界面，整个软件项目的组织，也都可以用这种方式。

第八章　RTS 游戏的平衡性

对 RTS 游戏来说，其游戏性中最重要的因素就是平衡性。所谓平衡性，就是使游戏中的各方的经济生产能力、军事生产能力、军事调动能力和战斗力，达到某种程度的动态平衡，避免出现一边倒的情况。因为如果能力不平衡，就会出现以下几种情况：或者玩家都去选择一两个好用的种族，其他种族根本不会有人使用；或者在玩的过程中一方享有压倒性的优势，使得游戏很快结束。这些都会影响 RTS 游戏的游戏性，特别是影响到网络对战效果和可重复游戏性。

平衡性又有广义和狭义之分。广义的平衡性，包括不同文明/种族之间的平衡性，这种平衡性主要包括了经济资源采集生产能力。狭义的平衡性，则主要是指不同武器系统之间的平衡性。本章主要讨论的是，狭义的平衡性。

系统论和谋略论

在具体讨论 RTS 游戏中武器系统平衡性问题之前，我们先换一个角度来看这个问题。任何类型的游戏都不是自己从石头里蹦出来的，游戏中所采用的各种思想或多或少地是从其他领域发展而来的，有其一定的传承性。RTS 游戏的武器系统平衡性也是这样，它本身是西方军事思想的产物。

大家在读历史书籍的时候，是否有这样的感觉：在史书上记载的中国古代战争中，强调智力游戏，谋略占很大篇幅，加上偶然因素太多，比如风向的改变就可决定一场战争的胜负（南唐之战），因此常令读者们觉得战争的规律无法捉摸，胜负难于理解，太过玄虚。读西方历史书的时候，则比较简单。战争中谁胜谁负是实力和客观因素的合理发展，比较令人容易理解。产生这种反差的一个原因就是在于东方讲究谋略和运气，务虚；而西方讲究实力，务实。务实的西医的原理要比务虚的中医的原理容易理解得多，对战争也是这样。

从历史角度看，西方对谋略讲究得不多。欧洲中世纪的骑士们只会正面冲锋，完全以实力决定胜负（西方所讲的公平竞争，即鲁迅先生评论过的“费厄泼赖”，就是从那时候发展起来的）。西方人不喜欢那些运气一类的东西，他们喜欢研究那些看得见摸得着的东西。贯穿整个西方战争历史的思想就是去研究那些不是由运气决定的东西，也就是技术/武器系统，研究不同系统之间的制约关系，改善自己的系统。当整个系统的优势达到一定程度后，谋略的作用就被降低了，系统受偶然因素的影响就降低了。可以说，西方采用的是很笨的方法，但这种很笨的方法，和他们在科学中采用的很笨的方法（比如动植物分类法，西医的头疼医头、脚疼医脚）一样，使得他们脚踏实地地走到了世界的前列。

我们可以把西方的体系称为系统论，而东方的体系称为谋略论。当使用计算机来表现战

争时，简单而实在的系统论，比较容易用计算机技术来模拟。它的数学模型比较简单，容易用软件来实现，而高超智慧的谋略论，无论是硬件和软件（算法），要用目前的技术来模拟都是太困难了。想构建令人信服的敌我双方主帅的思维模型简直是不可能的，而用软件模拟诸多偶然因素对战争的影响，也是难度重重。正因为如此，东方的游戏设计师们在近 10 年内并没有在提高战略游戏智能方面做出什么突破，而西方的系统论却在 RTS 游戏中得到了大发展，最终使得 RTS 成为游戏市场的主导之一。

战争史学家眼中的武器系统论

我们先了解一下在西方战争史学家眼中的系统，然后再介绍系统论在游戏设计中的应用。

图 8-1 Archer Jones 所著《西方战争艺术》，是美国 RTS 游戏设计师的“圣经”

Archer Jones 所著《西方战争艺术》(The ArtOf War in the Western World)一书，是西方武器系统论的代表作。其中阐述的思想，被美国的 RTS 游戏设计师们奉为金科玉律。Archer Jones 是美国陆军指挥及参谋学院的资深教授和战争史学家。此书不同于一般历史书之处，在于其忽略了士气、政治等随机因素和干扰因素，只把注意力集中到武器系统效能上。也就是说，完全没有偶然因素，战争的胜负是实力决定的。实力体现在武器系统的效能和相生相克关系上。

此书的另一个特点是以一种发展的观点对武器系统的演变做了阐述。介绍了随着技术的进步和革命，旧的武器系统如何被新的武器系统代替，新的武器系统又是如何影响和其他旧有武器系统的关系的。

在书中，Archer Jones 使用了类似于图 8-2 所表示的不同武器系统之间相生相克、相互制约的关系。这个示意图后来被游戏业所广泛采用。

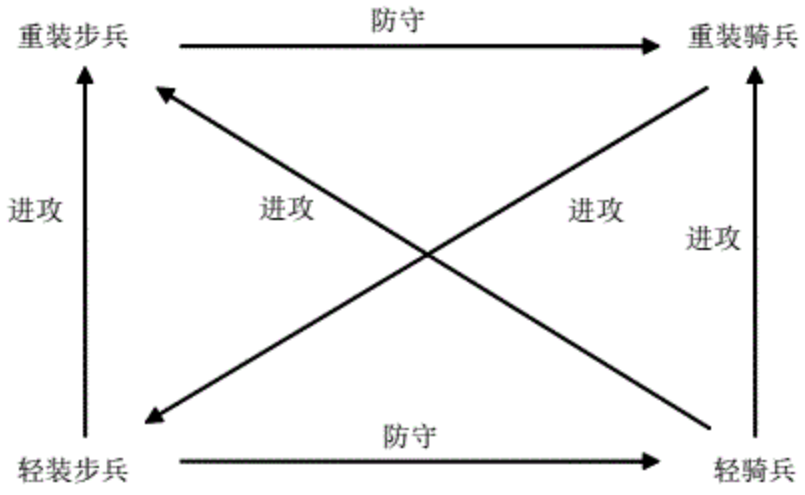


图 8-2 中世纪的各种武器系统战术效能比较图

对图 8-2，需要进行详尽的解释。它所表示的是中世纪欧洲的四种武器系统之间的关系，中世纪的战争中广泛使用了四种武器系统：重装骑兵、轻骑兵、重装步兵、轻装步兵。它们位于该图的四个角。

各武器系统的基本情况如下：

●重装骑兵：装甲厚重，武器威力大，以正面突击为主要任务。中世纪　的欧洲对重装骑兵情有独钟。尤其是法国骑士，在欧洲是赫赫有名。由于武器、铠甲和马匹的昂贵的装备和维护费用，重装骑兵是高贵的骑士们的特权，成为身份和地位的代表。

●重装步兵：训练有素，形成密集队形，有装甲的步兵。原来是希腊斯巴达和罗马军团的主力。但在中世纪衰落了，欧洲十字军中基本没有重装步兵。

●轻装步兵：在欧洲是由下层人民中征招而来，大多数没有装甲或者装甲很少，训练不足，以弓弩为主要武器，负责搭建营地、防守辎重等辅助任务。

●轻骑兵：在欧洲没有轻骑兵。轻骑兵是阿拉伯人的主要武器系统。装甲很少或者几乎没有，速度快，以弓箭为主要武器。

当两种武器系统对抗时，一定是有一方进攻，有一方防守，优劣体现在进攻一方是否能够瓦解防守一方的队形，或者防守一方是否能够承受进攻方的冲击。图中用带箭头的线段来表示这种关系，线段末端的箭头指向劣势一方，线段上的文字表示了箭头开始端的武器系统在对抗中的是进攻还是防守一方。以图 8-2 中轻骑兵和重装骑兵之间的线段为例，线段由轻骑兵指向重装骑兵，线段上是进攻，就是说当轻骑兵和重装骑兵对抗时，当轻骑兵是进攻一方时，享有系统优势。

图 8-2 中所有的箭头（即武器系统优劣关系）都是根据大量的历史资料来加以归纳判定的。比如说轻骑兵和重装骑兵之间，主要依据了十字军东征时的诸多战例。十字军东征是欧洲重装骑兵系统和阿拉伯轻骑兵系统的一次大碰撞。通过研究几次十字军东征时的战役，我们可以清楚地看到双方是如何斗法的，双方是如何逐渐熟悉敌我系统的效能和局限性，如何改进系统，力求获得系统最大优势的。

下面就以书中的几个战例说明一下：

当十字军最初向耶路撒冷进军时，欧洲的骑士们对他们的重装骑兵极为自信。他们对阿拉伯人的轻骑兵战术了解得不多，认为自己甲厚剑利，正面冲击威力很大，足够冲垮所有的抵抗。第一次使他们吃惊的遭遇战发生在 1097 年，当十字军由康斯坦丁堡进入了土耳其苏丹的领地时，他们遭遇了土耳其人。当时十字军成两列行进，之间相距 6~7 英里。当土耳其

人出现在左边一列的附近。左边的十字军立刻安营，分出步兵防守营地，骑士们开始集合，列成队形准备冲锋。可是土耳其人并不靠近十字军，而是保持一定距离并发射出一阵阵箭雨。十字军骑士厚重的甲胄给人员提供了有效的保护，但没有装甲的马匹损失惨重。由于土耳其人的队形松散，没有一个集中的正面可以让骑士们冲锋。小队骑士在忍无可忍的情况下进行了徒劳的冲锋，土耳其人迅速地拉开距离，同时射出箭，使得这些冲锋的小分队损失更重。这样对抗了几个小时后，十字军的重装骑兵开始后撤，并陷入了混乱。看来他们马上就要大难临头了。就在这时，右边一列十字军在接到了信使的急报后，出现在土耳其人的侧后，并对土耳其人的侧翼和后方进行了冲击。已经陷入绝境的十字军看到这种情况下，士气又鼓舞起来，进行了反冲击。这样的包围使得土耳其人的轻骑兵失去了机动的余地，他们没有装甲，武器威力弱，无法去和重装骑兵对冲。最终土耳其人狼狈地逃出了战场。

从上面的战例可以看出。当轻骑兵以松散的队形在重装骑兵周围用弓箭进行骚扰时，虽然重装骑兵的装甲可以有效地保护人员，但马匹却没有足够的保护。如果重装骑兵要反击的话，由于轻骑兵所惯长的松散队形，使得重装骑兵无法找到一个正面进行冲锋。加上轻骑兵的装甲少，速度快，可以很快逃逸，重骑兵也赶不上轻骑兵。因此，当轻骑兵使用弓箭攻击重装骑兵时，享有很大的优势。

侥幸获胜的十字军骑士们，在见识了阿拉伯轻骑兵的威力后，马上开始思考如何对抗这种武器系统。他们从这次的战役中看到，轻骑兵的优势在于其机动性和弓箭的远距离杀伤性，但其装甲薄弱，辅助武器（刀）威力小。在一定的地形环境下，当轻骑兵的机动性受到限制时，如果重装骑兵能够把握住机会去冲击轻骑兵的话，轻骑兵是完全没有招架能力的。

十字军在叙利亚成功地执行了这种战术。当时十字军的 700 名重装骑兵面对 10000 左右穆斯林轻骑兵。十字军选择了一个对穆斯林骑兵来说是致命的地形——穆斯林骑兵面对的是一条一英里长的两边是湖泊和河流的狭长走廊。当穆斯林骑兵通过这条走廊向十字军的营地进攻时，十字军的重装骑兵突然发起了冲锋。人数众多的穆斯林骑兵在狭长的走廊中相互拥挤，完全失去了机动力，没有装甲的轻骑兵面对十字军的重装骑兵，毫无防护能力，整个战斗有如一场屠杀。最后战场上留下了 2000 具穆斯林骑兵的尸体。

赢得了战斗的十字军，并没有满足。他们知道不能把胜利的希望寄托在地形上——如果在今后的战斗中找不到合适的地形呢？西方的军事家们从不心存侥幸，从不寄希望于偶然因素（地形在这里成了偶然因素）。既然重装骑兵面对轻骑兵处于劣势，也就是说在两种武器系统的对抗中我方的系统处于了下风，那就必须改进我方的系统，或者引入新的系统，使得新的系统无论在什么地形环境下都能有效地制约敌方的系统。

十字军的首领们在一些小规模的接触战中重新发现了轻步兵这个武器系统的重要作用。十字军中的轻步兵主要由下等阶级组成，使用弓和弩为主要武器，无装甲，训练不足，被由骑士阶层构成的重骑兵所轻视。但这些貌似乌合之众的衣衫褴褛的轻步兵，却能够有效对抗

轻骑兵。其道理很简单：轻骑兵在马上射箭，需要一心二用，既要控马，又要射箭，必然影响射箭的准确度和射程。轻步兵站在地上，不需要分心，射箭力度大，射程远，准确度高。因此，在轻骑兵和轻步兵的对射中，轻骑兵占不到便宜。在图 8-2 中，由轻步兵指向轻骑兵的线段，上面写着防守，就代表了轻步兵可以有效对付轻骑兵的骚扰战术。

发现了这一点后，当十字军面对轻骑兵的骚扰时，就把轻步兵在阵前展开，通过对射来驱散轻骑兵。

在阿拉伯人一方，在发现自己惯用的轻骑兵骚扰战术失效了后，他们也思考如何去抵消轻步兵在弓箭对射中的优势。既然轻骑兵的远距离武器对轻步兵失效了，轻骑兵只能用近距离武器冲击了——马刀。土耳其人尝试让骑兵使用马刀对列于阵前的十字军轻步兵进行冲击。这样轻骑兵就不再是远距离武器系统了，而是向重骑兵看齐，成为冲击系统了。

十字军马上发现，以弓箭为主要武器的轻步兵，无论是武器还是心理，都无法承受骑兵的直接冲击。他们同时发现，有些骑士因为在以前的战斗中马被射死射伤，只能被编入步兵序列。有骑士加入的步兵群一般都能比较好地承受住轻骑兵的冲击。其原因是当重装骑兵下马排成队列后，重装骑兵这种武器系统就转变成了重装步兵。重装步兵甲胄厚重，训练有素，心理承受力高。特别是重装步兵排成有纵深的方阵时，甲胄重重，枪林层层，几乎可以承受一切骑兵的正面冲击。

十字军在以后的战役中使用了这种武器系统转换和混合兵种的概念。他们的做法是在阵前布置轻装步兵，轻装步兵后面是由一部分下马的骑士组成的重装步兵，形成正面和纵深。在步兵序列的中间空出几条通道，供位于阵后的重装骑兵冲锋时候用。这样多种武器系统合成，威力更大，可攻可守。欧洲的十字军在和阿拉伯人的对抗中取得了优势，但武器系统之间的斗争并没有停止：埃及人的重装骑兵又出现了……

从上面的简单介绍中，我们对《西方战争艺术》一书中的主要思想方法有了基本了解。之所以这本书为美国众多的 RTS 游戏设计者所阅读并研究，就在于它把诸多偶然因素，包括谋略、士气、天气、地形，都尽可能地忽略或者减低其作用，而着重研究抽象的武器系统的效能。这种方法正好适应了计算机软件的局限性。游戏设计师在早期的尝试中，发现用计算机软件模拟那些偶然因素是十分困难的，而设定不同的武器系统以及它们之间的关系则容易得多，同样可以产生无穷的变化。其数学模型简单得多，而且可以构建成数据表驱动，实时性能好。由于以上的原因，这本书成了 RTS 游戏设计的“圣经”，其思路被美国的 RTS 游戏所贯彻。

当然，RTS 游戏中要应用这种思路，和历史上军事家们研究武器系统的思路有根本不同。在历史上军事学家研究武器系统制约关系的目的是为了使得自己的系统或系统组合能够在任何情况下都能获得压倒性的优势，而游戏设计师研究 RTS 游戏中武器系统平衡性的目的

是使得任何系统都不能对所有系统有压倒性优势。

游戏中的武器系统平衡性

RTS 游戏中武器系统平衡性有两个基础。第一个是前面介绍的武器系统论，即有专业分工明确（也是一种简化）并且相互制约的武器系统。而第二个基础，就是大家所熟悉的儿童游戏——石头、剪子、布。石头胜剪子、剪子胜布、布胜石头，这样构成一个死循环。RTS 游戏设计师的目的，是把游戏设计成滴水不漏、环环相扣的解不开的连环套，一物降一物，反反复复。你在这方面强，他在这方面强；A 是 B 的克星，B 是 C 的克星，C 又是 A 的克星。

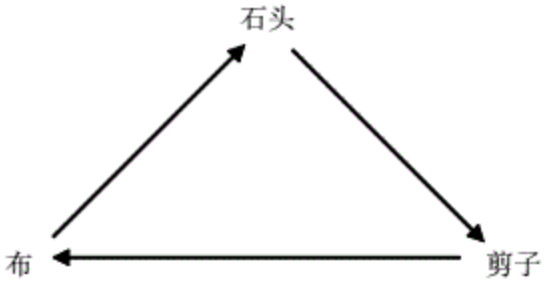


图 8-3 石头、剪子、布游戏中，三个竞争单元之间形成了死循环

游戏设计师一般从系统设定开始。首先要确定武器系统的种类。武器系统分为一般系统和专门系统。一般系统是指攻防兼备，十八般武艺样样都成，但样样都不是很突出的系统。专门系统，是指自身不平衡，某些方面很强，某些方面很弱的系统。

在基本的武器系统的种类确定下来后。需要考虑系统属性，比如说攻击威力、防守强度、行动速度等等。有了系统属性后，又需要数学公式来表达并预测两种武器系统之间的战斗及其结果。数学公式中一般包含时效和空间等变量（弓箭手总能最先从远距离给对手造成损伤，步兵需要接近才能给对手造成损伤，等等）。

这些数据和公式有了后，制定一个大数据表格把初始数值填进去。这些初始数值只是凭经验的估计值，肯定要经过不断修改。这个数据表格是调节平衡性的基础。

现在什么东西都是在纸上，无法验证数据是否合适。这时需要很快地编制游戏，尽可能早地搞出可运行的试玩版来，然后就是开始不厌其烦地测试和调节数据了。

	Archer1	Infantry1	Cavalry1	AntiArcher	AntiInfantry	AntiCavalry
Archer1	1.00	1.33	0.72	0.18	2.02	1.34
Infantry1	0.75	1.00	1.61	1.37	0.17	2.18
Cavalry1	1.39	0.62	1.00	1.87	1.24	0.16
AntiArcher1	5.55	0.73	0.53	1.00	0.98	0.98
AntiInfantry1	0.50	6.00	0.81	1.03	1.00	1.09
AntiCavalry1	0.75	0.46	6.34	1.02	0.92	1.00

表 8-1《帝国时代》的各种武器系统的效能比数据表

对武器系统来说，战斗威力并不是惟一的属性，其他因素很多，比如说建造成本、建造

速度、技术升级需求，等等。这些因素交织在一起，要用一个数学公式来计算效能比是不太实际的。而我们在前面几章介绍过游戏设计中的一个最基本的思路就是无法用理论指导的（数学公式）方面，用实践加反馈来调节（手工测试）。

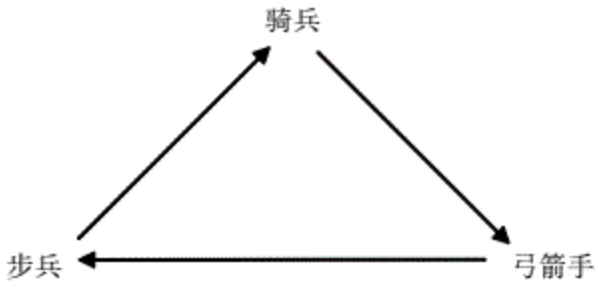


图 8-4 表 8-1 中的数据显示，弓箭手、步兵、骑兵三者之间也形成了类似石头、剪子、布的死循环关系

通过手工测试调节游戏中的各项参数是一项艰巨的工作，看似重复劳动，没有任何光彩耀人之处。但对游戏的成败，却起着重要的作用。可以说，一个杰出的游戏和一个一般的游戏，有可能基本系统差不多，优劣差别就在细微之处（也就是感觉）。这些细微之处若差之毫厘，则效果有天壤之别。细微之处的差别和优劣是如何出来的呢?就是杰出的游戏用了近一年左右的时间来调节参数，而一般的游戏只用了 4 个月。

为了帮助游戏设计师和测试人员摆脱繁重的手工作业，不少游戏项目都开发了辅助工具。《帝国时代》开发时采用了作战对比模拟器（Combat Comparison Simulator）。这样，设计师可以在作战对比模拟器的输入屏幕上选择两个武器系统，并选择双立数量（1 对 1，5 对 5，30 对 30），然后开始模拟战斗。模拟器即会在画面上显示出战斗的结果。设计师也可以通过两个系统的模拟战斗，找到平衡点。

第九章　人机界面设计

所谓界面，又称人机界面（human-computer interface）、用户界面（user interface），指的是一个计算机软硬件系统中用户看得见摸得着的部分。与之形成对比的，是软硬件的内部结构和运行机制，那些是用户看不见的，也不感兴趣的东西。用户使用一个计算机系统，每天是和这个系统的界面打交道。界面包括软件部分，如屏幕上的图像、文字、图标、窗口等，也包括硬件部分，如键盘、鼠标、手柄等。界面设计，对游戏机硬件厂商们来说，主要是设计游戏机的输入/输出硬件，特别是游戏机的手柄；对游戏软件公司来说，由于游戏机和 PC 的硬件是生产厂商定好的，除了很特殊的游戏（如某些钓鱼游戏专用的鱼杆），一般不用自己设计硬件部分，因此界面设计主要侧重在软件方面，即在系统现有的硬件环境下，去设计软件的外观和使用规则。人机界面的发展历史

人机界面的发展历史

谈到人机界面，就不能不谈 Windows 和苹果机。我们现在使用的界面，以 Windows 系列和 MAC OS 系列为代表，基本上都是图形用户界面（GUI）的一种，俗称 WIMP 界面。WIMP 是窗口、图标、菜单、鼠标的缩写（window-icon-menu-pointing device）。这四项现在看起来平淡无奇，但在 30 年前，所有人都在使用分时系统和字符界面时，它们的出现可是具有划时代意义的。但这个 WIMP 的概念既不是出于苹果公司，更不是来自微软，而是在施乐公司研究中心（XEROX PARC）诞生的。说到施乐公司研究中心，真可谓大名鼎鼎。目前 PC 上几乎所有的重要技术，如图形用户界面、鼠标、局域网、桌面排版（DTP）、客户/服务器、激光打印等都起源于那里。现在美国计算机界还有这么一种调侃的说法“我们不需要创新！光 XEROX PARC 的研究成果我们还没有完全消化呢！”其影响力可见一斑。关于这个研究中心的很多故事已经成为了计算机界故老相传的传奇。像苹果公司创始人 Steve Jobs 是如何从那里“偷”到 GUI 和鼠标的设计思想，然后开发出轰动一时的 Macintosh 的故事等。这个研究中心起源于 20 世纪 70 年代施乐公司的老板在一次会议上的吹牛，他当时对公司股东们谈到信息时代的时候，随心所欲地说了一句“我们施乐公司将会研究和发展信息时代的结构框架！”会议结束后，他找来他的助手说：“见鬼！我今天说了一些连自己也不

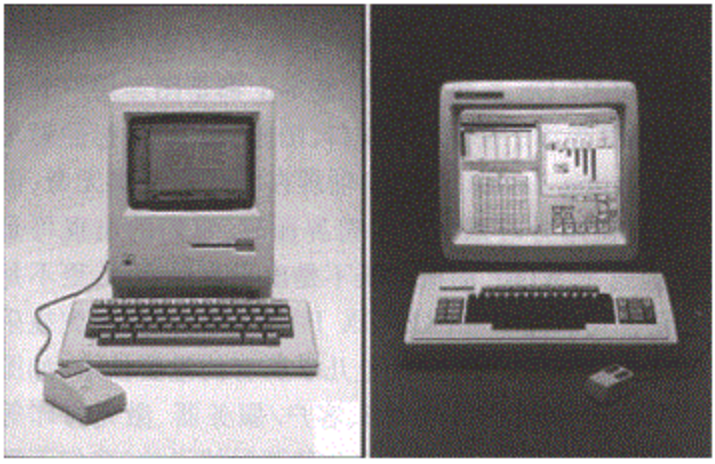


图 9-1 左边的是苹果公司 1984 年生产的 Machintosh 机，右边的是施乐公司 1981 年推出的 STAR 8010 机。从纯技术角度看，STAR 8010 强过 Machintosh 百倍，但耐人寻味的是 Machintosh 最终取得了商业上的成功

明白的话。我们还是建立一个研究中心，去搞明白我究竟说了些什么吧！”于是在美国加州 Palo Alto 这个地方设立了施乐研究中心。人们习惯性地将这个中心称为 XEROX PARC。由此开展了对未来信息时代所需要的各种技术的研究。尤其是在 PC 技术方面，其研究的深度、广度都是任何其他研究机构所无法比拟的。但令人不解的是，XEROX PARC 虽然创造出这么多高瞻远瞩的研究成果，但它本身没有从任何一项成果的商品化中获利。所有这些研究成果在商业上的成功，都是其创始人在离开 XEROXPARC 自立门户后发生的。由 XEROX PARC 的员工发展出来的著名公司，有网络公司 3COM、图像软件公司 Adobe，甚至 Microsoft Word 的起源，都可以追溯到 XEROX PARC。

XEROX PARC 的研究人员在人机界面领域所做的工作，在当时来说是极为前卫的。在字符式显示器和键盘一统天下之时，他们就看到了光栅式显示器和鼠标器的巨大潜力，并提出桌面（desktop）的概念，并在此基础上搞出了 WYSIWYG（What You See is What You Get，所见即所有），多窗口系统，下拉式菜单（pull-down menu）等许许多多新概念新技术。他们是图形界面（GUI）的创始者和倡导者。可以说，我们目前所使用的计算机系统，在人机界面方面，还没有任何地方能超过他们 30 年前的想象力！

在 XEROX PARC 做出划时代的研究之后，先是苹果，后是微软，应用了他们的研究成果，在商业上获得了巨大成功。极具讽刺意味的是，在微软的 Windows 操作系统获得巨大的市场份额并日益威胁苹果公司的领先地位时，苹果公司将微软告上了法庭，起诉微软剽窃了苹果公司的软件的人机界面设计思想。计算机界的人们都忍俊不已，暗笑道：“这不是小偷告小偷吗？你苹果公司的设计思想不也是从 XEROX PARC 弄来的吗？”后来果然一波未平，一波又起。施乐公司也以相同的理由将苹果公司告上法庭，形成了所谓的连环诉讼。

图 9-2 从这张照片我们可以清楚地看到 STAR 8010 的人机界面已经是相当高级了，包括了现在常见的图标、窗口、WYSIWYG 的文档显示等诸多特征。请记住这是在 1981 年，距离 Windows3.0 问世还有 10 年左右时间

在 WIMP 界面之后，目前研究的主要方向是非指令性界面（noncommand interfaces）。其基本思想是把人机界面从计算机屏幕、鼠标器和键盘的束缚中解放出来，利用人体自身及其周围环境，使界面成为用户自身的延展，并且融入用户的周围环境中，无所不在但又无形无踪。比如说一个大学教授的办公室可以是这样的：把屏幕投影到整个一堵墙上，然后在墙上安些小摄像机头用来捕捉用户的手势和脸部表情及眼睛所视方向。用户舒服地坐在沙发里，用手一指，墙上一个文件被选中了，然后一招手，文件就被打开并放大了。看完后反向推手，文件关闭缩小成图标。目前正在研究的多种技术，如语音输入、智能代理技术（interface agents）、多通道交互技术（multimodal）、眼球跟踪（eye tracking）、

姿势控制（gesture control）、三维虚拟现实系统等，都是向着这一方面的努力。

图 9-3 使任天堂初尝失败苦果的 Virtual Boy。其输出设备是一个全封闭式三维 VR 眼镜，即使是现在看来也是够前卫的了

对游戏业来说，人机界面设计的硬件方面的工作，主要是设计输入和输出设备。游戏机的输出设备十几年来一直没有任何改变，都是使用普通电视机，因此也没有什么可设计的。仅有的例外是横井军平在 Virtual Boy 上尝试了封闭式三维眼镜，但却使任天堂头一次遭到惨败。原因是这套系统设计得太前卫了，超过了当时技术所能允许的程度。目前看来电视在很长的一段时间内还将是输出设备的主流。虽然有很多更酷的输出设备，比如说往眼睛里直接打激光之类的东西（美国华盛顿大学的一些变态们搞的。据说他们每次演示和实验前都要花很大力气去说服战战兢兢的被试验者们，并向他们保证这种方法是万无一失绝对安全的。据有胆子一试的人事后说图像质量极好！），但短期内不会很容易被大众所接受。

游戏机的输入设备，主要是手柄。这是游戏机设计人员们需要花费大量心血仔细研究的，如何设计外形使得长时间握柄不致疲劳，如何安排按键的位置，如何控制移动方向等等问题。笔者曾经在人机界面的专业学术杂志上看到过关于任天堂 8 位机手柄的论文。论文作者通过具体的试验定量分析了任天堂手柄的效能，讲得头头是道。手柄设计的重要性可见一斑。

图 9-4 左边的是曾经在 20 世纪 70 年代末、80 年代初风光一时的雅达利 2600 主机。请注意它的操纵杆，很有现在流行的模拟摇杆的味道。但当时的二维游戏显然不需要这么多控制自由度，因此这种设计被更简单的十字键方案替代了。右边的是任天堂 8 位机的手柄。请注意它的十字键的配置。十字键后来成了二维游戏时代的标准配置。进入三维时代后，由于操纵复杂性增加，被长期冷落的操纵杆又复活了，不过这次是改成真正三维的模拟摇杆了。这种“发明—搁置—新生”的现象在科技发展的进程中十分常见。很多优秀的设计，由于超越了当时的市场和技术水平，加之和其他周边产品不配套，因而被长期搁置，被比较保守但简单实用的设计所代替。但当技术发展市场成熟、相关周边产品改进后，原有的设计又会获得新生，并吸取最新的技术得到改善。（好像有人说过“历史总是螺旋式上升的”）

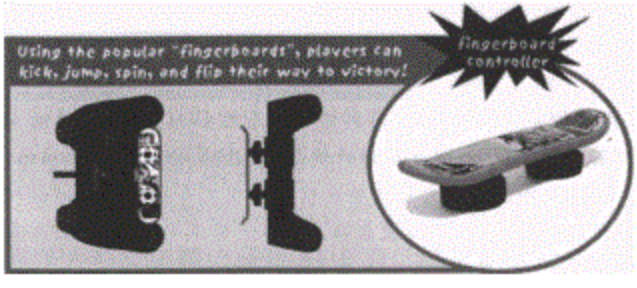


图 9-5 KOEI 为其滑板游戏所设计的附加硬件。使用时小滑板套在 PS2 的两个振动杆上。这个设计巧妙新奇，在游戏机旧有手柄的基础上进行了有限创新。通过改变输入设备，给游戏性带来了新的要素

游戏机游戏和 PC 游戏的人机界面

人们注意到游戏机游戏和 PC 游戏在人机界面的设计上有很大不同，这些差异主要是由于硬件的不同造成的。游戏机的输出设备使用电视机，无法显示高分辨率，显示文字也很困难。PC 游戏则没有手柄，一般要使用键盘鼠标来控制。我们还可注意到：由于输入/输出设备的不同，PC 游戏和游戏机游戏各自所擅长的游戏类型不同。精心设计的游戏手柄使游戏机游戏对 ACT 等类型得心应手，鼠标则使 PC 游戏在 RTS 方面占尽优势。

计算机屏幕的高分辨率使得 PC 游戏可以使用复杂的多窗口和菜单系统，其结果是 PC 游戏更多地使用了传统 GUI 界面的各种要素。我们可以称之为软界面。所谓软界面，是指不是用物理上实际存在的东西，如真实的控制面板上的按键 button，而是用屏幕上的东西，如 GUI 中的软按钮 button（在本章中，“软”button 翻译成按钮，“硬”button 翻译称按键），来构成界面，体现功能并接受输入。游戏机游戏的人机界面则可以称之为硬界面，因为其设计很大程度上依赖于手柄的物理外形，主要功能是围绕手柄上的按键来设计的。

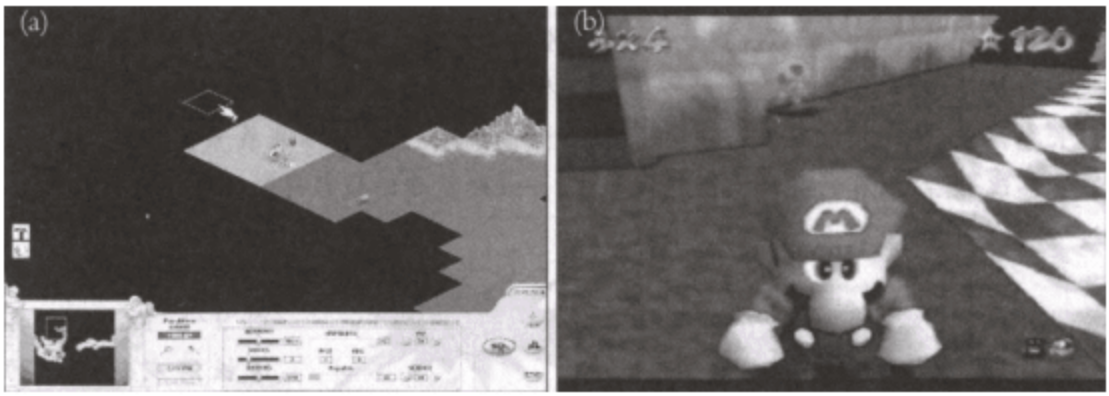


图 9-6(a)是一个 PC 游戏的界面，我们可以看到许多熟悉的 GUI 组件，如滑动条(slider)、按钮(button)、标签 (label) 等。这也是所谓的软界面。(b) 是《Mario64》的画面，其界面基本上没有使用 GUI 组件，人机交互主要是围绕游戏机手柄上的按键设计的，可以称之为硬界面。正是由于 PC 和游戏机在输入设备上的不同，使得它们各自擅长的游戏类型不同由于输入/输出设备是人机界面的物质基础，游戏机和 PC 之间输

由于输入/输出设备是人机界面的物质基础，游戏机和 PC 之间输入/输出设备的不同，决定了两者之间界面设计的不同，进而决定了游戏性的不同。长期以来困扰游戏业的问题就

图 9-7 游戏在不同平台之间的移植一直是困扰游戏业的大问题。移植作品很少能获得原作那样的成功。究其原因，输入/输出设备的差异是主要的障碍。这是《运输大亨》PS 版的画面，明显比 PC 版分辨率低。为了最大限度地保持原作的游戏性，《运输大亨》的 PS 版甚至随游戏附带了 PS 鼠标。即使这样，PS 版游戏性还是大打折扣人机界面的重要性

是如何把游戏从PC移植到游戏机上，移植的游戏在游戏机上一般很难获得原作那样的成功，而反之亦然。其实原因很简单：原作成功的原因是其游戏性，而游戏性又是和人机界面息息相关的，而人机界面的基础又是输入/输出设备。如果移植到一个完全不同的平台上，输入/输出设备的改变，必然导致人机界面设计的改变，最终必然导致游戏性的改变。也就是说移植作品实际上改变了原作赖以成功的游戏性，而新设计的游戏性是否能在新的平台上成功，当然没有100%的保障。从某种意义上说，移植一个游戏的难度要比设计一个新游戏复杂得多。因为设计一个新游戏，你可以从无到有，没有条条框框的限制。而移植一个游戏，特别是移植一个成功游戏到一个新的平台上，设计师既要考虑游戏原有的游戏性，如何最大程度地保持它，又得考虑新的平台的诸多限制和特点，这绝对是对设计师综合水平的一个挑战。

人机界面的重要性

人机界面的重要性很久以来都被游戏设计师们所忽视，它们把全部精力都贯注到图像和AI方面，结果是游戏开发出来后美则美矣，但操作复杂，上手困难，玩家不明白他在什么地方，也不清楚自己应该干什么。从理论上来说，人机界面的重要性在以下两方面：

人机界面决定了游戏性：人机界面是人机之间沟通的桥梁。在游戏理论一章中提到的游戏三层模型中，界面是中间一层。它起到连接玩家和游戏内核的作用。玩家在玩游戏时，所见所感的是游戏的界面，而非游戏的内核。只有通过人机界面，玩家才能够控制游戏的内核，因此对游戏来说，人机界面决定了游戏的游戏性的大部。

可以通过人机界面来定义并分析游戏性：长期以来，游戏业内关于游戏性有很多争论。其根本原因就是因为游戏性是个非常模糊的概念。如果问游戏设计师们什么是游戏性，他们会回答说：“当我坐下来玩游戏时，在我的手指触摸到手柄的一刹那间我就能感觉到游戏性，但你让我用语言非常规范地表述出来，这太难了！”游戏性成了只可意会不可言传的玄妙的东西了。我们从软件工程的基本理论中知道：如果你不能明确地定义并表达一个概念的话，那么你就根本不可能去改进它，更不要说什么使用规范化的方法了。比如说软件的质量，如果只是空谈而不去具体定义性能指标的话，那么一个项目完成了，谁也不知道这次的软件质量比上一个项目究竟是提高了还是下降了。因此，软件工程的根本思想就是定义一些概念，通过这些概念去反映软件本身和软件开发过程的质量，然后再定义一些针对这些概念的性能指标，然后再研究发展一系列的规范化的方法去评估并优化这些指标。对游戏性来说，如果无法对其确切定义的话，那么我们怎么能够改善它呢?如果还是用手工作坊的方法，我们怎么能够保证游戏软件的质量呢？如何把握住虚无缥缈的游戏性呢？也许人机界面能够给我们提供解决这些棘手问题的可能。我们知道：游戏性是虚的，但人机界面是实的。既然人机界面在一定程度上决定并反映了游戏性，那么通过分析人机界面，通过分析人机界面的动态交互性，我们也许可以找到定义并分析游戏性的途径，也许能够找到某些规范化的方法去改善游戏性。在目前人机交互学的研究中，有一些理论和方法是用来评测一般商用软件的可用性的。如果对这些方法加以改进，使它们适用于游戏软件的游戏性的分析，则有可能极大地

改善游戏开发水平，使其有章法可循。在前面游戏性一章中，对这种思路曾经有所阐述。

WIHP 类型人机交互示意图

用户和计算机软件之间交互的一个简单过程如图 9-8。

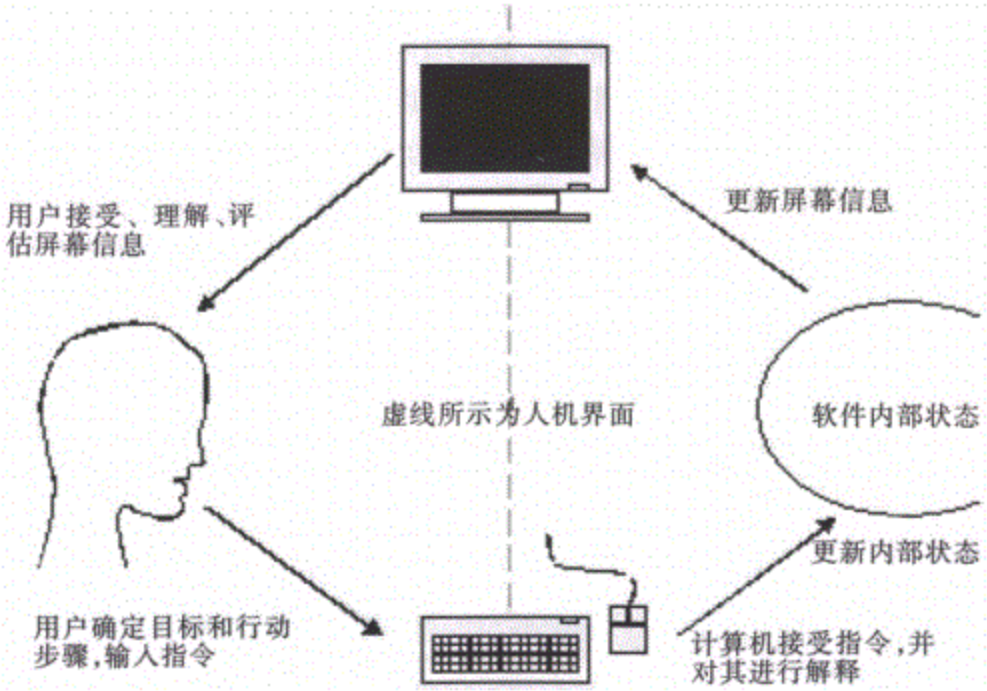


图 9-8 Don Norman 的人机交互环

这个图一般被称为交互环（interactive cycle），是由 Don Norman 提出的很有名的人机交互/人机界面的普适模型。所有 WIMP 类型的界面都是基于这个模型的。一个基本的循环如下：软件程序通过输出设备显示有关信息，信息所表达的是系统当前的状态，用户接受/理解/评估这些信息后，根据自己的需要，决定是否要改变系统的状态，如果是的话，则进一步确定总的目标(目的状态)和具体的实行步骤，然后通过输入设备输入指令，软件系统接受/解释/评估这些指令，再后改变自身的内部状态，最后通知输出设备更新显示信息。至此一个循环完成，用户看到了更新的信息，知道系统状态已经改变。如果需要，他可以再启动另一次循环。

人机交互，或者说人们每天使用计算机，就是通过无数次这样的循环往复。我们更可以看出，对这个交互环的成功运行起决定性作用的，一是在循环开始的阶段，输出信息必须能被用户准确无误地理解，因为这些信息正是用户决策的依据，如果用户对屏幕上的信息理解错误，他将做出不正确的决策，从而导致循环中以后的步骤完全错误；二是在用户正确理解了输出信息并有了明确目的之后，他必须选择正确的输入指令去达成目的，如果他不能选择正确的输入指令，同样也会导致循环中以后的步骤完全错误并引起误解。因此，人机界面设计的问题，归根结底就是两个问题：如何将系统信息明白无误地显示给用户，如何帮助用户

去选择正确的输入指令。

WIMP 类型人机界面设计的两个主要任务

数据可视化

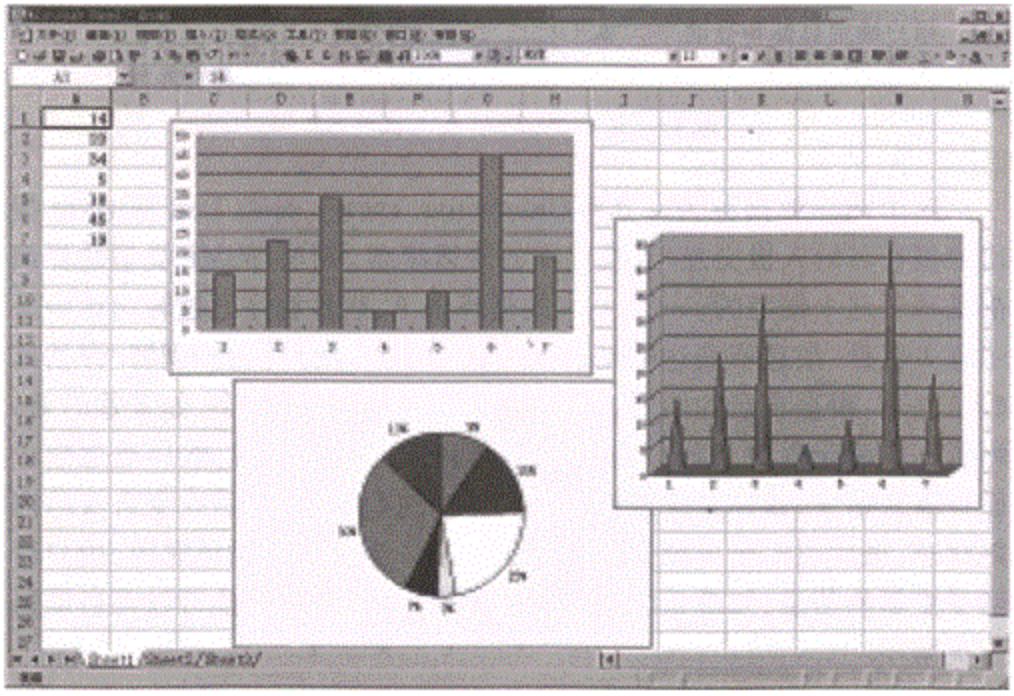


图 9-9 使用 EXCEL 对同一组数据采用不同的图表显示

人机界面设计的第一个问题，即如何将系统信息明白无误地显示给用户，实际就是数据信息的可视化问题（data visualization）。这个问题用大白话说就是：如果我们手里有一堆复杂的数据，如何对它们进行组织分类、编排取舍、并利用图形和图表，使得用户能够很轻松地从视觉上接受这些信息，并理解这些信息。一个最简单的例子就是数据报表。正如孔乙己说回字有数种写法一样，如何显示数据报表也有多种方法，比如用文字显示，或者用条状图饼状图等等。图 9-9 是使用微软的 EXCEL 做的基于同一组数据的三种图表。

一个更高级一点的例子如图 9-10 所示。

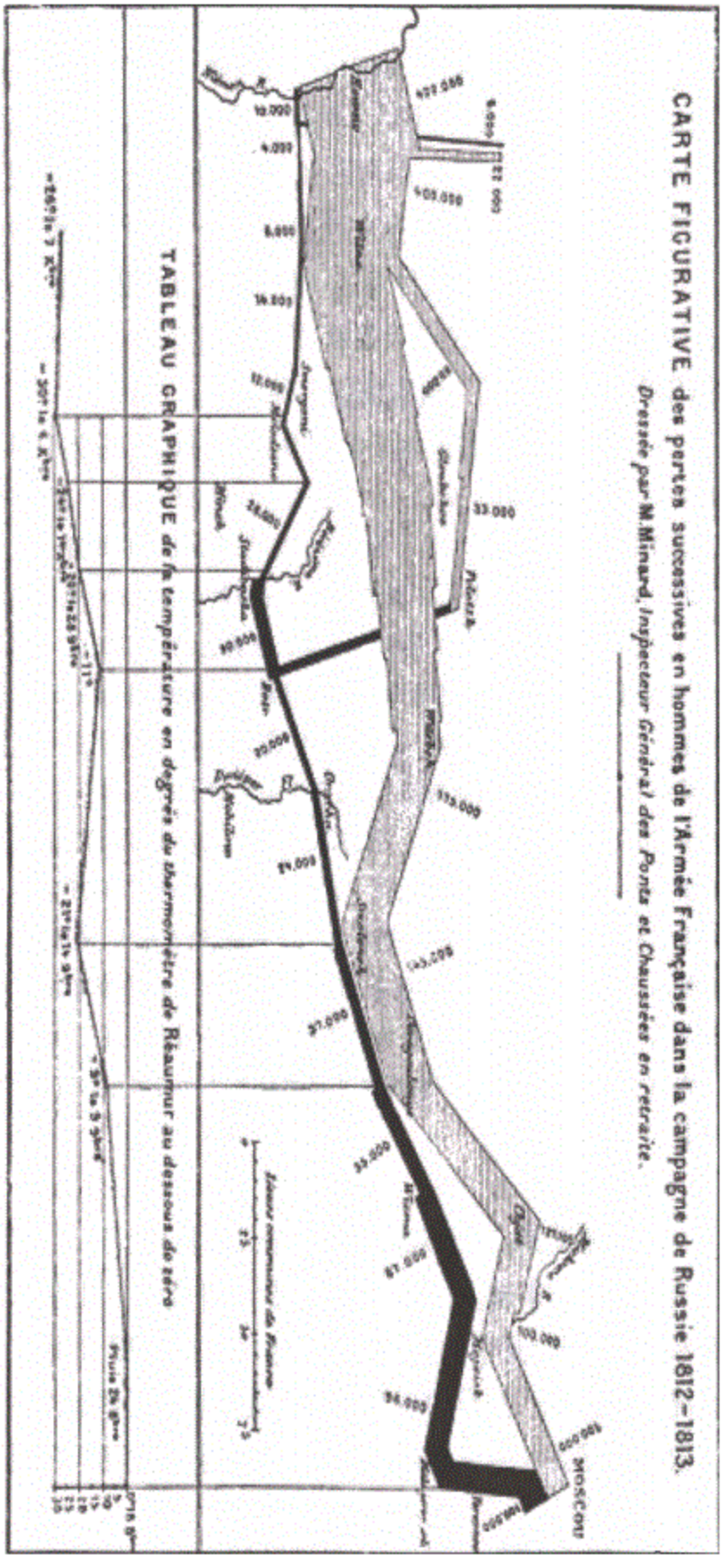


图 9-10 Charles Joseph Minard（1781-1870）所画的 1812 年拿破仑大军进攻俄国和撤退的示意图。图中上半部分是根据实际地图绘制的路线图，上方的灰色粗线是法军进军路线，下方黑色细线是撤退路线。图的最左边是俄波边界，法军由此进入俄国境内，我们可以看到法军在进军途中两次分兵，主力向东进发。图的最右边是莫斯科，是法军所到达的最远的地方，然后黑线一路折回，和留守的部队汇合，并一路逃向边界。图表下半部分的折线代表温度，我们可以看到自法军从莫斯科回撤开始，温度就一直下降，最下面是时间轴，显示了法军到达地图上各点的日期，在莫斯科是 9 月份，10 月到 12 月一直撤退。灰黑两线的宽度代表法军人数，在俄波边界集结之时，拿破仑大军是 42 万人，灰线最粗，到了莫斯科就只剩下 10 万了，而最后退到边境的只有窄窄黑线的了。这么一个图表包含了地图，进军路线，军队人数，温度，日期等诸多信息以及它们之间的对应关系，而这些都被组织得井井有条让人一目了然，实在是令人叹为观止设计者真是功力非凡啊这个图已经成了数据信息可视化的经典之作，它向我们展示了如何把庞大复杂的信息合理安排组织成能够让人轻松理解的图表

对于游戏来说，游戏越来越复杂，需要显示给玩家的、玩家需要掌握的数据也越来越多。比如 RTS 中的各种金银石铁的资源状况、敌我位置和数量对比、还有全局性的统计数据，而 RPG 中则有各同伴的健康状况、武器装备、携带物品和现有级数等等。这些游戏内部的状态或者说数据都需要井井有条地显示给玩家，否则的话玩家是两眼一抹黑无法进行游戏的。

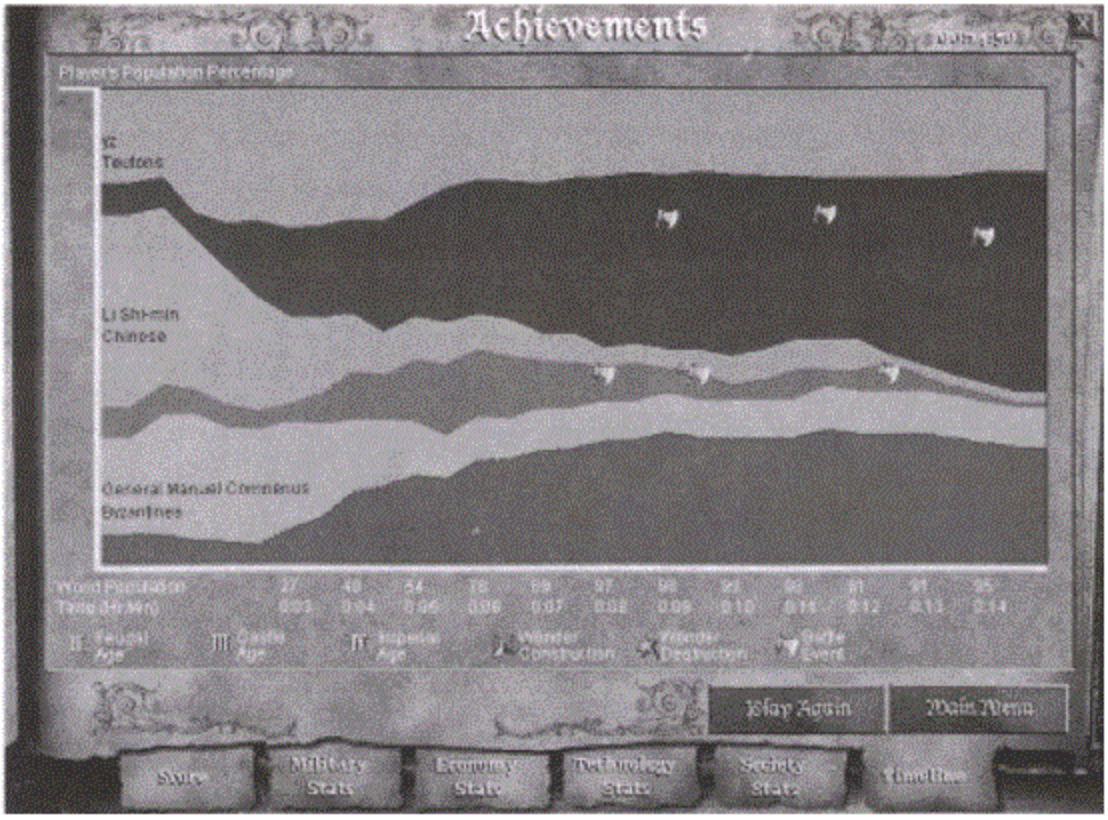


图 9-11 如何将游戏内部纷繁复杂的数据信息显示给玩家，是人机界面设计的首要任务之一。这是《帝国时代》中的一个图表，显示的是游戏中各种族的人口随时间改变的状况。这个图表的设计远非尽善尽美，比如说游戏中有三个种族，但图表上有 6 个不同颜色的区域，究竟哪个区域对应哪个种族不是很清楚。又如这个图表用斧头代表在某个时间段发生了战争，但我们从图表上无法知道这场战争是和谁打的，战况如何。和前面拿破仑 1812 年战争的图表比起来，这个图表包含的信息有限并且不是那么清楚。完全有可能把更多的信息塞进这个图表中的。读者们可以试着重新设计一下，看看如何改进之

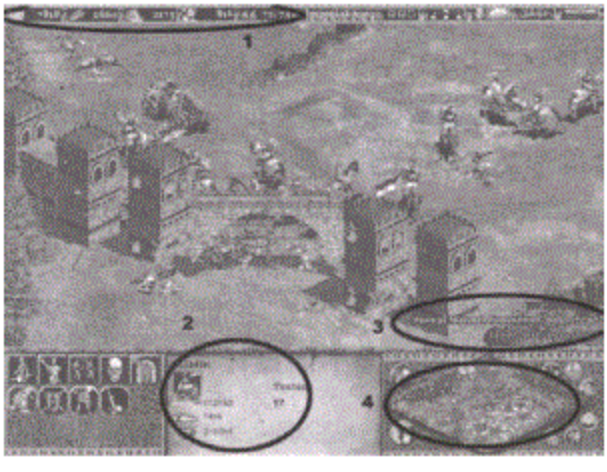


图 9-12《帝国时代Ⅱ》的主游戏屏幕。划圈的四个区域是用来显示游戏内部的数据信息的（也就是说是纯输出的区域，不接受输入）。我们可以看到既使用了纯数字，也用了地图和图表（如血槽），还有些辅助的图标（如金木等资源）。它们相互配合，共同任务就是使得玩家可以清楚地掌握游戏的状况

图 9-13 《可汗》(Kohan) 的界面，各种图像信息文字铺天盖地而来。这么多信息，如何组织安排确实是大问题。虽然这个界面总的来看色调统一，几个大的区域之间的关系也还明确，排版也一丝不苟。但如此庞大的信息量，玩家是否有足够的勇气去接受呢？

为了解决数据信息可视化问题，人机界面设计师手里主要有两个工具：由图书目录学发展出来的信息结构学（information architecture）和传统的平面设计排版技术。信息可视化的第一步当然是要信息结构化，然后再把信息摆放到屏幕上。因此信息结构学讲的是如何从复杂纷乱的信息中整理出层次结构，最后形成某种组织关系，还有命名规则等问题。而传统的平面设计排版技术中有很多关于如何在二维平面上安排信息的经验和技术，比如说格子系统（grid system）。以前软件界面设计根本就没有使用格子系统，其结果是按钮标签到处乱放，屏幕杂乱无章。近几年人机界面设计师们开始有意识地向平面设计师们学习，利用了格子系统的软件界面规整多了。

输入手段

人机界面设计的第二个任务，就是如何帮助用户去选择正确的输入指令。包括如何设计这些指令，以及如何在屏幕上显示这些指令。我们前面说过硬界面和软界面之分。硬界面是通过手柄按键来接受输入指令，而软界面是通过 GUI 组件来接受输入指令的。比较重要的有以下几种：

●菜单（menu）

菜单是比较常见的接受输入指令的 GUI 组件。每个菜单项对应一项功能。菜单可以分为固定式菜单、下拉式菜单、弹出式菜单三种。它们各有利弊，需要针对具体情况选择使用。固定式菜单一目了然，但所占屏幕面积大。弹出式菜单不占屏幕面积，但由于在屏幕上没有指示，玩家也许会不知道其存在。下拉式菜单综合两者优点，既有明确指示，又不占太大的屏幕面积。特别值得一提的是饼状菜单，如图 9-14 所示。

图 9-14 左边是饼状菜单示意图。饼状菜单也是弹出式菜单的一种。其中心的小圆是退出区，即在此区域内点击鼠标则关闭整个菜单。6 个饼状区域分别对应 6 个菜单选项。我们可以从数学上证明饼状菜单是所有种类的菜单中最优的。根据费茨定理（Fitts' Law）计算得出：使用饼状菜单鼠标需要的移动距离最短，定位精确要求最低，因此速度最快，效率最高。右边是《无冬之夜》（Neverwinder Nights）的菜单，采用了饼状菜单的概念，是个很有新意的设计！（饼状菜单来源于 10 多年以前的一篇专业论文。饼状菜单这么好但很少有人用，原因是设计师中没人去读论文，所以根本不知道有这么好的设计。大部分设计师认为科技论文太高深了，但实际上科技论文中有很多新奇的想法和思路可以借鉴。而到了 2001 年我们终于看到这一思想被游戏设计师们开始加以利用了。看来游戏设计师们应该真应该博览群书啊，特别是有关人机交互和人机界面的学术论文，一定会获益匪浅的！）

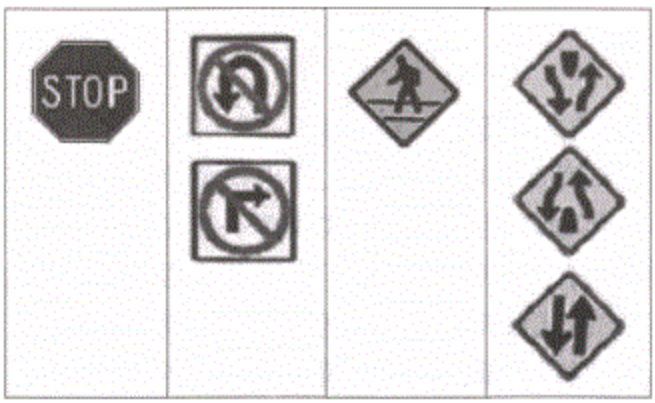


图 9-15 交通图标是图标设计的范例!这些图标是高度抽象的，但同时又是明白易懂的。人们无需任何文字说明就能马上理解这些图标的意思

●图标（icon）

不要小看这小小的图标。设计师们所公认最难设计的就是图标。因为图标需要在极小的空间里表达极大的信息量。既是高度抽象的，但又必须让人一眼看上去就能明白它所要表达的意思。其设计难度可想而知！交通图标是公认设计得最成功的例子了（不成功行吗？要出车祸的喔！）。图 9-15 是美国伊利诺伊州的几个交通图标。

世界上最有名的图标，估计要数 Smiley face 了。这个简单的设计（圆圈加上两点一线），自 20 世纪 60 年代问世以来，迅速风靡了全世界。

图 9-16 左边是 smiley face 的最初形态（下角有设计者签名），右边是其设计者 Harvey Ball。1963 年他接受一个客户的请求，设计一个图标来提高那个公司的士气。smiley face 由此诞生。设计出来后，不仅得到那个公司员工的好评，而且不胫而走，迅速征服了全世界各国人民。它的简单抽象的设计，超越了一切种族、肤色、文化、政治和宗教差异，表达着人类最美好的渴望，那就是欢乐的喜悦和乐观的精神。这个图标已经成了 20 世纪的经典之一

用在人机界面方面上的图标有几种，比如有的图标是没有任何功能的，只是起到提示的作用，如《帝国时代》屏幕最上方的各种资源图标。也有的图标实际上是起到按钮（button）的作用，每个图标对应一项指令（功能），比如《帝国时代》的建筑图标等。

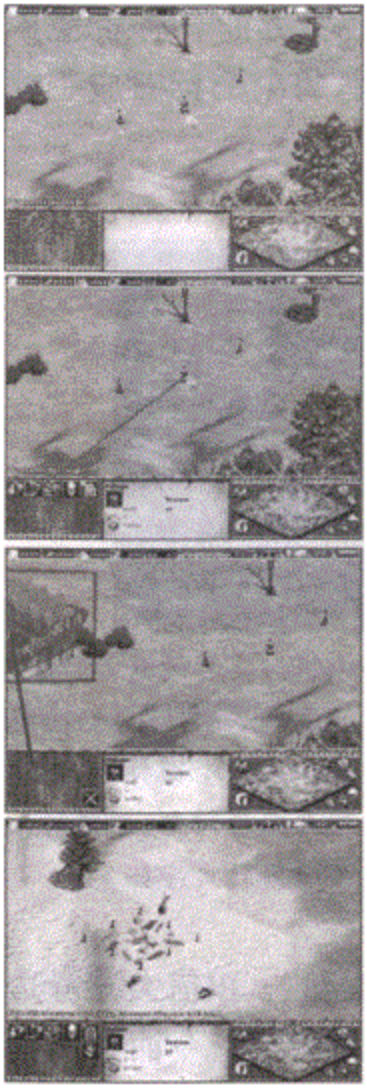
图 9-17 《帝国时代Ⅱ》的图标。这些图标的严重问题是如果不借助于文字标签，玩家很难搞清楚它们表达什么内容。比如第一行的前两个图标，一个代表巡逻，另一个代表防卫。但这两个图像之间有本质的一目了然的区别吗？单拿出第一个图标，又有谁能知道它代表巡逻呢，这也是目前所有游戏中图标设计所共同存在的问题，图像的细节还太多，不够抽象；不能正确表达它所需要表达的内容，还必须借助文字标签

●热健（hot key）

高手所用，速度最快（鼠标再怎么快也快不过键盘）。相信大家身边都有不少以能够记住并熟练地敲打 UNIX/LINUX 的复杂指令而顾盼自豪的高手或者变态（笔者是已经退化了，玩不转了）。热键的设计主要是针对游戏高手的，只有他们才会追求最快的速度，最快的反应，才会去花力气记住这些热键。

●指令序列（Action sequence）

所有这些菜单、图标、各种其他 GUI 组件、热键，最后都要被组合在一起，形成指令序列。所谓指令序列（action sequence），是指玩家要完成某个任务达成一定目的而需要遵循的一系列操作步骤。以我们天晴数码出品的大型网络游戏《征服》为例，模拟一个玩家要从物品箱里取出一件装备穿戴在角色身上，他必须完成以下几个指令序列：



假设我们有一个农民在屏幕中央，光标的初始位置也在屏幕中央，我们要造一个新的建筑

移动光标至建筑面板区

移动光标（此时建筑阴影随光标移动）到屏幕边沿，移动地图

在合适位置，单击鼠标键，开始建造。本指令序列到此结束

对农民单击鼠标键，屏幕左下方的建筑面板更新，显示一系列建筑图标

选择一个建筑图标，单击鼠标键

地图移动到合适位置，光标离开屏幕边沿

图 9-18 《征服》的一个基本指令序列

我们可以看到：一个游戏的游戏性或者说操作性，就是由许许多多这样的基本指令序列构成的。指令序列是游戏性的基础，其重要性可谓大矣！因此人机界面设计的重中之重就是设计并优化这些指令序列。玩家在游戏的过程中要重复无数次这样的操作。虽然一个指令序列看起来很简单，好像优化不优化无所谓似的。但重复几千次，则一个指令序列中任何小的疏漏都会极大地影响游戏性。

我们同样从这个例子可以看到，PC 游戏，特别是频繁使用标准 GUI 组件的 RPG、RTS 等类型的游戏的指令序列，基本上就是（鼠标移动+按键）n，即 n 次鼠标移动和按键的组合。

前面介绍了人机界面设计的两个基本任务，那么我们怎么去完成这两个任务呢？这就引入了下面的题目。

界面设计方法

人们也许会问：设计人机界面还需要什么方法？凭灵感和经验不就得了？答案是否定的！设计一个平面广告可以凭灵感，设计一个复杂的人机界面单凭灵感是不够的。为了保证人机界面的质量，保证玩家能够使用界面成功地进行游戏，人机界面设计必须遵循一定的方法和流程。

目前的人机界面设计方法，基本都是在 20 世纪 90 年代中后期发展起来的，已经相当成熟。惟一美中不足的是它们多为一般商用软件所使用，为游戏开发量身裁定的几乎没有。因此在使用中需要根据实际情况进行改进和变通。它们基本可分为两大类：设计流程（design process）和设计方法（design methods）。其中，设计流程是大框架，是设计师工作时所遵循的一系列步骤。而设计方法是则是在大框架上的枝叶，是在设计流程的每一个步骤中可以使用的一些工具。

目前最流行的设计流程，是快速原型法（rapid prototyping）和复进式设计（iterative design）。两者结合起来使用，关系密不可分，如图 9-19。

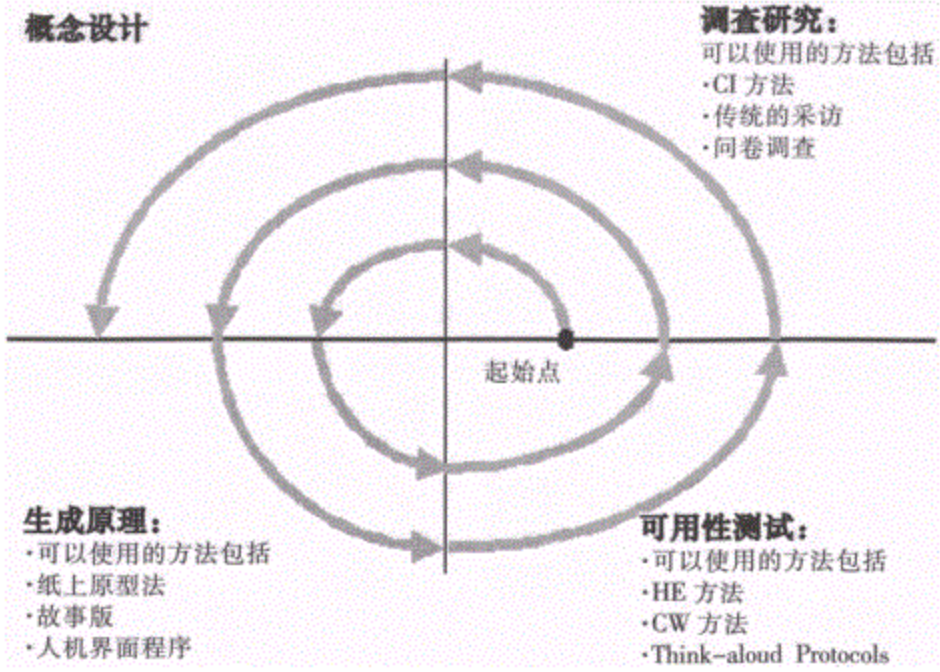


图 9-19：快速原型法和复进式设计示意图

我们可以看到，复进式设计和我们后面将要介绍的软件工程中的螺旋式模型有极大的相似性。实际上复进式设计可以看作螺旋式模型的一个缩小的版本，是其在人机界面领域的特殊应用。其基本组成步骤如下：

调查研究：在基本游戏内核设计完成并对游戏的基本功能和运行机制形成了一致意见后，才能开始设计人机界面。设计人机界面的第一步就是调查研究。这里所谈到的调查研究，

包括两方面：一是使用一种称为 CI 的方法（contextual inquiry）去从玩家那里获得第一手资料，包括他们是如何玩类似的游戏的，他们玩游戏时使用的策略，他们的心理活动和情感起伏等等。CI 方法是近几年 HCI 领域根据社会学的研究方法和软件人机界面的特殊需要发展而来的，专门用来观察分析用户的行为和心理，并把观察分析的结果用来指导人机界面的设计。二是观摩研究其他类似游戏的人机界面设计并获得启发。

概念设计：在调查研究的基础上，设计师可以自由地发挥想象力和创造力，进行人机界面的概念设计，包括基本的功能结构（人机界面要实现哪些功能，它们之间什么关系）、信息结构（信息如何显示，哪些重要信息需要放到一级菜单，哪些不那么重要的信息可以隐藏到二级菜单或者弹出式窗口里）、屏幕的安排（什么地方显示游戏世界，什么地方显示人机界面）、输入设备的利用（键盘的使用等）。在概念设计阶段，设计师主要的工具就是铅笔和纸张，完成一系列设计草图和文档。

生成原型：在概念设计基本完成后，需要把设计草图细化，使其成为能够用来测试的原型。目前比较流行的有纸上原形法，故事板、和用 Visual Basic，Director 等软件编写一些小程序。纸上原型法是指用 Photoshop 和 Illustrator 等绘图软件作出静态的人机界面屏幕截图（screenshot）。故事板是从动画电影业借鉴来的方法，是指把玩家使用人机界面的动态过程用看图说话的方式来表现（从某种意义上说很像连环画）。最重要的生成原型的方法就是用某些专用软件生成一些小程序了。像 Visual Basic 和 Director 等软件，都为目前业界所广泛使用。它们共同的特点是编程简单，易于快速生成 WYSIWYG 类型的单纯的人机界面程序（也就是只有人机界面而没有内核的程序）。由于能够真正运行，给下一步的测试自然带来很大便利。

可用性测试：完成原型后，需要做可用性测试。也就是预先设计好一些任务（每个任务包括一个目的和达成目的需完成的一系列步骤），请一些玩家到公司来，先给他们解释人机界面的基本情况，告诉他们最终需要达成的目的，然后给他们原型，让他们自己操作，看他们是否能够完成这些任务。从这个过程中，设计师可以发现自己设计中存在的诸多问题。

反复优化：通过可用性测试，设计师发现了设计存在的问题。于是他得修改设计，再生成原型，再测试。我们可以看到这是一个循环往复的过程，但每一次循环都是对上一次设计的优化和增强。人机界面就像滚雪球一样越滚越大。特别是在前几次循环时，开始可能我们只设计人机界面的一小部分，或者我们只用很粗糙的方法来生成原型（纸上原型法），随着循环往复的过程，一方面设计将越来越完整，另一方面原型也越来越复杂，越来越趋向人机界面的最终状态。前一个被称为覆盖率（coverage），后一个被称为真实度（fidelity）。这两个特性都随循环次数的增加而提高。

目前人们习惯把游戏的人机界面分成两部分：游戏初始及设置界面（Shell UI）和游戏中界面（In-game UI）。所谓 Shell UI，就是玩家在一开始时安装设置所使用的人机界面。游

戏中界面是指玩家开始游戏后在游戏虚拟世界中用来进行游戏的界面。两者之间显然有很大不同：Shell UI 的目的是使玩家能够轻松地安装好游戏，设置好相应的选项，尽快进入游戏。游戏中界面的目的是使玩家能够有效地进行游戏并体会到乐趣，也就是体现了游戏性。两者目的功能的不同，决定了设计上的不同侧重。对 Shell UI 来说，它和一般商用软件的人机界面没有很大不同，用我们前面所介绍的方法就可以应付了。而游戏中界面的问题则比较复杂。部分原因是它和游戏性紧密联系，而游戏性是很敏感的东西，比如说反应时间，1 秒 2 秒的差异也许就决定了游戏性的好坏，所谓差之毫厘，谬以千里。粗糙简单的原型很难真切地体现游戏性。而 Visual Basic 和 Director 等软件又先天不足，无法处理对反应时间等性能要求高的情况。因此在过去几年，由于工具的缺乏，快速原型法在游戏中界面的设计中并没有得到广泛使用。但近一年以来，游戏业对快速原型法是越来越重视了。新的更强大的适用于原型法的软件工具被设计出来（有些则是以大型软件的一个子系统或简化版的形式出现，如 MAYA Builder），基本上解决了我们前面所说的诸多问题。设计师可以使用这些工具，输入简单的三维模型，构建游戏场景，设置游戏人物，编写脚本（script），然后他们就得到了可

图 9-20 《帝国时代Ⅱ》的游戏初始及设置界面（左）和游戏中界面（右）

以真正玩起来的原型了。显然这样的原型能比较忠实地反映游戏的游戏性，不仅对人机界面的设计有极大帮助，整个游戏性的设计也将从中获益匪浅。可以预见，快速原型法将越来越被普遍应用于游戏开发，成为游戏设计师和人机界面设计师们的有力武器。

前面谈了设计流程，现在再谈谈设计方法。简单地说，设计方法就是在设计流程的各个步骤中可以使用的一些工具。比如说在可用性测试这一步，有几种测试方法可供选择使用：HE 方法（Heuristic Evaluations），CW 方法（Cognitive Walkthrough），费茨定理（Fitt's Law），这些方法各有利弊短长，好像主干上的枝叶，可以根据情况选择搭配使用。

界面设计原则

以下几条原则是设计师们应该遵循的：

人机界面不应该喧宾夺主：如果过分修饰搞得过于繁琐的话，人机界面反而会干扰玩家

的注意力，使它们不能集中精力于游戏世界。人机界面应该力求简单朴素，占用的屏幕空间应该越少越好，真正做到惨淡经营惜墨如金。记得一个著名的工业设计师曾经说过：一个出色的设计，是高度概括浓缩的，已经简化到不可能再简化!读者们也许会问："一方面游戏有这么多数据信息需要玩家知道，一方面你又说占用屏幕空间越少越好，这不是矛盾吗？"这的确是个矛盾。我们需要明确一点的是：任何设计都是对许多矛盾的折衷的产物!矛盾无法回避！正是由于这些矛盾的存在，使得人机界面设计的工作更富有挑战性，也逼着设计师们

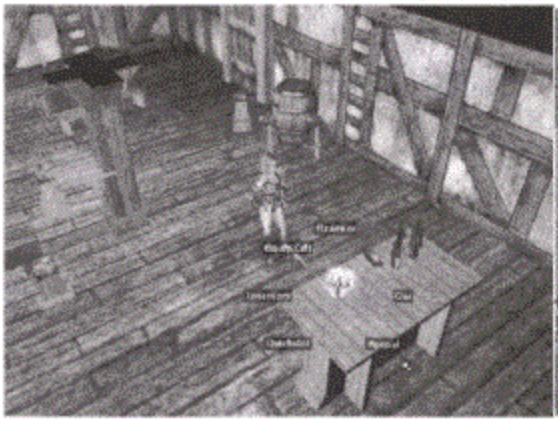


图 9-21 《无冬之夜》的界面。弹出式的图标和菜单在大部分时间是隐藏起来的，使得玩家可以完全沉浸于游戏的虚拟世界中不受干扰

图 9-22 一个典型的网上游戏的人机界面，各种图标、菜单、面板铺天盖地而来，毫无主次层次；颜色纷繁，令人眼花缭乱；白色和绿色的文字在褐色背景上很难阅读。总之，这是一个人机界面设计失败的例子

想出新的法子。比如说《无冬之夜》的界面，就是全部采用弹出的方式。平时不显示，不占用屏幕空间，要用的时候再弹出，用完关闭。如图 9-21。

人机界面和游戏世界应该风格一致：从色彩到质感，应该和游戏世界保持协调。即使要通过对比使得某一方更突出的话，也应该保持内在风格的一致。不能让人觉得双方风马牛不相及。

人机界面应该具有一定的自解释性（affordance）：所谓自解释性，是指一个设计能够通过自己的外形暗示自己的功能。最简单的例子如门把手，其形状本身就暗示了手应该握住它后向下按。音量调节钮上的凸凹不平，本身是为了增加摩擦，其外形就暗示了手可以很容易地转动它。在人机界面设计上，最典型的例子就是 WINDOWS 的窗口了，在窗口的右下角的三维凸凹不平，就是从音量调节器的设计中引申出来的，它了暗示鼠标可以拉动它从而

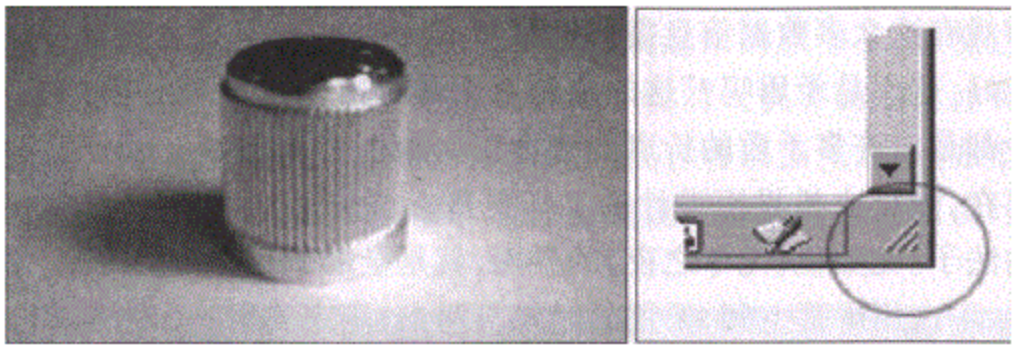


图 9-23 音量调节钮上的凸凹不平就暗示了其功能须增大摩擦，使得手可以很容易地转动它。而 Windows 系统的窗口右下角的图像同样也暗示了鼠标可以拉动它改变窗口大小。这是自解释性在现实世界和人机界面上的应用的绝好例子

改变窗口大小。

人机界面应该布局平衡：所有文字和图标应该摆放得恰到好处，形成平衡感。以往的游戏设计师们毫无平面设计的基本训练，在摆放图标和文字时随心所欲毫无章法。图 9-24 是《帝国时代》的一个信息面板设计。明显左重右轻，重要信息不突出。

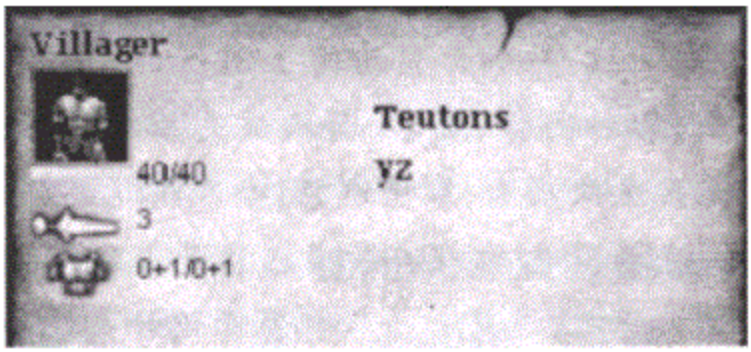


图 9-24　即使是成功的作品如《帝国时代》，其人机界面仍旧有诸多问题。这个面板的设计就是一例，显示出设计者没有细心安排布局，导致明显的不平衡感和重要信息不突出

应该以一种动态的观点来设计人机界面：人机界面设计和平面设计最大的不同点就是平面设计只是一种静态的设计，它的最终产品是一张广告，一本书的封面，或者一本宣传小册子。人们只是去阅读静态的页面。人机界面设计的最终产品是软件的界面，是要由用户来动态地使用的。设计人机界面，并不只是设计一个个窗口菜单和控制面板，更重要的是设计一种动态的交互（interaction），在设计时必须将用户的行为考虑在内。

人机界面设计的最高水平是达到无形入化：所谓人机界面无形入化，意思就是说人机界面非常自然，非常好用，玩家用起来得心应手，全身心地投入到游戏世界中，似乎人机界面是透明的了，似乎感受不到它的存在了。这是所有人机界面设计的终极目标，也是所有游戏设计师们努力的方向。

界面设计举例

学习界面设计最有效的方法是分析成功游戏的界面设计。看看它们哪些地方设计得好，哪些地方有可能改善。下面就以几个知名游戏为例。

从 1 代到 2 代田标的演变分析《帝国时代》的界面设计

由 Ensemble Studios 设计开发，微软发行的《帝国时代》系列，被游戏界誉为近年来 PC 游戏人机界面设计的典范。而《帝国时代》的设计师们能够取得这样的成功，是有其原因的，Ensemble Studios 的老板 Tony Goodman 是从商用软件领域改行过来的。他把原来开发商用软件人机界面时使用的一些成功方法和经验带到了游戏业，在《帝国时代》开发的全过程都着重强调了人机界面的重要性，并且不遗余力地重复改进，力求使其易学易用。工夫不负有心人，他们的努力得到了回报。《帝国时代》发售后，很快赢得玩家的一片好评声。有很多

从不玩游戏的对计算机软件怀着敬畏心理的中老年人，也买来《帝国时代》，怀着对世界历史的浓厚兴趣，玩得废寝忘食不亦乐乎，去体会“游戏”（作动词）历史的新奇和刺激。

《帝国时代》人机界面设计之成功，就在于设计师们采取的策略是正确的。他们的目的就是要把《帝国时代》设计成老少咸宜雅俗共赏的游戏，但老年人和年轻人，铁杆玩家和一般玩家显然能力不同，要求也不同。为了适应不同层次玩家的需要，《帝国时代》的人机界面也按不同层次来设计。对入门级玩家来说，整个游戏用一个鼠标器就可以操作自如，不需指导就能上手。而对铁杆玩家来说，则有复杂的热键和策略供他们使用。不同层次的玩家，使用同一个人机界面，都能从游戏中得到乐趣。这是游戏人机界面设计最基本最重要的，同时也是最难的要求！《帝国时代》的设计师们显然成功地达成了这一要求。

《帝国时代》的设计者们还曾提出所谓的“前 15 分钟法则”。“前 15 分钟法则”是指对一个游戏来说，如果入门级玩家不能在前 15 分钟顺利地弄明白基本操作和策略并开始游戏，或者铁杆玩家不能在前 15 分钟感到有趣和挑战的话，他们都会永远地离开这个游戏，不再进行下一步的尝试。因此，前 15 分钟就决定了一个游戏的命运：是被安装以后玩 15 分钟就惨遭删除，还是可以陪伴玩家共同度过几天甚至几个月。《帝国时代》的设计师们特别强调这一点。他们为此还专门调整了游戏的初始状态，力求游戏的前 15 分钟引人入胜。

当然，《帝国时代》的人机界面也并非完美无缺。从一代到二代人机界面的演变中，我们可以看到设计师们否定自我和超越自我的魄力和胆识，我们更可看到他们是如何对已经很成功的作品进行再优化的。最显著的例子，就是建筑的图标。在一代中，设计师用了每个建筑自身图像的一部分来作图标。单从图标本身来看，有几个问题：首先图像过于复杂，我们都知道图标的细节越多越不容易识别；其次有几个图标的图像太相似（如第一行前三个和第二行最后两个），玩家很难把他们区分开。更致命的弱点则是在玩家使用人机界面的过程中，这些图标非但没有起到帮助玩家的作用，反而阻碍了玩家的心理认知过程。以玩家需要骑兵为例。玩家修建马厩的心理过程分三步。第一步，因为需要骑兵，玩家的脑海里首先会出现马、马掌、马靴、马刺、骑手、马刀等和骑兵有关系的影像。这些影像信息是玩家从他大脑中的长期记忆中获取的，是从他以前的人生经历和常识知识中长期形成的，因此是一个非常迅速和自然的过程，在下意识中就完成了。第二步，他必须从短期记忆中找到骑兵和马厩的建筑图像之间的映射关系。从他第一次玩这个游戏开始时到目前为止，他从游戏中所获得的信息一般被存储在短期记忆中。这些信息（如马厩的建筑图像），不是普遍通用的常识性知识，只是这个游戏特有的。除非经过长时间考验，才能被升级存入长期记忆。而他在第一步从长期记忆中读取的诸多和骑兵有关的影像都没有用，都浪费了。第三步，在获得了骑兵和马厩建筑图像之间的映射关系后，玩家的眼睛接受屏幕上图标的视觉信息和短期记忆中的信息对应，如果吻合，则选择正确的图标。显然，要成功修建马厩，第二步至关紧要。如果玩游戏一段时间后，还不能建立骑兵和马厩建筑图像之间的正确的映射关系，并将其存入短期记忆的话，则玩家在选择图标时就会遇到很大麻烦。游戏高手和庸手的区别，就在于是否能

够快速正确地在短期记忆中形成这样的映射关系，并在需要的时候快速正确地从短期记忆中提取，作出正确的选择。但这第二步同时有个问题：玩家的心理活动过程，即“骑兵→马厩建筑图像→骑兵”的过程，可以看成“抽象信息→具体信息→抽象信息”，“功能→图像→功能”和“普遍常识→特殊知识→普遍常识”的过程。既然玩家是从普遍常识出发，最后又回到普遍常识。那么为什么要绕一道弯呢?

我们再来看看二代的设计。图标变得抽象化了，更加简化浓缩了。不再使用复杂的易混淆的实际建筑图像，而是使用代表其主要功能的简单的图像。马厩的功能是训练骑兵，因此用马掌来表示。这样就减少了心理认知过程的第二步。玩家整个的心理过程简化为“功能→功能”和“普遍常识→普遍常识”。这对入门级玩家的帮助是显而易见的，因为他们无需在短期记忆中去费力地建立建筑图像和其功能的映射关系了。另外简单的图像容易分辨，不易混淆。这些都减轻了玩家的负担。

图 9-25 (a)是《帝国时代Ⅰ》的图标，和(b)是《帝国时代Ⅱ》的图标。我们可以看到二代较之一代有了极大改善。一代的图标过于复杂，难于分辨，给玩家带来不少麻烦。二代的图标则一目了然，简单易用。

《幕府将军》的界面设计

EA 推出的《幕府将军》（Shogun：total war），也是 RTS 类型的游戏。值得指出的是这个完全以日本战国时期为历史背景的游戏是由英国的一个制作组开发的，可以看作是西方人对日本文化的某种崇拜吧。从游戏本身来看，制作组对日本历史和文化确实是下了一番工夫的。我们来分析几个屏幕。

主菜单屏幕：日本风格的信札（菜单）徐徐打开，白色的信札和红色背景相衬，几个菜单选项简单明了，重点突出。背景有两层，有很复杂的景深动画。整个背景向左卷轴移动，暗红色的士兵和战马的剪影则不停地向屏幕右半部分移动，增大了动感。每个士兵和武士的

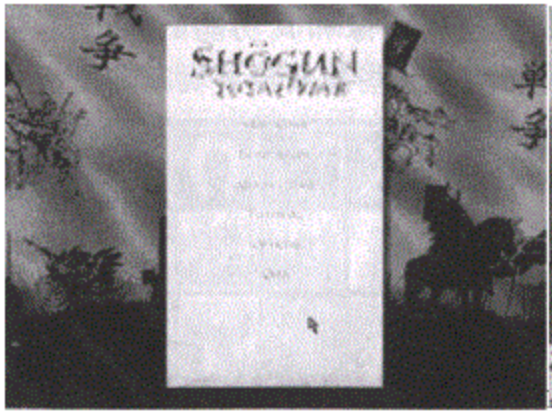


图 9-26 《幕府将军》的主菜单屏幕

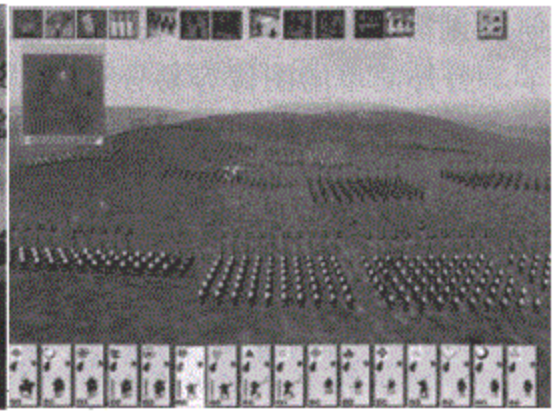


图 9-27 《幕府将军》的主游戏屏幕

动画相当细致，如衣甲和旗帜的飘动（可以说这些剪影动画达到了极高的水平，很有重量感），虽然背景有很复杂的动作，但由于色调比较暗，没有对前景产生干扰。前景背景，一动一静，相辅相成，再配上日本风格的音乐音效，使得这么一个简单的“被动”的主菜单屏幕，充满了动感和魄力。这是 Shell UI 设计得很成功的一个范例。

主游戏屏幕：透明的弹出式窗口很有新意，而图标则有一些问题。屏幕上角的一排大图标，包括了控制部队的主要功能（队形、状态等）。但这几个图标显然设计得太保守，太像 Windows 标准的三维图标了。其实，用纯二维图标代替这些三维的“盒子”又何尝不可呢？二维图标和这个游戏整个的日本文化氛围显然更和谐一些。

其他人机交互手段

前面主要是介绍了基于 WIMP 类型的人机界面设计。除了 WIMP 外，其他各种形式的人机交互手段方兴未艾。它们更加接近人类的习性，更加自然。随着技术的进步，它们将在未来的游戏中占有一席之地。下面简单地介绍几种最有希望的交互手段。

语音识别与合成

从技术角度讲，语音识别与合成已经不是什么新鲜的东西了。其技术已经趋于成熟，或者说暂时没有太大的发展余地和突破性进展了。但在应用方面，却是刚刚起步。目前使用这种技术的游戏很少，SEGA 的《人面鱼》就是一个很难得的例子。

图 9-28 《人面鱼》中使用了语音识别技术

语音识别，是指通过麦克风采集玩家的声音，然后通过处理，了解玩家所要表达的意图。语音合成，是指不使用预先录制的语音，而是采用计算机合成语音，传达给玩家一定的信息或指令。

图 9-29 语音识别和语音合成经常被用来制造能够和人直接交流的机器玩偶。图中玩偶为 SONY 所设计制造的智能狗。

语音识别基本要分三步：第一，辨别声音；第二，辨别词语；第三，辨别语意。

使用语音作为交互手段的优点是：第一，更加直接自然，使得计算机更加人性化；第二，易于适应个人，因为语音识别技术可以逐渐熟悉某个玩家的语音特征，使用的时间越长正确率越高。其缺点是总会有一定的误判，另外单独使用时有极大困难——比如在玩 RTS 游戏时，玩家要命令一堆士兵从某地移动到另一地。他不能只

说：“第一集群，从这移动到那！”因为计算机虽然能够理解“这”和“那”的意义，但却缺乏足够的指示说明“这”究竟是指地图上哪个位置，“那”又是指地图上的哪个位置。所以语音识别与合成技术，还需要和其他技术结合起来使用，比如下面要介绍的手势识别技术。

姿势识别与控制技术

人和人之间的交流是通过多种途径进行的。平时人们之间的对话，除了字斟句酌选择合适的词语外，眼神表情手势也很重要。《红楼梦》第二十六回“蜂腰桥设言传心事，潇湘馆春困发幽情”中有这么一段对话描写：

“薛蟠道：‘要不是我也不敢惊动，只因明儿五月初三日是我的生日，谁知古董行的程日兴，他不知那里寻了来的这么粗这么长粉脆的鲜藕，这么大的大西瓜，这么长一尾新鲜的鲟鱼，这么大的一个暹罗国进贡的灵柏香熏的暹猪。你说，他这四样礼可难得不难得？那鱼，猪不过贵而难得，这藕和瓜亏他怎么种出来的。’”

这段对话描写十分生动，我们可以想象薛蟠边说边指手画脚的样子。对话的另一方贾宝玉是可以准确无误地接受薛蟠所要表达的全部信息的，因为他看着薛蟠在那用手比划。如果这段对话改在电话里进行，就绝不会是这个样子。我们设想一下薛蟠在电话那头唠叨“这么大的大西瓜，这么长一尾新鲜的鲟鱼”，在电话这头的贾宝玉肯定越听越烦，因为他无法知道薛蟠说的“这么大”究竟有多大。这个例子就是要说明人和人的对话，除了声音外，手势也是表达信息的重要部分，对声音具有辅助作用，使双方更容易理解和沟通。大家总觉得打电话不如当面说得清楚，其原因就是电话只能传播语音，而不能传播这些辅助的信息。

手势在演讲时的作用也很大，可以起到发泄感情煽动群众的作用。像列宁在讲台上探身挥手，希特勒在麦克风前双手握拳歇斯底里的镜头，相信大家都很熟悉。手势甚至可以取代语音，自成体系来表达复杂的信息，比如旗语和手语。

前面说了手势在人和人之间交流（交互）时所起的作用。我们知道研究人机交互的目的就是要让人和计算机能够自然地交流，因此现实中人和人之间交流的方式对人机交互的研究有借鉴作用。从传统的键盘鼠标，到近几年的大热门语音识别，人们逐渐认识到单一的交互手段都有各自的局限性，比如说语音识别的出错率比较高。研究人员不仅想到是否可以再加以其他辅助手段来相互补足，于是姿势（手势）识别与控制就被提到台面上来了。

图 9-30 20 世纪最著名的手势—丘吉尔的 V-sign，代表胜利

所谓姿势识别（gesture recognition），又称为姿势控制（gesture control），主要指两种技术：一种是完全抛开键盘鼠标，使用摄像机捕捉人的手势或其他肢体动作，通过神经元网络

和模式识别等技术来分析手势的意思，并以这种方式来操作计算机。这种技术叫做基于摄像的姿势识别技术（video-based gesture recognition）。它主要应用于三维虚拟现实系统中，并不适用于目前商用计算机的鼠标、键盘、屏幕格局。第二种技术是使用现有计算机的鼠标或者数据笔（stylus），进行在时间上连续的输入。所谓连续的输入，是指鼠标输入不再是离散的了——我们在 WINDOWS 中使用鼠标的时候，把鼠标移动到一个地方，按下鼠标按键，触发一个事件。这时候决定所触发事件的是光标当前所处的位置。系统并不需要知道鼠标是通过什么样的轨迹，以什么样的速度移动到现在的位置的。这种输入叫做离散的输入。而连续的输入，是指在一段时间内的鼠标的移动轨迹和移动速度决定触发的事件。最直接的例子，如使用数据笔在 PDA 的屏幕上书写，图例如下。《黑与白》中使用的也是这种技术，只不过鼠标代替了数据笔，如图 9-31。

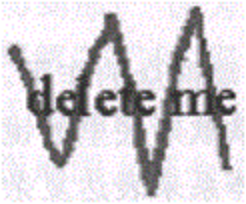


图 9-31 最简单的姿势识别技术示意 使用数据笔 在屏幕上的一段文字上面画折线 就意味着要删除这段文字

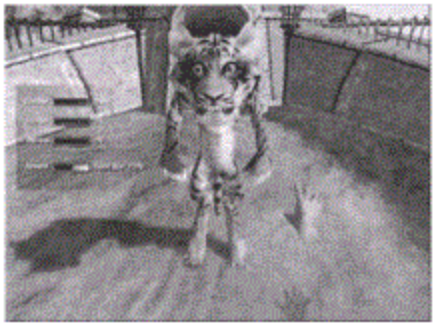


图 9-32 《黑与白》是第一个成功使用了姿势识别和控制技术的 PC 游戏

姿势识别和控制技术是十分强大的。首先它具有多样性和扩展性，可以设计出种种新奇的操作方法。因为如果只是接受离散的事件，则操作方法只是在屏幕的某个位置按下鼠标按键。现在加入了时间因素，操作方法可以更趋多样。其次是其表达力更丰富，因为加上时间这一维后，可以用鼠标表达更复杂的意图。最后一点是使用这种技术比较直观。比如在《黑与白》中对宠物的惩罚——掌嘴。就是按住鼠标按键，在宠物脸部左右快速来回移动。这种操作非常直观自然，玩家一旦学会以后就不会轻易忘记。

当然，这种技术也有其弱点。首先它需要一定的学习时间，要花很大力气才能掌握全部操作方法。在《黑与白》中，为此专门设计了一个白胡子老头和一个邪恶的魔鬼。他们负责指导玩家，帮助玩家逐步掌握各种操作。其次姿势控制需要精心的设计，因为每一种姿势都是既有轨迹又有时间，设计变得更加复杂，这就给设计者提出了很高的要求。最后，因为姿势识别和控制目前没有实际的标准，玩家对这种操作方式还不熟悉（对标准的 WIMP 界面，玩家可以拿来就用，因为那已经形成了标准，并被所有计算机用户所熟知）。这样各个公司都搞出自己的一套姿势识别技术，容易在玩家中造成混乱和迷惑。

道具或者玩具

在东西方都曾经有这样的迷信：用木头削成或者草秸扎成一个小人，用来代表仇人，贴上咒符，用针扎，可以使远在其他地方的仇人七窍流血而死。也就是说通过这个小人可以实现对真人的控制。在西方，这叫做巫毒玩偶（voodoo doll）。现在当然没有人会信这些了，但经常胡思乱想的计算机研究人员把却对这古老的迷信发生了兴趣，他们想到：这种通过一

个玩偶来控制它所代表的物体的方法是否可以用来控制计算机呢？结论是可以，而且很好！

图 9-33　屏幕上显示的巨嘴鸟所“生活”的虚拟世界

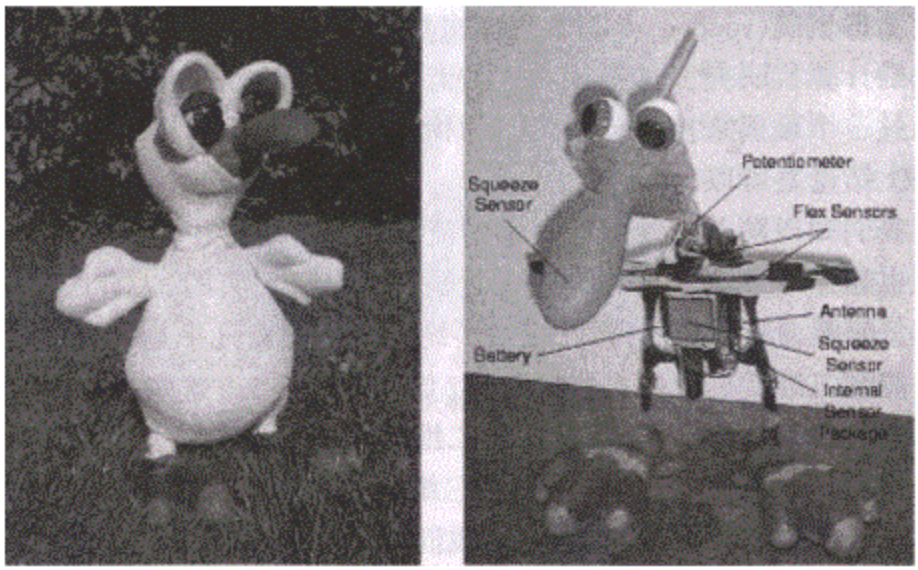


图 9-34　巨嘴鸟的实体。左边图片是其外部形状，右边图片显示其内部构造。有电池、各种各样的传感器和天线

我们知道随着三维图形技术的流行，三维世界中的人物和生物的模型也越来越复杂。从其构造来看是完全模拟现实的，人有四肢，鸟有翅膀。但要控制他们的行为，仅靠传统的鼠标键盘几乎是不可能的。如何去控制三维世界中的人物抬右腿？如何让他摇头？这些高自由度的在三维空间内发生的动作，很难通过二维鼠标和键盘按键去控制。而如果我们使用一个玩偶，通过它来控制三维世界中的人物，则非常直观——要三维世界中的人物抬右腿的话，我们只需要抬起玩偶的右腿即可。这种控制方法难度不高，任何拥有毛绒玩具的小孩子都能够顺利地操纵三维世界中的人物。从情感上说，使用憨态可鞠的毛绒玩具或者其他实体来控制计算机，可以吸引更多的女性用户，使孩子们更喜欢使用计算机。

MIT 媒体实验室的巨嘴鸟玩偶就是这种技术的一个尝试。巨嘴鸟外表上看是普通的毛绒玩具，但内藏各种传感器，使用无线网络，把自身状态传给计算机，然后反应到三维世界中。

图 9-35　巨嘴鸟玩具是这么被使用的：玩家位于屏幕前面，屏幕上显示巨嘴鸟和它的朋友们所生活的虚拟世界。玩家通过扇动翅膀使巨嘴鸟飞翔，摇动它的脑袋向它的朋友打招呼……基本上一个小女孩对毛绒玩具的呵护方法都能被反映到虚拟世界中，并影响巨嘴鸟和它的朋友们的行为，并触发出各种事件

虚拟现实

虚拟现实技术是最近 10 年曝光度很高的一项技术，但在其应用方面现在遇到很大的问题。从技术的角度讲，这项技术已经相当成熟。各种产品如数据手套和三维立体眼镜都已经商业化。但对虚拟现实技术的行为模式的研究却是极大地缺乏。所谓行为模式，是指当人戴上三维立体眼镜，戴上数据手套，完全置身于一个虚拟世界中时，他失去了现实世界的参照，必须用一种新的行为模式来代替在现实世界中的日常行为。他如何在虚拟世界中行走（显然不能戴着三维立体眼镜“瞎”走），如何在虚拟世界中和虚拟的人或物交互？这些问题不是技术问题，很大程度上是心理学和行为科学的问题。如果解决不好，用户在三维世界中可能方位感全失，甚至会引起不适和恐慌。

美工篇

第十章　人物设计

近几年来，各种游戏人物形象日益深入人心，马里奥、刺猬 SONIC、春丽（街霸）、克劳德（最终幻想 7）、皮卡丘等可以说是家喻户晓、众人皆知，劳拉（古墓丽影）甚至成为美国风靡一时的公众偶像，上了《时代周刊》的封面。可以说，设计出为广大公众所喜爱的成功的游戏人物形象，是所有游戏设计师的梦想——即使上不了时代周刊的封面，能够成为 SONY 的标志宠物也行啊！

图 10-1 成为《时代周刊》封面人物的劳拉（左），和成为 SONY 标志宠物的古惑狼（右）。古惑狼是由美国一家游戏公司独立设计，设计出来后大获好评。恰巧当时 P5 刚问世还没有自己的标志宠物，SONY 迫切需要一个形象来和 SEGA 的刺猬 SONIC 和任天堂的马里奥对抗。古惑狼幸运地被选中，一跃成为 SONY 的标志宠物

游戏人物的重要性

对于游戏人物的重要性，业界有不同的看法。任天堂的宫本茂曾在被问到马里奥的设计时说：“对游戏来说，好玩是第一位的，我们所有的努力都是围绕着这一点。设计游戏人物，只不过是所有这些努力中的一环。如果游戏不好玩的话，游戏人物即使设计得很好，也是不可能成功的。”而他的美国同行，大名鼎鼎的古惑狼的设计者 Jason Rubin 却说：“大部分玩家买游戏时，受的是广告、宣传品、游戏包装封面的影响。如果印在封面上的游戏人物不能吸引人的话，游戏肯定卖不动！”

这两个人大相径庭的观点和电影界的某些争论有些相似。电影界有人认为，大牌明星决定票房，有明星出演的电影一定叫座，收益比较有保证。也有人认为电影是一门艺术，重要的电影自身的“语言”表达（如 auteur 理论），演员只是导演的一个简单工具。

我们可以把这两个人的观点，看成代表了两种不同的制作理念。一种是比较理想主义的观点，把游戏视为能够给玩家带来欢乐的有一定艺术价值的载体；另一种是商业化社会

的产物，将游戏视为大众化消费品，以卖动游戏为主要目的。两种观点并无优劣高下之分，并且都在实践中获得了一定程度的成功。

几个影响人物设计的外部因素

游戏人物设计并不是一项简单的任务。设计者的思路不能如天马行空般自由，而是要受到很多因素的影响和制约。下面我们就来考察一下这些影响因素。

硬件机能

对游戏中的人物设计起最大制约作用的，就是硬件机能。在 8 位机、16 位机时代，由于分辨率限制，游戏人物大多是大头小身子（3 头身），而且根本无法做出鼻子和嘴等细节，为了表达感情，只好把眼睛做得很大。像马里奥就是由于嘴做不好，只好用大胡子来掩盖了。由于分辨率太少，如何使自己的游戏人物跃然而出给人留下深刻印像就成了艰巨的任务。于是业界流传着这么一种说法：你的游戏人物设计出来了，如果其黑白剪影（silhouette）能被人准确无误地认出的话，那么这个游戏人物就成功了。这种说法实际是从动画界传过来的，确实有一定道理。它逼着游戏设计师抓住一个人物角色的本质和最主要的形体特征，简化简化再简化，浓缩浓缩再浓缩。

进入 32 位机时代后，三维图像成为主流。但机器所能实时处理的多边形数目有限。这就逼着游戏设计师们为同一个人物角色搞两套模型，一套多边形多的模型，用于高分辨率的过场动画；另一套多边形少的模型，用于低分辨率的实时画面。两个版本之间差别显著，其高分辩率模型有多边形上万个，而其低分辨率模型可能只有多边形几百或者几十个。

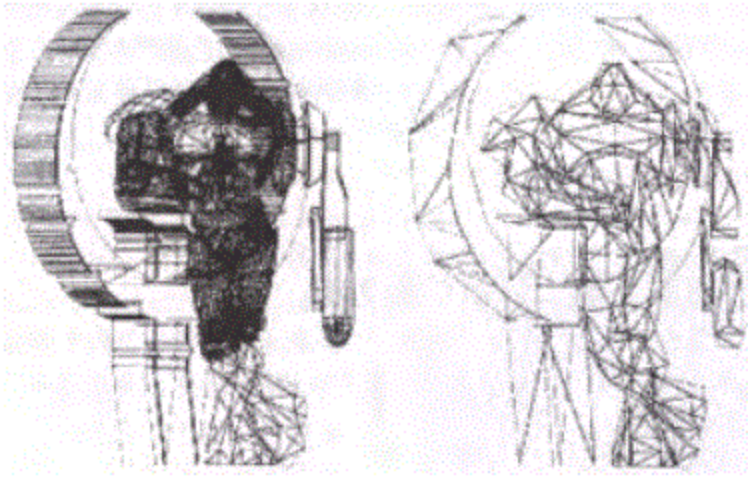


图 10-2 《Unreal》中同一个人物的高分辨率模型和低分辨率模型

随着机能的提高，特别 PS2 问世后，机器可以实时处理的多边形数量大大增加了。低分辨率模型的重要性将越来越削弱，最终退出游戏舞台。在多边形数量的瓶颈被克服后，设计者拥有了更大的自由度，但同时更新的挑战又提到了面前——如何使得高分辨率模型

具有更真实的动作和表情，甚至行为模式？目前的一些公司已经向这方面努力，最显著的例子就是一些电影所使用的更为复杂的建模技术。这些尖端技术将被逐渐转移到游戏业中并获得广泛使用。

总之，硬件机能的限制永远是套在设计者头上的紧箍咒，也是对设计者的最大的挑战。设计者必须明确知道在现阶段的技术条件下，什么是可行的，什么是不可行的，在限制之下发挥最大的设计自由。

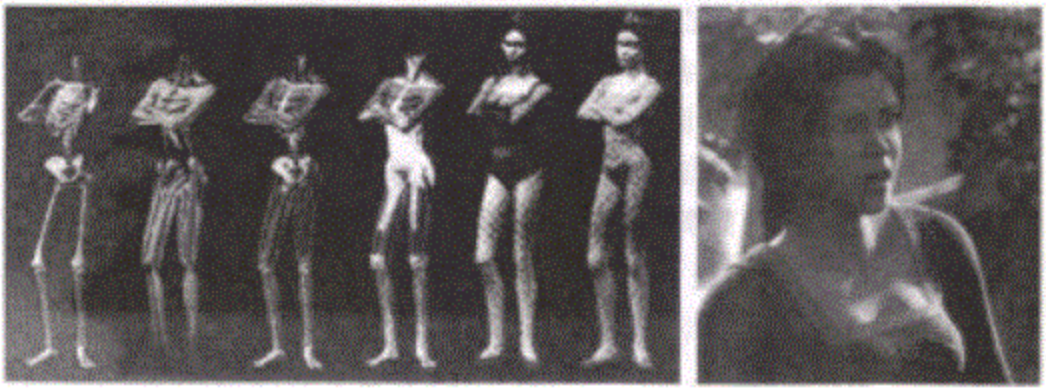


图 10-3 《怪物 SHREK》中费娥娜公主的模型，是从骨骼到肌肉到皮肤分步构建的高度复杂的模型，和传统的表面模型不同。人物动作由肌肉收缩驱动，更加真实生动

游戏类型

不同类型的游戏，对游戏人物设计的需求不同。RTS 类型的策略游戏，显然不需要非常细致的人物设计．一般是有成百上千的没有独特性格特征的公式化形象（stereotype）出场。而 RPG 和 AVG 游戏中，则需要有很多性格比较丰富多样的人物，并要反映出主要人物成长的历程。FTG 类型的游戏，则要人物个性张扬，一出场亮相就能够吸引玩家的注意，并通过一些小动作来强化人物，有时候还要杜撰一些背景故事来增加人物的厚度。因此，由游戏类型所决定，人物设计的任务有轻有重，侧重也不同。

图 10-4 RTS 游戏中的人物，大多是千人一面、性格模糊的公式化形象。图为《帝国时代》中的人物

文化背景

不同文化背景的玩家，审美观点不同，导致对游戏人物接受程度的不同。

以 PS 的标志宠物古惑狼为例。为了适应不同的市场，古惑狼的形象设计有细微的改动。我们看到在日本发售时的海报和杂志封面上，古惑狼更单纯天真一些。而美国本土的古惑狼，更有美国街头坏小子的狂放劲。两个版本的古惑狼笑起来就不一样，日本的古惑狼笑起来是孩童般阳光灿烂的微笑，而美版的古惑狼笑起来是捣蛋鬼似的狡黠的笑，更成人化一些。由于调整了人物设

计，更照顾到日本人的审美观点，古惑狼在日本获得了美国公司以前无法想象的极大成功。

更说明美日审美观点的差异的是《古惑狼》游戏中的配角。当时开发组里的几个毛头小伙子都是二十三四岁，在封闭的环境下开发游戏，自然很感寂寞。于是决定给“古惑狼”加个绝对性感的女配角。她就是 Tawna。设计出来后给 SONY 日本总公司送去审查，他们回话说你们把一个老女人放到游戏里干什么？设计者们面面相觑，哭笑不得。结果只好把她改成适合日本口味的，像小妹妹一样的 Coco 了。从附图上我们可以看出，原来 Tawna 的设计，是标准的美国漫画中女性形像，骨骼巨大，身体丰满，极为夸张。就亚洲观众来看，确实有些过了（所以他们认为是个老女人）。而新的 Coco 的设计，显然更年轻单纯，更适合亚洲市场。亚洲观众喜欢的是更童稚（childish）的形象。

图 10-5 《古惑狼》游戏中的配角，左边的是 Tawna，右边的是 Coco

亚洲设计的人物形象，很多都在美国流行开，而美国自己设计的人物形象，如果不做修改的话，很难推销到亚洲去。

人物设计的诸多方面

形体造型

设计一个人物，最重要的是人物的造型。Disney 公司的著名动画师 Preston Blair 在他的《卡通动画》（Cartoon Animation）一书中，对如何设计动画人物有极为精辟的阐述。他的阐述对游戏人物的设计也有一定的借鉴作用，在这里简单介绍一下。

●身体比例

设计一个人物，最重要的是身体比例。所谓身体比例，就是身体各部分之间的相对大小。对动画来说，就是头、胸、臀三大块。不同的比例，适用于不同类型的人物。可爱的宠物，一般是采用婴幼儿的比例，即大头小身子。而强悍类型的人物，则是小头大胸小臀。图 10-6 显示了不同类型的人物的不同比例。

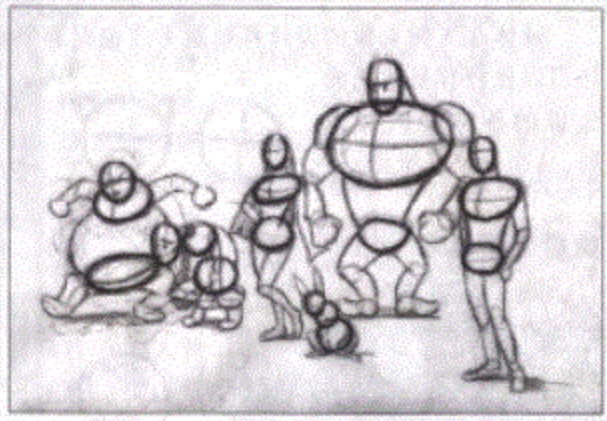


图 10-6 头胸臀三大块之间的大小比例关系

●球体组合

Disney 动画片中的人物最讲究的是三维感——在二维画面上显示出令人信服的纵深来，使人们感到动画中的人物不是二维的纸片，而是三维的有血有肉的人物。为了达成这种三维感，球体组合技术就应运而生了。Disney 动画中的人物，在设计和绘制时，一般以球体为基本构建单位，也就是把不同大小的球体组合到一块，形成人物的复杂的形体。这么做是有其深刻原因的：因为球体简单，其三维关系容易控制。把几个大的球体的大小、取向、相互关系抓住了，整个人物的形状就不会太走样。图 10-7 显示了球体组合的例子。

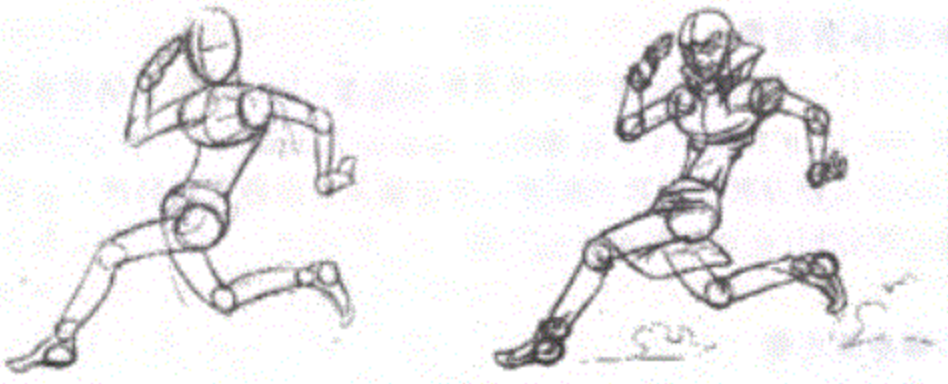


图 10-7 球体组合

球体在人物头像的设计和绘制方面也起着很重要的作用，利用球体可以比较容易地把握头像的各种仰俯和偏转，从而保证三维感。

图 10-8 球体在人物头像中的应用

服装道具

除了人物造型外，服饰道具也很重要。俗话说得好：“人凭衣裳马凭鞍！”人物的服饰很大程度上反映了一个人的社会地位、生活习惯和审美取向。

要设计好服饰和道具，对基本资料的掌握和熟悉极为重要。目前市面上有各种古代及少数民族服饰的图书，可以成为设计者很好的帮助，而游戏中需要的各种武器、道具、小物饰，则有各种专业书籍来填补设计者知识的不足。

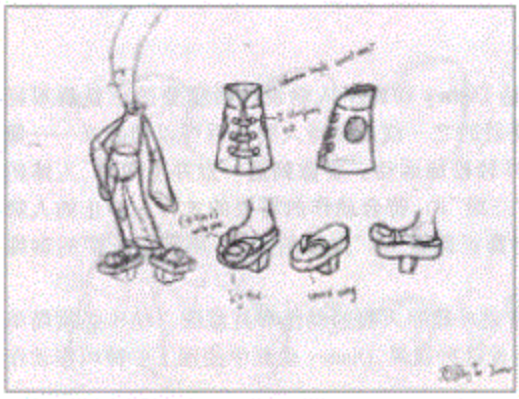


图 10-9 服装道具设计草图

在以前的 RPG 游戏中，虽然游戏人物可以频繁更换防具（头盔铠甲等），但由于技术的限制和成本的考虑，这些并不能在画面上反映出来。也就是说无论你给主人公装备什么样的装备和服饰，顶多是在菜单窗口中显示一下，在游戏画面上的主角还是原来的装束。现在随着三维图形技术的提高，存储能力的增强，现在越来越多的游戏通过多种材质，甚至改变多边形，使得游戏中的人物可以真

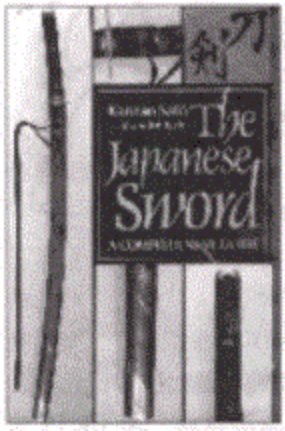


图 10-10 《The Japanese Sword - A Comprehensive Guide》(日本刀大全),类似的书籍在市场上很多。它们彩页众多,介绍详尽,是游戏设计者的重要帮助

图 10-11 在 XBOX 游戏《Project Ego》中,玩家可以在三维实时画面中给主角换装,甚至改变主角的发型

正地改变装束,让玩家感到眼前一亮、焕然一新。

动作特征

前面说过三维感是 Disney 动画中人物设计最重要的也是最艰巨的任务。三维感的主要目的之一就是使得人物今后可以动起来——像现实世界中的人物那样轻松地运动。要做到这一点并不简单,人体的各个主要部分必须是"三维"的,符合动作的需要的才行。静止的人物谁都会画,而要设计出真正能够"动"起来的人物,则需要长期的训练和实践。

要让观众明确地看到并理解人物的动作和其意图,动作必须简单并且重点突出。为了达成这种效果,Disney 动画中使用了一种叫作动作基线(line of action)的技术。所谓动作基线,是一条虚拟的线段,它表达了人物主动作方向。人体各部分的形体动作,要围绕这条主动作线,而不是干扰这条主动作线。这样动作的效果才能被观众理解。图 10-12 显示了一些人物动作和其动作线。

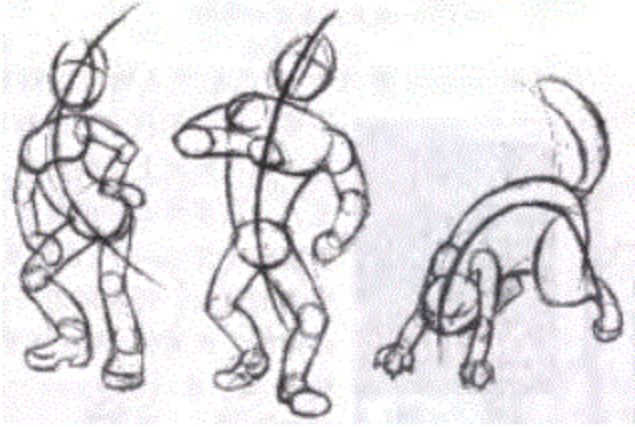


图 10-12 动作线

动作线同时也是达成形体动作的韵律和节奏的基础。图 10-13 显示了动作线的美妙的变化。

对游戏人物来说,造型的时候也必须考虑人物如何运动。而不同的人物应该有不同的运动特征——不同类型人物行走的姿势肯定不一样。卖花姑娘在伦敦街头畏畏缩缩地走的姿势,和上流社会的淑女在闲庭信步自然不同。这些在设计时都要考虑。

图 10-13 动作线的美妙变化

操作方法

比动画更进一步,游戏人物不

仅需要考虑动作，还要考虑操作，即玩家是如何控制人物的。实际上，人物的操作决定了人物的动作，人物的动作又决定人物的形体。我们可以看到矛盾出来了：人物的设计过程是与此相反的！因此需要不断返回修改，使得人物的三维模型能够动作流畅无碍。这点在游戏设计时是不可避免的。

性格属性

有了人物的躯壳，有了动作特征和操作方法，人物就有了一定的特征。但要使得人物形象更丰满更突出，则需要更多的表达人物性格特征和其他各种属性的手段。比如说让人物有自己的口头禅。DVD 的大容量存储更使得人物有自己独特口音的可能。

背景故事

游戏中的人物不应该是从石头里蹦出来的，需要有其出生及成长的历史。人物档案和背景故事是人物设计时要考虑的一环。这也是周边产品及续集开发的需要。

人物设计的工具

目前游戏业从动画业逐渐学到了很多工具和方法。下面就把最常见的几种工具介绍一下。

图 10-14 背景墙上为《Jack & Daxter》的设计速写和模型板。NaughtyDog 公司直接从动画电影业雇到了很多有经验的美工。这些美工把动画业成熟的方法介绍到游戏业

模型板

在以前游戏还不很复杂的时候，一两个美工就可以完成所有的工作。随着游戏容量越来越大，一个游戏所需要的美工越来越多，他们之间协调就成了一个问题。目前各游戏公司们都在向动画业学习各种工具和方法，模型板就是其中之一。

模型板是从动画业借用过来的工具。制作一部动画片，需要上百美工。他们分工不同，水平不同。有的负责画关键帧，有的负责画中间帧。这么多不同的人画同一个人物，如何能够把握好，达成一致呢？模型板的目的就在于此。模型板就是人物设计的标准，所有人画同一个人物的时候都要对照模型板以保证一致性。

三面图

从不同角度看同一个人物，是达成人物的三维感的重要工具。

配色图

配色图用来试验各种色彩配置。

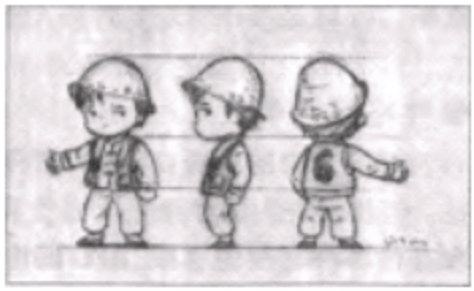


图 10-16 配色图

人物对比图

用来显示各人物之间的大小比例。

图 10-17 人物对比图

表情图

用来显示人物的各种表情。

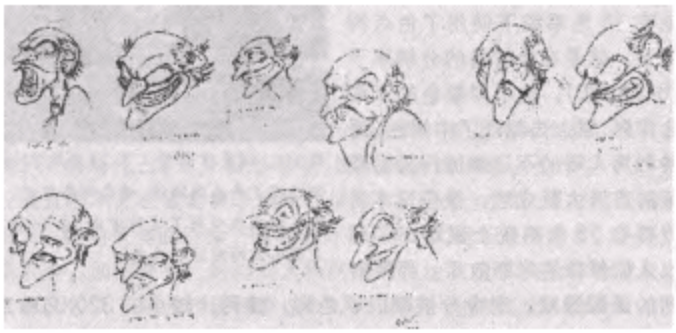


图 10-18 表情图

第十一章　色彩配置

色彩是最重要的视觉元素之一，在时装业中起到很重要的作用，甚至还有什么流行色之类的说法。色彩同时也是最具争议性的，因为每个人对色彩的喜好不同。色彩对游戏来说是至关重要的。早期的游戏由于硬件限制只能允许同屏显示 16 色或者 256 色。游戏设计者们必须反复斟酌取舍，才能把自己想要表现的效果展现出来。那时的游戏美工，在设置调色板方面要耗费大量时间精力。

当时的一个矛盾就是在分辨率和色彩之间要做出取舍。由于 PC 硬件的限制，分辨率和色彩不能兼得。设计者或者取高分辨率（640X480）加小调色板（16 色），或者取低分辨率（320X200）加大调色板（256 色）。日本的美工采取了前一种策略。他们别出心裁地在 16 色系统下使用了色点间隔法。就是在比较高的分辨率下（640X480），把不同颜色的点混合排列，看上去就成了中间色。这种利用人眼的不足来进行视觉欺骗的方法大获成功，很多日本游戏都在 16 色系统上显示出了超出人们想象的华丽效果。而同时期的美国游戏，则舍分辨率而取色彩，使用比较小的 320X200 或者 320X240 的分辨率，却拥有 256 种色彩可以选择。

图 11-1 《美少女梦工厂 2》只用了 16 色，但使用了色点间隔法，视觉效果华丽。很多日本游戏都使用了这种方法，在 16 色系统上显示出绚丽的画面来

随着硬件机能的提高，这种窘迫的状况一去不复返了。现在的游戏可以使用 8 位以上的真彩色，设计者们获得了前所未有的自由。但这种自由是不能滥用的！给游戏配置合适的色彩，仍然是一项艰巨的任务。游戏设计者们想出了浑身解数，其中不乏新奇之举。比如说在一堂设计课上，老师曾经让同学们去找一片树叶，然后扫描，提取其中的色彩作为调色板。

图 11-2 大自然的色彩，于多样中孕育着和谐。采下一片枫叶，用扫描仪扫描到计算机里，可以获得复杂的色彩层次和变化。把这些扫描到的颜色作为网页设计的调色板，整个网页将给人极为自然和和谐的感觉

又如 Sid Meier 的《文明 3》，使用的是一幅文艺复兴时期的名画作为调色版的根据，使得整个游戏有一种油画的厚重感。

一般来说，选色应该遵循少即是多的原则。色彩过多过滥，反而会使得画面凌乱琐碎。限制一定的色彩数量，可以使画面达到某种和谐的效果，对比鲜明，重点突出。《Emperor:

Battle for Dune》就是一个很好的例子。

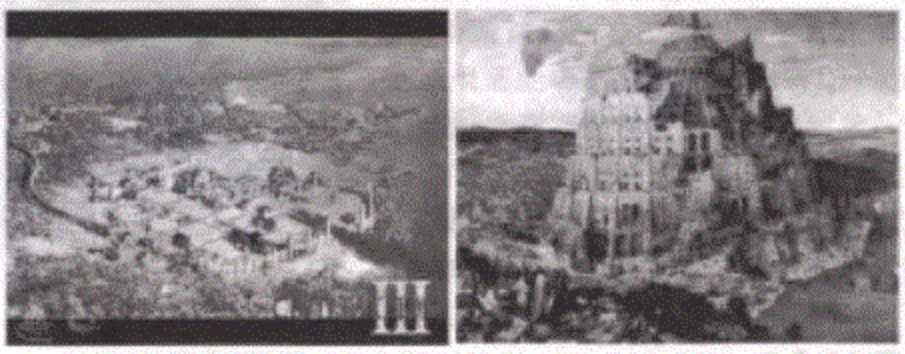


图 11-3 左边的是《文明 3》的图像。右边的是文艺复兴时期 Pieter Bruegel 的名画《巴比通天塔》(The Tower of Bagel)。巴比通天塔的故事出于《旧约创世纪》,讲述人们妄想修建一座高塔,以通到天国,后来触怒了上帝一事。《文明 3》的调色板就基于这幅画,甚至其视觉风格都有诸多共同之处

图 11-4 《Emperor : Battle fOr Dune》是一个视觉风格与众不同的游戏。它是那种能够让人眼睛一亮说:"以前从没见过这种效果"的游戏。其秘密就在于其调色板的设定和半透明的人机界面。以左图为例:暗粉色的背景,蓝色的作战单元,半透明的蓝紫色的界面 这三种颜色构成了游戏画面的主色调(游戏业一种说法是画面上主色不能超过三种) 它们在色环上邻近,过渡自然,没有跳跃感。整个画面给人一种和谐明澈的感觉 形成了现实世界中所没有的、如梦如幻般的效果。

针对不同人对色彩的喜好不同,有些游戏更进一步提供了多种色彩配置方案。比如《最终幻想》系列的界面,预设的蓝底白字,玩家可以根据自己的喜好修改。但这种方法并不流行,因为很多情况下玩家缺乏自己调整色彩配置的勇气。另外,他们也并非专业设计师,选择一种自己喜爱的色彩是容易的,但设定一整套色彩方案对他们来说是太难了。于是设计师们就根据不同基色来配置了不同的色彩方案,供玩家选用。最简单的例子,就是 Windows 操作系统的外观设置,如图 11-5。

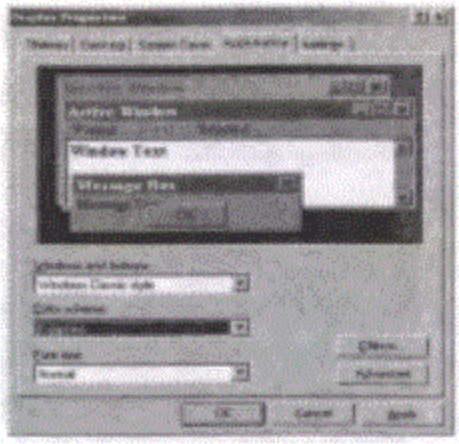


图 11-5 Windows 系统的外观设置窗口。用户可以选择不同的色彩配置方案

色彩系统

色彩系统(color systems)的作用是系统化地精确量化定义色彩,从而使得辨识和交流成为可能。在此基础上才能够进行工业化的合作开发。色彩系统在印刷方面,在平面设计和工业设计领域中,都起到极其重要的作用。

游戏设计者所接触的色彩系统,主要是 RGB 和 CMYK 两种。计算机显示器和电视机所使用的是

RGB 色彩系统，即三个阴极射线管发出红、绿、蓝三种颜色的光，当三种最强的光汇聚在一起时，就是白色；反之为黑色。RGB 色彩系统因此又被称为增色系统。而 CMYK 色彩系统主要用于印刷业，其中 C 代表青色，M 代表洋红色，Y 代表黄色，K 代表黑色。

CMYK 是一种减色系统。游戏设计者一般主要要和 RGB 色彩系统打交道，而在设计游戏相关资料的印刷品时才需要用到 CMYK 色彩系统。

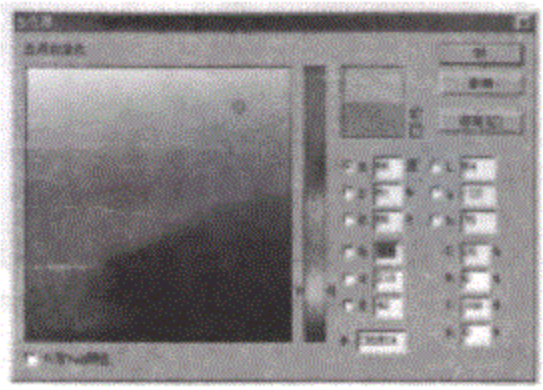


图 11-6 Photoshop 的色彩选择窗口，显示了几种不同的色彩系统的数值

色彩的作用

色彩可以表达感情

色彩可以表达感情——红色使人兴奋，蓝色使人平静。通过屏幕上的色彩，可以有效地影响观众的情绪。电影《Woman in Love》就是一个很好的例子。影片开始的时候，使用高度饱和的原色和互补色，给人一种冲突和不协调感，来表现片中人物所过的肤浅的城市中产阶级生活。随着故事的发展，当男女主人公在乡间找到了爱情后，田园般的柔和色彩便占据了屏幕，给观众们以温馨的感觉。

色彩因文化而差异

色彩因文化而差异。比如在中国结婚穿红，在西方结婚穿白；在中国丧礼穿白，在西方葬礼穿黑。诸如此类的例子数不胜数。

具体到游戏方面，美国游戏用色厚重，使用混色多，有时感到昏暗污浊。日本游戏一般比较明快，色彩比较纯。这也是传统使然——美国游戏和日本游戏在色彩和其他视觉因素上的差异和几百年前西方油画和日本版画的差异是一样的。

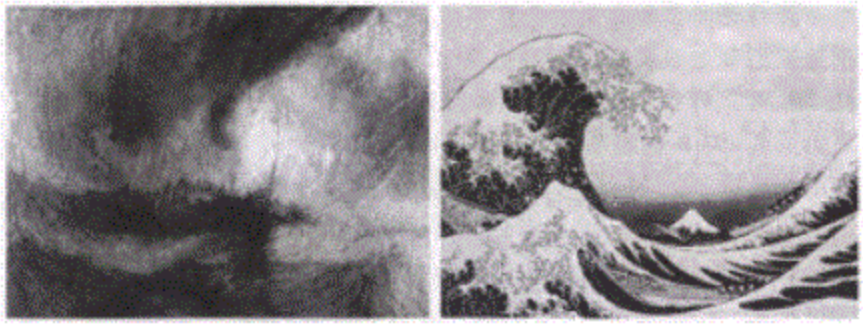


图 11-7 左边的是英国画家透纳（Turner）的《暴风雨中的汽船》，右边是日本葛饰北斋的《神奈川冲浪》。同样画大海巨浪 两幅画的色彩运用和视觉风格完全不同

中国的国画中，色彩运用也有独到之处，比如说传统的青绿山水和浅绛山水，都具有自己独特的色彩配置，实现了特别的视觉效果。近人王叔晖的仕女画，也很值得研究。她的画的色彩配置有其特色，形成一种明丽淡雅的风格，在任何其他绘画中找不到这样的效果，如果能够借鉴并应用于游戏中，必然会产生和现有的美日游戏迥然不同的效果。

色彩可以被用来引导玩家的注意力

在一幅黑白为主的画上有一点彩色，这个彩色的区域就会马上吸引人们的目光。在一幅彩色的画上，人们的目光会最先投向最亮（饱和色）的区域。齐白石画白菜，在旁边用曙红色点两个辣椒，整幅画立刻就活了起来。而马路上的停车标志牌是红色的，目的是在周围的各种颜色中仍能保持醒目。

图 11-8 红色的停车标志牌

图 11-9 美国影片《乐乡幻境》(Pleasantville)中，使用计算机技术进行了后期处理，在同一个画面中混合了黑白人物和彩色人物。由于观众的目光会自然地投向色彩部分，为了使得在前景的黑白主角能够成为主导，需要细心地选择并调节背景人物的色彩，使得背景人物不那么突出

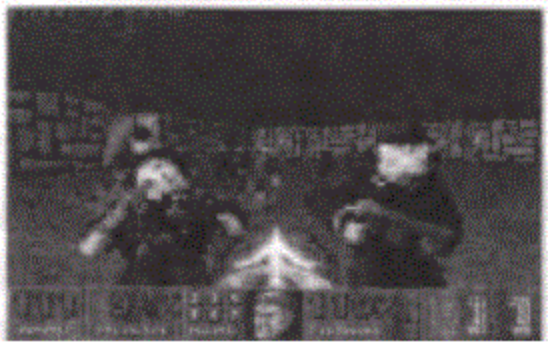


图 11-10 一个很老但是很恰当的例子：在《Doom2》中，灰暗的背景上，血值等最重要的数字用红色来显示，敌人用亮蓝色，玩家在进行游戏时，对两个最重要的信息——与之对抗的敌人和自己的状态，都了如指掌

在游戏中，色彩可以被用来引导玩家的注意力，突出重要的内容和信息。比如图 11-10 的例子。

色彩配置技术

游戏中的色彩设置，首先要考虑的是究竟先设定游戏人物的色彩，还是先设定背景的色彩。无论先设定谁，两者都无可避免地有可能冲突，需要在设计过程中不断修改。

背景的色彩配置，以动作游戏为例来简要说明一下。动作游戏一般有一系列关卡，每个关卡都有自己独特的任务和敌人，更有自己独特的环境、建筑、物品。每个关卡更应该有自己的独特的主题和色彩设置。

一般来说，一个关卡的调色板有 2～3 种主色彩就足够了。然后在这 2~3 种主色彩的基础上进行深浅明暗的变化，并加以日夜雨雾等效果。

虽然在计算机中使用软件可以随意地变换颜色。但一开始进行设计的时候，很多设计师还是喜欢用传统的方法，用画笔和颜料，绘制大色块效果图。所谓大色块效果图，就是用粗大的笔触，刷出关卡的大效果。忽略各种细节，只考虑 2~3 种主色彩和其层次变化。大色块

效果图，一般被用来试验关卡的色彩配置，获得直接快速的反馈和意见。

各关卡中的物品和财宝，一般用亮色，并且可以使它们不时闪耀发光，使得玩家能够比较容易地找到它们。

使用色彩来标识某些特定功能区域，也是一种常用的方法。这种方法在平面设计和网页设计方面也很常见，就是使用某种色彩去标识某项功能，使

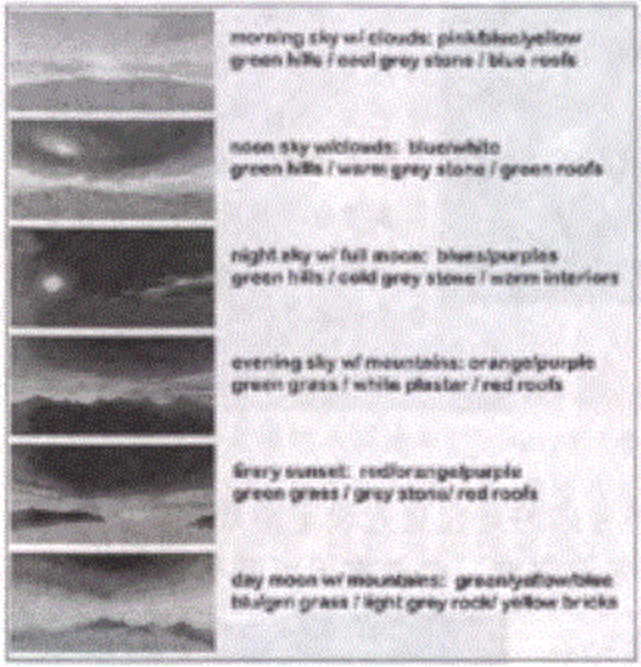


图 11-12 在《小恐龙 Spyro》的设计过程中，设计者绘制了各关卡的大色块效果图，悬挂在墙上，供人们品评

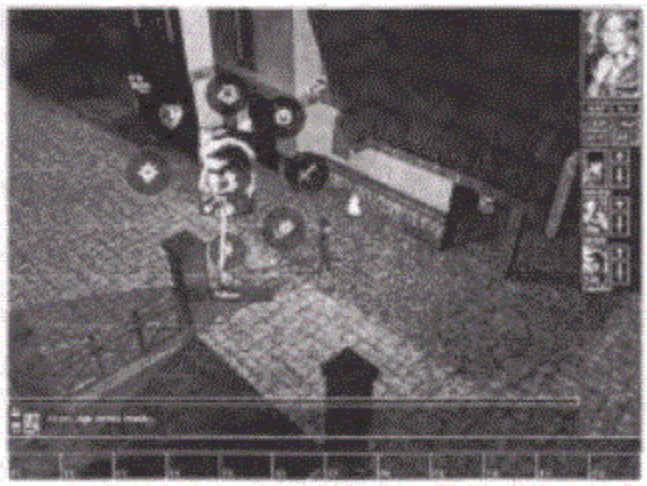


图 11-13 《Neverwinter Nights》的半透明人机界面，最大程度地减少了界面对游戏世界的干扰

图 11-11 小恐龙 Spyro 的身体原来定为绿色，但设计者后来发现当它在绿色草地上跑的时候容易和背景混在一起，于是就把它的身体改成了紫色

得用户（玩家）在一定时间后形成色彩→功能的条件反射。这样当他进入一个新的网页或者关卡时，可以马上找到相同功能的单元或者物体。

人机界面的色彩是一个比较棘手的问题。因为人机界面实际上不属于游戏世界，它是浮游于游戏世界之上的一层，如果色彩设定得不好，对游戏世界是个干扰。更棘手的问题是各个关卡的色彩配置不同，但界面的色彩是不应该改变的，这样就有如何使得界面的色彩配置和所有关卡的色彩配置都协调一致的问题。目前比较流行的半透明的人机界面，最大程度地减少了界面对游戏世界的干扰，是比较好的解决方案。

第十二章　迪斯尼经典动画理论

在三维图形大行于世的今天，介绍 70 多年前由 Disney 公司所创立的针对二维手绘动画的传统动画理论似乎有些过时。但实际上，这套理论仍然是现在一切形式的动画片制作时的最根本的指导法则。无论是二维的还是三维的，手工绘制的还是计算机生成的，要想达到真实生动的动画效果，就必须在一定程度上遵循这些法则。作为一个自成体系，经过长期实践考验的理论系统，Disney 动画理论的重要性是无可替代的。

图 12-1 三维动画《玩具总动员 2》（Toy Story 2）。其制作流程仍然遵循传统动画片的制作流程，并在前期采用了很多传统动画片的制作工具和方法，比如故事板

对游戏美工来说，虽然不是专业动画师，但也需要对这套理论有一定的了解。其原因有三：第一，动画和游戏有很密切的关系。游戏中广泛使用了片头动画和过场动画。对中小型公司来说，如果没有专业的动画师参与，这些片头动画和过场动画必然要由游戏美工来完成。如果缺乏对动画理论的认识，必然导致片头动画和过场动画的低劣乏味。而目前的游戏市场上，这种劣质的片头和过场动画比比皆是。第二，游戏的实时画面中的角色动作，除了使用成本比较高的动作捕捉技术（motion capture）外，很多情况下还是需要由美工来手工完成其设置。如果能够了解传统动画理论中的有关法则，可以使得角色动作更加生动真实。第三，虽然目前游戏的主流是三维，但二维是三维的基础，传统动画理论是计算机动画的基础。只有了解了二维动画的规律，才能制作出高水平的三维动画来。

Disney 动画理论的核心是 12 条法则。我们先从历史的角度来看这些法则的由来。

传统动画理论的建立

在 20 世纪 20 到 30 年代的时候，动画短片开始盛行起来。这些几分钟的动画短片一般是在无声电影上映前插播放映，起着点缀和调剂的作用。这时候的动画，是一种新奇的玩艺儿，而不是一种成熟的艺术形式。动画师们对动画独特的艺术规律没有一个理性的认识，动画的技法粗糙，品质没有保证。比如那时的动画师还不知道如何真实地描画肢体的动作，因此产生了橡皮管动画（rubber hose animation）这种效果——动画角色的四肢成了可以任意伸缩的橡皮管。

随着动画制作的规模越来越大，人们开始尝试制作更复杂的题材和更长的动画电影。显然再画这种橡皮管动画是不行了。Disney 意识到总结动画艺术规律的迫切性。于是，他先是和洛杉矶的 Chouinard 艺术学院合作开设了一些课程，用来培训 Disney 公司里的动画师，并研究动画艺术的独特规律。由于动画艺术本身的规律并不清楚，这种课程在一开始的时候实际上相当于研讨班。Disney 的动画师去那里交流思想，通过观察模特的动作，并反复播放电影片段，来研究如何更真实地表现角色动作。通过集思广益，越来越多的规律被整理发掘出来，专业术语被标准化了。当这些 Disney 的动画师们回到公司后，一方面通过实践对这些规律进行了充实和修正，使得它们更加完善；另一方面开始系统化地对新来的动画师进行了培训和指导。经过了一段磨合的时期，Disney 公司的传统动画理论正式地问世了。这个理论的核心是 12 条法则。它们成了指导 Disney 公司所有动画制作的理论基础，并进一步影响了整个美国动画界，成为美国动画片的实际标准。从 Disney 的第一部动画电影《白雪公主和七个小矮人》直到 80 年代的《小美人鱼》，Disney 公司所有的十几部动画片，无论题材如何变化，角色如何多样，从动画技术角度讲，都是一样的，都是遵循 30 年代确立的动画理论。

图 12-2 橡皮管动画

在 80 年代中后期计算机三维动画问世后，一开始由于搞三维动画的人中技术人员占绝对多数，他们对电影基本技术和二维传统动画理论所知甚少。因此这时候的三维动画更多的是一种技术上的尝试和炫耀，而不是一种艺术表达。这种情况一直维持，直到 Pixar 公司开始在其动画短片中尝试采用了传统动画的工作流程和动画理论，包括故事板的使用和 Disney 的 12 条法则，才使得三维动画真正进入艺术的殿堂。Pixar 公司的人员还专门撰写学术论文，以 Pixar 的三维动画短片为例子，阐述二维传统动画理论在三维动画中的应用。而以《玩具总动员》系列为代表的三维动画电影大片，虽然其技术令人瞠目结舌，但其整个动画的基础也还是 Disney 的传统动画理论。

游戏中的短动画

Disney 传统动画理论中的 12 条法则，其根本目的就是探讨如何生成真实自然的人物动作。因为动画和真人电影的不同之处就在于真人电影由真人来表演，演员通过肢体语言和表情来表达人物。而动画没有真人演员，所有动作和表情都需要由动画师在白纸上画出来。而其他方面，如镜头剪辑等技术，真人电影和动画电影基本差不多。因此动画的独特的艺术规律，主要就体现在如何真实生动地描绘人物的表情和动作。我们后面介绍 11 条法则时读者们就会发现它们基本上都是处理人物动作的问题。

对于游戏来说，游戏的实时画面中各种角色的动作实际上是一段段事先设置好的短动

画。比如说在 RPG 游戏中的对战，当玩家下达指令使用某项魔法后，游戏程序播放一段事先设置好的动画，显示角色的释放魔法的动作，和敌方挨打之后的反应。这实际上就是两段预先编制好的动画。而更简单的例子，比如在一个第三视角的游戏中，按下方向键，角色前进一小步，从按键下去到动作停止，这个移动的过程也是一段事先编制好的短动画。这些短动画短的不过几秒，长的不过一分钟，但它们是构成整个游戏中人物角色各种动作的基础。只有通过这些细致的动作，才能彰现游戏人物的个性，使得游戏角色不只是一堆多边形，而成为有血有肉的呼之欲出的人物。

既然游戏中的人物动作短动画如此重要，而 Disney 传统动画理论又是专门讲人物的动作问题的，那么显然 Disney 传统动画理论对游戏可以起到一定的指导作用。

从历史的角度看游戏中的短动画。在早期 PC 和游戏机上的二维游戏中，人物动作的短动画和动画片中一样，都是绘制许多帧静止画面，然后连续播放形成。但游戏中的短动画十分原始，比动画片中简单得多。首先因为硬件机能的限制，游戏中的人物不可能像动画片中那样精细，分辨率比较低，色彩单调些。而且因为内存等限制一个动作的帧数要比动画片中的要少得多。早期 RPG 的人物行走动画循环（walk cycle）也就 2 帧到 3 帧，而在 Disney 动画中行走动画循环的关键帧至少是 8 帧。

当然也有极少数例外，比如 PC 游戏中的早期经典《波斯王子》（Prince of Persia）。这个二维游戏中的人物动作极为柔和。其原因就是制作者对二维传统手绘动画片的各种技术比较熟悉，并有意识地应用到了游戏中。甚至还使用了在动画电影制作中才能见到的 rotoscoping 技术。

三维游戏问世后，短动画不再需要一张张绘制，而是在计算机中设置好参数，然后自动生成。也就是说对动画师的绘画本领的要求降低了，但仍需要他们对动画理论有很深刻的理解，才能够调整好各种参数，生成满意的动画效果。

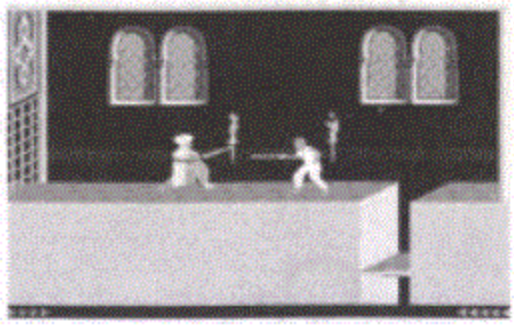


图 12-3《波斯王子》借鉴了二维传统动画，并使用了 rotoscoping 技术，使得游戏角色的动作极为柔和自然。所谓 rotoscoping 技术，就是先用摄像机把真人的动作拍下来，然后用一种专用的投影仪把影像一帧帧投射到动画师的玻璃画板背面。这样动画师可以把真人的动作描下来，达到极好的效果

三维游戏中又有真三维和伪三维游戏之分。伪三维游戏，如《帝国时代》，是先用 3D Studio Max 做出三维模型，然后设置参数，渲染后输出为短动画。其本质还是动画片段的播放。而真三维游戏，不预先渲染生成短动画，是实时计算然后改变三维模型的位置和形状。也就是说短动画不是预先渲染好然后被线性播放，而是以各种参数的形式存在，并被实时处理。

12 条法则

下面逐条介绍并图示 Disney 传统动画的 12 条法则。

挤压和拉伸

挤压和拉伸是 12 条法则中最重要的也是最基础的一条，同时它也是美国动画和日本动画的主要区别之一。美国动画片中挤压和拉伸的效果更明显更夸张。我们知道在现实生活中有些物体在运动时是保持形状不变的，比如桌椅、汽车等。它们在现实生活中形变就很微小。而任何有生命的物体，不管骨架如何，肌肉组织如何，在运动中都会出现一定的形变。比如当人弯起胳膊时，上臂变粗。这些形变在动画中得到了夸张的体现，就是挤压和拉伸效果。

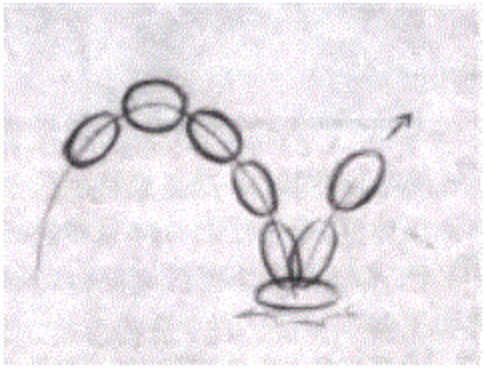


图 12-4 最简单的跳跃小球，有些画鸡蛋的感觉，简单但是有内涵

在所有的动画学校里，第一堂课都是学画跳跃的小球（bouncing ball）。你可能会说，这还不简单？可是在简单的跳跃小球中，体现出来的就是 12 条法则为首的最重要的挤压和拉伸法则。如图 12-4 所示。

跳跃小球有两个要点：第一，为了显示正确的重力加速度，在小球跳跃的最高点附近间隔比较密，下落弹起过程间隔比较疏松；而第二点就是使用挤压和拉伸，在小球下落和弹起的中间阶段拉伸小球，这样可以显示速度感。而在接触地面的时候挤压小球。这样小球才显得有弹性，可以再次跳跃得起来。这就是挤压和拉伸的基本使用。

在跳跃小球的基础上，我们再来看看如何处理一个下落然后摔在地上的角色。如图 12-5 所示。这样的人物就要自然生动得多。

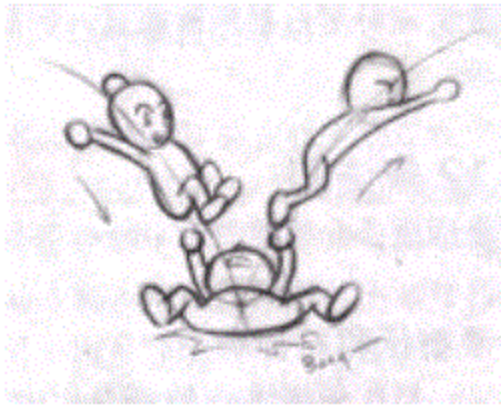


图 12-5 落下和弹起的时候纵向拉伸，落地的时候横向挤压

在使用挤压和拉伸效果的时候，特别需要注意的一点就是要保持物体体积不变。这就意味着当物体在一个方向上受到挤压时，在交叉的方向会出现拉伸的效果。就像一袋装满大米的口袋，把它垂直向下丢在地板上后，口袋的高度会降低，中部的宽度会增大。如果不保持物体的体积，只沿一个方向进行挤压或拉伸，将给人以物体突然缩小或者胀大的感觉，是不正确的。

拉伸效果的另一个作用就是沿运动方向进行拉伸可以增强速度感。

早期的三维游戏的角色是基于骨架结构的，没有考虑到肌肉的挤压和拉伸，结果做出来的行走战斗等动作都非常僵硬，如机器人一般。随着三维图形技术的进步，可以表现更复杂的效果。比如 SEGA 的 VR 战士，已经能够在一定程度上表现肌肉的变形了。

预示

人物角色的任何动作，都可以分为三个阶段：动作的准备，动作本身，动作的终止（实际上这三个阶段是人类长期观察自身运动的规律所得到的常识，比如明朝将领戚继光在他的兵书中讲述格斗要领，就提到三个字“起”、“当”、“止”，实际就是动作的准备、动作的本身、动作的终止）。预示（anticipation）处理的就是准备动作的问题。

在动画中使用预示有两个原因。一是生理上的。因为肢体运动需要靠肌肉的收缩来完成，而肌肉要进行收缩的话必须先放松，也就是说肌肉的放松是肌肉收缩的准备动作。二是心理上的。如果角色要做一个动作，应该在做动作之前给观众一个明确的提示，给观众一个心理准备的时间。这样接下来的动作对于观众才是可接受的不突兀的。预示就是起到了这个作用。

预示可以是微妙的表情变化也可以是明显的肢体动作。比如当一个角色开始跑之前，他会先做好准备姿势，持续一小段时间，然后冲出。有些动画中就会更夸张，在真正冲出之前要原地踏步很长时间。又如在角色踢人之前，总要先向反方向使劲抬腿，蓄足力气然后踢出。可以看出，在现实动作中很少有动作是不需要准备动作的，而且大多数准备动作都是反向动作，在跑之前要反方向，蓄力，在跳之前要下蹲，用拳打击的时候要先收回手臂。

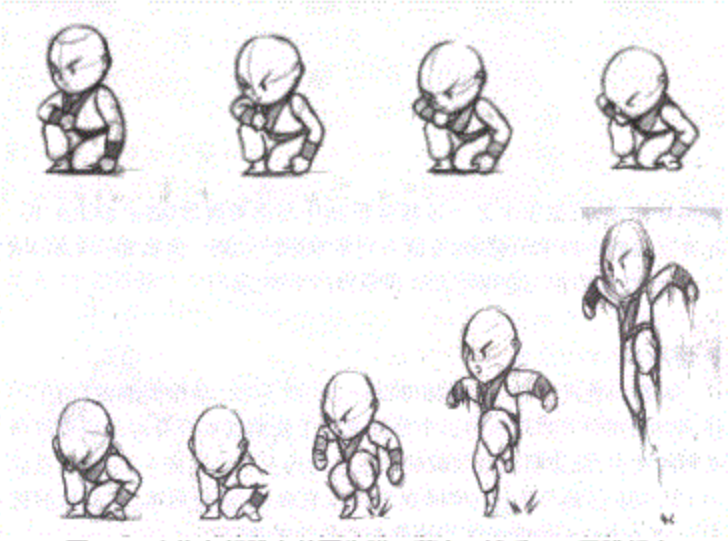


图 12-6 人物在起跳之前要先蹲下蓄力，然后一下子跳起来

预示和动作之间的关系要遵循以下的准则：如果动作本身很快，则需要比较长时间的和比较大的预示效果。如果动作本身很慢，则预示效果不需要太长太明显。其原因是显而易见的：如果动作很快，一下子就过去了，观众很容易忽略掉。所以必须用长的预示效果来让观众有所准备，才能观察到动作的一瞬。而如果动作本身很慢，观众不会轻易忽略掉这个动作，预示就没有必要做得很长了。

格斗类型游戏的人物动画就很注重预示。比如《街霸》系列中，飞行道具的代表波动拳，准备动作就是双手蓄气于腰部，持续一小段时间，然后再推出。甚至《街霸》的操作也借鉴了这条法则，尤其是蓄力型人物，先后向蓄力，才能向前发出招式。

展示

所谓展示就是要完整清楚地表达你想要表达的意思，把需要明确告诉观众的事情明白无误地展示在他们面前。这个概念是从话剧中得来的。在话剧中，人物的动作是被精心地设计并展示到观众面前的，人物的性格也是被精心地展示到观众面前的。

展示最重要的一条就是必须有意识地操纵观众的注意力，引导他们的目光投向你要他们注意的地方。在一个静止的场景中，观众的目光将被运动的物体吸引。而在一个很多物体运动的场景中，静止的物体能够得到观众最多的注意力。又如当一个新角色登台亮相之时，所有观众的目光都会集中到他身上。这时通过人物的表情和动作，可以展现人物的性格特点，从而让观众领会人物的心情。如图 12-7：

图 12-7 Ken 的招牌动作

在操纵观众的注意力的时候要注意在同一时间只能有一个重点。如果出现多个重点，观众的注意力会被分散。在话剧中，当一个角色正在说话的时候，其他角色不会有太大的动作。因为，如果其他角色动作频繁的话，观众的注意力将被分散，正在说话的角色所要表达的重要内容将被观众忽略。

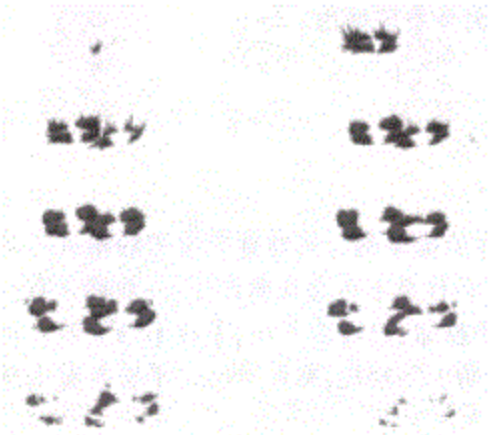


图 12-8 东西落下溅起灰尘就是非关键帧动画

非关键帧动画和关键帧动画

所谓非关键帧动画，就是在画动画的时

候，从第一张开始，一张挨一张的画，直到结束。动画师随着感觉一直画下来。

关键帧动画，动画师会在一开始设计好人物的几个关键的姿势，即关键帧（keyframe），然后再根据动作的快慢在关键帧之间加入一些中间帧。在关键帧之间加入中间帧的工作叫做 inbetweening。

图 12-9 一个武打序列的若干关键帧

这两种方式各有千秋。在早期的动画片中一般采用非关键帧的方法，因为那时候的动画还是一种自由散漫的个人探索。而随着动画越来越复杂，越来越商业化。很多人共同制作一个动画片，带来了分工合作的问题。关键帧的方法就取代了非关键帧的方法。使用关键帧的方法，容易对动画，包括人物动作、姿势、镜头等进行严格的控制，也为分工合作带来了可能。可以让有经验的动画师画好关键帧，然后让水平一般的新手去填入中间帧。目前关键帧的方法成为动画片制作的实际标准，而非关键帧的方法只是在个人制作实验型的动画艺术短片（experimental animation）时才有所使用。

三维动画都是关键帧动画。

跟进和重叠运动

图 12-10 轻拳转身踢，注意人物转身时候的动作

前面介绍的预示是关于动作的准备的，而现在要介绍的跟进运动则是用来处理运动的终止。任何动作都不可能戛然而止，一般都要有一个动作终止的过程。比如棒球手投球，当球出手后，手并不马上停止运动，而是要沿原来的轨迹继续运动一段距离，然后停止。手臂在抛出球之后的运动就是跟进运动。

当一个人物运动的时候，其身体各部分并不是同时同步运动的。总是有一些部分先开始运动，一些部分后开始运动。比如说人物的转身动作，先是腰部动，然后带动上肢，然后带动肩膀。也就是说腰部运动是主导

（lead），而其他部分的运动是跟进（follow）。当停止一个动作，准备开始下一个动作的时候，也不会是所有部分同时停止，而是有的部分先停止，有的部分后停止。一个人物的各个部分之间运动的不同步就是跟进和重叠。

角色身上的柔软宽松的部分和附加装饰物，比如兔子的长耳朵、女孩的长头发、围巾等，这些东西启动慢，制动也慢。当身体其他部分已经开始运动后，它们还落在后面。而当身体的主要部分停止运动后，它们的运动还要保持一段时间。滞后的时间由其重量决定，重量越大，滞后时间越长。

图 12-11 人物转身后头发的摆动

显然，在动画中使用跟进和重叠运动将使得动画更加真实生动。当然这同时也带来了更大的难度，动画师们需要做更多的工作。

慢进慢出

慢进慢出处理的是两个关键帧之间的中间帧的步距问题。在关键帧动画中，动画师花了极大的努力去刻画角色在一个个关键帧的姿势（pose），自然希望让观众明确地注意到这些重要的姿势。如果在加入中间帧（inbetweens）的时候，均匀地加入中间帧，那么关键帧的重要性将被削弱，关键帧就不突出了。因此，要在关键帧附近加入比较多的中间帧，而在两个关键帧的中间位置加入少量的中间帧。这样关键帧和其相邻帧所表现的姿势在屏幕上停留的时间就长些了。这就是对于关键帧的慢进慢出。

图 12-12 伸手的动作，在开始和结束的位置动作比较密

弧线运动

生物体的动作很少是机械的沿直线的。因为骨骼的运动一般是轴运动，所以生物体的动作都是若干轴运动的组合，其轨迹都是圆滑的呈弧线的。比如在人行走时，头和躯干就

是按照波浪线在前进过程中上下移动的。在动画制作中，通常会先画出弧线然后在弧线上标出关键帧的点做定位，然后画出的人物动作就会很自然流畅，不僵硬。

图 12-13 马的全身关节的运动都呈现曲线状，图中画出了头部曲线

辅助运动

动画角色在做一个动作的时候，除了体现他主要意图的主导动作之外，加上一些辅助动作可以增强角色的表现力。当然这些辅助动作是要以帮助主导动作体现人物个性和意图为唯一目的，不能与主导动作发生冲突。一些简单的辅助动作如一个悲伤的角色用手抹掉眼泪，一个受到惊吓后狼狈不堪的角色扶扶眼镜企图稳定情绪，起床的时候一定要在床上先伸伸懒腰等等。这些动作并不是主体动作，但是就是这些小动作才能把角色表现的活灵活现。

在动画中加入辅助动作的难点在于把握好分寸。辅助动作既要能够增加动画的生动感，又不能喧宾夺主，妨碍主体动作的表达。一个从实践中总结出来的行之有效的绘制辅助动作的方法如下：第一步，先要按照故事的需要将角色需要做的主导动作准确地画出。第二步，在第一步的基础上，按照人物的性格和当时的环境，给人物加上各种合适的辅助动作。因为在第一步已经准确定位和画出了主导动作，所以在第二步修改的过程中加入辅助动作的时候就不会出现主导动作走形或者不突出的问题。第三步，不断地修改，并且调节其他相应的部分，使得辅助动作和主导动作协调一致。

时间控制

时间控制是动画的灵魂。它决定了动作的速度。而动作的速度决定了动作的涵义，决定了观众是如何理解这个的动作以及这个动作所要表达的意图。速度同样反映了物体的重量和大小，甚至能够表达人物的情感。

动画是以均匀的速度（每秒 24 帧）播放一系列静止的图像。因为播放的速度是不变的，动作的快慢只能用加入图像的多少（帧数）来控制。所谓时间控制，就是在播放速度固定（每秒钟帧数确定）的情况下，通过控制在两个关键帧之间加入中间帧的数量，来表现不同的动作。

同样的两张关键帧，在中间加入的中间帧数量不同，就会导致动作的巨大差异。一个最经典的例子就是 Disney 公司最著名的动画师 Frank Thomas 和 Ollie Johnston 在他们的书里所举的摇头动作：两张关键帧，一张画着一个人头摆向右边，第二张画着他的头摆向左

边。然后我们在这两张关键帧中间加入中间帧。我们可以看到，随着每一张中间帧的加入，这个摇头的动作就有了新的涵义，所要表达的意思完全改变。下面列出了不同中间帧数所产生的各种动作。

●没有中间帧：这个人的头被置于其耳边的大炮所轰击。
●一张中间帧：这个人的头被人用炒菜锅扇了一下。
●三张中间帧：这个人在躲避飞来的砖头。
●六张中间帧：一辆豪华跑车从这个人眼前驶过，他转过头去，目光追逐着跑车。
●十张中间帧：这个人脖子崴了，他正在努力地慢慢活动他的脖子。

夸张

夸张是美国动画片的特点之一。相比之下，日本动画片中的夸张不像美国动画片中那样明显。在美国动画片中，如果人物需要悲伤，则痛哭流涕得感天动地；如果人物需要开心，就要张开嘴哈哈大笑前仰后合；如果人物被惊吓，就要吓得连眼睛都凸出来头发倒竖。除了这些对现实世界中的原型动作的夸张外，还可以生成很多现实中无法做到的夸张的效果，比如被雷击中后的角色化成粉末，受到打击的角色变成石块然后碎掉，受到惊吓的角色连自己的心脏都从嘴里跳出来说：“吓死我了！”等等。

为什么要在动画中使用夸张呢?夸张的目的是要强化人物的动作和表情，从而把人物的心理和情感状态表达出来。因为动画角色无法像真人演员那样有很细腻的表演，要把情绪表达出来，必须用夸张的动作才能让观众明白发生了什么和角色的反应。

立体感

立体感体现在两个方面。一是背景要符合透视原理。在 Disney 动画片中，虽然背景的街道房屋是经过一定变形的，但还是要遵循基本的透视原理。第二是对角色来说，在从各种角度表现他的各种姿势的时候，必须让人信服，让人感到他是一个有血有肉有重量感的三维的角色，而不是二维的纸片。

图 12-14 3/4 视角的人物走动，始终控制在长方体内

吸引力

所谓吸引力，并不一定指酷男靓女，而是说动画角色应该具有吸引观众的独特的个性和外表。吸引力取决于独特的造型设计，富于变化的表情，具有活力的动作等等，以及一切可以抓住观众目光的元素。

第十三章　三维建模

接下来两章将介绍关于三维动画的内容：三维建模和三维动画。理论上来说，模型的存在是制作动画的基础，不过在实际的操作中，三维建模和动画制作往往是环环相扣同时进行的。因为现在的建模技术还不是特别完美，所以在调试动画的时候会发现模型的缺陷，回过头来重新调整模型是经常的事情。

本章分为两个主要部分，建模和渲染。

三维建模

三维建模是三维动画的基础。三维模型按用途可以分为表面模型（surface modeling）和实体模型（solid modeling）两类。所谓表面模型就是在建模的时候，只创建物体表面而不考虑物体内部，创建出来的物体是一个空壳；而实体模型在建模的时候不只考虑物体表面，而且也考虑物体内部。举个例子，同样是创建一个球体，表面建模出来的是个空心球，而实体建模建造出来的是实心球。

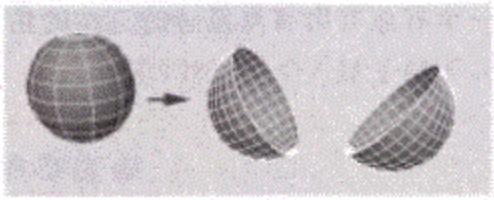


图 13-1 表面模型的球体，切开后是空心的

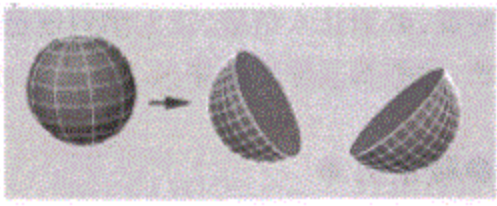


图 13-2 实体模型的球体，切开后是实心的

表面建模主要用于各种三维动画软件，而实体模型主要用于工业软件（主要是 CAD 软件）。

三维建模按常用的技术可以分为多边形建模（polygonal modeling），曲面建模和比较新的 Subdivision。还有一种比较特殊的是粒子模型（Particle-system modeling）。

多边形概念

三个基本元素是点（Point）、线（Line）、面（Plane）。

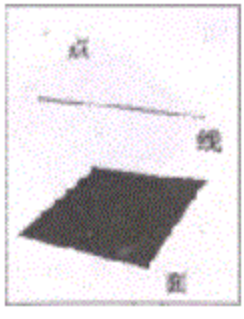


图 13-3 最基本的点线面

最简单的面就是由三个顶点（vertex）构成三角形平面，通过拼接多个三角形平面就可以构成更大更复杂的面。

图 13-4 由三角形平面构成的面

所谓多边形（polygon）就是用顶点定义的面来构成的物体模

型。现在三维游戏中多使用的建模技术就是多边形技术。如果物体模型的多边形越分越细，数量越来越多，就可以用来模拟柔和的曲面，这种技术就叫多边形近似（polygonal approximation）。

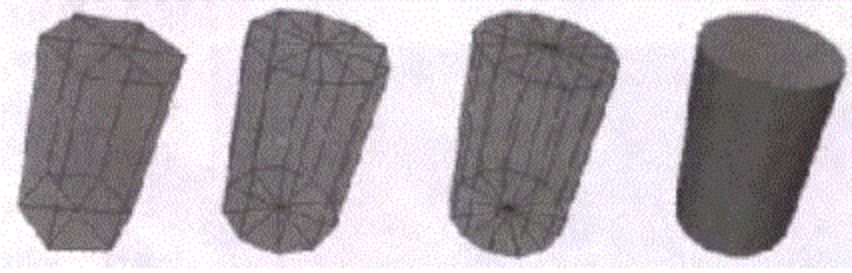


图 13-5 如何用细致的多边形来模拟一个圆柱体的

这种近似方法虽然方便，但是要模拟复杂的物体就需要大量的多边形面，每个面至少要三个顶点，导致数据组织的复杂和数据量的增长。

曲面概念

在现实中曲线和曲面比直线要普遍得多，但是使用多边形近似始终无法完美的表现曲面，所以产生了另一种主要的建模方法曲面建模。一般采取几种方法来绘制曲线。最基本的方法是线性近似（1inear approximation），就是用连续的直线段来近似曲线，类似于多边形模拟曲面。

图 13-6 线性模拟曲线，线段越多曲线越柔和

另外一种就是 Splines，一般高端三维软件都会提供 Splines 曲线的绘制方法。Splines 一词来源于早期制船业，为了使木板适应船体外形的曲线，工匠在有韧性的木条的不同位置用压铁（Ducks）加以固定使其弯曲。

对应三维软件中的 Splines 曲线，木板就是曲线本身，而压铁就是控制点（control point），所有控制点连接的折线段称为外壳（hull）。

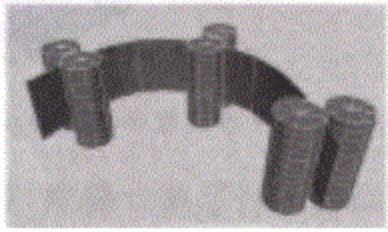


图 13-7 造船业中的 Splines ,圆柱体就是压铁

Splines 曲线有比较多种类，常见的有 Cardinal spline，B-spline，Bezier spline 等等，它们主要的差别在于控制点的位置和如何控制。比较新的是 NURBS（Non-Uniform Rational B-Spline）曲线，它比较好的综合了各种曲线控制方式的优点。

通过两个方向上的曲线就可以构成曲面（curved surface）。如图 13-9 所示，最初的一条曲线在 U 方向上用黑线表示，扩展曲线将最初的曲线在 V 方向上扩展从而形成曲面，可以称为面片（patch）。

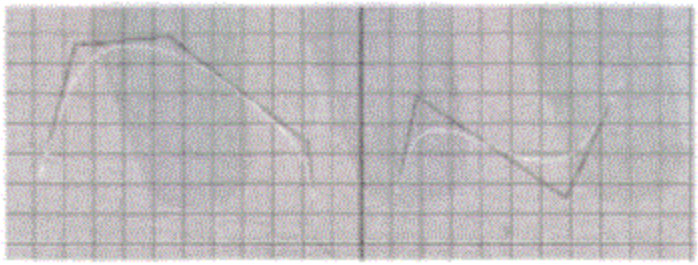


图 13-8 被外壳包在内部的 Splines 曲线和与外壳有交叉的 Splines 曲线

曲面继承了创建其的曲线的性质，比如控制点和外壳，可以通过控制点和外壳方便地来调整曲面形状。

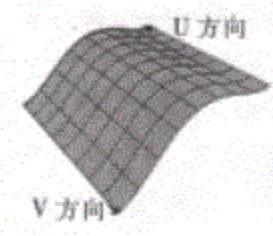


图 13-9 利用 U 方向的一条曲线在 V 方向扩展的面片

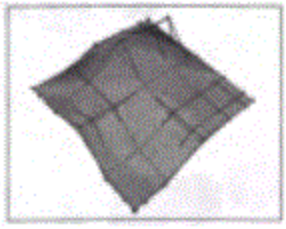


图 13-10 利用控制点和外壳来调整曲面

Subdivision 概念

新出现的 Subdivision 是综合了多边形模型和曲线模型优点、以多边形为基础的建模技术。它的第一次使用是 Pixar 的著名短篇《Geri's Game》中老人的模型（例子!!!!!）。Subdivision 的一个优点就是对应不同细致程度的部分可以有不同的复杂度，类似于将多边形的面逐渐细分从而达到更柔和的曲面模拟。而 Subdivision 又有类似于曲线模型的控制点和外壳，更容易控制表面的形状。可以说是一种非常灵活的建模技术，所以 Subdivision 在三维领域越来越得到普及。最新的 Maya4 中 Subdivision 已经不再附属于多边形而独立成为与多边形建模曲面建模并列的建模技术了。

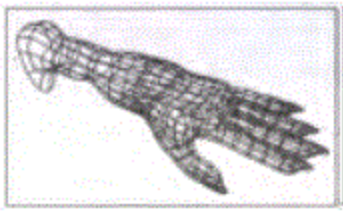


图 13-11 利用 Subdivision 技术构造的手臂

图 13-12 带有多边形外壳显示的 Subdivision 手臂

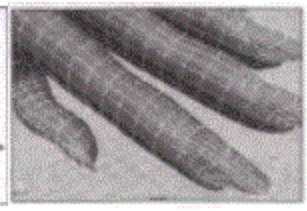


图 13-13 在不同的部分使用不同的复杂程度

常用的建模技术

一般的三维软件都会提供最简单的几种几何体，比如球面（sphere）、立方体（cube）、圆柱体（cylinder）、圆环面（torus）、圆锥体（cone）、去顶圆锥体（truncated cone）、平面（plane）等等。也有比较特殊的，比如 3D Max 就提供著名的茶壶。

图 13-14 多边形建模的简单几何体

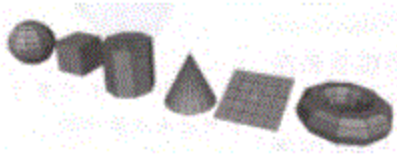


图 13-15 Nurbs 曲线建模的简单几何体

全手工建模（free-form modeling）是最实用最基础的建模方法。首先创建系统提供的几何体，通过基础操作，如调整控制点、边或者外壳的位置，添加、删除或拼接点线面，完全手工一点一点修改直到达到要求，有些类似现实的雕塑。

图 13-16 通过手工调整控制点和外壳来建模原来的脸部就是一个多边形的立方体，简单地修改后转化成 Subdivision，然后再细致地修缮

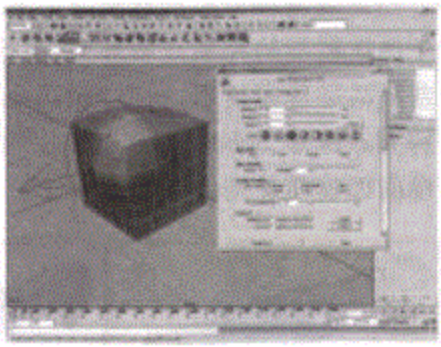


图 13-17 Maya 所提供的雕塑工具

多边形的模拟雕塑（virtual sculpting）是简化了的手工建模。有些三维软件提供基本的雕塑操作，可以对多边形物体进行雕塑。一般用来定物体的大外形，如果要细致调整细节.雕塑工具就显得太难掌握了。

除了对系统提供的几何体进行加工，还有各种各样的方法可以创建初始模型，下面一一介绍。

纺梭面建模（lathed surface），也可称为车床旋转面建模。方法是在二维空间先做出剖面图，再以轴旋转而生成物体。类似圆柱体的物体都可以用这种方法方便的生成。

伸展（extrude）是很有用的建模技术。在二维空间做出剖面图，然后延法线方向伸展从而生成物体，有些类似于纺梭面方法。这两种方法都是从现实车床操作中衍生出来的三维模型技术。

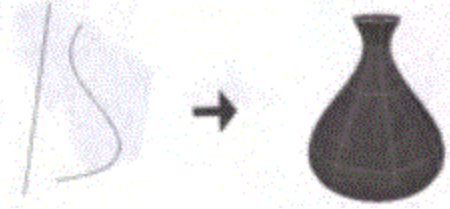


图 13-18 轮廓线绕轴旋转生成瓶状物体

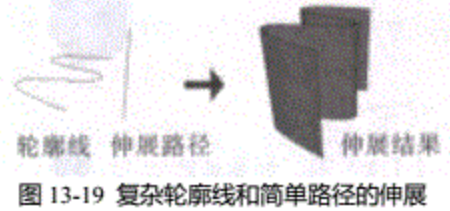


图 13-19 复杂轮廓线和简单路径的伸展

稍微复杂些的伸展建模是将二维剖面延一条曲线路径伸展，可以用来制作像电线绳子一类的物体。

对于多边形物体，伸展技术是修改编辑物体的一个有效方法。因为对于多边形物体的任何一个面，都可以进行伸展操作来制作突出和凹陷。

梯形面（lofted surface）有些类似于复数的伸展技术，也是从二维剖面图伸展出三维物体。在二维描绘不同高度的剖面，然后像梯田一样按不同高度

伸展出来。

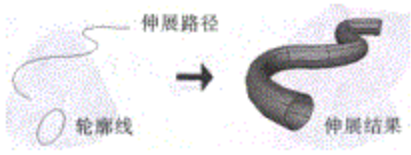


图 13-20 另外一种思路的伸展建模

边界面片（boundary patches）是利用两条或四条曲线作为边界来构成曲面，调整作为边界的曲线就可以调整整个曲面的形状。

图 13-22 使用四条曲线做边界构成的曲面

图 13-21 利用三条剖线伸展出来的曲面

布尔运算（Boolean operations）是主要用于多边形模型的建模技术。包括三种操作、并集（union/addition）、差集（difference/subtraction）、交集（intersection）。布尔运算适合于实体模型，对于表面模型则会将表面模型当作实体模型来运算，之后在需要的位置自动添加上新的面使物体封闭起来。

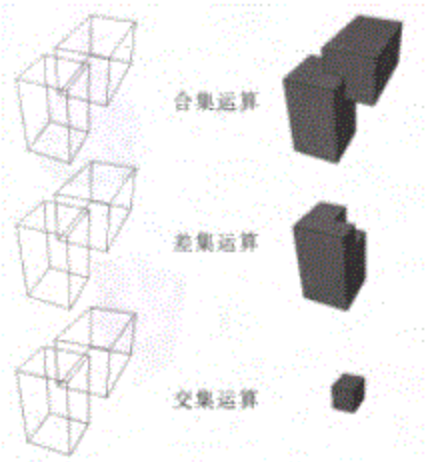


图 13-23 布尔运算，要注意的是差集运算中是有先后顺序的

裁减操作(trims)可以说是对于由曲线定义的曲面的布尔运算。通过对两个曲面交界的计算进行曲面裁减，由交集产生出来一条新的曲线称为交集曲线(curve Of intersection)。但是裁减操作并不会在被剪裁物体上自动补上新的曲面，所进行的一切只是曲面之间的操作，没有模拟实体模型的操作。

有了裁减就要有缝合操作(merging)，两个曲面在各自边界的行数或者列数相同的情况下可以缝合成一个曲面，同样的行或列上的两条曲线也会合并成为一条曲线。在基于曲线的人脸模型中，曲面的裁减和缝合比较常用在眼睛附近和耳朵的建模上，一般会把眼眶附近和耳朵从整个头部模型上剪裁下来，然后缝合上新的曲面作为眼眶和耳朵。

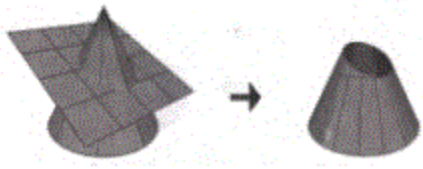


图 13-24 在圆锥体上用平面进行裁减操作，被裁减后的圆锥体上面是镂空的，操作并没有自动地给圆锥体加上一个裁减面

方便的削斜角操作(beveling)和边缘制圆(rounding)。这两个操作都是来处理模型边界的。削斜角操作对应多边形的角点，可以自动在角点位置削一个斜面，把角点变得柔和。

边缘制圆对应多边形的边，在边界附近自动增加一些面让原来棱角分明的边界变得柔

和。

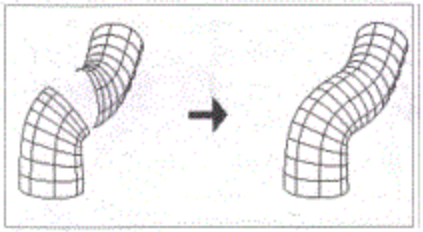


图 13-25 缝合两个边界列数都为 5 的曲面

图 13-26 眼睛附近的裁减和缝合

融合(blending)是一种合并两个表面的操作，并不是像布尔运算的合集操作在交界处直接合并，而是在边界附近融合处自然的曲面。

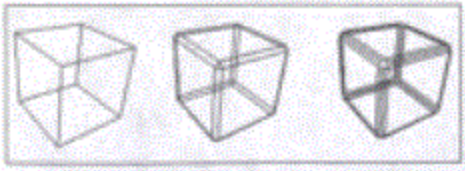


图 13-27 对于多边形立方体不同程度的削斜角操作

图 13-28 对于立方体一条边的不同程度的边缘制圆

变形(deform)是调整物体外形的基本操作，有很多种变形方式，比如斜拉(shear)，锥形放缩(taper)，凸凹(bulge)，各种弯曲(bend)等等，不同三维软件提供的种类各有不同。

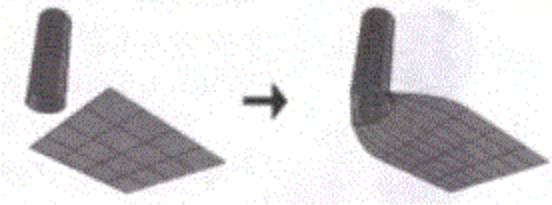


图 13-29 圆柱体和平面的融合，在交界处形成自然的曲面

网格操作(lattices)是变形的延伸，非常灵活实用。在已经建好的模型或者其某一部分的外面包裹一定细致程度的网格，就像用网格将其包裹住，之后只要操作网格进行各种变形操作，模型就会做出相应变形。

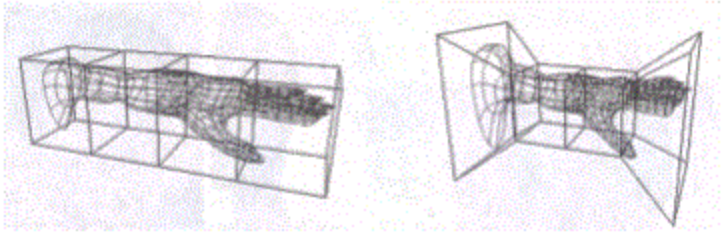


图 13-30 在手臂上创建网格，调整网格的同时手臂也会相应整体变化，就不用直接对手臂表面进行操作了

上面基本囊括了常用的建模技术，大多数三维软件都会包含这些基本操作，不过好用与否就各有千秋了。

比较特殊的建模技术

密度球体技术(blobby surface)是一种特殊的建模技术，虽然支持它的三维软件并不是很多，但是却是一个有趣的概念。密度球像一个实心球，不同直径的地方密度不同，而且它的显示表面并不是密度球的表面，如图 13-31。

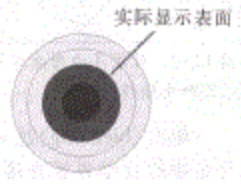


图 13-31 基本示意

图 13-32 显示表面在表面交汇的时候就会开始融合

图 13-33 利用密度球概念塑造的模型，可以看到各种不规则椭圆体

阴影部分才是真的显示出来的表面。当两个密度球接近的时候，表面有交汇的话密度球就会相互吸引，根据密度的不同显示表面就会有所变化甚至相互融合。

实际应用中，密度球不见得非要是球体，可以是不规则的椭圆体。

植物生成器(plant generators)利用数学公式自动生成各种类型的植物，发展的比较全面，只需要输入参数，比如树的种类、年龄、茂盛程度等，软件就会自动创建出所要求的植物。非常省事的工具。有两方向的生成技术，基于空间的过程模拟技术(space-oriented procedural techniques)是以环境对植物的影响为侧重点来生成植物模型；基于结构的过程模拟技术(structure-oriented procedural techniques)偏重考虑植物个体自己的生长情况和该种类植物的特点。

在这里要提一下的是著名的 L 系统(Lindenmayer systems)，L 系统非常适合各种平行分叉的植物，如图 13-34。

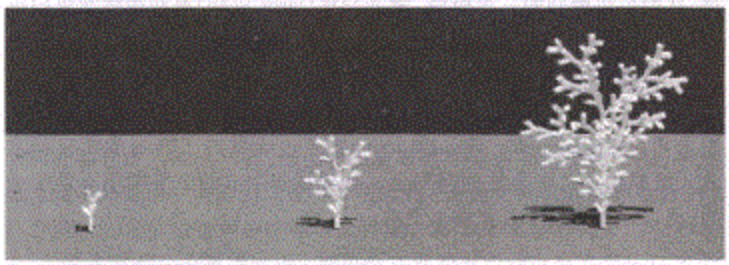


图 13-34 简单的 L 系统，模拟分叉植物的生长（www.cs.umbc.edu）

不规则碎片几何体(fractal geometry)是通过不断的随机细分多边形的边角来实现一些计算机才能做出来的特效。

这种方法的一个优点是可以无限细致下去，放大任何部分都可以得到足够精细的图像。比较常用于建模中的例子是将一个面无限细分再加上噪声做出不规则的起伏，用来模拟无规则复杂地表。

粒子系统(particle systems)将在下一章三维动画中详细介绍。

物理模拟建模(physical simulations)是利用复杂的物理公式来模拟现实中的物体，比如流动的水面。系统会自动根据输入的参数来创建动画，一般很难手工调整。

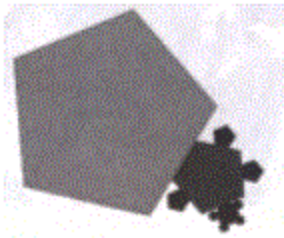


图 13-35 不断细分，在五边形的边上复生出新的小号五边形，然后在小号五边形上再复生更小的.....如此继续

数字化建模技术

这里只简要介绍一下数字化建模技术，因为这类建模技术所需设备都很昂贵，一般接触不多。这些方法一般常用于工业三维模型建模，只有数字笔比较常用于三维动画的制作。

数字笔(3D digitizing pen)是比较常见的设备，利用模型雕塑(现实中真实的)，在上面绘制网格，然后用数字笔逐点点击，系统就会接受数据，从而形成雕塑的三维模型。激光扫描(laser scan)是利用激光束扫描模型雕塑，得到类似剖面图的外边界截面，然后形成三维模型。这两种方法的共同缺点是没办法处理中空物体。于是，产生了改良的激光扫描，真正的将模型的剖面图得到，从而形成中空的模型。还有一种三视图重建法(orthographic reconstruction)，通过绘制的三视图系统自动生成三维模型，这种方法的问题是，从三视图生成的三维模型不是唯一的，所以还需要附加其步骤来更改。

图 13-36 数字笔

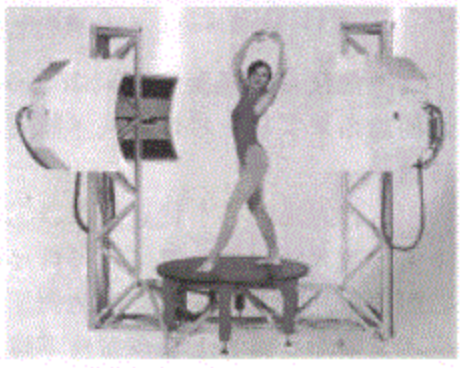


图 13-37 激光扫描

图 13-38 三视图重建法。左图是根据三张不同角度的照片进行描点处理。右图是最后生成的三维模型

建模中的层次结构

层次结构对于越来越复杂化的建模来说，是个至关重要的部分。大多数现实中的物体都不是简单几何体，总是由若干部分组合而成的。每个部分都有自己的坐标系，当你移动旋转或变形整个物体的时候，所面对的就不只是单一坐标系下的一个几何体，而是多个几何体的集合，它们之间的位置就需要用层次结构来控制。举个最简单的例子，一个木桌，由两条腿和一个桌面组成。如果腿和桌面都以自己原有的坐标系为准，相互之间没有联系，在旋转和

放缩整个凳子的时候就会出现问题。

正确的方法是建立层次关系，将两条腿作为孩子节点连接在作为父亲节点的桌面上，这样在进行旋转和变形操作的时候就以统一的父亲节点的坐标系为准，呈现的效果才和现实中的一样。

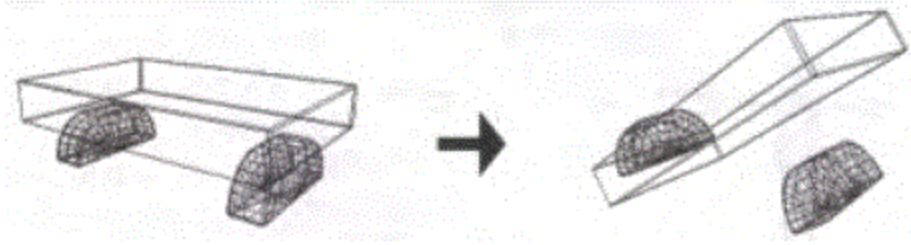


图 13-39 因为没有建立层次关系，在旋转桌子的时候出现的错误

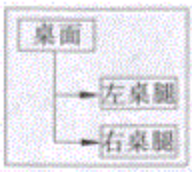


图 13-40 建立后的层次关系图

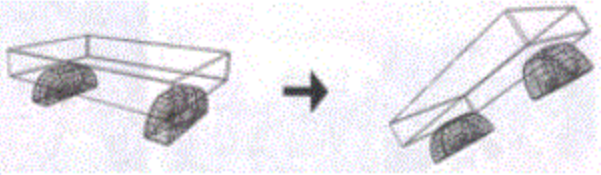


图 13-41 建立层次关系后，一切正确

建立层次结构不光是为了统一坐标系，更重要的是它是一种有效管理模型各部件的方法。层次结构一般是树状模型。创建具有细致层次结构的模型，就可以对每个部分或者几个部分分别操作，清楚明了，而且灵活。

一个比较重要的模型层次结构就是人体的层次结构。一般来说，人的骨架结构以腰部为根结点，下面延伸出左右臀部，然后是大腿和小腿之间的膝盖、脚踝和脚趾；向上则是脊柱，脊柱的块数按精细程度可以从 2 到 5 块不等，脊柱的尽头就是脖子，然后是头部；从脊柱还要分出两个肩关节，然后是手肘、手腕、手指。给出基本的人体层次结构，见图 13-42。

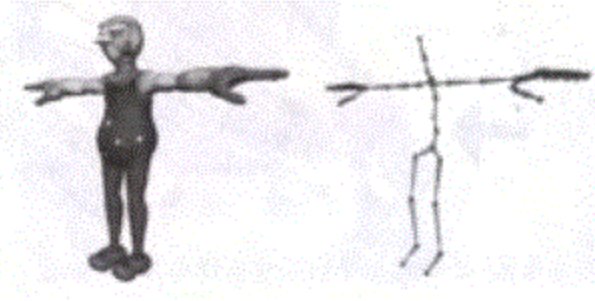


图 13-42 人物和基本的骨架模型

有些三维软件提供更为复杂的层次结构功能，比如 Maya。这种层次结构包含三维空间的所有物体，不光是三维模型，也包含了灯光、摄像机、材质、贴图以及动画数据等等。在物体模型的连接之外，灯光与物体的连接，材质和贴图甚至渲染手段与物体的连接，灯光与贴图的连接等错综复杂的连接使这种结构很难用树状模型来表现。在三维软件越来越复杂、各种参数越来越多的情况下，如果不理解层次结构在整个三维制作中的重要之处，不掌握合理安排各要素的层次结构，就会自己陷在大量混乱的信息中。

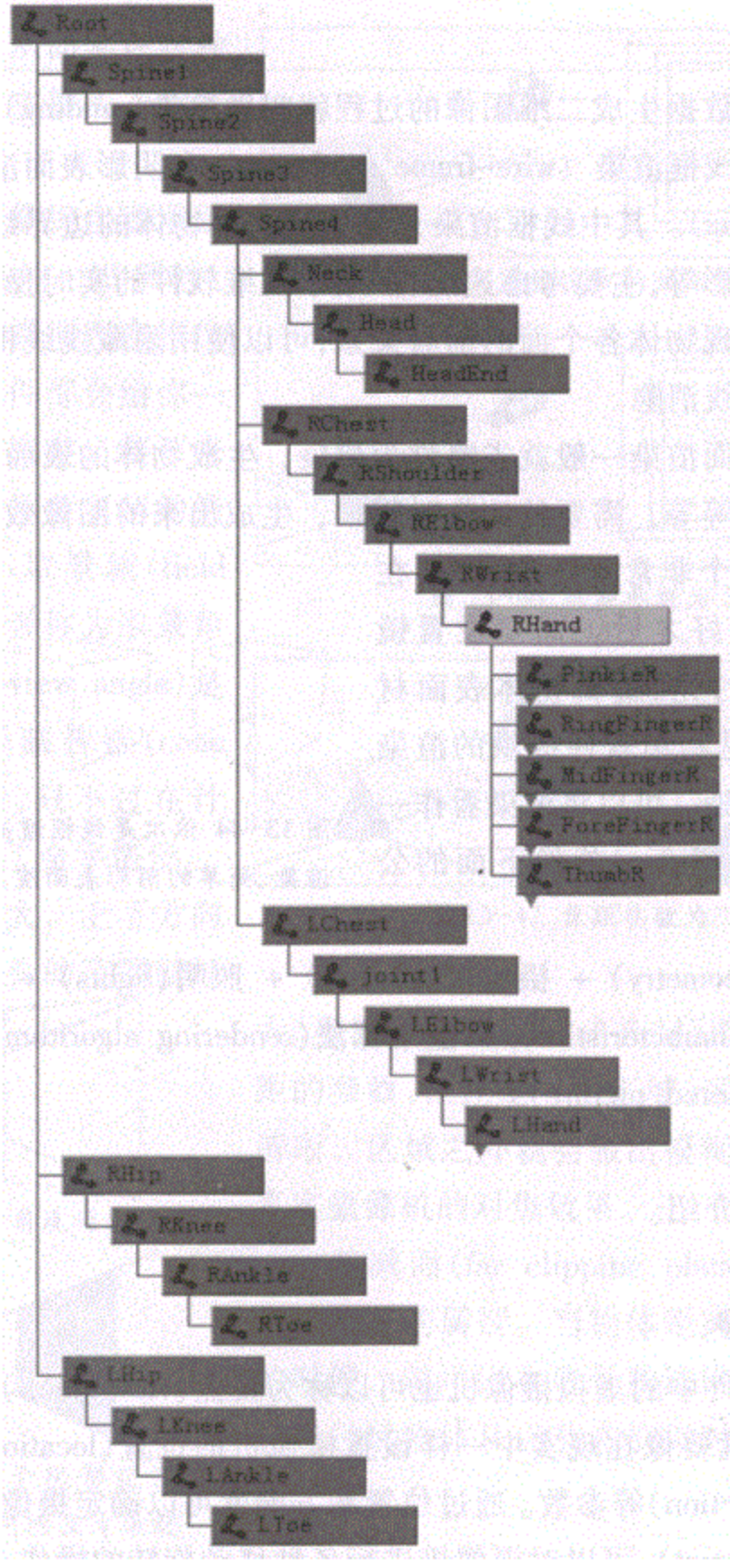


图 13-43 三维软件 Maya 中所建立的人物骨架层次结构，对应图 13-42 的人物

渲染

由三维数据生成二维图像的过程就叫做渲染(rending)。按繁简渲染可以分为线框渲染(wire-frame rendering)和阴影表面渲染 shadedface rendering)。其中线框渲染只会生成三维物体的边界线图，不考虑表面材质光影等，主要考虑速度，多用于三维软件的实时操作界面。为了更好地表现物体各个面的前后关系，可以使用隐藏线线框渲染，将应该看不到的线消隐。

图 13-44 依次是线框渲染、隐藏线线框渲染、简单的阴影表面渲染

阴影表面渲染一般就省略称为渲染，生成物体的表面而且考虑材质光影特效等等，需要的渲染时间长，生成出来的图像效果也要好得多。渲染是个非常复杂的过程，在物体模型建好之后，需要设置镜头，在场景打光，编辑物体表面材质，最后才是利用软件提供的渲染算法开始渲染。可以将渲染看作一系列操作的集合，就像下面的公式：

三维模型(geometry)+摄像机(camera)+照明(lights)+物体表面性质(surface characteristics)+渲染算法(rendering algorithm)=渲染结果图像(rendered picture)下面就依次介绍。

摄像机

三维软件中的虚拟摄像机也可以称为视点(view point)。要取得想要的镜头，就要像在现实中一样设置摄像机的位置(location)和摄像机的方向(direction)等参数。通过位置和方向就可以确定摄像机的兴趣点(camera interest)。可以对摄像机进行各种移动旋转的操作，就像对三维模型一样，从而实现各种镜头技术(在镜头一章有详细介绍)。

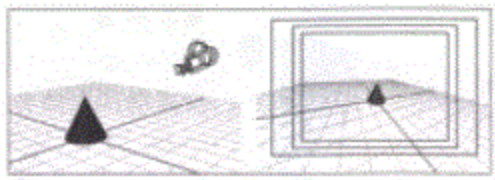


图 13-45 模拟真实摄像机的焦距，焦距参数为 70

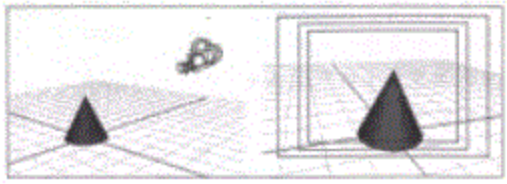


图 13-46 焦距参数为 35

虚拟摄像机也继承了真实摄像机的一些属性，从而更好地模拟真实摄像机，一般软件都会给你一些参数供你调整。接下来介绍一下必须要了解的摄像机属性。取景域(field of view)或者称为取景角度(field-of-view angle)是用来模拟眼睛视锥(cone of vision)的，只不过在计算机里变成了金字塔形。

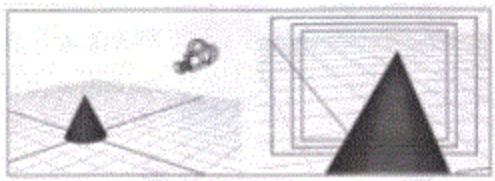


图 13-47 焦距参数为 70

角度越大，上下方向所能收入镜头的范围就越大。景深距离(depth of field)是用来控制焦距的参数，往往是一个区域，区域之内对焦清晰，区域之外就会做出模糊效果来模拟真实摄像机的对焦效果。

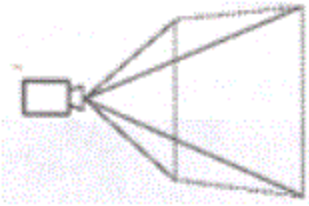


图13-48 取景角度的金字塔

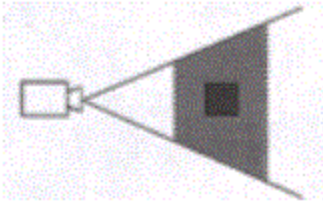


图 13-49 处于阴影范围(景深距离)内的立方体就有清晰的效果

裁减面(far clipping plane)是虚拟摄像机特有的属性。当物体距离摄像机过远的时候，就可以在取景和渲染计算的时候忽略过远物体从而节省渲染时间和系统资源。

照明

和摄影机一样，三维软件中的灯光主要是模拟现实中的各种灯光。基本上有点光源(point light)、局部光源(spot light)、区域光(area light)、环境光(ambient light)和方向光源(infinite/directional)这几种。点光源是最简单最普遍的光源，模拟类似灯泡的光源。

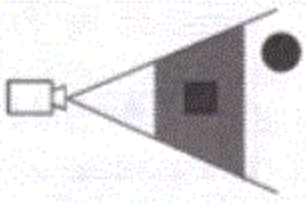


图 13-50 阴影面的上边界代表裁减面，生成图像中只会有立方体，而球体因为距离镜头太远所以在渲染的时候被忽略

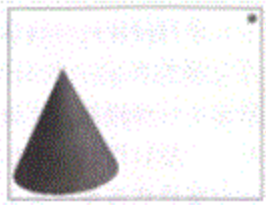


图 13-51 点光源很简单，不用调整入射角度

局部光源类似有灯罩的台灯，用来照明某个局部，一般是个圆锥形，特有的参数是锥形的角度和边缘淡化(drop off)。

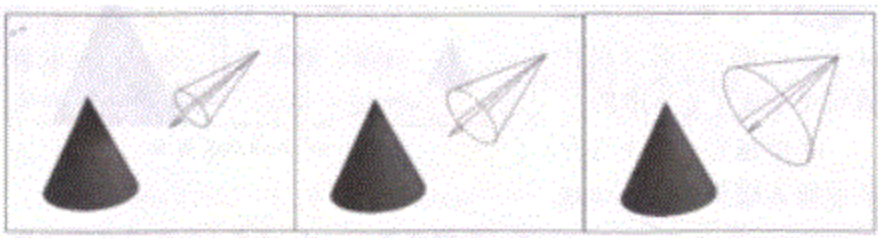


图13-52 不同锥形角度的局部光源

绚丽的舞台灯光就是用各种颜色的局部光制作的。

区域光主要用于照明小区域，长方形或锥形的较多，比如日光灯。

环境光(ambient light)主要来制作弥漫性光源。

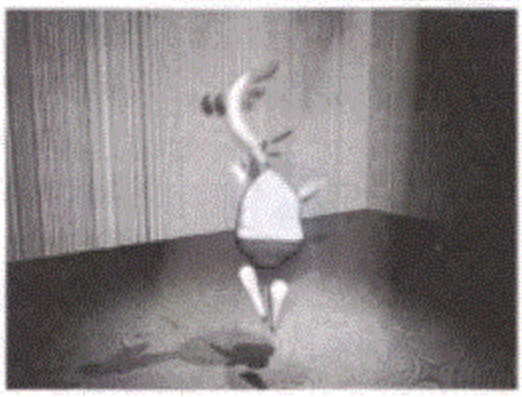


图 13-53 利用多盏局部灯光营造舞台效果

方向光源也称为无限远光源，用来模拟类

似太阳光，方向光的光线是平行而没有角度的。

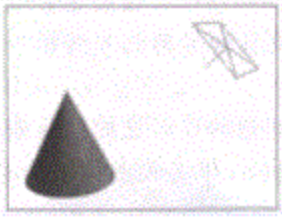


图 15-54 区域光，需要设定入射方向

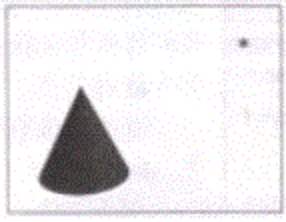


图 13-55 环境光

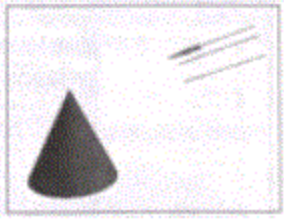


图 15-56 方向光源具有完全平行的光线

除了光源的位置颜色强度等基本参数，一个比较重要的参数是衰减(decay/fall off)。光束不会在空气中永远穿行，会慢慢的衰弱直到消失看不到，衰减就是来模拟这种效果的。

从作用来看，主光源(key light)是照亮物体的主灯，强度大作用明显而且有投影，用于照亮你想要照亮的地方，一般用局部光源。弥漫性光源(fill light)用于定义整个场景的色调和融合场景内其他不同的光源。比如处理露天夜景的时候，在所有灯光之上打上一盏蓝色的弥漫灯光，将整体色调处理成夜晚的蓝色。

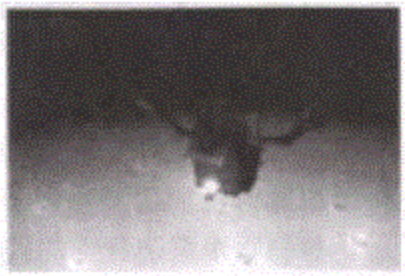


图 13-57 在所有光之上使用蓝色的弥漫性光，使整个场景的色调统一

如何摆设场景内的灯是处理照明的重要问题。三维中的灯光设置理论和方法也是从电影摄影的照明理论中衍生并发展过来的，但是也有它自己的特点。简单的说想照明一个物体一般要三个光源，主光源、侧面光源(side light)从相对主光源的另一侧射向物体，强度要淡得多，背面光源(back light)强度也很弱，作用是从物体背面照明物体阴影部分(具体如何处理不同场景不同物体摆设可以借鉴比较完善的电影照明理论)。相比现实灯光，三维中灯光所特有的优点是可以灵活设置灯光与物体甚至材质贴图的关系。在很多三维软件中，照明连接(lighting links)是层次结构中的一部分，可以让某个光源只照射在某些物体或者材质之上，而对其他物体和材质没有影响，这在现实中是做不到的，也是弥补三维软件永远无法真实模拟自然光线的一个手段。

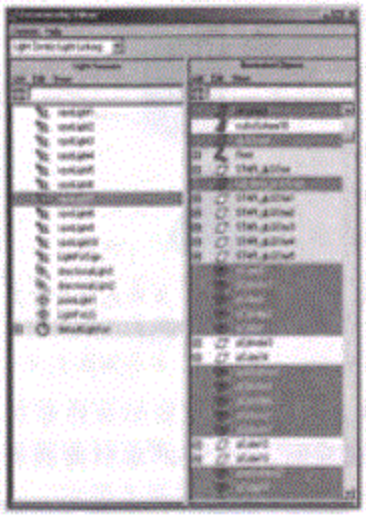


图 13-58 Maya 中提供物体和灯光之间的关联

有些三维软件也提供灯光的特殊效果，比如自发光(glow)和镜头光晕(lens flare)等效果。

图 13-59 使用自发光做出的效果

影子(cast shadows)，阴影(shading)和光源紧密相连，是体现场景真实性的主要元素。一般只有与物体连接的主灯才设置

影子和阴影属性，对应不同的情况可以调节影子边缘的模糊度甚至影子的颜色等参数，也可以根据设置不真实的影子和阴影来营造特殊的效果。

表面性质

像现实中一样，表面性质也是物体一类重要的参数，比如颜色、反光度、凸凹性质等等。物体表面各种性质的集合称作 shader。将物体面性质定义为独立的数据类型 shader，好处很多：定义好一个 shader，可以复数使用在不同物体上；作为独立的数据，方便调整和衍生出新的 shader；构成 shader 库(shader library)，方便他人和不同的项目调用。很多三维软件会提供基本 shader 库给客户，客户也可以根据自己的需要创建出新的 shader 来充实自己的 shader 库。

对于物体表面的描影法(shading)从简到繁可以分成碎面描影法(faceted shading)、柔和描影法(smooth shading)和反射描影法(specular shading)。碎面描影法对于多边形的每一个面只使用单一的颜色深度值，面与面之间有明显的边界，常见的有 Lambert 描影模型；柔和描影法在每个面内采用连续的颜色深度值使每个面看起来要柔和得多，可是面与面之间的边界仍然明显，大多数三维软件的柔和描影法都基于 Gouraud 描影模型；反射描影法计算更多的表面性质，所生成的多边形面之间则没有边界而且强调了镜面反射，它则基于著名的 Phong 模型。

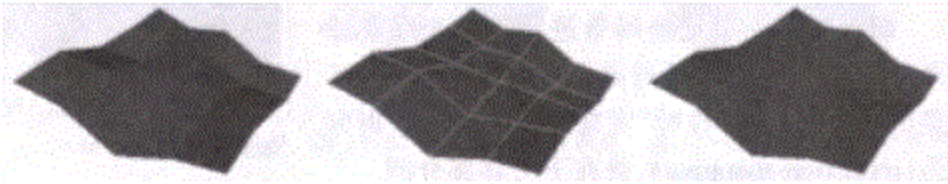


图 13-60 依次是碎面描影法、柔和描影法和反射描影法

说到这里，就不能不说道专业渲染工具软件：渲染者(RenderMan)。渲染者是功能强大的渲染软件，包括了各种渲染算法及工具，特别是提供特有的描影法语言(shading language)。描影法语言是一种类似 C 语言的编程语言，用于描述物体表面性质，有很大的可塑性及其灵活。用户完全可以通过描影法语言创造全新的视觉效果。

下面介绍一下基本的物体表面属性。表面颜色(color)的定义和二维软件中一样，使用 RGB 或者 HSL 值。发散率(diffuseness)或吸收率定义有多少光会从物体表面反射出，控制着物体表面的光亮度。虽然有发散率控制光亮度，但是它不会营造高光效果，需要另外一组参数高光(specularity/highlight)来定义高光的面积大小(highlight size)、颜色(highlight color)、衰

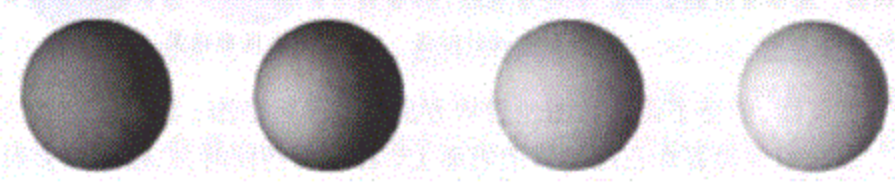


图 13-61 对同一环境下的球体调整不同的表面属性

减(highlight decay)等属性。透明属性(transparency)可以控制物体的透明性质，如果需要物体局部透明就需要用透明贴图(transparent mapping)的方法。折射率(refractivity/bend)决定光线穿过透明物体后变形的程度。炽热(incandescence)用来模拟自发光或者自发热物体的表面，比如灯泡。反射(reflectivity/mirror)用于模拟类物体表面的反射效果，根据材质的不同又分为环绕反射(ambient reflection)、漫反射(diffuse reflection)、镜面反射(specular reflection)等等。营造反射的效果，就要用到光跟踪算法(ray tracing)。

如果要对一个复杂场景进行追踪光线算法来生成各个表面的反射是非常消耗资源的，于是出现了另一种“作弊”实现反射的方法反射贴图(reflection mapping)，将在下节详细介绍。

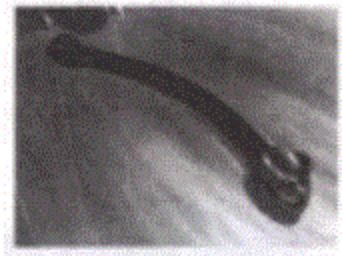


图 13-62 Ray tracing 所做的地板反射

2D 贴图

材质贴图(texture mapping)是模拟物体表面的重要技术。二维贴图可以投影或者包裹在物体上，对应的就是投影贴图(projection texture mapping)和参数贴图(parameterized texture mapping)。

投影贴图按投影方法可以分为平面投影贴图(planar projection mapping)、立方体投影贴图(cube projection mapping)、圆柱投影贴图(cylindrical projection mapping)、球投影贴图(spherical projection mapping)等。投影贴图虽然很简单方便容易计算，但是不足的是，在边界顶角等地方都会出现拉伸的条纹或者急速收缩的效果。

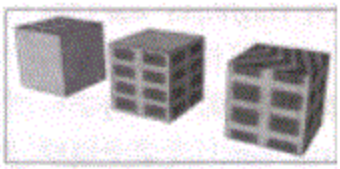


图 13-63 对立方体的平面投影贴图、圆柱投影贴图和球投影贴图

图 13-64 对圆柱体的平面投影贴图、圆柱投影贴图和球投影贴图

图 13-65 对球体的平面投影贴图、圆柱投影贴图和球投影贴图

更灵活、更便于调整贴图位置的方法就是参数贴图。基本思想就是将二维的图像包裹在三维物体的表面上。在包裹的同时，二维图像会根据具体三维物体的外形而拉伸匹配。几个基本的参数有贴图位置(placement)，定义二维图像在三维物体表面的具体位置和角度等等；放缩比例(scale)用来指定二维图像在贴图的过程中是否自身要放大、缩小，因为往往二维图

图 13-66 简单的平面投影

图 13-67 对贴图位置的调整

图 13-68 对贴图大小放缩和旋转等操作

像在贴图过程中要改变自身的尺寸。

图 13-69 重复操作

重复(wrap)来设定二维图像是否在三维表面重复贴图来实现类似地板砖的效果。

利用二维贴图可以实现一些特殊的表面效果，比如物体的局部透明(transparency mapping)。利用二维图像上不同的颜色或者深度值来标示物体的透明程度，越透明的区域颜色越淡直到完全透明的白色。经常利用透明贴图制作铁丝网或栅栏围墙等繁琐的中空物体。

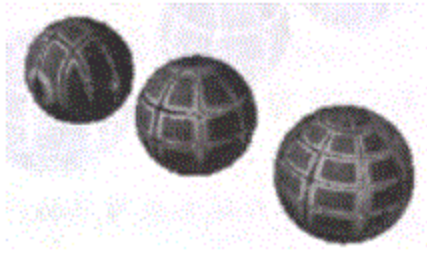


图 13-70 对球体使用网格透明贴图

凹凸贴图(bump texture mapping)可能是使用最频繁的特殊二维贴图了，用于处理物体边面的凹凸不平，在不需要特写镜头的情况下，对于表面凹凸不平的物体不需要在建模的时候将凹凸表面建模出来，一般采用的办法是利用凹凸贴图。

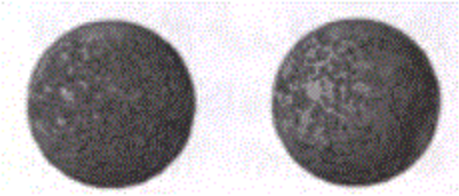


图 13-71 不同程度和密度的凹凸贴图

同透明贴图一样，凹凸贴图是利用二维图像上不同颜色或者深度值，来标示物体的凹凸程度，在渲染的时候三维软件会自动的将物体表面处理成有阴影效果的凹凸效果。不过凹凸贴图有个很大的缺点，就是在物体凹凸表面结束的边界并不会显示凹凸，还是直线一条，而且当你几乎平视凹凸表面的时候，会发现表面根本是平的，这是因为凹凸贴图只是在模拟凹凸表面，而不会改变实际的三维模型。

位移贴图(displacement texture mapping)可以改善凹凸贴图的缺点。位移贴图也是利用二维图像来得到凹凸程度，和凹凸贴图不同的是，它会真正调整三维物体的表面。因为改变了三维物体模型使其过于复杂，位移贴图的用途也和凹凸贴图不一样，不再是模拟物体的凹凸表面，而是多用于制作三维地表，比如山峦、峡谷等等。很多三维游戏的地形，都使用位移贴图这种思想来制作的。

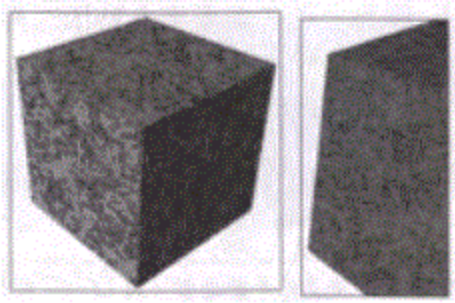


图 13-72 远看边界，没问题，放大后边界还只是直线

还有些三维软件提供比较特殊的贴图功能，比如自发光贴图(incandescence mapping)、高光贴图(specularity mapping)等等。它们的原理和上面的几种贴图一样，都是利用二维图像标示物体表面上需要做效果的区域。

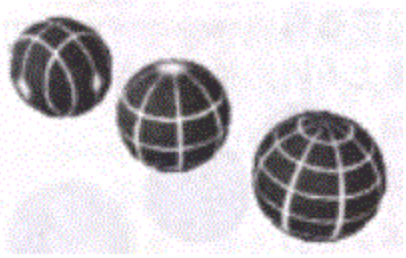


图 13-73 对球体使用网格的自发光贴图

关于贴图的大小。游戏和电影中的贴图在三维原理上基本相同，但是在二维图像的尺寸效果等方面有很大差距。首先是精致程度，一般三维游戏需要即时(real-time)运算，对于贴图尺寸的大小和容量有严格要求，一般来说都是些同样大小的图片序列。256*256 或 128*128 点阵是比较普遍的大小。每张二维图片一般都会被最优的分割成若干小区域来放置物体不同部分的贴图。图像大小也受内存的限制。而对于三维电影的贴图，就根本不用考虑这些，在成片渲染时间允许的前提下效果好才是第一位，而且贴图的大小不必要相同也不必要进行分割。有些好莱坞电影的三维特效的贴图竟然达到 3072*3072 的高解析度。

3D 贴图

两种主要的三维贴图：环境反射贴图(environmental reflection mapping)和实体材质贴图(solid texture mapping)。

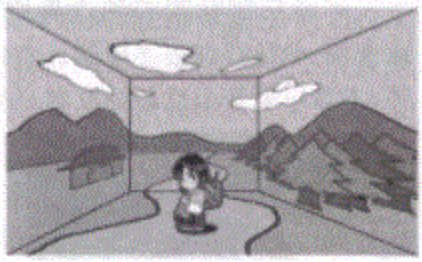


图 13-74 立方体的屋子内部的六个面

如果要让金属类的物体在一个场景当中显得真实，就要考虑到金属表面的复杂反射。可以在渲染的时候使用光线跟踪算法(ray tracing)来计算反射，但是光线跟踪算法极其复杂耗时，于是在不是特别追求反射准确性的情况下，可以利用三维环境反射贴图来模拟环境反射。三维反射贴图的基本思想是在物体表面上贴上类似周围环境的二维图像来模拟环境反射。最普遍的环境反射贴图是立方体反射贴图(cubic reflection mapping)。

图 13-75 利用已经建好的场景来制作贴图

举例子来说明，在一个封闭的立方体屋子里，墙面、地板还有天花板这六个面分别贴着六张二维图像模拟一个空间，注意这六张图像相互的边界应该是无缝连接的(seamless)。如果在屋子中间放一个金属球，金属球应该反射周围的环境，所谓立方体反射贴图就是把这六张模拟周围环境的二维图像贴在球体上来模拟金属球的反射。如何制作这六张无缝连接的二维图像是立方体反射贴图的难点。最有效普遍的方法是先创建好三维场景，确定金属球在三维场景中的位置，然后在金属球的位置设置摄像机，按照六个面的方向渲染，之后再将这六个面渲染所得的二维图像用立方体反射贴图法贴在金属球上。

步骤很麻烦，不过算是最简便有效的办法了。还有一种环境反射贴图是球体反射贴图

(spherical reflection mapping)，原理和立方体反射贴图一样，只不过屋子换成中空球体，六张二维贴图换成一张贴图。虽然贴图的数量减少了，但是因为这样图的上下左右边都要能够无缝连接，所以贴图的制作难度增加了。

图 13-76 一张反射贴图，左右是连续的

环境反射贴图的优点是渲染速度快，但是存在很多限制。首先，如果在场景中有两个以上的金属物体，不可以制作出反射的反射；如果场景中金属物体外有物体的明显运动的话，很难再使用环境反射贴图来做反射效果，如果要做，就要先渲染出二维贴图的动画序列，然后按时间将图像序列贴在金属物体上，就算是一段持续两秒的短动画，一秒 30 帧，就要 2*60*6=720 张贴图!所以说用环境反射贴图做动画很不实际。根据金属物体的外形不同，有时候二维贴图并不需要做得非常细致，只要色调和明显物体的位置一致就可以。

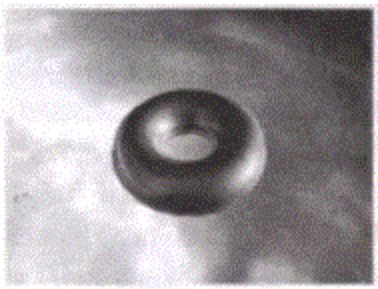


图 13-77 金属圆环，周围环境很复杂，但是反射在金属圆环上根本辨别不出什么

另一种三维贴图实体材质贴图是用来模拟贯穿物体类材质的贴图方法。最有代表性的这种材制就是木纹，木纹在木板的六个表面是连续有规律的，就好像它的纹理穿过整个木板，用普通的二维贴图方法很难实现这样的效果。

实体材制贴图的基本思想是，模拟一个三维材质的容器，然后将物体浸入容器，物体的表面留下的痕迹就是最后的贴图效果。

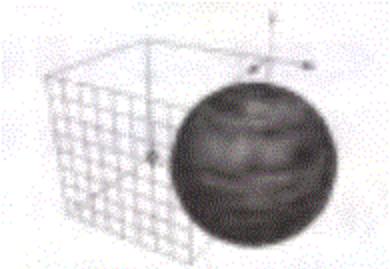


图 13-78 Maya 中实体贴图的外框，也就是下面的容器了

需要注意的是，如果没有将三维材制连接在物体之上，在移动旋转或者放缩物体的时候，三维材制的容器不会自动一起变化，当然也可以用这个特点做出材质浮动的特殊效果。在实体材质贴图之上再利用透明贴图或者凹凸贴图，就可以制作出各种不同的材质效果。

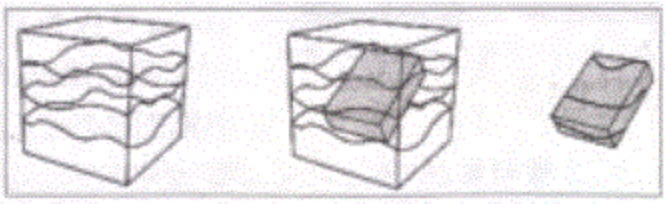


图 13-79 将长方体浸入三维材质的容器中

空气效果

三维软件的渲染图像都可以达到非常精细清楚的程度，但是现实中因为空气的存在，根据不同的空气密度等因素，会使图像有不同程度的模糊效果或者颜色的变化。举个例子来说从不同的距离来看远山，会因为空气的原因使山的颜色逐渐淡漠。这种效果被称为明暗效果(chiaroscuro)，这是因为光线在空气中穿行

的过程中，光线的一些颜色逐渐被吸收，穿行得越远，被吸收掉得越多。很多三维软件都会提供生成各种空气效果的工具。

使用最多的空气效果就是雾(fog)。有些三维软件可以自动生成雾效果，只需要设定各种相关参数，比如雾的密度、颜色、范围等等。这种方法不易调整，很难按照一定设计来实现雾化效果，比较灵活而且使用较多的密度图方法(density map)。如图是一张典型的密度图。

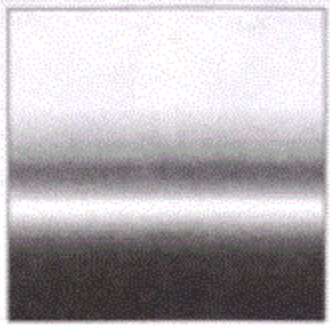


图 13-80 雾的密度图

通过颜色的深浅来标示雾的浓淡关系，然后系统根据这张图产生不同浓度的雾化效果。有些三维软件还可以根据时间来调整雾的浓度变化。如图 13-83，为两张不同的密度图，代表两个不同时间雾的浓度，系统会根据设定的时间在这两张密度图之间自动生成中间密度图，从而营造出雾缓慢飘动的效果。

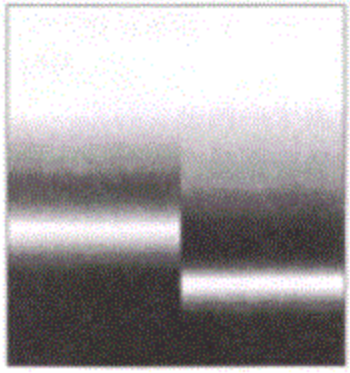


图 13-81 两个时间雾的密度图

另一种体现雾效果的方法是雾灯(fog light)。在浓雾弥漫的海边灯塔所打出的光束，还有舞台上的绚烂光柱都是雾灯效果。之所以可以看到光传过空气形成光柱，是因为空气中水气大或者大量的小尘埃。雾灯效果实际上是灯的一个属性，多用于局部灯，同样可以调整雾的浓度、颜色、衰减快慢等参数。使用雾灯也可用来模拟大面积雾的效果，如图 13-82 所示，使用巨大的雾灯来模拟地面上淡淡的雾化效果。

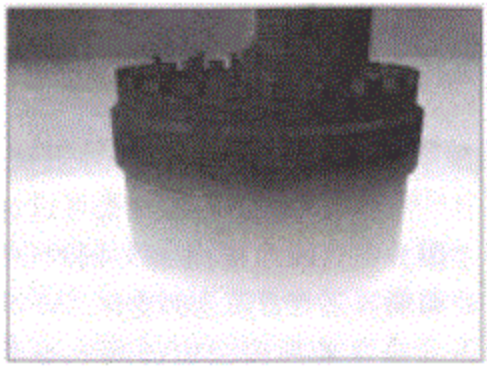


图 13-82 使用雾灯，很容易做出淡淡浮雾的效果

第十四章　三维动画

三维动画的原理和二维动画一样，都是利用视觉暂留原理(persistence of vision)来“欺骗”观众的视觉，通过连续快速播放(一秒 24 或 30 帧)画面序列形成动感来模拟现实中的运动。它们之间所不同的就是画面生成的方法：二维动画的每帧画面一般为手工绘制而成；而三维动画则是利用三维软件制作模型，配置参数，然后渲染出各帧画面。

三维动画大多是基于关键帧动画的原理，三维软件提供了设定关键帧和插入中间帧的多种工具。也有一些特殊情况无法通过设定关键帧的来生成动画，只能使用类似于非关键帧动画的方法，设定好参数后让软件自动按时间顺序生成动画，比如利用粒子系统(particle system)生成的火焰烟雾效果。

三维动画的基础技术

关键帧动画

关键帧动画是三维动画的基础。三维动画软件一般都提供很多方便的工具来设定关键帧。设定关键帧的对象可以是三维模型、摄像机、灯光、表面材质甚至具体的某一项参数。

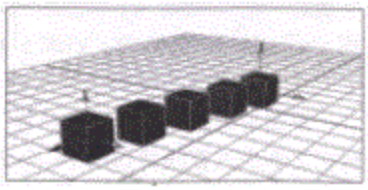


图 14-1 在关键帧中间自动生成中间帧

先用个例子说一下最基本的位置移动动画。如图 14-1，在动画的开始和结束时刻，将立方体放置在不同的位置，分别设定关键帧(set keyframe)之后，软件可以在两个关键帧之间自动生成中间帧，有了这些帧，就可以预览动画了。

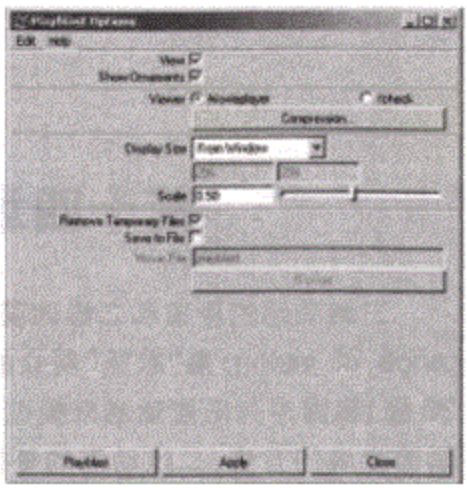


图 14-2 简单快捷的 Playblast

很多软件在编辑时就可以预览动画，不过有的时候因为模型过于复杂或者系统不够快，三维软件就会提供动态预览(motion preview)的功能，可以用来播放简单的渲染动画，来粗略地观察动画效果是否是正确的，比如 Maya 中的 Playblast。

你会发现三维软件会根据每秒钟帧数(frame per second)等参数，自动计算出中间帧中立方体的位置，这个计算的过程叫做插补操作(interpolating)。

在三维动画软件中经常使用二维坐标图像编

辑的手段来调整插补操作。如图 14-3，Y 轴代表了位置移动、旋转角度或者其他参数变化，而 X 轴代表时间，关键帧在坐标图中显示为点，点之间的连线就是插补操作所生成的中间帧。

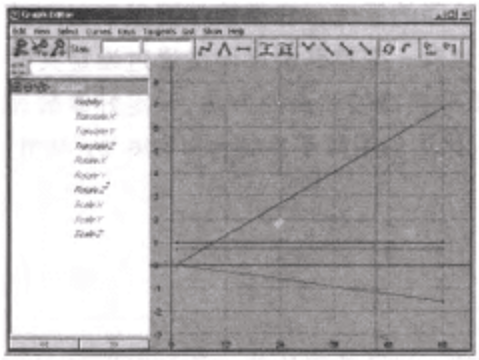


图 14-3 Maya 的 Graph editor，显示上面立方体简单运动的曲线图

对于物体运动来说，比较常见的参数有空间中 x、y、z 方向移动 Tx(Transition x)、Ty(Transition y)和 Tz(Transition z)；x、y、x 方向旋转 Rx(Rotation x)、Ry(Rotation y)和 Rz(Rotation z)；x、y、z 方向放缩 Sx(Scale x)、Sy(Scale y)和 Sz(Scale z)；比如上面的例子，Y 轴代表了三维空间中 X 方向的位置移动 Tx，X 轴代表时间。

最简单的插补操作是线性插入(linear interpolation)，就是在相邻关键帧之间插入直线。如图 14-4，三个关键帧之间用直线相连，构成折线段。

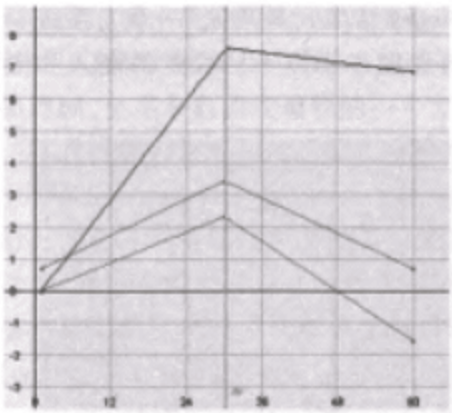


图 14-4 线性插入

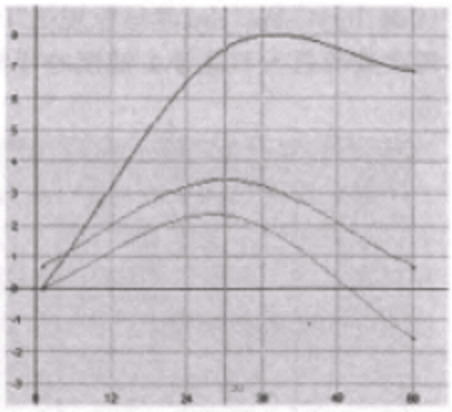


图 14-5 曲线插入

但是现实中物体的运动大多数都不是一段一段的僵硬改变，要自然柔和得多，所以更好的办法是用曲线来模拟现实中运动。如图 14-5 所示，同样三个关键帧之间用的就是复杂一些的曲线插入。

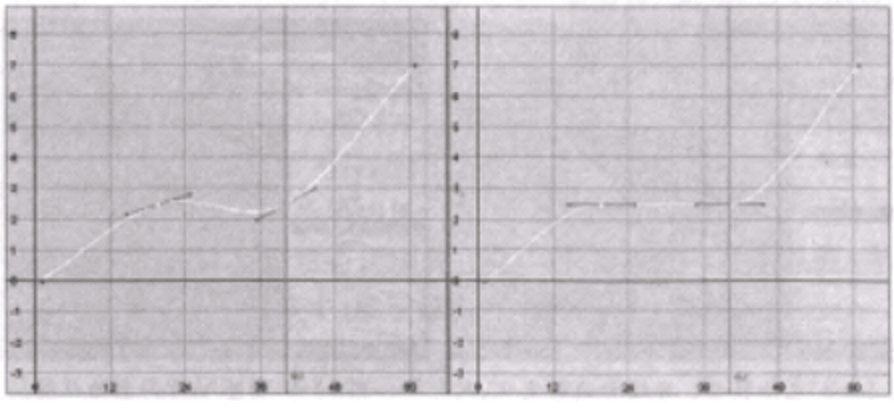


图 14-6 对于同样关键帧下的曲线形状可以采取不同的处理方法

上一章三维建模中所介绍的多种曲线被广泛应用在插补操作的坐标图显示中，它们自然地模拟了物体运动的各种变化，而且利用它们本身的特点也非常容易来调整曲线，比如控制点和外壳。

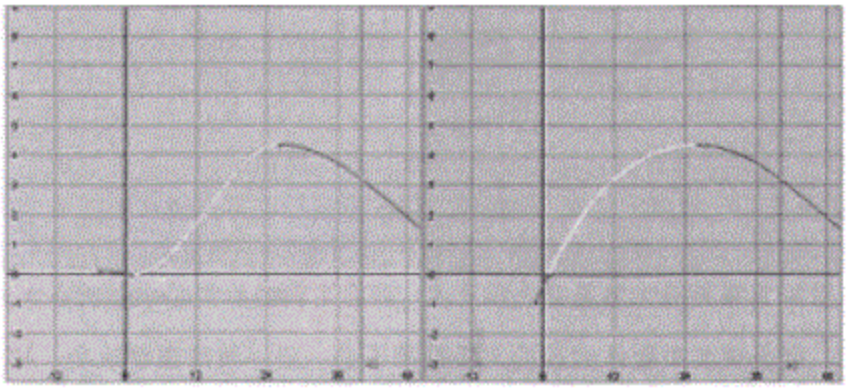


图 14-7 对于左边开始点不同程度的慢出

曲线插入的另一个重要的用处是模拟渐入和渐出(ease in/ease out)。这里所说的渐入和渐出类似于经典动画理论中的慢入和慢出(slow in/slow out，具体原理见“Disney 传统动画理论”一章)，系统利用曲线的进入和离开关键帧的弯曲程度来模拟不同程度的渐入和渐出。

参数曲线编辑(parameter curve editing)是对应曲线插补方法的各种编辑方法的总称。

一类操作是对于代表关键帧的点的操作。点的上下方向的移动对应 Y 轴所表示属性的变化，如图 14-9。

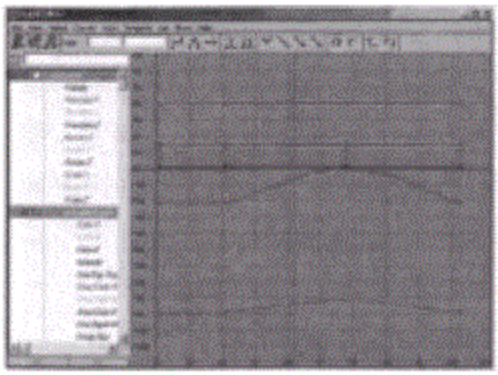


图 14-8 包括移动、放缩和旋转交化，所对应坐标系中的 9 条曲线

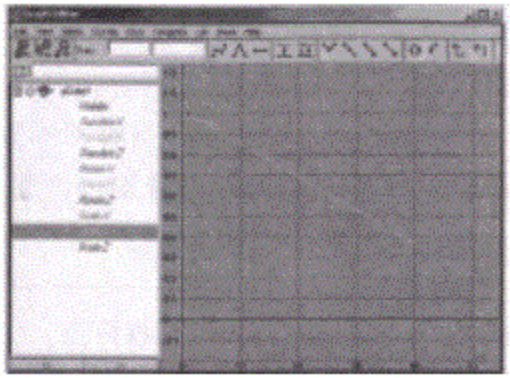


图 14-9 物体 Y 方向上的放缩

Y 轴代表物体 y 方向上的放缩，通过上下调整关键帧实现调整物体体积的放缩；点的左右方向的移动对应时间，通过调整点的位置可以调整关键帧出现的时间，从而调整动画的速度；可以直接在曲线上插入新的关键帧(insert key frame)，而不非得要在可视化的视图下设置新关键帧，也是参数曲线编辑的一个方便之处。

另一类编辑则是对于曲线本身的操作。因为曲线的特性，曲线在点上有切线矢量

(tangent vector)，通过调整切线矢量也可以调整曲线在点附近的曲率；在曲线中某两关键帧点之间也可以单独使用线性插入方法，而其他部分仍保持曲线插入；部分曲线放缩是对曲线一部分进行放缩，从而调整这部分动画的整体速度；还有类似文本编辑器中的复制(copy)、剪切(cut)和粘贴(paste)等操作。

这里介绍一下 Maya 所附带的图示编辑器(Graphic Editor)。在 Maya 里基本所有的对象都可以设置关键帧，普遍的比如三维物体的各种外形属性、摄像机和灯光的位置和照射方向、对于材质的大小位置变形等等；深入细致的比如每个具体的细致属性，如灯光的亮度、衰减的程度等极其细小的属性。所有这些都可以作为动画的一部分，而图示编辑器就是有效管理和编辑所有这些数据的利器。当选择某个对象，图示编辑器就会显示所有关于这个对象的关键帧，并可以按用户的需要显示部分关键帧。Maya 的图示编辑器包含很多的曲线编辑手段，比如可以折断或者拼合切线矢量、自由地复制粘贴曲线等等。在实际编辑三维动画的过程中，图示编辑器的使用总是很重要的。

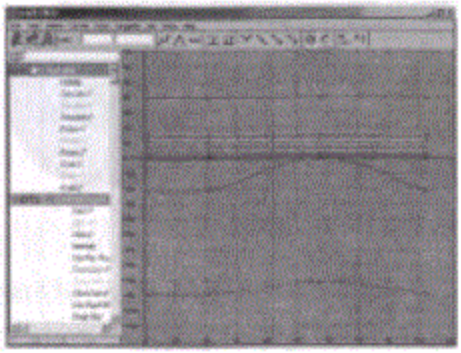


图 14-10 在左半部分选择一个对象，右半部分就显示了其动画曲线

层次结构动画

在三维建模一章中已经介绍了层次关系在建模中的重要性，对于动画来说层次结构同样是非常关键的概念。现实中物体的层次结构对其运动规律有极大的影响，三维世界中的层次关系也是模拟现实物体的运动的基础。

因为层次结构的存在，各个层次的运动也存在着继承关系。父亲结点的运动要被所有孩子结点所继承，而孩子结点的运动并不影响父亲结点。现实中的很多运动都是继承的，比如人移动手臂的时候，手要继承小臂的运动，而小臂和手都要继承大臂的运动。再举个通俗的例子，地球和月亮的运动。月亮可以看作是连接在地球上的一个孩子结点，整体可以看作一个父亲结点和一个孩子结点的运动。地球的运动有围绕太阳的公转和以自己为轴的自转，月亮的运动有围绕地球的公转和以自己为轴的自转。继承关系体现在月亮在实现自己的两种运动之外，还要继承父亲结点地球的公转，和地球一起围绕太阳运动。

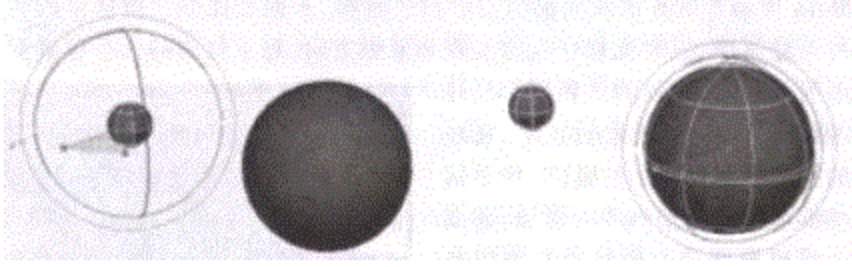


图 14-11 地球和月亮的自转和公转关系只有通过层次结构才可以完全表现

关节的自由度(degrees of freedom)是限制运动范围的参数。比如人的小臂，是不可能作3个方向360度的旋转的，实际的运动是受到限制的。有些三维软件对于某个物体的旋转运动方向可以设置自由度，从而方便地限制运动范围。

图 14-12 手臂部关节的自由度就很受限制

前向运动学(Forward Kinematics，简称FK)是一种基于层次结构的运动方法。这种运动是从根结点向叶子结点驱动的，按走向来说是向前的，所以称为前向运动。在总体结构所处位置越向根部的运动就会牵动越多的叶子结点运动，而叶子结点的运动影响不了根结点。利用这种方法来设置动画中物体的关键帧姿势，是从根部结点开始调整，最后调整叶子结点。比如调整手臂的姿势，就要先调整大臂，然后调整小臂，最后才是手部。前向运动的优点是方便灵活，只需要依次调整关节的旋转角度，但是越是灵活自由的，越不容易控制。前向运动方法在处理某些问题的时候，就需要动画师具有专业水平(动画界有这么一种传闻，真正一流的动画师绝大多数只用FK技术)。比如用手臂去拿一个杯子的动作，开始帧的手臂姿势是很容易摆设的，有难度的是如何通过旋转关节让手到达位置然后握住杯子，每个关节要用多长时间旋转大概多少度，到什么位置哪个关节的运动就应该停止，只有具有一定专业素质的动画师才可以正确地来设置这些。为了解决类似触摸等这样的问题，就产生了反向运动学。

图 14-13 完全使用FK完成的动画

反向运动学(Inverse Kinematics，简称IK)不同于前向运动的运动方式，应用相当普遍，举个例子，当你想去拿一个杯子的时候，你会想要把手移动到杯子的位置，而不是先旋转大臂然后旋转小臂然后旋转手腕，这就是反向运动和前向运动思想的根本区别。对于这个动作来说，动作的引导者是手，手移向杯子从而带动了小臂的运动，小臂运动带动了大臂的运动，而对于传统的层次结构模型，只能实现大臂带动小臂，小臂带动手，而无法实现反向的运动传递。反向运动方法提供了反向运动传递，可以更好地来实现这类动作。将这种基于反向运动学的连接结构叫链结构(chain)。链结构的尽头，就是可以用来移动的点叫做受动器(effector)；而链结构被固定的那一段叫做根结点(root)。

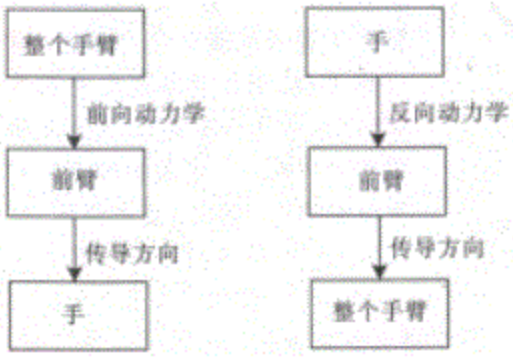


图 14-14 FK 和 IK 不同的运动传导方向

S 反向运动方法经常用于模拟人或动物的骨架关节，而现实中人和动物关节的旋转是受限制的，所以在定义反向运动关节的时候总会定义关节旋转角度的限制，也就是上面提

到的关节的自由度。但是受限制的 IK 经常会导致出现问题。

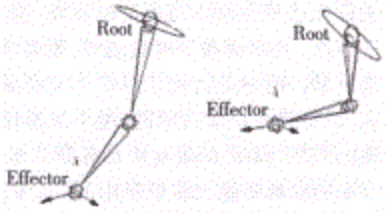


图 14-15 最基本的链结构

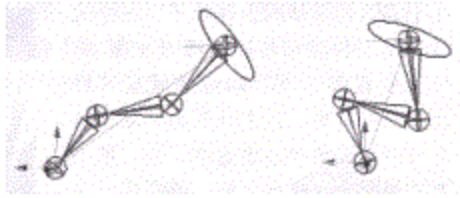


图 14-16 三个关节的链结构

因为人小臂的旋转角度被限制在 180 度范围之内，同样是移动小臂上下运动，如果在 180 度范围之内，大臂就不需要旋转；而如果超过 180 度范围，大臂就要以自己为轴旋转。在处理类似这样变化中的手臂运动。经常会发现 IK 无法正确生成中间帧，甚至旋转的方向和所设想的完全相反。这种时候，就需要动画师尽量详细地设置关键帧，避免 IK 的错误。对于 IK 问题的另一种手工限制是限制关节只能在二维平面中旋转，这样即避免了整个链错误的旋转又节省了运算时间，当然也限制了 IK 的应用。

图 14-17 人物的小臂动作的限制

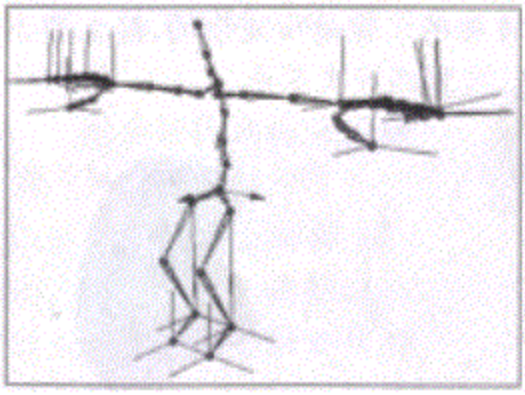


图 14-18 三维人物的基本骨架，多处使用 IK

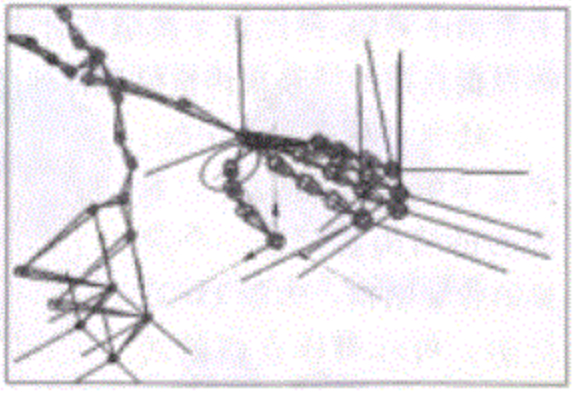


图 14-19 五个手指也可以使用 IK 驱动

图 14-20 连接好皮肤的人物骨架

图 14-21 关节处的网格

当定义了链结构或者骨架结构之后，就可以把三维模型连接(attach)在链结构或骨架上。简单的方法是制作分开的三维模型，每一件对应链结构的一部分，然后一一对应分别连接在链结构上。比如作人的手臂，就是小臂一个模型，大臂一个模型，手掌一个模型……这种方法在关节处是分裂开或者重叠的，没法很好的表现表面连续的物体，更好的方法是将一块整体模型连接在整个链结构之上。一般系统的自动连接并不能完全正确地连接物体和链结构，所以三维软件提供很多种工具来编辑，比如对与骨架连接好的物体表面逐点编辑，制定每个点属于哪个关节，甚至该点受那个关节的影响程度等等。

对于关节弯曲，系统往往会自动让模型自然变形，更好的方法是在关节处加上网格(lattice)，通过调整网格的属性来实现不同程度的变形。

在实际动画制作中，如果三维模型连接到骨架结构连接得不好，就直接导致后期动画制作中三维模型的表面相互交叉或者变形不正确，所以这个模型与骨架连接的步骤是很复杂的，经常要不断根据连接的结果来调整模型再连接，反复四五次不止。

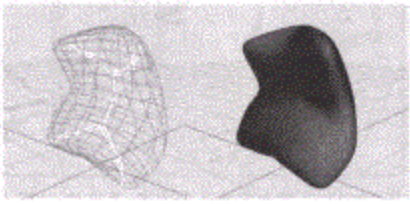


图 14-22 内藏骨架的布袋类型角色

还有一种有趣的手法，将简单三维模型连接在复杂的链结构甚至是一个完整的骨架结构，如图 14-22 所示，可以做出有趣的动画，比如拟人的口袋角色。

运动轨迹动画

运动轨迹(motion path)动画是常见的一类动画。三维软件中一般是先创建一条运动轨迹，然后将一动物体连接在轨迹之上，让物体沿轨迹运动。运动轨迹也是一条时间曲线(tlining curve)，利用图示编辑器可以编辑不同时间物体在轨迹上的位置，从而实现不同的移动速度。物体在路径上移动，物体的方向最简单的是保持不变总指向一个方向。

也可以将物体与一个移动的目标点连接，这样在物体移动过程中总指向目标；最常用的是让物体指向路径的切线方向，顺着路径移动。

图 14-23 立方体不改变方向沿路径移动

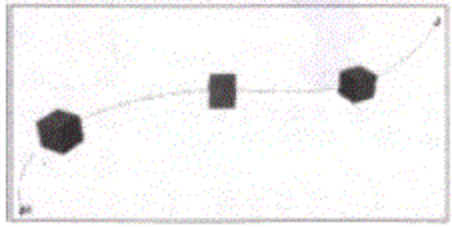


图 14-24 立方体顺着路径方向移动

外形变化动画

前面所讨论的多是物体运动的动画，而相对于物体本身形状改变的外形变化动画(shape changing animation)也是三维动画重要部分之一。现实中物体外形变化非常普遍，比如花蕾绽放成花朵、泥塑的变形、肌肉的收缩等等。三维软件提供各种方法处理物体的外形变化，基础的思想还是关键帧动画，在变形开始和结束时设定关键帧，然后自动生成中间帧。

最基础的方法就是对物体外形的控制点或者外壳设置关键帧。如图 14-25，对球体上的控制点设置关键帧，然后调整球的外形，再

图 14-25 对球体的控制点设定关键帧

设置关键帧。

这种方法简单方便，但是如果每个时间要调整太多的控制点，就太过繁琐，改进后的方法称为融合外形(blend shapes)。

图 14-26 利用关键帧生成的变形动画

融合外形的思想是，在原有物体的基础上，复制出不同变形的复制体，也称为关键外形(key shapes)，然后直接利用这些复制体来设定关键帧，就不用对每个更改的控制点来设定关键帧了。

融合外形方法经常使用在人物表情动画(facial expression)上。一般在完成人物脸部模型后，就复制人物脸部然后在不同的部位加以调整做出不同的表情，然后用融合外形的方法在动画中设置关键帧。具体有两种办法，一种是在做关键外形的时候每个复制体只做局部变化，比如这个复制体光做眼睛睁开而其他部分保持原样，另外一个光做张嘴呼吸的，然后在融合外形后设置关键帧的时候，利用不同部分的变化来组合出某种表情；另一种方法是每张关键外形的脸只是一种表情，是什么表情就是什么。第一种方法的优点在于可以减少工作量，只制作有限的局部表情，然后通过排列组合就可以创建出很多变化；缺点是因为人在做表情的时候总是会带动很多面部肌肉，各个部分总是相互关联的，而这种办法组合出来的表情相互之间的关联变化表现不足，并不非常自然。第二种方法的优点是可以很好地表现整个面部表情，缺点是工作量太大，要为每个表情制作一个关键外形。所以在实际应用中经常混合使用这两种方法。

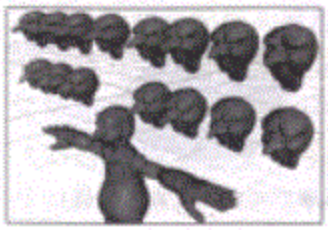


图 14-27 人脸的多个 key shapes

图 14-28 Maya 的 BlendShape 界面

还有一种类似的技术是(3D morphing)。类似融合外形技术，不过被融合的两个物体并不是通过复制然后修改而出的，而是完全不同的物体。3D morphing 有两个限制，一个是两个物体的控制点数要一样多，这就要求在建模的时候就计划好具体的点数和位置；第二个就是要求两个物体的控制点要一一对应，为了更好的控制点之间的连接关系，有些软件提供逐点连接的工具。

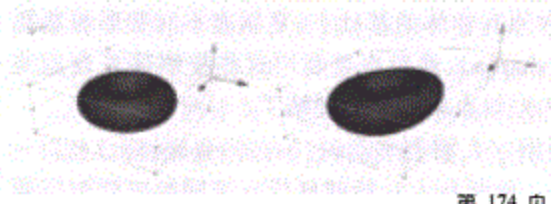


图 14-29 对圆环体加上网格来设定关键帧

另外一种变形动画技术是网格动画(lattice animation)。在三维建模的章节中已经介绍过网格，利用网格可以方便地调

整整个物体的大外形特征，很容易来制作整体的弯曲压缩拉伸等变形，而且只需要对相比模型简单得多的网格来设置关键帧。

路径变形(path deformation)可以算是加入变形要素的路径动画，当柔软的物体沿着一条曲线路径运动的时候，物体的外形也会符合曲线的形状而变形。这种方法经常用来做蛇或者鱼类动物的游动动画。

图 14-30 双头蛇沿路径移动

镜头动画

镜头动画(camera animation)主要是调整摄像机的各种参数来模拟电影中镜头的移动。除了对摄像机位置角度和镜头焦距等参数设定关键帧，比较常用就是上面章节说过的运动轨迹，只不过这里将摄像机连接在运动轨迹上。有些三维系统不可以将摄像机直接连接在轨迹上，一般采取将物体连接在轨迹上之后再将摄像机连接在物体上的方法来实现路径动画。还有可以将摄像机连接在复杂运动的三维物体上，来制作肩扛式摄像机的效果。甚至可以自动调焦，这样不管摄像机怎么运动都还是对着需要摄制的物体的。

灯光动画

灯光的变化可以很好的表达情绪变化，而且通过灯光强度的变化可以吸引观众的注意力。一般的灯光动画就是对灯光的各种属性设置关键帧，或者像摄像机的路径动画一样将灯光连接在物体或路径上，还有就是用来模拟日光变化，通过投射在物体上的影子变化来显示时间的变化。

对于自然现象中光影的模拟，比如火焰、爆炸、水面的光影反射等等，三维软件通常混合使用灯光动画和粒子系统等复杂系统(将在后面具体介绍粒子系统等)来制作，但是因为计算这些复杂系统需要消耗大量时间，于是出现了一些简单的消耗小的窍门来模拟这种自然的光影效果。

一个很普遍使用的技巧是用来模拟火光在建筑上的投影的。

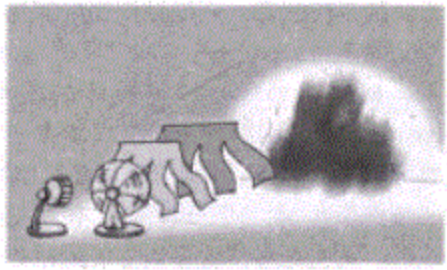


图 14-31 小窍门

如图 14-31，使用局部灯提供光源，用风扇吹动红色、黄色或橙色半透明的布条，最后投影在建筑物上做出晃动的火焰效果。还有用来模拟波光荡漾的技巧，就是在局部光源外套上两层半透明材质，材质则是相互垂直的波纹状图像，如果材质做得好可以做成边缘无缝循环的，这样就可以循环动画了。有些三维游戏中则直接用两层波纹材质来表现水面。

类似的窍门还有很多，但是随着计算机系统速度的不断升级，这些为了减少资源消耗的方法不再显得那么必要，而且这样方法所做出来的效果的确也不如复杂系统所做出来的，倒是在各种实时三维游戏中大量地使用这种窍门方法来有效地制作出“实时”灯光效果。

表面材质动画

表面材质动画主要是通过设定表面材质各种参数的关键帧来制作关键帧动画，比如材质的位置大小角度等等。还有就是利用各种特殊材质贴图，比如凹凸贴图(bump texture mapping)上下幅度的变化制作地表的震动，利用自发光贴图(incandescence mapping)动画做出物体表面光怪陆离的感觉。三维贴图中的实体材质贴图(solid texture mapping)变化更丰富。

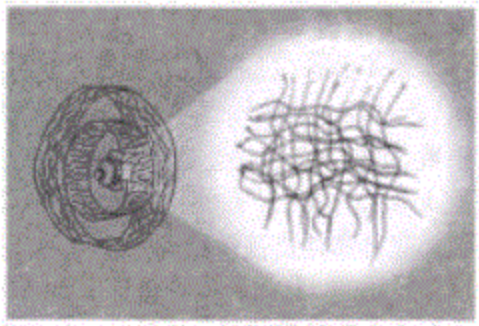


图 14-32 水面的表现方法

材质动画中有一种特殊的技术是图像序列动画(images sequence)，最常用的就是用来制作在三维场景中正在播放的电视或者电脑屏幕类的东西。基本思想就是用二维图像序列按时间顺序，依次贴在三维模型上，而为了节省资源，使用的二维图像序列经常是可以循环播放的。其实这种图像序列方法可用处很大，在很多实时游戏中，因为灯光处理太过消耗资源，有时候就用图像序列动画来模拟灯光的变化。比如篝火周围的光影忽明忽暗就可以用在地面上的循环图像序列来表现。

图像序列动画也经常用来做人物表情，很多卡通类人物的面部并不是用建模实现五官的，而是通过贴图来表现五官。所以要通过设置不同的五官贴图来实现表情变化，只是这里的二维图像就不再是按时间顺序贴图，而是按关键帧来贴图的了。

高级动画技术

动力学模拟

一个小皮球落在地上弹起来，然后跳呀跳，到最后慢慢停止。如果我们要用三维动画来表现这种运动，有经验的动画师完全可以用手工完成。但是更复杂的运动，比如说一个箱子从楼梯上翻滚落下，就不是那么容易凭手工完成了。因为要生成真实生动的动画，就要使箱子的落下遵循物理原理，要考虑很多物理因素，比如重力、密度、体积等等。动力学模拟动画(motion dynamics)就是解决此类复杂动画的较好的办法。

动力学模拟是基于物理模型(重量、速度等)，对三维空间中物体的运动进行模拟，通过计算得出效果。大多数三维软件允许用户通过界面来设定各种参数，然后系统自动计算生成动画。使用动力学模拟，首先要定义一个物体的各种物理参数。比较重要的物理参数

有质量、密度、弹性、静摩擦力和动摩擦力等等。这些参数不仅可以对应场景中的物体，也可以对应整个环境，比如环境密度(environmental density)在模拟水中运动时是不可缺少的一个参数。

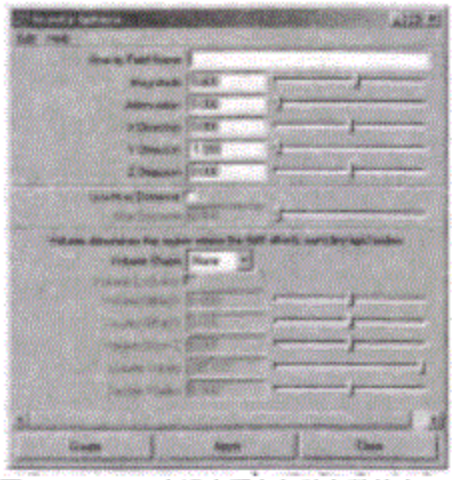


图 14-33 Maya 中设定重力各种参数的窗口

另一个动力学模拟中重要的概念是力。大家最熟识的力就是重力(gravity)，重力影响着空间中所有的物体。对于大多数三维场景，简单来说可以理解为将所有空间中物体向 Y 轴负方向拉的力量。

风力也是经常使用的力量来源。一般用位置、方向矢量和强度来定义风力。大多数三维软件生成的风力可以均匀地影响整个场景。有的三维软件提供更复杂的风源：风扇。和现实中的风扇一样，其作用范围是一个以自己为顶点的圆锥体，风的方向则是以顶点向外发散性发射而不是平行的，风的强度也随着与顶点的距离而逐渐减小。

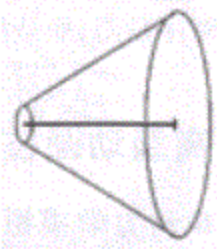


图 14-34 风扇的作用范围在梯形区域内

还有一种力量来源称为关键帧加速度(keyframed acceleration)，就是在某一个关键帧处使用外力给物体一个初速度或者加速度，然后用动力学模拟来计算这一关键帧之后一段时间内的物体运动。

动力学模拟的另一个重要用途是处理碰撞模拟和碰撞检测(collision detection)，比如前面箱子从楼梯上滚下的例子，箱子落到台阶上然后弹起的运动就是碰撞模拟，而计算什么时候什么位置发生碰撞则是碰撞检测。

碰撞要考虑参与碰撞的物体各自的速度、质量、弹性等属性，然后根据物理定律计算出物体的变形程度和反弹速度等结果。碰撞检测则主要是根据物体外形计算是否相互碰撞。碰撞模拟和碰撞检测是非常消耗计算资源的，尤其是外形复杂的物体之间的碰撞检测，所以大多数三维动画软件都会提供各种折衷简化的方法。一种方法是在碰撞检测前判断空间中的其他物体是否为此物体的障碍物，也就是根据此物体的可能的运动轨迹，将绝不可能与它碰撞的物体设置为非障碍物，从而在碰撞检测计算中减少了需要考虑的物体数目，节约了计算资源。另一种方法是使用包容立方体(bounding box)来简化物体外形，如图 14-36。

图 14-36 人物模型的原型和用包容立方体简化表现的人物

使用包容立方体简化了物体的外形，可以节省大量的计算时间。由包容立方体衍生出

来的还有包容球体(bounding sphere)、包容平面(bounding plane)等等。

对于结构复杂的物体，可以利用层次结构，只对物体的局部进行动力学模拟从而节省资源。比如被风吹动的窗帘，如果计算风力和重力对整个窗帘的影响，计算过于复杂，可以采取这样的简便方法：给窗帘加上平行排列的 IK 骨架，然后只对 IK 骨架的每个关节进行动力学模拟，骨架运动则带动窗帘运动。

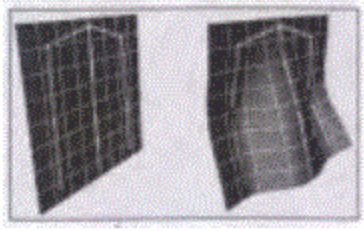


图 14-37 加上骨架的窗帘

粒子系统和类粒子系统

自然界的许多现象，比如烟雾、火焰、云彩等等，此类自然现象中的个体存在时间非常短暂，总是不断地进行新旧更新，或者个体极小乃至看不到，只有当个体数量达到足够多后才可见其整体效果，很难用表面建模的方法来实现。粒子系统(panicle system)是模拟此类自然现象的最好办法。使用粒子系统，不需要去构建独立的表面，而是定义粒子系统各种参数，系统会自动模拟产生效果。

粒子系统需要设定很多参数，首先要设定的是粒子系统所模拟的自然现象的类型，比如烟雾、火焰、云彩等等。不同类型的自然现象，有不同的控制参数。比如烟雾，就可以设置烟雾飘散的方向、移动的速度、和扩散的速度。对于总体效果的控制，则有个体的数量、密度等参数。最后要设定粒子个体的属性，有颜色、大小、形状、生命周期(life span)等等。

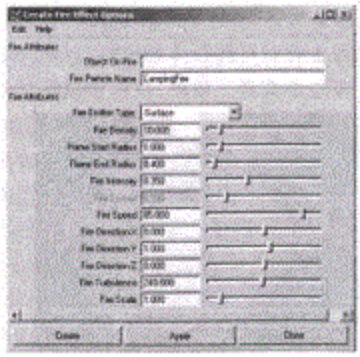


图 14-38 Maya 创建火焰窗口

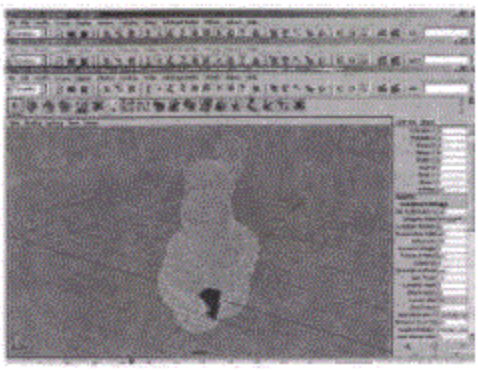


图 14-39 调整各种参数，来达到想要的火焰效果

生命周期是决定个体存在时间长短的属性，以火焰为例：如果个体的生命周期很长，总的效果就是火慢慢地燃烧，更新得很慢；如果生命周期短，则是火苗冒得很快，新生的火苗一下子就燃烧到火焰的顶点然后消失。

一般来说，粒子系统的个体是不可见的或者极其简单的，只能通过调整其参数来加以控制，而无法对其个体进行自由修改。为了克服粒子系统的局限性，人们在粒子系统的基础上发展出了类粒子系统(particle—like system)。两者的主要区别是类粒子系统可以直接对表面模型进行操作。

类粒子系统最常见的应用是在爆炸(explosion)效果生成方面。其基本思想是这样的：物体表面模型是由多边形组成，而多边形一般都要再细分成一个个三角形，那么我们可将物体的模型“打破”成这些小多边形或者三角形，从而形成爆炸效果。三维动画软件一般都可以通过调整参数控制碎片的面积、范围、或者形状，比如对某种类型的爆炸要保持一些面积较大的四边形面。对于爆炸点在物体上的位置和爆炸的强度等等参数也可以调整。

图 14-40 Maya 的火焰效果

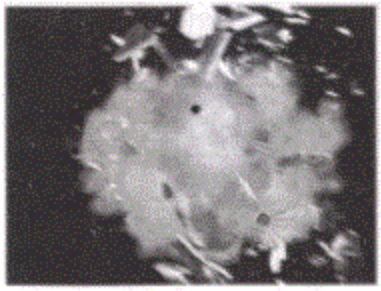


图 14-41 使用 Maya 生成的爆炸效果

类粒子系统的另一个用途是模拟群体移动(flocking)，比如鱼群、羊群或者两军交战之前的冲刺等等。对群体移动来说，首先每个个体不再是极小的，而是有复杂的三维模型，甚至可以是一个三维模型的动画序列。比如鱼群游动，就可以将建立好的鱼的三维模型定义为个体。这种群体运动的特点是总体上有大的移动方向和群体的形状，而对于每个个体来说移动速度方向则各个不同。使用三维动画软件生成群体移动效果，一般只需要设定群体的大致外形、开始和结束的位置以及群体移动的路径。

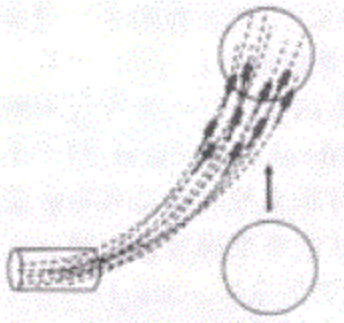


图 14-42 鱼群移动

图 14-43 蜂群移动

如图 14-42 所示，鱼群的移动被限制在管状外形中，而每条鱼的路径却不同于群体的路径。对模拟群体移动的方法进行变化，也可以做出无规则运动的群体效果，比如蜂群的满天飘舞(图 14-43)。

过程动画

过程动画(procedural animation)是一种通过编写程序来生成动画效果的技术。一般三维动画软件创建三维模型和制作三维动画都是在视图界面下完成。实际上，所有人机界面下的操作都可以用该软件所附带的专用编程语言来实现。因为 C 语言的流行性和可读性好，三维动画软件的专用编程语言大多数都类似于 C 语言。

使用三维动画软件的人机界面进行的所有操作，都会被存储在场景文件(scene file)里。这些场景文件就是用专用编程语言写成的，可以在文本编辑器中编辑。也就是说可以直接在文本编辑器中用编程的方法创建物体和修改动画。有些软件比如 Maya，在人机界面下也提供编程语句输入框或编辑器(script editor)，可以直接输入语句。

前面说了在三维动画软件中一般提供专用编程语言，可以使用编程语言来生成动画。那么使用编程的方法有什么优越性呢?使用编程的方法，可以生成很复杂的、需要精确控制和特殊效果的动画。

上面提到的粒子系统就是一种简单化的过程动画，基于物理原理的程序是早就完成并且封装在粒子系统中的，动画师只需要调整各种参数就可以了。如果不满意只能调整有限的参数，有些软件则可以用编程的方法来控制更多的属性，以营造出更有新意的效果。

使用程序可以完成很多难以通过手工生成的动画。一个有代表性的例子就是那些需要精确定位的动画，比如《Metal Gear Solid 2》片头中 DNA 双螺旋的合并，用手工设定关键帧的方法来控制运动和物体变形难度很大，只有利用数学公式并通过编程才能够准确地控制螺旋运动和合并方向。过程动画也经常被用来模拟植物的生长。如图 14-44 所示，利用过程动画模拟植物从种子萌芽、根部生长、开花和结果一系列生理过程。模拟不同类型植物的生长和外貌形态，或者模拟同类植物在不同环境下的生长区别，都是过程动画的优势。比如利用过程动画来生成一片森林，在几种树的个体模型的基础上，利用程序对日光角度、树的年龄、土壤状况等等自然因素的设置，再加上一定随机量，就可以创建出有丰富变化的森林。

图 14-44 植物发芽

目的驱动动画系统(goal oriented animation system)是一种尚在实验室中的更高级的动画系统。使用这种系统，只需要指定一定的目的，人物或机械的动画就可以自动生成。比如说一个人，给他的命令是去拿桌子上的杯子，系统就会自动让人物走到桌子附近，然后让他弯腰伸手拿到杯子。显然这种系统和传统的三维动画软件比以起来有极大的优越性。使用传统的三维动画软件操纵动画人物去实现一定的目的，你需要“告诉”它如何一步步去达成目的，每一步的细节都要输入；而目的驱动动画系统，所有这些步骤都省略了，你只需要制定它最终所要达到的目的即可。

这里简单介绍一下动画专用编程语言中比较有代表性的 MEL。MEL(Maya Embedded Language)是三维动画软件 Maya 所定制的类似 C 的编程语言，有着强大的功能，可以直接控制 Maya 底层。用 MEL 构建人机界面，创建各种控件，比如菜单、输入框、下拉选择框、滚动条等等。这样可以构建适合自己的更简单快捷的界面。

利用 MEL 也可以控制物体的各种属性，在各种属性之间建立其联系来。如图 14-45

和图 14-46 所示，火车车轮的转动就是根据火车的行进速度来控制的。

图 14-45 火车轮子的转动手工很不好调整

使用 MEL 对 Maya 的底层进行控制，可以实现很多复杂的动画，如图 14-47，一个小项目 Smart Camera，利用电影镜头理论，对三维空间中摄像机的位置、焦距和运动进行自动控制，自动进行镜头切换。

使用 MEL，可以改写新的表面属性或者制作复杂的群体动画，更好地控制粒子系统和动态模拟系统等等。

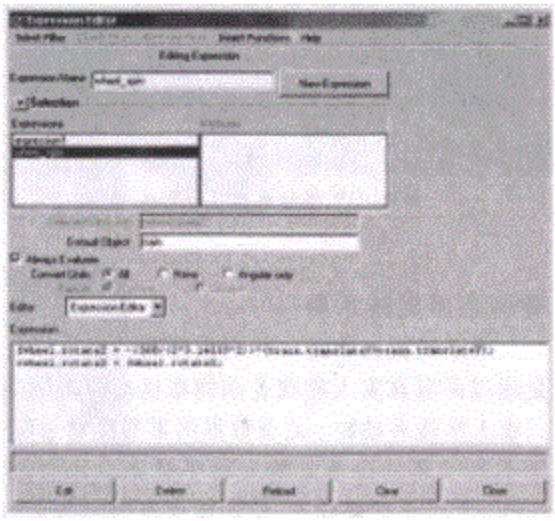


图 14-46 通过编程则可以很轻松的实现，使用 Maya 的 Expression Editor

过程动画可以说是比较高难度的动画制作手段。在商业动画片的制作中，经常是由动画师和程序师协同工作，动画师提出要求，比如某特殊效果或者特殊的材质，程序师就要利用各种模型来编程实现这种想法。因为现有三维动画软件所提供的工具经常满足不了创新的要求，每一部出名的三维动画片都是伴随着几种新技术而诞生的。只有使用过程动画这种高级的方法，才能实现创新，而不是简单地使用工具。

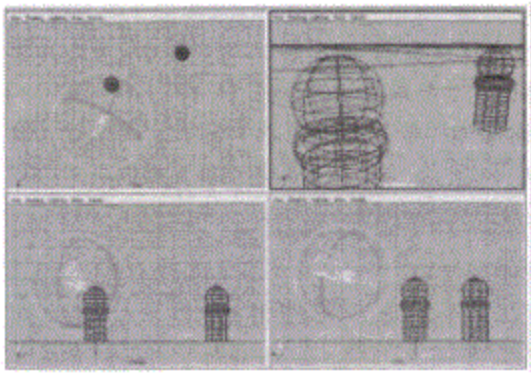


图 14-47 摄像机的自动控制

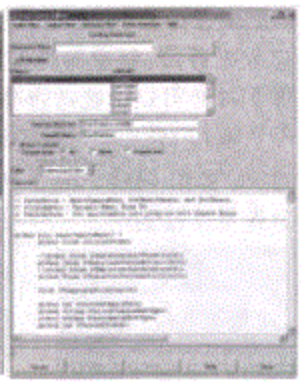


图 14-48 在 Maya 下编辑程序

动态捕捉技术

动态捕捉技术是一种应用普遍的获得动态信息的方法。基本原理是通过捕捉真实人物或者动物在运动时关键部位的运动数据，来驱动三维人物或者动物。动态数据的采集经常采取分层的方法，比如利用精度不是很高的设备采集人物手臂身体等大范围动态(basic tracks of motion)，而在此基础上，使用精度高的设备采集人物表情或手指运动等细节动态(Secondary motion)，最后合成动态来驱动三维人物。也可以在大范围动画的基础上，手工

调整细节动画。

动态捕捉中一个基本概念是感受器的数量。感受器就是附着在演员身上不同部位的数据采集器，根据精细程度和性能的不同，感受器的数量和位置也不同，从 70 多个感受器的高级系统到十几个感受器的廉价系统。

大多数动态捕捉系统都是对应人体运动的，其实对于人物表情也可以利用这种技术来捕捉，就是表情捕捉(face tracker)。使用数十个脸部感受器来取得关键表情的皮肤位置，在结合上面提到过的关键帧动画来制作脸部表情。

上面所说的动态捕捉，基本上都是实时的。在演员动作的同时，数据就按时间顺序被记录下来。对于数据的编辑，都会有专业的编辑器方便地对每个感受器各种属性曲线的调整修正。比如对于人体的关节，因为没有办法将感受器放在关节中心，而只能贴在表面，所以需要消除一段距离上的误差。

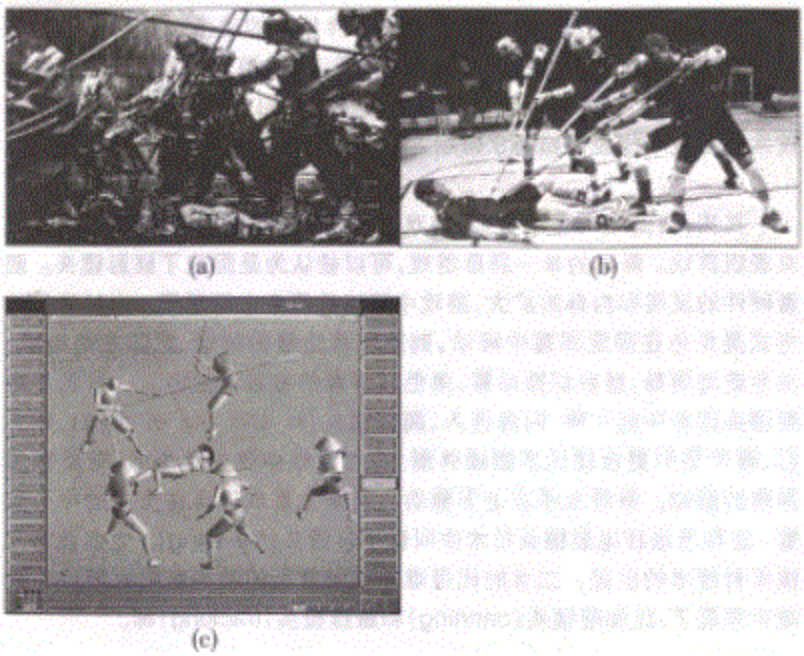


图 14-49 (a) 为《鬼武者》的游戏片头动画。(b) 为动态捕捉时的情景。(c) 是将动态捕捉获得的数据输入三维软件中并就用到简单的三维模型上的效果

第十五章　镜头与剪辑

硬件技术的进步总是不断地推动软件技术的进步，从而进一步提高艺术表达手段的多样性和复杂性。随着 32 位游戏机和 PC 上高性能显示卡的问世，三维图形也已经成为现在游戏的主流，从而为电影镜头技术在游戏中的应用扫清了道路。本章将介绍基本的镜头和剪辑技术。

其实，自游戏问世之日始，游戏设计师们就尝试着使用电影的方式来表现游戏。最早的单一屏幕游戏，可以被认为是固定了摄影镜头。随着硬件的发展和内存的扩大，游戏中可以使用多个场景了。当时常见的方式是角色在固定屏幕中移动，到达屏幕边缘的时候，比如左边边缘，角色走出屏幕，然后切换场景，角色从屏幕的右边边缘进入。这就是电影镜头技术中的一种，叫做进入、离开镜头(in and out of shot)，再之后，程序员们更合理地发掘硬件潜力，在游戏中使用了卷轴，就是随着角色的移动，背景水平或上下卷动，而角色基本保持在屏幕的中心位置。这种方法在电影镜头技术中叫做平移镜头(traveling)。之后随着三维实时技术的出现，二维时代很难使用的复杂的镜头变化也可以在游戏中实现了，比如摇镜头(panning)和跟踪镜头(tracking)等。

在三维游戏中应用电影镜头技术，既包括游戏中的实时画面，也包括事先做好的动画片头和过场动画。对前一点来说，最具有代表性的就是 AVG 游戏了。原因很明显——目前的 AVG 从构成上来说最接近电影。从早期的 PC 游戏《鬼屋魔影》(Alone in the Dark)开始，三维技术的应用，使得 AVG 中的主角可以在虚拟三维空间做纵深的移动，解决了以前二维游戏必须平行设置镜头的问题，使得镜头的使用更为多样化，从而实现更强烈的视觉效果，这对恐怖游戏是至关重要的。对后一点，动画片头和过场动画可以被看成是电影中的一个片段，因此可以完全照搬电影的镜头和剪辑技术。

图 15-1 PC 上的《鬼屋魔影》(Alone in Dark) 是最早的三维游戏之一。镜头的运用也是第一次变得复杂多样。

基本概念

在介绍具体的镜头技术之前，需要搞清楚一些电影方面的基本概念。

三种场景

电影是由一系列场景组成的。所有场景又可分为三种：

●单纯对话场景：对话的双方没有剧烈的运动，比如说一对恋人坐在露天咖啡厅闲谈。

●单纯动作场景：比如说在追逐场景中，意大利黑帮的两个人在小巷里一前一后地奔跑。观众只能听到奔跑者的沉重的呼吸声，和碰倒垃圾桶时发出的清脆的声响。整个过程没有对话，整个屏幕上充满了紧张的气氛。

●既有动作又有对话的场景：警察小组在攻人一栋房子时，各部分一边搜索前进一边紧张地相互汇报情况。

三种运动

电影中可以有三种不同的运动方式。

●镜头固定，人物或物体在镜头前运动。

●静止的人物或物体，镜头运动。

●镜头和人物同时运动。

镜头和剪辑

本章要介绍的两个问题就是如何配置镜头和如何剪辑。镜头的配置主要是按照一定的法则，把镜头摆在合适的位置上。然后开动镜头，拍摄一些片段。而镜头的剪辑就是使用已有的拍摄好的片段进行排列、调整、连接，从而把一个完整连续的故事情节讲述出来，使得观众能够理解因果关系。这两种技术是紧密相联相互影响的。一般是先设计好镜头位置和要拍摄的镜头序列，在这时候要预先考虑到后面剪辑的问题。然后按计划拍摄，最后根据拍摄的实际结果和先前的方案来对照着进行剪辑。下面就以镜头的拍摄和剪辑为主介绍一些基本的技法和注意事项。

镜头技术

镜头距离

根据镜头与所摄物体之间距离的不同，镜头基本可以分为特写镜头(close up)、近景镜头(close shot)、中景镜头(medium shot)、全景镜头(full shot)和长镜头(long shot)。图 15-2 以一个人物角色为例展示了各种镜头的区别。

图 15-2 不同镜头距离示意图

当然，这种分法是相对的，并不是以严格的距离测量为基准的。一个房屋的近景和一个人的近景，摄像机和房屋之间和摄像机和人之间的距离显然是不一样的。

图 15-3 镜头高度示意图

对人物角色来说，又可以细分成极度特写镜头(extreme close up)、中度特写镜头(medium close up)、腰部镜头(waist shot)、中景镜头(medium shot)和膝盖镜头(knee shot)。如图 15-3 所示。

一个小窍门：在使用针对人物角色的长镜头的时候，一定要把角色的脚收入镜头。从脚踝剪辑的长镜头会给人以不舒服的感觉。

镜头角度

按镜头的角度来分，还可以分为高角度镜头(High Angle Shots)和低角度镜头(Low Angel Shots)。高角度镜头也是大家常说的俯视镜头，从上面俯视场景，很容易营造一种居高临下的感觉。而低角度镜头就是仰视镜头，抬头向上看的视角，给人一种被压制的感觉。

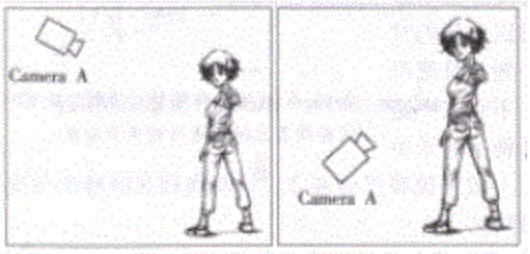


图 15-4 高角度镜头配置 图 15-5 低角度镜头配置

利用仰俯视给人的压抑感可以营造很有意思的感觉。影视作品中最常用也用得都烂了的但是还是十分有效的，就是一个彪形大汉面对一个个子矮弱小的人的时候，会对比的连续使用彪形大汉的俯视视角和小个子的仰视视角，更突出大汉的魁梧和小个子的弱小。


图 15-6 高角度镜头效果，角色处于劣势位置，也反映了角色当时的境况


图 15-7 低角度镜头效果 ,仰视地看角色可以表现出角色的自大

三角形法则

三角形法则是用来处理对话场景的镜头配置的。先介绍一个基本概念：兴趣线。

兴趣线(line of interest)是一条沿着两个正在交谈中的演员相互对视方向的直线。两个交谈的演员和他们之间的兴趣线可以从三个位置(使用三个镜头)来观察，即两个极端位置和一

个侧面位置。如图 15-8 所示。

这三个镜头位于兴趣线的同一侧。它们构成了一个三角形，这个三角形的底边平行于兴趣线。三角形法则就是说：在摄制对话场景的时候，要选择兴趣线的一侧并且把所有镜头都放到这一侧，不可以中途穿越兴趣线。这是电影镜头技术中一条非常重要的法则。一般来说要严格遵守。只有在极为特殊的情况下，才能有所变化甚至打破。

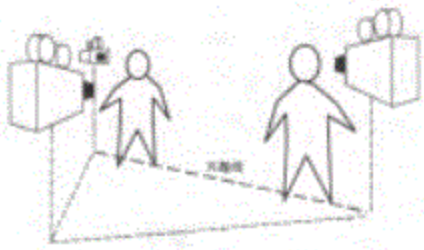


图 15-8 兴趣线和三角形法则。两个对话的演员之间连线就是兴趣线

一般来说，三角形法则里镜头的排列有五种基本的变化。它们是：

- 外部相反角度(external reverse angles)
- 内部相反角度(internal reverse angles)
- 平行位置(parallel positions)
- 右角度位置(right angle positions)
- 普通视觉轴(common visual axis)

以下分别介绍：

●外部相反角度(external reverse angels)

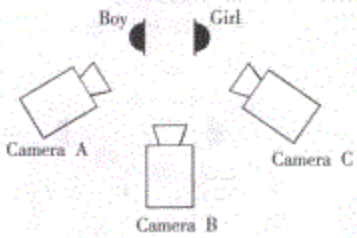


图 15-9 镜头配置示意图

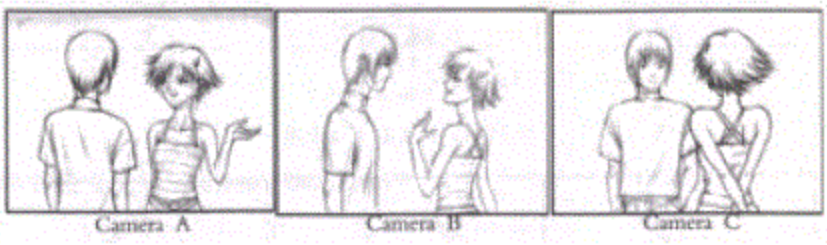


图 15-10 效果示意图

●内部相反角度(internal reverse angels)

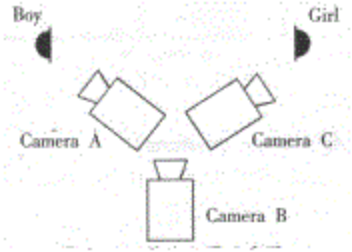


图 15-11 镜头配置示意图

图 15-12 效果示意图

平行位置

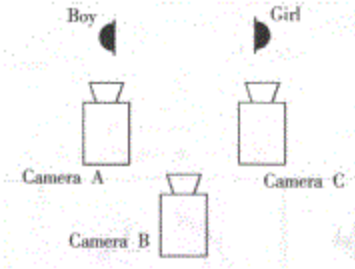


图 15-13 镜头配置示意图

图 15-14 效果示意图

●右角度位置(right angel positions)

当演员并肩站立成“L”型的，较常使用右角度位置。

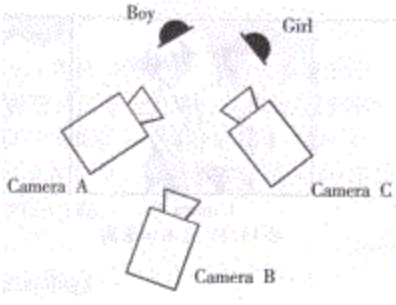


图 15-15 镜头配置示意图

图 15-16 效果示意图

●普通视觉轴(common visual axis)

当镜头里只需要容纳一个演员的时候，位于三角形底边上的摄像机之一，就要在它自己

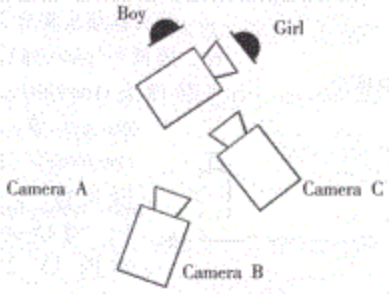


图 15-17 镜头配置示意图

图 15-18 效果示意图

的视觉轴上拉近与演员的距离。

镜头移动

前面介绍的三角形法则，是用于指导对话场景中镜头的固定配置的。下面我们要介绍的是镜头移动的技术。在介绍具体技术之前，需要强调一点：镜头移动就像一个威力无穷但是没有保险的武器，不光会带来丰富的变化，也极可能破坏画面的视觉效果。不正确的使用会打乱整体的节奏，甚至完全篡改你想表达的意思。以至于著名导演约翰·福特曾说过要完全取消镜头移动，用剪辑来代替。当然，这种回避并不是一个好方法。我们需要先了解镜头移动的特殊性和其规律，然后合理地使用之。

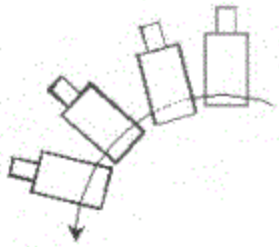


图 15-19 摇镜头

图 15-20 水平镜头移动

镜头移动主要有 3 种：摇镜头(panning)，平移镜头(traveling)，和调焦镜头(zooming)。分别如图 15-19、图 15-20 和图 15-21 所示。

图 15-21 调焦镜头

●摇镜头(panning)

摇镜头经常用于覆盖全景，比如介绍故事发生时的环境。最简单的例子就是在硝烟弥漫的战场上，从站在高处观战的将领的视角扫视全景。摇镜头也可以用来跟踪

一个运动的物体，比如说在一条小道上一个武士骑马而过，固定在小道一侧的镜头摇镜头60度，记录了武士由远到近到离开镜头的全过程。

●平移镜头(traveling)

平移镜头经常可以作为一种追踪镜头来追踪演员的移动，尤其是在较拥挤的复杂场景中，比如地铁里、集会场景。最简单的例子如战场冲锋，镜头对着骑兵们的侧影，跟着一同平行前进。

●调焦镜头(zooming)

调焦镜头是最常用的技术之一。包括拉近和拉远。拉近镜头可以用来把观众的注意力从群体引导到个体上。比如说镜头前有很多人，镜头拉近到一个人身上，这个人是主人公。拉远镜头可以用来揭示人物周围环境。比如说一个主角的特写，他正在往四周看，似乎在寻找什么。然后我们使用拉远镜头，显示他周围的环境，包括他寻找的目标。

调焦镜头一般比较短暂。因为在调焦镜头中，基本的内容是不变的，观众没有耐心看太长时间的调焦镜头效果。另外，如果调焦镜头的时间特别短的话，可以产生十分强烈的效果，甚至可以吓唬观众们一下。

一般来说，当上面的三种镜头移动组合使用时，效果更好。最常见的如一边进行摇镜头一边拉近。

●主观视角(subjective view)

如果将镜头的移动做成像从演员的眼睛的角度看一样，叫做主观视角(subjective view)。这种方式给观众很强的代入感。从演员的视角看整个场景，从镜头的移动中体会人物的感触。但这种镜头不是平稳的，是蹒跚摇移的。如何把这种不稳定表现得恰如其分，需要很多经验。因为，如果处理不好，观众会明显感到不适，好像你在玩一种小把戏来欺骗他们。有时候，使用主观视角的镜头来表现人物刚进入新场景时对场景的环视。

我们可以利用摇镜头和平移镜头，在镜头移动结束的时候引导出新的剧情。比如用摇镜头来拍摄屋子里的设置，慢慢移动到壁炉旁边，发现地上有血液在慢慢地流，继续顺着血液移动，停在一具趴在地上的死尸上。这里面的技巧就是如何利用移动过程中所拍摄到的事物和流动感，来引导观众的悬念，然后展示出观众期待中或者没有想到的景物，达到某种程度的惊喜。

摇镜头和平移镜头也可以作为转移故事的兴趣中心的一个手段。比如镜头先定位在一个

静止的演员身上，然后另一个演员走过，镜头跟踪移动的演员，移动的演员移动到另外一个静止的演员的时候停止，再定位在这个新的兴趣中心。这样的摇镜头或平移镜头，由三个部分组成：开始的时候镜头是静止的，中间的镜头移动阶段，和最后的结果阶段镜头再次静止。

使用镜头移动是有一些窍门的：

●在拍摄一个从静止到移动再到静止的人物的时候，要让人物在镜头移动之前移动，让镜头在人物停止之前停止，人物在镜头停止后一定要再在屏幕上移动一些距离。

●使用移动镜头拍摄移动中人物，当你想从移动镜头剪辑到同一个人物的静止镜头的时候，有两点一定要遵守：在移动镜头和静止镜头里，一要将人物保持在屏幕的相同区域里，二要人物的移动方向保持一致。

●要让镜头移动的轨迹尽量简单，把复杂的移动交给演员或者交通工具去完成。

●摇镜头和平移镜头的一个作用是重新建立画面的平衡。当原来场景中的某个演员离开屏幕，剩下的画面会变得不平衡，这时候，稍微的镜头移动就可以有效地重新建立画面的平衡。

最后再强调一下：在你决定要使用镜头移动的时候，自己一定要事先有一个明确的概念如何让镜头移动，而这一切都是为了更清楚、更富于动感、更准确地来描述你的故事。

动作场景

没有动作场景，电影就不会受到观众如此的欢迎。观众们喜欢在电影中看到汽车追踪、战场搏杀等令人血脉喷张的场面。动作场景的感染力是很大的。但是动作场景一定要非常简单明了，不能让观众产生任何的疑惑或者感到混乱。简单地说，动作的动机应该清楚，并且要有足够的细节。

●四种动作场景(four basic types)

一般来说，动作场景有四种：追逐场景，搏击场景，机械对抗场景和事故场景。前两个比较好理解，不用在此介绍。机械对抗场景指的是人和机械的对抗。比如说侦探发现了定时炸弹，并试图拆除。这就是一个经典的机械对抗场景。这时候屏幕上就会出现主人公脸部的紧张表情，手部和炸弹的特写(各色导线什么的)，嘀嗒嘀嗒的声效，经过多次努力终于排除了炸弹。而更大的机械对抗场景的例子，如《泰坦尼克号》中，发现冰山后，大副命令锅炉房的工人把主机倒转。这时只见锅炉房的工人们紧张地操作，和巨大的机器对抗，主轴慢慢停止并开始倒转。事故场景，简单的如追逐过程中引起车祸碰撞；复杂的，如山崩、桥断、楼塌等。

这四种动作场景都是双方或者多方之间的相互作用，如再引入时间因素，就不仅有双方的冲突，又有和时间的冲突，效果就更强烈了。

●标准公式(standard formulas)

下面介绍一些动作场景的标准公式。所谓标准公式，实际上是经过长时间实践人们总结出来的各种经验。你可以把它们作为设计动作场景的依据，也可以用它们来评价你设计的动作场景因果是否清楚，节奏是否合适。

●对话在动作场景中的负面影响：观众们需要看到发生了什么，而不是听你的讲解。在激烈的动作场景中，不会有长篇累牍的对话，只有简单明快的命令。

●对话可以让情绪紧张的观众得到短暂的喘息：持续激烈的动作让观众的情绪饱和，这时候就需要通过一点对话或者其他东西来打断连续激烈的动作来给观众一个喘息的机会，好让他们在进入动作场景的最后高潮之前缓口气，而且让观众有个短暂的时间来思考和理解当前态势等等。

●一个动作场景可以用很长时间来建立来渲染气氛，然后结束得很快：比如常见的西部片对决场景，在真正开枪之前，要花很多时间来建立这个场景，然而实际的对决，也就是开枪部分，几秒钟就结束了。

●除非动作场景的目的是夸张搞笑，否则的话动作必须是可信的：动作不能过分做作和夸张，其因果必须可信，也需要符合物理规律。当然这是根据不同的题材来决定的。在神话故事中，飞翔术和瞬间移动等魔法就是可信的。而在现实题材的故事中，就不可以胡乱安排。

●双重追踪：当一个猎人在追寻一个猎物的同时，对他们任何一个感兴趣的第三方同时也在追踪他们两个，这就是双重追踪。和螳螂捕蝉，黄雀在后异曲同工。这样的安排，让动作场景更为复杂富于变化。如果有多组人同时在追踪主角，就更容易让观众感到紧张。

剪辑技术

剪辑，简单地说就是把不连续的场景和片段凑在一起，从而形成故事的流动和一定的因果关系。电影和戏剧的不同之处，就在于电影是不连续的。当我们去剧院看歌剧话剧的时候，我们是从一个角度连续地看完所有的表演，就好像在看一部用一台摄像机不停地连续拍摄的纪录片。镜头的角度不变，距离不变。在早期电影中，电影工作者们也是采用这种方法。他们还没有意识到电影作为一种独立艺术形式的特殊性。慢慢地他们开始尝试使用不同距离和不同角度的镜头，使得观众可以从各个方面来观看表演。而那些从不同距离，不同角度拍摄的片段，需要连接起来，需要保持一定的因果关系，使得观众能够理解并接受。剪辑技术就

这么发展起来了。

视觉标点法

不同的镜头片段或者场景之间一般用两种方法来剪辑在一起。第一种是直接的剪辑，就是直接两段头尾相接，中间没有过渡。而第二种是通过各种视觉效果来作为过渡，也就是所谓的视觉标点法。我们在写文章的时候，在句子与句子之间要加上标点，来表示它们之间的关系。同样地，在镜头片段与片段之间，我们也要加上这种“视觉上”的标点，来使得它们更好地连接在一起。下面先介绍最常用的几种手法。

●淡入和淡出(fade out and fade in)

这是最常用的一种方法。前一个场景逐渐暗化变黑，后面的场景逐渐显露。这个变化效果可以是发生在一个静止的帧上，也可以由许多帧完成。

●白屏和彩色淡入淡出(white-outs and color fades)

这是普通的淡入淡出的一种变化，场景可以淡入成白色屏幕，然后新的场景再显现。也可以使用其他单色，比如说前一个场景中的最主要的颜色。先由一种这种颜色逐渐扩张浓化直到完全遮挡住原来的画面然后再慢慢透明显示出下一个场景。

●溶解效果(dissolve)

溶解效果是淡入和淡出的组合应用，在前一个场景淡入的同时后一个场景淡出。

●擦抹效果(wipe)

在介绍新场景的时候，可以使用擦抹效果。有两种擦抹方法：一种是新场景从屏幕的一边进入将旧场景推出屏幕；另一种新场景进入的时候，旧场景不动，由新场景直接把旧场景擦掉。第二种方法有表示时间推移的作用。擦抹方向可以是水平垂直或者斜向，擦抹的边界也不见得非要是平的，可以按画面需要使用不用的边界形状，比如波浪形等等。

●虹彩效果(iris)

比较卡通化的效果，使用各种形状的边界放大或缩小进行屏幕的切换。这种效果较常用于电视连续剧或卡通的过场。

以上这些都是比较普通的明显的方法，还有很多类似的方法，你都可以在 Adobe Premiere 里面找到细致的图示。

下面介绍的方法和上面的方法相比更为自然和隐蔽，可以从前一个场景中非常自然地过

渡到后一个场景。

●使用阴暗的区域(use of dark areas)

使用摇镜头或者平移镜头将镜头移向场景中的阴暗区域，直到画面全黑，然后再从下一个场景中阴暗区域移出。两次镜头移动的方式和方向一致就非常流畅自然。这种方式的一个变化就是让主要人物走向镜头，直到他完全遮挡住镜头，屏幕变成黑色，然后剪辑。

一个很著名的例子是希区科克的《Vertigo》。在影片的后半部，男女主人公在旅馆里拥抱并亲吻时，镜头围绕两个人做 360 度旋转，先由房间的背景转到一个阴暗的背景，然后再转出来时，观众和男主人公都发现背景变了，背景变成了男主人公和已死的情人最后亲吻时的背景了。男主人公脸上刹那间露出怀疑惊惶的神态，但马上又低下头去，继续亲吻女主人公。这时候镜头继续围绕两人旋转，背景再次转入黑暗，然后回到旅馆里。这段剪辑得天衣无缝，很见功力。它所达成的是空间时间上的跳跃，但维持了视觉镜头的连续和流畅。(对故事不太清楚的读者：这段说的是通过这么一吻，主人公第一次对女主人公的身份产生了怀疑。因为他在接吻时感到女主人公和他“死去”的情人实际上是同一个人。)

●视觉相似欺骗(deceptive visual match)

图 15-22 视觉相似欺骗

所谓视觉相似欺骗，是指被连接起来的两个片段，前一个的尾和后一个的头的场景相同或相似，给观众感觉是两个片段是一个片段，但实际上他们马上会发现两个片段之间空间上相同，但时间上跳跃。一个很好的例子如《Life is Beautiful》中，镜头对准花房的入口，男女主人公走近花房，夜幕降临。然后天亮了，一个小孩子从花房里走出。这样的衔接非常紧凑，把大量的信息很自然地浓缩在短短的一个场景内，也很好地处理了时间的跨越——男女主人公组建了幸福的家庭，有了孩子。

夜景片段马上就要结束了，接上了新的白天片段，但画面的视觉效果和前面的夜景保持一致(月亮的形状 ,位置 ,云彩等), 所不同的是色调。这样就保持了连续感，过渡得很柔和

两个片段衔接处的画面也可以不是同一个场景，但具有类似的视觉效果。图 15-22 所示的是《FF8》片尾的一段。

●道具(props)

使用在前一个场景中已有的道具，达成一定的效果，来遮盖住屏幕，然后切换到下一个场景。图 15-23 所示是 FF8 的一段片头动画，使用了羽毛做掩饰来切换两个不同场景。

●时间道具(time props)

在镜头中使用时间道具(time props)可以表示时间的推移，比如点燃的蜡烛、点燃的雪茄、篝火、钟表、日历和有着明确表示时间的报纸等等。从光影的变化也可以表现出时间的推移，比如天边云彩场景匹配对话场景的剪辑较难剪辑的场景颜色的变化和影子的变化。人物可以在第一个镜头移动出屏幕，然后时间道具或者光影变化，显示出时间的推移，人物再进入屏幕开始新的情节。

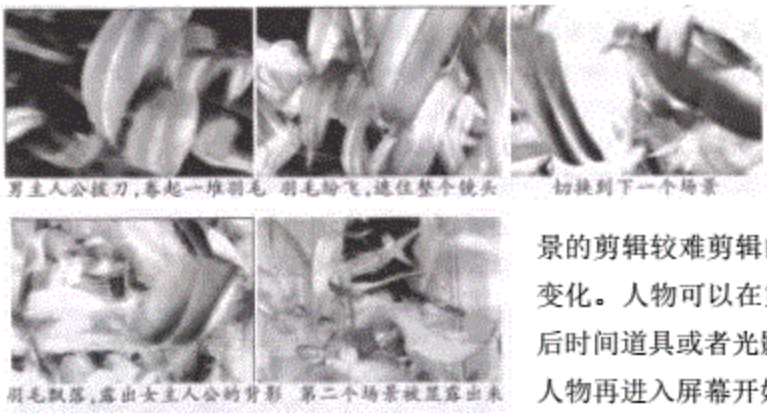


图 15-23 使用道具

●虚假的因果关系(question and answer)

这是一个非常有趣的方法。比如一个人物冲着屏幕问道：“我是不是很帅呀？”镜头切换到一个回答者：“别开玩笑了。”但是这个回答者所处的时间地点和上面一个人物的时间地点完全不同，实际上这个人物回答的问题也不是提问者所提出的问题。这种方法在喜剧中很常见，把貌似有因果关系但实际上八竿子打不着的两段剪辑在一起，可以出现很好的效果。

还有其他很多种方法，比如使用平行剪辑、使用近景镜头切换、镜头变焦使画面模糊等等。

场景匹配

所谓场景匹配，就是说要被剪接在一起的两个场景，应该在视觉上有一定的连续性，不能互相冲突。场景匹配，有三个必须满足的因素，就是：位置(position)、运动(movement)和注视的方向(look)。

●位置匹配(The position)

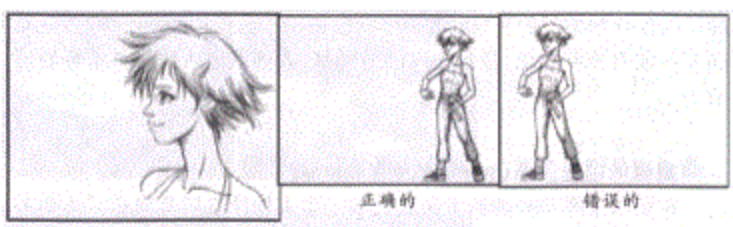


图 15-24 位置匹配

电影屏幕是一个固定的区域，如果一个演员在全景镜头中出现在屏幕的右侧，那么下一个近景镜头中他也要出现在屏幕的右侧。这个就是位置匹配。如果没有遵守位置匹配，就会导致屏幕上突兀的视觉跳跃，观众们不得不被动地把注意力从屏幕的一个区域转移到另一个区域。

一般要将屏幕划分成两个或三个区域，把演员安置其中，达到位置匹配的要求。

●运动匹配(The movement)

运动匹配也是基于相同的思路。在两个连续的镜头里，记录演员持续运动时，运动的方向应该保持一致。否则的话，观众会对演员的运动方向产生疑惑。

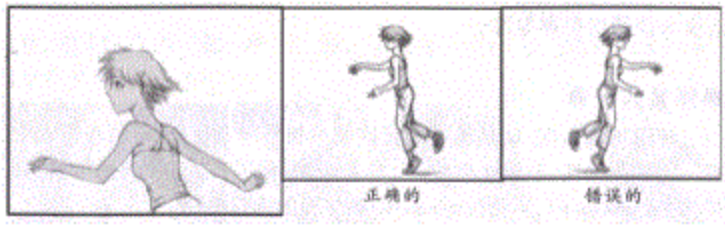


图 15-25 运动方向匹配

●注视方向的匹配(The look)

对话片段开始的主场景(master shot)中包含了两个演员，之后每个演员的单一场景中，要保持两个演员注视方向是相对的。

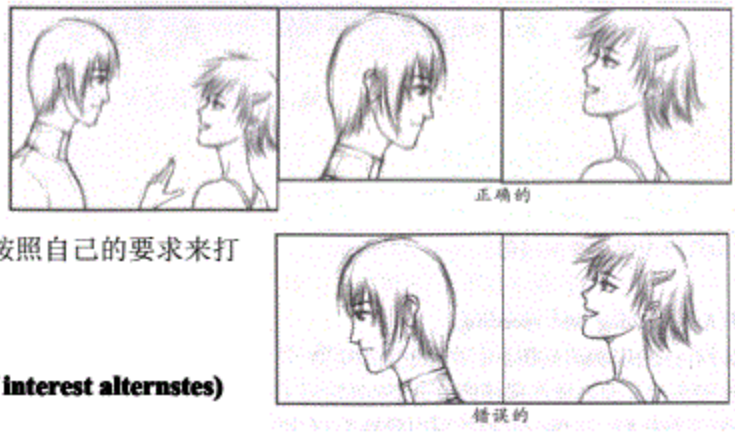


图 15-26 注视方向匹配

需要说明的是：这些是基本的准则。在熟练掌握的基础上，自然是可以进行变化或者按照自己的要求来打破它们的。

●视觉中心的变换(Center of interest alternstes)

在多人以上的场景中，一般是要有一个视觉中心的，就是观众注意力集中的地方。如果注意力中心长时间地固定，观众就会感到乏味，所以要合理的转换视觉中心。需要注意的是：视觉中心的转换要给观众以提示，而不应该给观众带来困惑。例如，两个演员在对话，第三个演员的注视方向，引导着观众注意力的转变。在处理舞会等人物众多的场景时，同一个场景中有若干组演员，每个组要有自己的两个中心，以便视觉中心的转换，然后在这些组里面再有一个占主导地位的组，是视觉中心组。

对话场景的剪辑

对于没有镜头移动的静态对话场景，一般由一个主镜头(master shot)来覆盖整个场景。

所谓主镜头，是指一个静止的镜头，将对话的双方和周围环境都包含在内。对话者在这个镜头前交谈。但是使用一个镜头是远远不够的，因为过于呆板，没有足够的变化。除非这个对话场景或者对话本身就充满了戏剧化效果。有的时候也可以使用两个主镜头进行平行剪辑。但无论如何，光使用主镜头是不够的，因为观众总是希望能够近距离地观察演员，从他们的表情和小动作来更深刻理解体会人物对话时的思想感情。所以，主镜头必须要以近镜镜头和中景镜头等来辅助之。

下面就介绍一些在对话剪辑中要注意的问题。

●镜头拉近和拉远交替(Approaching and receding)

比较长的对话场景应该有一些明确的兴趣点，在这些兴趣点上需要给观众更细致的刻画。不能平铺直叙地使用同样的镜头。简单的例子就是中景镜头和近景镜头的交替使用。对话一开始使用中景镜头：在说到关键处时，使用近景镜头或者特写镜头来强调重点，把观众的兴趣引导到重点上；然后再回到中景镜头，给观众一个机会放松一下。值得注意的是：交替使用中景和近景镜头的时候，应当避免切换得太生硬。一般来说可以让演员同时做适当的动作，来隐性地实现镜头的切换和对观众的引导。

●如何开始一个对话插景(Starting a sequence)

在一般的对话场景中，一般不会让演员先完全坐好，然后开始对着念台词。这样显得比较突兀。要更自然地安排对话的发生。在对话的开始和结束应该让演员有一些动作，或者使用声音先人人物后人的方式

这里介绍几种常用的方法：

●两个演员从边缘沿相反方向走进屏幕，停住然后开始对话。

●在屏幕里已经有一个演员，另外一个走进屏幕然后在他旁边停下，对话。

●使用移动镜头跟着一个演员，直到他停下，和另一个演员开始对话。

●使用移动镜头跟着第三个演员，他在两个主要演员面前停下，给他们什么东西或者其他动作，然后走开，这个时候镜头停在两个主要演员上，他们开始对话。

●开始镜头中没有演员，只有声音。然后镜头逐渐移动，显示出演员。

这些只是一些常用的基本方法。对于一个对话场景的结束，也可以使用上面的方法，不过要倒过来使用。

●重新建立对话插景(Re-establishing shots)

在利用近景和特写镜头突出表现对话双方的细微动作和表情之后，我们需要反过来提醒观众一下对话所发生的地点和周围环境。这就是所谓的重新建立对话场景。如果一段对话很长，则至少要在对话进行到一半的时候重新建立对话场景。重新建立对话场景目的如下：

●当使用近景镜头或者特写镜头的时候，观众可能会因为放太多注意力在演员身上，而忽视了对话所在的周围环境，重新建立场景起到了提醒观众注意演员和地点之间关系的作用。

●重新建立场景是叙事中的一个停歇，在视觉上也给观众一个稍微放松的机会。

●在对话场景结束的时候使用，给了演员足够的空间来进行下面的行动。

●如果因为对某个演员的特写镜头，其他一些演员暂时地被放在屏幕之外，重新建立场景提醒了观众那些演员的存在。如果这些演员在一半的时候要走掉，重新建立场景则可以展示他们的离开。

●聆听者的反应(Silent reactions)

聆听者的反应往往要比说话者的表情更有表现力。两个演员 A 和 B 的对话，镜头中 A 开始念台词，几句之后镜头给 B，A 的说话持续着，当 A 结束 B 开始念台词，B 念几句之后，镜头转给 A。这样交替进行。值得注意的是，确定每次切换镜头的时候，要考虑到台词上下文的内容，还有周围环境。有必要的话，可以专门加入一些动作或停顿。

如果演讲者面对的是一组人，听众的反应就更为重要。如果镜头始终停在讲演者身上，是极其枯燥单调，而且也不知道别人对他说话的反应。需要不时地插入听众的镜头。

当然前面所介绍的原则并不是铁板一块，有时候也需要变通。当演讲者有很重的表演戏的时候，就没有必要把镜头从他身上移开了。比如美国影片《巴顿将军》的一开始，巴顿走上讲台，开始做煽动性的演讲。

这时候镜头中只有巴顿一个人和作为背景的巨大的美国国旗。镜头里没有听众——他的士兵们。这么做的原因是由于这段要集中体现巴顿的性格，主角的表演本身就很精彩了，没有必要再分散观众的注意力。

上面介绍的比较杂，不过都是真正经过实践考验的剪辑手法。读者可以找来一些片子，有意识地去对照分析其中的例子，就能够获得更深刻的理解。

较难剪辑的场景

所谓较难剪辑的场景，是指那些完全缺乏动作的场景和动作过多过于复杂的场景。它们是两个极端，在剪辑的时候很难配合好。比如说由一幅静止的画面切换到另一幅完全静止的画面时很容易造成跳跃感。下面介绍一些经典的办法供参考。

●在镜头和静止物体之间的运动(Movement between camera and static object)

举个例子(参见图 15-27)，在一个小饭店里，一个演员一个人坐在屏幕中心的桌子边。使用全景镜头来拍摄整体的场景，介绍环境。然后使用中景镜头，让观众看清楚人物的表情从而了解她的心情如何。演员没有动作，因为剧情没有个这需要。那么，如果你直接把全景镜头切换到中景镜头，因为画面中都是静止的物体，直接的剪辑就会产生视觉跳跃感，让观众感到不舒适。那么如何处理这个从全景到中景的剪辑呢？方法就是在镜头和静止物体之间加入运动的物体。

图 15-27 在镜头和静止的物体之间加入运动物体

在坐着的演员之前加入一个横穿屏幕走过的人物，可是是和剧情完全无关的人物，比如一个侍者。在远景镜头中，侍者从左边走向右边，当他遮挡住坐着的主要人物的时候，镜头切换到中景镜头，侍者继续行走，直到出了屏幕，这时候观众就可以清楚地看到坐着的主要人物了。注意的是，要在运动的人物或物体要基本全部遮挡住静止人物或物体的时候才可以剪辑。

这是一个很常见的解决方法，用来遮挡的人物或物体也可以是原来就在屏幕里的人物，只要在运动中遮挡住静止人物的时候，就可以剪辑。变化很多。

●在静止演员身后的运动(Motion beyond the static players)

这是上面例子的一个变式。在前方的人物是静止的，他身后有一个演员走过。先用全景镜头拍摄两个演员，当静止人物完全遮挡住运动的演员的时候，镜头切换到中景镜头来显示静止人物的细部。需要注意两点：一个是静止人物要能够基本遮挡住运动的人物，一个是切换的时候两个演员都要在屏幕的中心位置。

●中景静止镜头到全景运动镜头的切换

前面的例子讲的是由全景到中景的切换。如果要从中景切换到全景，则有时会遇到以下情况：先是一个对着静止演员的静止中景镜头，然后切换到对他的全景镜头，另外一个运动的演员从他和镜头之中走过。在第一个镜头中完全没有运动，而在第二个镜头一开始就有新的运动引入，这样会显得有些突兀，没有给观众足够视觉的适应。为了解决这个问题，我们可以在第一个镜头中加入背景运动，比如说远方的人或者物体在移动。切换到第二个镜头后，背景运动还将继续。这样第二个镜头中前景运动的引入就不会很突兀。不用担心背景运动会干扰前景运动，因为离镜头越近的运动在观众眼中是占主导地位的。

●在第一个镜头里隐藏向纵深移动的物体

在屏幕中的物体如果保持在屏幕中心位置做纵深运动，很少可以直接将其切换到近一些的同样角度的镜头去的。因为移动物体在屏幕中心纵深运动，它的体积逐渐变小，直接剪辑之后，体积又突然变大然后又慢慢变小，带来的视觉跳跃会很不舒服。这时需要做一些处理，可以使用前景人物或物体在第一个镜头中遮挡住纵深移动的人物或物体。当前景人物完全覆盖住运动物体的时候，就可以切换到对运动物体的近镜头了。这个手法也经常在群众场面中使用。

●利用场景中的物体

如果两个镜头之间直接切换比较突兀，给观众带来不舒适的感觉，则我们可以利用场景中物体的各种效果来做掩饰。可以使用很多自然效果，比如溅起的水花或沙子、弥漫的灰尘和烟雾等等，用他们来巧妙而自然地遮挡屏幕，然后切换镜头。

上面主要从镜头和剪辑两个方面介绍了一些电影中常用的方法，游戏设计者是如何在游戏中使用这些比较成熟的电影方法的呢？下面会根据一些例子来解释。

游戏实时画面中镜头的使用

下面主要就游戏实时画面中镜头的应用来做一个简单的介绍。之所以着重强调实时画面中镜头的应用，是因为对游戏片头动画和过场动画来说，基本上可以把它们看成小电影，可以直接照搬电影镜头技术。而游戏的实时画面有其独特的要求，并不能直接照搬电影镜头技术，有很多需要注意的地方。

二维时代的电影镜头的应用

二维游戏时代，镜头的配置基本就是水平、俯视、斜视几种方式。而镜头移动只能是平移镜头，没有可能实现真正的摇镜头和调焦镜头。不过在 16 位游戏机上，可以巧妙地利用硬件的 zoom in / out 功能做出伪三维效果，实现调焦镜头；还可以利用多层背景做出一定的

摇镜头的效果。当然这样做出的并不真正意义上的调焦镜头和摇镜头。

水平移动的跟踪镜头在动作射击游戏中最常用。移动方向可以水平或者垂直或者按一定角度，主角总是位于屏幕的中心。比如 Capcom 的 ACT《三国志 2》和 Konami 著名的《沙罗曼蛇》。大多数游戏的人物移动都是遵循一个法则：就是如果不是剧情的特殊需要，人物的移动一般都要从左到右、从下到上。

16 位游戏机的硬件旋转放缩功能使得简单的调焦镜头成为可能，伪三维游戏应运而生，最著名的例子是赛车游戏。SFC 上的《F-Zero》就是使用了这种假的调焦镜头功能实现了很好的效果。

而利用多重卷轴实现摇镜头的效果，在二维动画片中也是经常使用的。早期 Disney 动画电影中就使用多重背景实现三维感的镜头移动。但是，因为游戏机当时硬件的限制不可以使用过多过大的背景，所以这种技术在游戏中应用得很少。而到了三维时代，也就很少需要使用这种技术了。

三维技术提高了游戏中使用镜头的灵活性

到了三维时代，游戏中镜头的使用就变得更为灵活了，从而真正向电影镜头技术靠近。但硬件上的限制还是有，所以产生了两种折衷的解决办法。第一种是使用渲染(render)好的图像做背景而集中硬件实力处理三维人物，比如 FF7。这种方法无法处理复杂的镜头移动。复杂的镜头移动只好交给穿插的过场动画。另一种方式就是牺牲精度做真正的全三维空间，这种方式镜头的设置更自由而且可以更生动地处理镜头移动，比如《Metal Gear Solid》。随着硬件技术的进一步发展，第二种方式将会被更广泛地采用，比如 PS2 上面的 FFX 和《Metal Gear Solid2》。

在使用渲染好的图像做背景的游戏中，镜头是固定的。所以要注意的问题就是镜头如何设置和镜头间的切换。不同游戏类型对镜头设置的要求不同。RPG 和 SLG 等游戏，经常需要展示比较宽广的场景，使用俯视的远景镜头比较多。在要表现特殊剧情的时候，一般会用过场动画来表现。AVG 游戏的场景主要是封闭空间，常常需要展示狭小空间的压迫感，所以使用的大多数为中景、近景甚至特写镜头。恐怖 AVG 经常会使用一些经典恐怖电影的镜头设置方式来制造紧张感和悬念。

游戏中的镜头切换一般都使用如下方式：人物可以在画面中做纵深移动，设置几个地图切换点，当人物进入切换点范围就切换到另外一场景。这样做频繁的切换是因为内存的限制，但也是为了在不同场景中使用合适的镜头。但是这就导致了一个问题，镜头设置和镜头切换的矛盾。镜头设置是独立的静止的，能覆盖需要的场景，看上去有表现力就可以。而镜头切换则是相关联的，不光是要联系多个场景，还要考虑到玩家的视觉感受，就像电影中考虑观

众一样。在 Square 的 FF7 中，就存在着玩家有时候从一个场景进入另一个场景的时候找不到主角在哪里的情况，所以迫不得已用图标表示人物所在的地方。不光是因为人物太小和分辨率不够高，因为人物没有出现在玩家下意识希望的区域里。这就是为了照顾特定场景的镜头设置，而在切换的时候不得不做出些牺牲，没有遵循电影剪辑的法则，比如运动和位置的匹配。好在 FF7 中明显不适的地方不是很多。而在 AVG 游戏中，它的镜头设置更为接近电影，在切换的时候也不像 RPG 那么有多切换点等等，所以切换都比较容易遵守电影剪辑的法则。

而对于真正全三维的游戏来说，镜头设置和镜头切换在技术上都不受限制，问题在于如何安排，自由有时候不是优点。在刚刚使用三维技术的时候，很多游戏厂商都不知道如何使用镜头，为了显示华丽的效果使用大量移动轨迹复杂的运动镜头，再加上本身物体运动，结果让玩家看了头晕目眩找不到北。在使用渲染图做背景的游戏中，镜头是固定的，是事先精心设计好的。而在全三维游戏中，镜头经常要根据人物与人物之间、人物与背景之间的关系进行调整，而这些又经常是变化的。所以镜头设置只能按一定的模糊的规则，而不是按照预先定好的精确的点位置。这里就涉及到如何利用总结出来的电影镜头理论定制一套规则，来实现非人工但专业化的镜头设置的问题。这一点目前来说还没有哪个游戏可以说自己做得很完美，比如在玩《古墓丽影》类的时候，你经常会遇到莫名其妙地突然镜头移动或旋转，使你无法马上掌握自己的方向，甚至看不到自己的角色了。涉及到多人物对话场景的时候，如果不是实现安排好镜头设置，而是按照规则来设置镜头，难度更大。　Konami 的经典游戏《Metal Gear Solid》在如何使用镜头技术上有独到之处。特举例子说明。《Metal Gear Solid》里面基本的技巧动作都和镜头有着关联。它很好的利用镜头和动作配合展示游戏气氛。平时以俯视视角为主，在特殊情况下就移动镜头转换成最适合的视角。当你趴着身子藏在桌子底下的时候，可以切换主观视角来观察；当你往通风管道里爬的时候，镜头会自动切换为主观视角，让你体会在封闭空间内的压抑感；当你站着近靠墙壁的时候，镜头自动切换成间谍片中常见的平视视角，紧张气氛骤升。

游戏实时画面中剪辑手法的应用

剪辑手法在游戏中的几种用途

在游戏中使用剪辑的目的和电影中不太一样。电影中主要是衔接两段镜头序列，而游戏中的应用要广泛的多。除了游戏实时画面本身的镜头切换，还有游戏画面和菜单的切换、移动画面和战斗画面的切换等。这些都是游戏所特有的。

在游戏实时画面中剪辑手法的使用比较简单，场景镜头之间主要就是直接剪辑或者使用移动镜头连接在固定镜头之间。在二维游戏中，绝大多数剪辑都是直接剪辑。人物移动到屏幕某个边缘，然后移出，屏幕直接切换，人物从对称边缘移入。而三维游戏中，有时候会使用移动镜头来衔接两个固定镜头。

以 CD-ROM 为媒体的游戏出现后，视觉标点法成为很重要的工具，用来掩饰极慢的读盘速度。最常见的也是最苍白的就是 Loading 画面或者完全黑屏。有的游戏会利用增长的淡入淡出，或者穿插简单的过场画面来掩盖长时间的读盘时间。

RPC 游戏中移动画面和战斗画面的切换广泛使用了各种特效，如放缩、擦抹、扭曲、融合、彩虹变换、变焦距等等。

总的来说，剪辑在游戏中的应用不是很多。一个重要原因，电影是需要在短时间展示长时间的故事，所以要拼接在不同时间、不同地域摄制的镜头序列。而对游戏来说，基本是跟随主人公经历整个故事，需要用剪辑来实现空间时间的跨越的情况不是很多。

在游戏叙事中使用平行剪辑法

在电影中平行剪辑法是经常使用的剪辑手法。简单来说就是由两条有一定关联的主线 A 和 B，发生在同一地点或者不同地点，相同时间或者不同时间，两条主线的镜头序列按照 A1、B1、A2、B2、A3、B3……这样的方式平行剪辑。

对于如何在游戏中有效地使用平行剪辑法来展示剧情，还需要摸索。因为游戏是玩的，不光是看的，与电影很多不同。电影可以在 10 分钟内平行剪辑两个场景切换多次，而游戏就不可以，让玩家开始熟悉第二个场景的时候再切换到第一个场景的时候，可能连记都记不起来了，这也就失去了平行剪辑的效果。而且作为游戏，不可以在接近高潮的时候突然剪辑到其他部分。在电影里，可以在你马上就可以看到凶手面容的时候剪辑到其他镜头，而在游戏中就很难这样做——马上就要面对 Boss 了，画面切换到其他场景开始其他情节，然后没一会儿再切换回 Boss，玩家非得气死不可。

在 FF 系列中可以看到一些平行剪辑的尝试，主要是处理大块情节。在 FF 系列中，经常会出现众多角色分组的情况，最明显的就是 FF6。分组队员在平行时间不同地域进行需要玩家来控制的冒险，在这里 Square 就大胆地使用平行剪辑的办法在各组之间切换控制(也就是镜头)，因为游戏本身性质的限制，这种切换不是非常频繁，两三次之后所有组就会汇集在一起。FF6 中经典的歌剧场景就是使用了平行剪辑的方法，在盗贼的营救行动和歌剧演出两条主线中进行切换，把那种潜伏在动听歌剧下的危机生动地表现出来，之后两条主线合并在一起，众人一起对付 Boss 水怪。

本章简单介绍了电影镜头的基本概念和剪辑方法，和它们与游戏的关系。随着技术的进步，游戏的表现方式将越来越自由，很多电影理论都会被借鉴到游戏制作中。关于电影镜头和剪辑手法的使用，包括如何自动设置镜头、如何在实际游戏中有效地使用平行剪辑法等等，都将成为游戏制作的重要部分。

编程篇

第十六章　游戏编程基础

在介绍具体的三维图形编程和 AI 技术之前，我们有必要先介绍一些基本的游戏编程技术和思想。这些都是前人根据游戏编程的特殊需要所总结的经验之谈。了解了这些思想和方法，才能使我们完成从一个程序员到一个游戏程序员的转变。

游戏程序员

很多人都认为，游戏程序员的工作是最具挑战性的，因为他们必须把 PC 或者游戏机的硬软件能力发挥到极限!游戏程序员们必须时时面对挑战，力求百尺竿头，更上一层。因为谁能够将硬软件能力发挥得更彻底，产生更高分辨率更美丽的画面，更流畅更自然的动画，更富有挑战性的 AI，和更稳定的网络对战功能，谁就更有可能在激烈的市场竞争中存活并取胜。

由于游戏的特殊性，游戏的设计总是在开发的过程中被不断地修正和改变，因此游戏程序员们时时做无用功，把自己已经写好的东西推倒重来。几乎所有的游戏程序员都有几个月心血白费的经历。这对游戏程序员的心理素质也是个考验。

游戏编程的工作量巨大，游戏程序员们的工作负担较重。以美国来说，一般的白领一年工作 2000 小时，而游戏程序员一般要工作 2500 小时，甚至 3000 小时。游戏业的传奇人物，《QUAKE》的主程序员 John Carmak 一年工作超过 4000 小时!很多人都认为游戏程序员们是天生的工作狂，习惯于与黑夜和咖啡为伴。

因此可以这么说：游戏程序员们是一群对技术最精益求精、心理素质最好、工作最勤奋的人!

但另一方面，游戏程序员们也普遍地存在一些缺点和问题，比如他们是整个软件业里最臭名昭著地不愿意给代码加注释的，不愿意遵守代码标准(一般游戏公司也没有所谓的代码标准)。总体来说，游戏程序员们更有西部拓荒牛仔的气质，更有艺术家桀傲不驯的个性，而不愿意循规蹈矩。在目前来看，随着游戏项目越做越大，游戏程序员们也到了必须作出某种改变的时候了，这样才能适应更规范化过程化的游戏开发的需要。也就是说，游戏程序员们必须摆脱西部拓荒者的牛仔模式，走一条更正规化的道路了。

游戏程序员如此辛苦地工作，承受巨大的压力，盖因他们处在工作链的最后紧要关头，是他们把设计师的抽象蓝图具体实现，最后交出成形产品。其责任巨大，自然带来巨大的

权力。他们的权力就体现在他们在最后关头可以以保证进度和技术难度为借口删改设计。此等删改使其和游戏设计师关系紧张。程序员们倾向于认为不懂技术之人对其工作指手画脚为典型的外行领导内行，而同样游戏设计师视程序员为无思想之“机器”，而今居然要妄图修改设计，实是可恶。可以说，对游戏项目危害最大的，莫过于游戏设计师和程序员之间的隔阂。如果处理不好，将严重影响组员士气。

解决程序员和游戏设计师之间的矛盾，应从两方面下手。一是制作过程的制度化和规范化，包括实现“改动控制”(在软件工程一章中会谈到)。二是双方都应该对对方有所了解。程序员们必须明白：任何工作都必然是由抽象到具体，都是由只懂得大面或者抽象概念的“外行”去管理懂得具体技术细节的“内行”。因此，他们应该尽可能地去帮助“外行”调整技术细节并提出合理化的建设性的意见，而不是以“外行”不懂技术细节为由采取不合作态度。而游戏设计师也必须认识到，游戏程序员并非无大脑之“机器”。他们之所以选择辛苦的游戏编程事业就是因为自己要为游戏尽一份力，对他们的意见和想法，要尊重并认真对待，不能因为认为他们只是技术人员就把其创造性想法丢到一边不理。

游戏程序总体结构

在动手开始写第一行代码之前，我们必须先为我们的游戏程序选择一个好的总体结构，这对程序开发来说至关重要。有了好的总体结构，则程序条理清楚，各模块之间关系明确，易于扩展，易于纠错。如果总体结构有问题，则各模块之间关系复杂，信息传输混乱，编译纠错困难，越往后问题越多。可以说，软件质量的高低，很大一部分决定于一开始确定程序总体结构的时候。而中国游戏业和美国游戏业的差距，也主要在于美国有一批有经验的、水平很高的软件结构师(software architect)，而我们相对来说缺乏这方面的人才。

提到程序总体结构，就得谈谈目前也是一个很热的话题——软件模式(design patterns)。软件模式是近几年提出来的概念。其思路是在大量总结近几十年各个应用领域各种类型的程序的开发经验基础上，总结归纳出一系列模式(pattern)，也就是软件的普适结构。就好像是建筑蓝图一样。这样当我们要开发一个新的系统时，我们可以把这本厚厚的软件蓝图书拿来，看看在这个应用领域，前人曾经使用了何种结构，各自的优缺点是什么，其中哪些比较适合我们这个项目的实际需要。然后根据书中的指导来构建自己的程序。

著名的模式，例如 MVC，即 Model-View-Controller，最早在 SmallTalks 中应用，是公认的最适合于一般 GUI 程序的软件结构。微软的 MFC 的整个设计思路就来源于 MVC 模式。

那么游戏是否也有其统一的普适结构，或者说游戏程序是也有其特有的模式呢?答案是有的。所有的游戏，基本上都可以看作一个循环结构，每次循环都处理玩家输入，然后根

据游戏内部逻辑，改变游戏状态，最后更新画面。图 16-1 就是游戏的总体结构示意图。

下面来具体地介绍图中的各部分：

程序初始化：在这一步，需要分配内存，获取资源，读取文件，并建立速查表等。

进入游戏主循环：进入游戏的主循环后，程序就一直循环往复，直到玩家发出退出游戏的指令。

接受玩家输入：玩家的输入，通过键盘、鼠标、游戏手柄等，被处理并存入系统消息队列(对 WINDOWS 而言)。

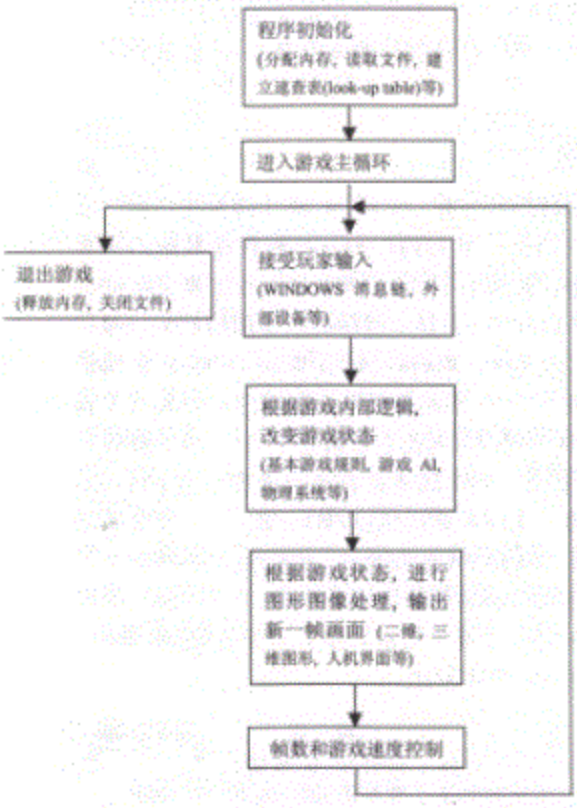


图 16-1 游戏程序总体结构图

游戏内部逻辑处理：这部分是游戏的重中之重，游戏的规则就是在这里体现的。在接受了玩家的输入后，根据游戏设计师事先定义好的游戏规则，再加上高级的游戏 AI 和物理系统的配合，来决定游戏的新的状态。所谓新的状态就是玩家的输入所产生的后果。

图像更新：游戏的状态改变需要在屏幕上显示出来。因此，需要根据游戏状态，使用二维三维图形技术，去计算产生新的一帧画面。

帧数和速度控制：新生成的帧，并不能马上放到视缓存，而需要放到另一个缓存区里(这就是所谓的双缓存)。然后根据实际游戏运行速度决定何时将其拷贝到视缓存，显示到屏幕上。其目的是使得游戏速度能够比较稳定，不受各种因素的影响。比如说在 RTS 类型的游戏中，我们不希望看到游戏画面的更新速度随着屏幕上的小兵数的改变而时快时慢。

由上面的介绍我们可以看出，游戏程序是一个循环结构。一般我们可以把游戏编程任务分成三大块：图形图像部分、AI 部分、基本逻辑和 UI。而其中最重要的，也是最复杂的，编程难度也最高的，则是图形图像和 AI 这两部分。关于这两个部分，我们会分别有专门的章节阐述。而游戏的基本逻辑、规则机制和 UI 部分，在编程方面难度相对低些，但和游戏的设计息息相关，已经在本书的设计部分着重提及，所以在编程部分就略去了。

编程语言和编译环境

游戏编程中所使用的编程语言，有 C / C++、汇编、JAVA 等几种。其中，C / C++语言是一种中级语言。其对内存的直接操作是其威力所在(也是其最大的毛病——指针错误的根源)，可以最大程度地发挥硬件潜力，使其成为游戏程序员的首选。而目前程序员所使用的 C / C++编译器，主要是微软的 Visual C++，Code Warrior，还有 Watcom C++。汇编语言由于其近于机器语言，难于掌握，不利于书写大型程序。但其效率很高。汇编语言比 C 要快 2-10 倍左右，因此在目前还未消亡，有些程序员还要在程序的某些部分用到嵌入式汇编来提高运行速度。而 JAVA 由于是一种非编译性语言，需要 JAVA 虚拟机(JVM)才能运行，速度比较慢，一般不太适合游戏编程。但 JAVA 的安全性、分布性、和多线程等特性使得其成为网上编程的首选。随着网络游戏的兴起，有些公司也开始使用 JAVA 编一些网上游戏了，另外手机游戏中有不少是用 JAVA 的。表 16-1 列出了各种编程语言的优劣对比。

	优点	缺点	适用机种
C/C++	可以直接对内存地址进行操作，快适合系统软件或对运行性能要求高的软件的编程	指针容易出错，并不是100%地面向对象编程语言	PC，游戏机，掌机
JAVA	可移植性好，安全性高，是真正的面向对象编程语言，无指针，有内存垃圾自动清理功能，支持多线程，适合网上编程	速度慢，需要 JVM 才能运行	手机游戏，网上游戏
汇编语言	速度最快	难学难记，很难用来开发大型软件	早期游戏机和掌机

表 16-1 不同编程语言对比

需要指出，在不同的硬件平台上，各种编程语言的应用普及程度不同。比如说 PC 上的游戏总是最先采用新技术，因为 PC 平台开放，而且更新升级速度快，这就给 C++，STL(Standard Template Library)和各种高级数据结构的应用提供了更广阔的空间。而游戏机游戏因为内存和处理能力的限制，加上开发环境比较封闭，所以在新技术的应用上比 PC 游戏总要慢半拍。比如 C++在游戏机游戏编程中应用并不普遍，PS 和 PS2 都还是基本上用 C 语言。而掌机 GBC 还是用的汇编语言，到 GBA 才开始用 C 语言。而由于内存的限制，游戏机游戏一般也没有采用 PC 游戏中使用的很复杂的 AI 技术。所有这些导致了 PC 游戏编程和游戏机游戏编程的不同，也就是说一个很好的 PC 游戏程序员，当他面对游戏机捉襟见肘的内存时可能会束手无策；而一个很有经验的游戏机游戏程序员，也许并没有掌握 PC 游戏编程方面最新的技术。

当然，这种现象现在也有所变化。随着游戏机硬件水平的提高，特别是 PS2 和 XBOX 等新一代游戏机的问世。PC 游戏编程和游戏机游戏编程之间也在逐渐融合。其显著特征就是专业垮平台游戏开发引擎的问世，这些引擎一般都是同时支持 PC 和游戏机。大家都使用同样的工具，调用相同的函数。这样把 PC 游戏的编程和游戏机游戏的编程拉近了许多。

算法评估

前面我们说过，游戏程序员们在硬件软件的诸多限制下，要力求把它们的潜力发挥出来。因此他们对于一段程序运行所需的时间和所占用的内存的要求是十分苛刻的。而要完成同一个功能，写程序可以有很多种方法，也就是说有不同的算法可供选择。程序员们当然希望选择速度最快，占用内存最少的算法。而如何知道哪种算法快，哪种算法慢呢？这就引入了算法评估问题。

程序的运行速度和所占用的内存

为了说明完成同一功能的程序可以有完全不同的运行速度，我们使用一个最经典的例子，就是菲波那切数列(Fibonacci numbers)。其基本数学表达如下：

$$F_n=\begin{cases}0 & n=0\\ 1 & n=1\\ F_{n-1}+F_{n-2} & n\geq 2\end{cases}$$

使用上面的公式，我们可以得出菲波那切数列为 0,1,1,2,3,4,8,13,21,34……

现在我们要写一段程序来产生菲波那切数列中任意一个数。我们需要的是根据一个数在数列中的位置，得出这个数的数值。从菲波那切数列的数学表达，我们可以很自然地得到以下的程序：

程序 1：

```
int fib(int n)
{
if (n<=2) return 1；
else return fib(n-1)+fib(n—2);
}
```

似乎问题解决了——我们提出了一个抽象的问题，然后用具体的程序实现了。但是游戏程序员们是不会就此满足的，他们必须思考：这个算法究竟是快还是慢呢?是否是最优的呢?还有没有其他更好的方法呢？这才是编程真正的挑战所在!一个优秀的程序员是不会满足于仅仅找到问题的解答的，他必须力求找到问题的最优解答。一般来说，要考虑 3~4 种不同的方法，然后择其中最好的一种去实现。

仔细考察我们的程序，我们不难发现，这是一个递归调用的程序。它的致命弱点是每次计算 F(n)的时候，都要重复计算 F(n—1)和 F(n-2)。如果以 n=5 为例，则我们得到以下的树结构。

从这个树我们可以看出一个问题。我们重复计算了很多结点。比如说 F(3)在两处出现，

第二次出现时我们又重头到尾算了一遍(递归)。当 n 很大时，这种重复计算使得运算时间慢得超乎我们的想象。

明确了上面算法的问题所在，自然就要改进。一个简单的想法就是把前面算过的结点存储下来，这样第二次遇到就不用重复计算了。并且先解决小问题，从小问题到大问题，而不是相反。考察下面的算法：

程序 2：

```
int fib(int n)
{
int f[n+1];
f[1]=f[2]=1;
for (int i=3;i<=n;i++)
    f[i]=f[i-1]+f[i-2];
return f[n]
}
```

显然这个程序大大简化了递归循环的次数。树的每个结点只需要计算一次就够了。但这个算法是否还有改进的余地呢？从另外一个角度思考，在这个算法里面，我们为了计算一个数值就使用了一个大数组。能否把程序改进一下，节省内存呢？考察以下程序：

程序 3：

```
int fib(int n)
{
int a=l，b=1；
for(int i=3； i<=n； i++)
    {
    int c=a+b;
    a=b;
    b=c;
}
return a;
}
```

我们在这里用了两个中间变量代替了数组，从而节省了内存。

从上面的简单的例子，我们可以领会到：游戏程序员的任务不仅仅是实现一定的功能，更重要的是他要寻求最优化的方法去实现这个功能。区分一个优秀游戏程序员和一个平庸程序员的，就是程序的效率——运行速度多快，占用内存多少。

BIG-O

为了提高程序的效率，提高程序的速度，我们首先要有方法来衡量一个程序的运行速度，这样我们才能在不同的算法之间进行比较。而为了衡量程序的运行速度，我们要定义一个概念——BIG-O。如果读者们还记得的话，我们在游戏性一章中所提出的重要思想——要改进一个东西的某项属性，首先要明确定义一个概念去反映这种属性，然后设计出一些评估方法去利用这个概念去衡量这种属性，然后才有可能持续地改进这项属性。如图 16-2。

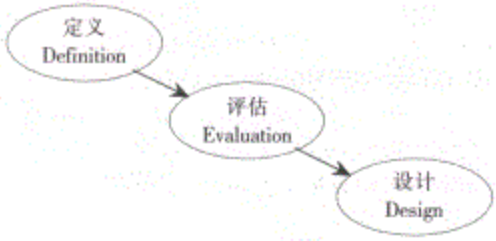


图 16-2 定义—评估—设计

BIG-O 的定义，也是为了算法的评估。而算法的评估，是为了更好地选择和设计算法。

BIG-O 定义：

假设 f(n)和 g(n)是两个函数，n 总为正，如果 n 很大时，存在一个大于 0 的数值 c，使得 f(n)≤cg(n)成立，则我们说 f(n)=O(s(n))，读作 f(n)是 g(n)的 BIG-O(大 O)。

按照上面的定义，n2+4n=O(n2)。因为当 c=2，并且 n 大于 4 时，n2+4n 总要小于 2n2，如果我们画出这两个函数的曲线，我们发现，当 n 大于 4 时，n2+4n 的曲线总在 2n2 曲线之下，也就是说，2n2 的曲线是 n2+4n 曲线的上限。可见 BIG-O 所处理的是上限，并且是 n 很大时候的情况，这就意味着我们实际上分析的是算法在最差(最慢)情况下的运行情况，也就是说，我们将知道这个算法最差将差到什么程度。

正如所有的数学概念一样，BIG-O 也有自己的一些法则。比如说以下两条：

f(n)+g(n)=O(max(f(n)，g(n))

kxf(n)=O(f(n))

算法评估

现在我们定义了 BIG-O，这个定义似乎很数学化，如何使用这个概念呢?使用 BIG-O 这一概念，我们可以从分析算法的结构人手，对程序的速度作出定量评估。

考察以下程序：

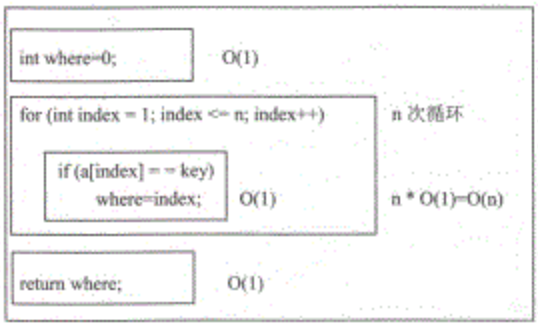
```
int LinearSearch(int n，int key，NumberArray a)
{
    int where=O;
    for (inl index=1；index<=n；index++)
        if (a[index]==key)
```

```
        where=index;
    return where;
}
```

这段程序搜索一个 n 元数组，返回其中数值等于 key 的元素的位置。我们可以把这段程序所需的运行时间看成一个函数 T(n)，其参数为 n，即数组的元数。显然 n 越大，程序所需运行时间越长。

为了简化起见，我们假定上面程序中每一单行语句运行时间相同，用 O(1)表示。O{1)意味着无论 n 多大，这行语句的运行时间都是固定的。

我们主要考虑这个算法在最差情况下的运行速度。所谓最差情况，是指运行时间最长的情况(有点循环论证的意思)。对这段程序，最好的情况是每次数值为 key 的元素都位于数组第一位，这样根本无需循环了。而最差的情况，则是数值为 key 的元素位于数组最后一位，或者根本没有数值为 key 的元素，在这两种情况下，需要 n 次循环。



经过简化，并且只考虑最差的情况，分析这段程序，我们得出以下的结果。

结论是 T(n)=O(1)+O(n)+O(1)=O(n)。也就是说，这段程序的运行时间是 BIG-O(n)。那么我们得出这个结论有什么用呢?这是一个很抽象的结论，单独看来好像没有什么用处。但其作用是和其他算法比较，比如另外一个算法的运行时间是 O(n2)。比较 O(n2)和 O(n)，显然 O(n)的算法在最差情况下要快。如果不考虑其他条件，我们将倾向于 O(n)的算法。

上面的程序极为简单。遇到复杂的程序，其运行时间的评估则不会这么直观简单。这时则有更复杂的工具，可以用来分析其 T(n)。比如 master theorem 和各种迭代替换方法。具体的内容，请见各种讲述计算机算法的专著。

正如我们前面指出的，这种使用 BIG-O 对算法的运行速度进行评估的方法，着眼的是最差的情况。为什么我们只考虑最差的情况呢?主要原因有以下三点：

第一，最差情况给算法的运行时间设定了上限，知道了这个上限，我们可以舒口气，说：“再坏也不会比这更坏了。”

第二，从概率上说，很多算法经常运行在最坏的情况下。

第三，对很多算法来说，在“一般情况”下所需的运行时间和在最差情况下差不多，

属于一个量级。

这种假定最坏情况，从最坏处着眼，而不是假定最好的情况，一相情愿地从最坏处着眼处理问题的思路，在西方比较常见。很有意思的是美国的政府组织，就是从最坏处设想(假设官员个个有私心，人人想腐败)，设计制度规则，使得政府运行在最差的情况下不至于太差。为了保证最差情况下不至于太差，甚至牺牲其在最理想情况下(官员个个大公无私，兢兢业业)的效率。而前苏联的政府，则从好处着眼，目的是使其在最理想的情况下能够利用所有资源高效运行，但其缺点显而易见——你不知道它在最差情况下的“上限”究竟差到什么程度。到头来，上面两种“算法”在大部分情况下哪种运行得比较好，已经被历史证明。

代码优化

对游戏程序员们来说，经过长期实践，他们已经自己总结出了一套优化代码的经验。这些经验完全是根据游戏编程的特殊需要而量身订做的，并不能放之四海而皆准。有很多甚至和我们从课堂上所学的编程习惯完全相反，比如对全局变量的态度。以下只是简单地罗列了一些比较浅显的经验。

●使用二进制移位代替乘除

研究过汇编语言的人都知道，CPU 进行乘除运算是最慢的。在游戏程序中，能够不进行乘除，就要尽量避免。而使用二进制的移位可以代替部分乘除运算。如下例子：

```
//y 乘以 64
y=(y<<6);      //2?6=64，2 的 6 次方是 64，所以将二进制的 y 左移 6 位
的结果就是 y*64
//y 除以 8
y=(y>>3);      //2?3=8，将 y 右移 3 位，相当于 y 除以 8
```

●适当使用全局变量

我们在课堂上学计算机编程的时候，都知道有了 C++和 JAVA，就应该避免使用全局变量，像 JAVA 更是完全不支持全局变量。因为全局变量容易把程序搞得很乱，难以跟踪纠错。但对游戏来说，使用全局变量可以避免调用函数时的参数传递，可以提高速度。

●使用 32 位变量

尽量使用 32 位变量，因为新的 INTEL 处理器是 32 位优化的。

●尽量避免类型转换

尽量使用相同类型的变量进行运算。比方说整数乘整数，浮点数乘浮点数。如果用整

数乘以或除以浮点数的话，在进行类型转换的时候要花去不少时间。如果必须进行类型转换的话，把它留到最后进行。

●使用 Inline Function

我们知道调用函数是一个非常复杂的过程，要用到栈。当函数比较简单时，我们不希望每次调用它的时候都兴师动众。使用 Inline 伪指令，可以避免不必要的函数调用。

●使用速查表(Look-Up Table)

如果程序中用到一些数学函数，比如 y=f(x)，实时计算太耗费时间。可以预先计算出对应的 x，y 的数值，放到一个大表里。在程序运行时只需到表里查出数值，这样是以存储空间换取运行时间。

其他各种类似的经验有很多，在这里就不一一列出。这些经验是程序员在工作过程中不断积累的，需要细心和留心。有经验的程序员，在游戏业绝对是金子。而金子的价值，既在于雄厚的基础知识，更在于灵活的实用技巧和经验。

第十七章　三维图形编程

自 PS 和土星等 32 位游戏机发售后，三维游戏逐渐取代了二维游戏的地位，成为了市场上主流。三维图形编程成了游戏开发中的重中之重。游戏从二维到三维的转变，带来了游戏性的重大变革，也给原来的二维程序员们带来了巨大的挑战。

三维图形编程的复杂度，比传统的二维图形编程大多了。首先是对程序员的数学水平要求提高了。起码你要懂得线性代数和矩阵运算。其次，构建三维世界所需处理的信息量大增，要处理坐标位置、多边形、光源、表面特性等等诸多不同类型的数据。最后，三维图形的算法一般比较复杂，需要更多的技巧去优化和调制。

我们同时看到，新技术的采用是有一个过程的。谁也没有能力一开始就采用最复杂最先进的技术。在 PS 的第一代游戏中，由于程序员们还没有真正掌握 PS 的硬软件环境，因此他们只用了最简单的平面明暗处理(flat shading)。而当他们逐渐积累了经验后，在 PS 的第二代游戏中，才使用了比较复杂的 Gouraud 明暗处理(Gouraud shading)。如图 17-1 所示。

图 17-1 左圈是 SQUARE 的《TOBAL 1》，使用的是最简单的 flat shading，使用单色涂满整个多边形。由于多边形数较少，这样的单色平涂使得一个个多边形被一览无遗，多边形与多边形之间分界明显，角色显得方头方脑的。右图是《TOBAL 2》的图像。由于使用了更高级一点的 Gouraud shading，每个多边形上有不同明暗的颜色过渡，使得图像更柔和细腻，多边形也不那么明显了

下面就一步步介绍三维图形编程的基本技术。

几何变换

几何变换几乎是后面所有三维图形技术的基础。我们先介绍二维几何变换，然后再介绍三维几何变换。

二维几何变换

几何变换有三种：平移、缩放和旋转。首先介绍平移变换。假设在一个平面上的一点 P(x,y)，我们要把它移动到一个新的地方 P'(x',y')。我们沿轴移动 dx 的距离，沿 Y 轴移动 dy 的距离。则我们有以下的公式：

x'=x+dx　　y'=y+dy

我们也可以用向量来表示：

$$p=\begin{bmatrix}x\\y\end{bmatrix}\ ,\ p'=\begin{bmatrix}x'\\y'\end{bmatrix}\ ,\ T=\begin{bmatrix}dx\\dy\end{bmatrix}$$

$$P'=P+T$$

对于一个物体，我们可以对它上面所有的点都使用上面的公式，其结果是整个物体移动了。当然物体上可能有无数个点，比如一条线段由无数点组成。但其中最主要的是两个端点，它们定义了这个线段的位置和其他特性。我们可以只对它的两个端点使用上面的公式，这样的结果是线段被平移了。

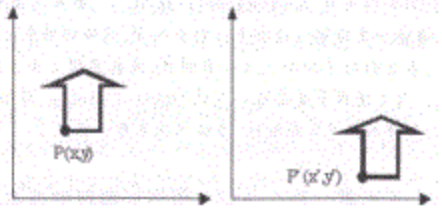


图 17-2 一个二维坐系下的房子被平移了，这是通过对 7 个顶点的平移变换实现的

而缩放变换，是对一个点的 X，Y 坐标分别乘以数值 Sx，Sy。

x'=Sx ● x　　y'=Sy ● y

可以用矩阵表示为：

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}Sx&0\\0&Sy\end{bmatrix}\bullet\begin{bmatrix}x\\y\end{bmatrix}$$

又可以简写为 P'=S・P，其中 S 代表公式中的矩阵。

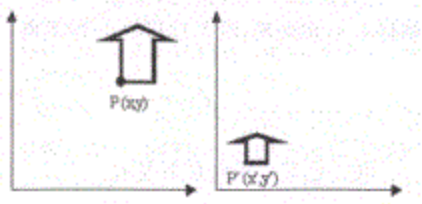


图 17-3 缩放变换

我们也可以使一个点绕着坐标原点旋转。旋转的数学定义如下：

$$x'=x\bullet\cos\theta-y\bullet\sin\theta$$
$$y'=y\bullet\sin\theta+y\bullet\cos\theta$$

以矩阵的形式表达

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

或　$p'=R\bullet P$

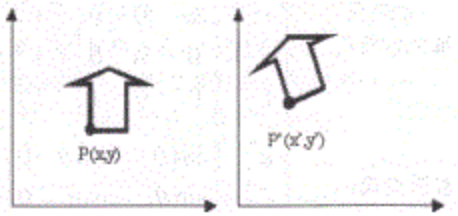


图 17-4 旋转变换。当角度为正时，围绕原点逆时针旋转

如图 17-4 所示。

齐次坐标系

前面介绍了三种二维几何变换，我们注意到如果用矩阵表示，这三种变换分别是：

P'=P+T

P'=S ● P

P'=R ● P

在游戏中，我们同时要用到这三种变换。比如说一架飞机，既要沿直线飞行，也要能够转弯，当它飞远的时候，它的图像还要能够被缩小。既然我们同时要用到三种变换，那么我们是否可以找到比较好的方法，把这三种变换用统一的方式来表达呢?如果能的话，那我们的计算和编程都会变得简单多了。这就引入了齐次坐标系的概念。

使用齐次坐标系可以把三种几何变换用矩阵相乘的形式来统一表达。在齐次坐标系中，我们把在一个平面上的点的二维坐标上再加一维，也就是说，平面上一个点(x,y)现在是用(x,y,W)来表示。如果 W=l 的话，则平面上的点由(x,y,1)来表示。而三种二维几何变换可以改写为统一的矩阵形式了。

$$p=\begin{bmatrix}x\\y\\1\end{bmatrix}\quad p'=\begin{bmatrix}x'\\y'\\1\end{bmatrix}$$

平移变换：
$$\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}1&0&dx\\0&1&dy\\0&0&1\end{bmatrix}\bullet\begin{bmatrix}x\\y\\1\end{bmatrix}$$

缩放变换：
$$\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}Sx&0&0\\0&Sy&0\\0&0&1\end{bmatrix}\bullet\begin{bmatrix}x\\y\\1\end{bmatrix}$$

旋转变换：
$$\begin{bmatrix}x'\\y'\\1\end{bmatrix}=\begin{bmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{bmatrix}\bullet\begin{bmatrix}x\\y\\1\end{bmatrix}$$

这样一来我们就把三种变换都用矩阵的形式来表达了。而这种矩阵表达的真正威力，在于我们只需通过简单的矩阵相乘就可以自由组合各种变换，达到各种效果。比如说旋转变换是对原点逆时针旋转，如果我们希望让一个图形绕它自己的左下角旋转，我们应该如何运算呢?

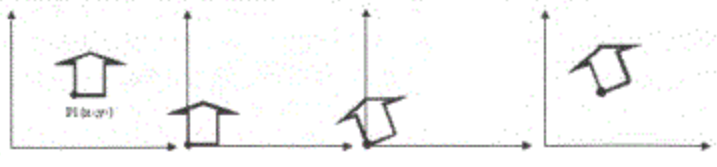


图 17-5 将房子平移到原点，然后旋转，再平移回原来的位置。这样的效果是房子绕自己的左下角旋转了

我们需要做的就是把三种二维几何变换组合起来，达成我们需要的效果。我们可以先把这个图形平移，使得它的左下角和原点重合。然后使用旋转变换，绕原点旋转。最后再使用平移变换把图形移回原来位置。这样这个图形就是绕自己的左下角，而不是绕原点，旋转了一个角度了。

其矩阵相乘如下：

$$\begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

注意到最右边的矩阵代表最早进行的变换，即将图形平移到原点。从右往左，每一个矩阵对应相应的变换。从这个例子我们可以看出，对一个在二维平面上的图形进行无论多么复杂的操作，我们都可以将其表达为一连串矩阵的相乘。这样的话对计算和编程都简化了。

三维几何变换

三维的几何变换，就是在二维变换基础上再加一个坐标，即 Z 坐标。我们同样使用齐次坐标系下的(x，y，z，W)来表达一个点，一般用(x，y，z，1)。我们在本书中一般使用右手坐标系，如图 17-6。

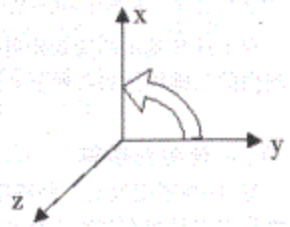


图 17-6 右手坐标系：将右手握拳，四个手指的旋转方向如图中箭头，即由 x 轴到 Y 轴。伸出拇指，则拇指所指方向为 Z 轴

而三种变换矩阵分别如下：

平移变换：
$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

缩放变换：
$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

对于旋转变换来说，则有些麻烦。因为我们现在有三个坐标轴，因此也就有三种旋转的可能，分别是绕 X 轴旋转、绕 Y 轴旋转和绕 Z 轴旋转。这样我们就有三个不同的矩阵。下面仅列出绕 Z 轴旋转的矩阵，其他两个矩阵读者可以自己推出。

围绕 Z 轴的旋转变换：

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

有了这些基本的变换矩阵，我们就可以通过矩阵相乘，实现对三维空间内的三维物体的各种复杂操作了。

窗口—视图变换

窗口-视图变换是一种特殊而又重要的几何变换。实际上所有的游戏中我们起码要有两种坐标系，一种被称为世界坐标系，一种被称为屏幕坐标系。比如说二维 RPG 游戏中，在内存中存在的，是一个大的世界地图，这个地图可以比计算机屏幕的分辨率大许多倍。而在屏幕上显示的，只是这个大地图的一部分。具体如图 17-7 所示。

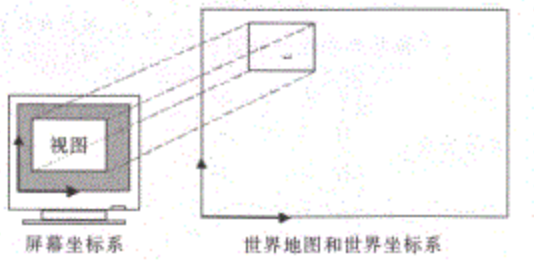


图 17-7　窗口-视图变换示意图(为简化起见 ,坐标原点都取在左下角)

从图中我们可以看出，在屏幕上的一个白色方形区域，叫做视图，用来显示世界地图的一部分(其他灰色区域可以是界面菜单区)。而其对应的世界地图上的一个方形区域，叫做窗口。一般来说，游戏世界中的人物、物品的当前位置数据、相互关系和移动等操作，都是在世界坐标系中进行的，然后才转换到屏幕坐标系中，输出到屏幕。正因为如此，窗口-视图变换才如此重要。其简化的示意图如图 17-8。

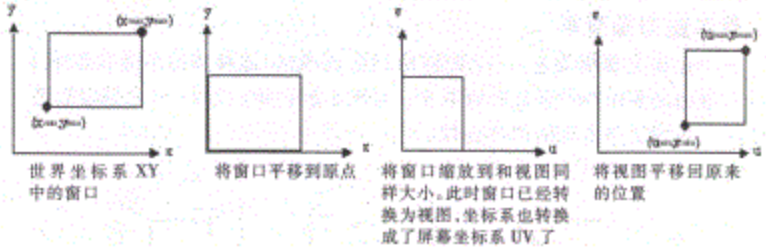


图 17-8 窗口-视图变换示意图

用我们前面学到的矩阵相乘的方法，窗口-图转换可以表达成：

$$\begin{bmatrix} 1 & 0 & u\min \\ 0 & 1 & v\min \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \frac{u\max - u\min}{x\max - x\min} & 0 & 0 \\ 0 & \frac{v\max - v\min}{y\max - y\min} & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x\min \\ 0 & 1 & -y\min \\ 0 & 0 & 1 \end{bmatrix}$$

对三维游戏来说，窗口-视转换更是重要。这时世界坐标系是三维的，而屏幕坐标系还是二维的。如何做转换呢?方法是在三维世界牛取一个平面，先把三维世界中的物体都投影到那个平面上，这个平面就相当于一个二维的世界坐标系了，然后再做二维的窗口-视图转换。这个平面的具体构建方法将在下面的投影一节讲到。

三维物体

为了构建三维游戏世界，我们首先需要某种方法来表达三维物体。也就是说用某种数据结构，或者数学模型，来定义一个物体。目前游戏中普遍使用的，是所谓的表面模型(surface modeling)。即通过定义物体的表面特性而定义一个物体。表面模型不关心物体的内部，一般认为物体内部为空。而表面模型中最常用的，就是多边形模型。

多边形模型

在多边形模型中，一个物体由一系列顶点、边和多边形连接而成。一条边最多由两个多边形所共享。一条边连接两个顶点。一个多边形是由一系列边所形成的闭环连接。

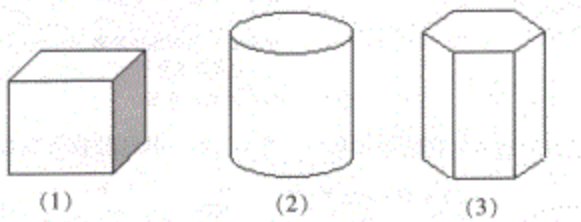


图 17-9 图中(1)一个正方体，可以用多边形模型完美地表达。8 个顶点、12 条边、6 个面(6 个多边形)，它们共同构成一个中空的正方体；而(2)一个圆柱体，用多边形模型表达的话就有困难。当然我们可以用多边形来近似上下两个圆形面，如(3)；当(3)的上下两个多边形的边越多越细，则多边形越接近圆形，而多边形模型(3)也就越接近圆柱体

我们可以采用多种数据结构来表达一个多边形模型。最简单直接的方法就是使用一个链表来存储多边形，而每个多边形的数据结构如下：

P=((x1,y1,z1),(x2,y2,z2),……,(xn,yn,zn)

这是一个有 n 个顶点的多边形，从(xl，yl，zl)到(xn，yn，zn)。很明显，这种方法会造成很大的存储空间的浪费，因为有很多顶点是同时被几个多边形所共享，这样它们就被重复存储了好多次。我们可以改进一下，再使用一个链表来存储顶点。如图 17-10。

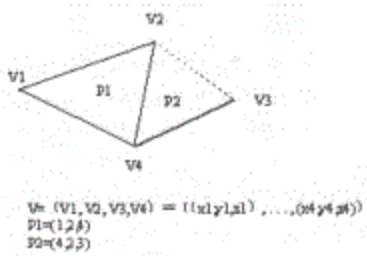


图 17-10 多边形的表达

如图 17-10 所示，两个多边形 P1 和 P2，四个顶点。V 是顶点链表。而 P1，P2 分别用指向链表的指针或者下标来表示。这种方法比前一种显然要优越些，节省了存储空间。但还有个问题：我们无法一下子知道 P1 和 P2 是否相邻?相邻的边是哪个?为了解决这个问题，我们又引入了一个链表，用来存储各条边的信息。这样的话，就可以一下看出两个多边形是否相邻了。如图 17-11。

从图 17-11 我们看出，除了顶点链表 V 外，我们又建了一个边链表 E。其中每个边由两个顶点的指针或者下标，和这个边所在的两个多边形组成(有可能一个为空)。比如 E4，是由(V2,V4,P1,P2)表示。

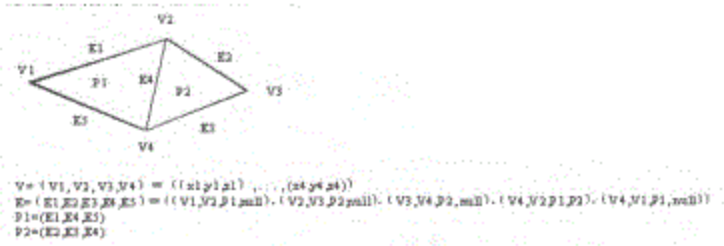


图 17-11 多边形的表达

曲线与曲面

前面介绍了多边形模型，用多边形和多边形模型可以近似地表达曲面和复杂的物体。但随着精度要求的提高，多边形的数量也需要急剧增多。比如说用多边形来近似圆形，显然多边形的边数越多，每条边越短，则我们所得到的多边形越近似于圆形。但这样带来的后果是对存储空间要求多了，而且做任何变换操作都是很复杂的，因为你要对所有的多边形进行变换。

下面我们要介绍的方法，是用三次方多项式来表达一条曲线段。这么做的好处是更加精确，也容易对曲线段进行操作。需要指出的是，这么做也还是有某种近似，并不是 100%地精确的。

我们知道，在数学上表达一个平面上的线段可以用 x,y 之间的函数来表示，即 y=f(x)。这是所谓的显式表达。而隐式表达则为 f(x，y)=0。这两种表达都有自己的问题，不方便于运算。而参数表达，则为 X=X(t)，y=y(t)。很明显，这种表达方法比较方便我们对 x,y 进行变换操作。

在三维空间里，一段曲线可以用如下的形式表达：

$$x(t)= a_xt^3+b_xt^2+c_xt+dx$$
$$y(t)=a_yt^3+b_yt^2+c_yt+dy$$
$$z(t)=a_zt^3+b_zt^2+c_zt+dz$$

图 17-12 为了确定曲线段的参数，需要 P0 和 P1 的坐标，以及曲线段在 P0 和 P1 两处的斜率。共四组数据

为了求出 12 个系数，我们需要四组数据，一般来说需要这个曲线段头尾两点的坐标，和曲线在这两点的斜率。如图 17-12 所示。

曲线在某点的斜率，实际上是曲线在这点的切矢量。其计算公式如下：

$$Q(t)=\begin{bmatrix}x(t) & y(t) & z(t)\end{bmatrix}$$

$$\frac{d}{dt}Q(t)=\begin{bmatrix}\frac{d}{dt}x(t) & \frac{d}{dt}y(t) & \frac{d}{dt}z(t)\end{bmatrix}=\begin{bmatrix}3a_xt^2+2b_xt+c_x & 3a_yt^2+2b_yt+c_y & 3a_zt^2+2b_zt+c_z\end{bmatrix}$$

通过上面的方法，我们可以根据四组数据，确定 12 个系数，从而确定了一段曲线的特性。而几段曲线相连接起来，就可以形成很复杂的形状。

而确定四组数据的方法，也有许多种。不同的方法，绘出的曲线也不尽相同。两种比较常用的，一是 Hermite 曲线，是根据两个顶点和两个切矢量来直接计算得出。另一种是 Bezier 曲线，是根据四个点，包括两个顶点和两个控制点来得出曲线的。如图 17-13。

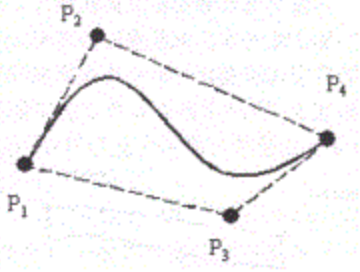


图 17-13 Bezier 曲线。P1 和 P4 是顶点，P2 和 P3 是控制点，并不在曲线上。实际上 P1P2 和 P3P4 分别确定了两个切矢量

而曲线的矩阵表达为：

$$Q(t)=\begin{bmatrix}x(t) & y(t) & z(t)\end{bmatrix}=\begin{bmatrix}t^3 & t^2 & t & 1\end{bmatrix}\begin{bmatrix}m11 & m12 & m13 & m14\\ m21 & m22 & m23 & m24\\ m31 & m32 & m33 & m34\\ m41 & m42 & m43 & m44\end{bmatrix}\begin{bmatrix}G1\\ G2\\ G3\\ G4\end{bmatrix}=T\bullet M\bullet G$$

G 是由四个单元构成的矢量。这四个单元分别对应确定曲线特性的四组数据。比如对应 Hermite 曲线，就是两个顶点和两个切矢量。而 M 为 4×4 的矩阵，是预先定好参数的基本矩阵，Hermite 和 Bezier 分别对应不同的数值。

把这种曲线的概念推广，我们可以用同样的方法来构造曲面。既然一段曲线用 Q(t)=T·M·G 表示，其中 G 是由两个顶点和切矢量决定的，那么我们可以设想：保持 M 不变，在三维空间里移动这两个顶点，同时可以改变切矢量。那么我们实际上就是移动并改变了这这条曲线段。它的轨迹，或者说把这些改变并移动了的曲线段都连接起来，就形成了一个曲面。这种方法的数学表达如下：

$$Q(s,t)=S\bullet M\bullet G(t)=S\bullet M\bullet\begin{bmatrix}G_1(t)\\ G_2(t)\\ G_3(t)\\ G_4(t)\end{bmatrix}\qquad \text{其中：}0\le t\le 1$$

这个公式是直接从曲线的公式转化过来的。我们为了简化起见，把 T 改写成 S(实际就是换一种符号)。也就是说当 G 固定时，曲线随 s 参数的变化而变化。而 G 由原来的不变的矢量，改成了也随参数 t 变化的矢量了。即两个顶点和两个切矢量随 t 变化。这样我们有两个参数 s 和 t 了。因此曲面就可以表达为 Q(s，t)，即 Q 由 s 和 t 两个参数决定。

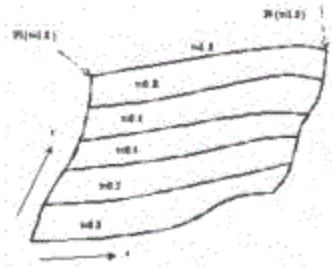


图 17-14 曲面示意图：箭头表示 s 参数和 t 参数由 O 到 1 的方向。当 t 固定时，s 由 0 到 1 定义了一条曲线。而 t 由 O 到 1 定义了一系列曲线 这些曲线连接起来，就是一个曲面

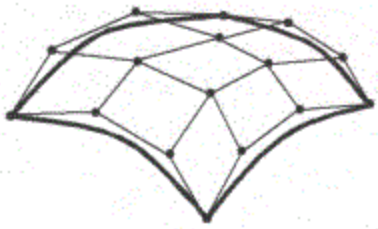


图 17-15 由 16 个控制点构成的 Bezier 曲面

层次关系

通过前面介绍的多边形模型和曲面，我们学会了构建基本的形体，如立方体等。我们也可以构建出具有曲面的形体。然而要构建更复杂的由多个部件组成的形体，如车辆和人体，单一的物体就不够了，必须把许多物体组合起来，形成某种层次结构。

举一个简单的例子，一辆汽车的层次结构如图 17-16。

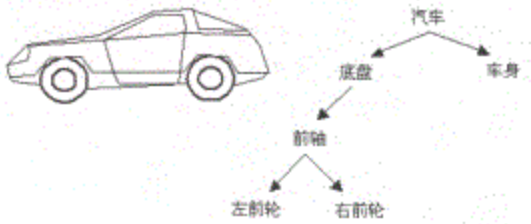


图 17-16 汽车的(部分)层次结构

从图中我们看出：这辆汽车由很多部件构成，底层的部件可以组合成高一层的部件，高一层的部件在一起又可组成更高一层的部件。这样由底向上，构成整个复杂物体，这种结构是一种树状结构，也可以看成是有向无闭环图(directed acyclic graph)。节点是部件，而节点间的连接代表了包含(连接)关系。在树状结构中有所谓父节点和子节点的关系，如前轴是前轮的父节点，而前轮是前轴的子节点。

在实际编程中，一般是用链表来存储这种树状结构。各部件单独存放。链表中包含指向一个个部件的指针，还有表达部件之间关系的信息。

各部件之间的连接，一般来说都有其空间上的限制。也就是说低一级的部件和高一级的部件之间的位置关系不是完全自由的。比如说人的大臂和小臂之间只能旋转约 90 度，而汽车轮子只能绕轴旋转等。

当我们对高一级的部件做某种几何变换时，作为它的子节点的部件，也同时继承了这种变换。也就是说，当我们平移了汽车时，它上面所有的部件，也即子节点，如底盘轮子等，也都被平移了。也就好像是平移变换的矩阵沿树状结构下传一样。但当我们对于节点做某种

变换时，父节点不受影响。如我们只是转动轮子的话，则车身不受影响，不会跟着转动。

投影——从 3D 到 2D

现在我们有了在三维世界里的一些多边形模型，这个三维世界的数学表达是在程序中存在的。我们必须把它显示出来。而计算机的屏幕是二维的。如何把三维世界转换到二维屏幕上呢?这就引入了投影的概念。

简单地说，投影就是把一个 n 维坐标系中的一点，转换成一个 n-k 维坐标系上的一点。我们所感兴趣的，是如何把三维空间的一点，转换到二维平面上。有两种方法，分别如图 17-17。

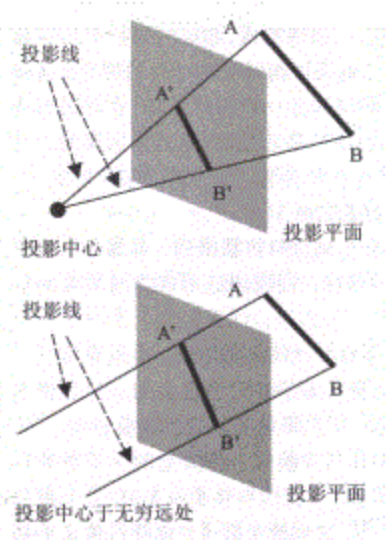


图 17-17 透视投影(上)和平行投影(下)

如图 17-17 所示，投影是通过以下步骤完成的：首先在三维空间确定一点，这个点称为投影中心，再在适当位置确定一个平面。这个平面被称为投影平面，三维空间中的物体将被转换到这个二维的平面上。然后自投影中心向目标物体发射直线，这些线被称为投影线。假设我们在三维空间有一个线段 AB，那么我们可以将 AB 分别和投影中心连接起来，连线和投影平面相交，交点为 A'和 B'。A'B'就是线段 AB 在投影平面的投影。这样我们就把一个三维空间内的线段转换到了二维平面上。这种投影就叫做透视投影。透视投影的效果比较符合我们日常所见，即近大远小。因此一般游戏中都使用透视投影。

而当我们把投影中心移到无穷远处时，这样所有的投影线都变成了平行线。这时透视投影就变成了平行投影。平行投影看起来不太真实，但它可以准确表示出物体之间的平行关系和相对大小。因此平行投影在三维工具软件中应用很多，比如说地图 / 关卡编辑器。

下面主要介绍透视投影。

透视投影

透视投影可分为三种，分别为一点透视、两点透视、三点透视。如图 17-18 所示。

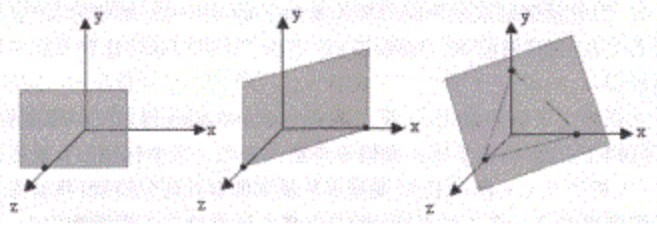


图 17-18 图中依次为一点透视、两点透视、三点透视

而三种透视的区别，在于一点透视中，投影面只和一个坐标轴相交，两点透视中和两个

坐标轴相交，三点透视中投影面和所有三个坐标轴都相交。而从一点透视的效果上看，最后的二维平面上只有一个消失点。也就是说原来平行的线，现在不再平行了，要在很远的地方汇聚于一点，这个点就是消失点(vanishing point)。三种透视的效果图如图 17-19。

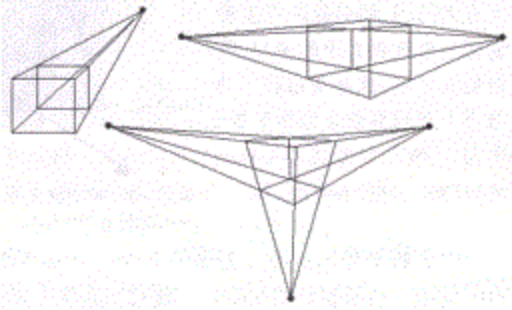


图 17-19 三种透视的不同效果。图中各点为消失点

前面介绍了投影透视的基本概念，现在我们要开始研究如何从数学上表达和如何在程序中实现这种透视。我们已经有了一个三维的世界坐标系，在这个三维坐标系所定义的三维空间里有一些物体，投影透视的第一步，显然先得确定投影点和投影面的位置。它们的位置都是根据游戏的实际需要确定的。其中投影点只需给出其 XYZ 坐标值就可以了。而投影面则相对麻烦些。我们知道要在三维空间确定一个平面有很多种方法，其中比较简单的是找一个点和一个矢量。这个点是面上的一点，被称为 VRP(视平面参考点，自此往后，我们也把投影面称为视平面)。而矢量则是这个平面的法矢量(normal vector)，被称为 VPN(视平面矢量)。这样就能惟一地确定一个平面了。我们就用这种方法来确定投影面。

在确定了投影面之后，在二维的平面里我们还得定义二维坐标，为了区别于三维 XYZ 坐标，我们将投影面上的二维坐标用 UV 坐标来表示。u 相当于 x，v 相当于 y。确定 UV 坐标的方式比较特别，我们先在三维空间里定义一个矢量 VUP(指上矢量)，认为这个矢量是指向上方的，也就是说 V 的方向。然后我们把这个矢量投影到投影面上，它的投影就是 V 坐标轴。然后我们可以通过 V 坐标轴和 VRP 得出 U 坐标轴了。

在确定了视平面和视平面上的坐标之后，由于平面是无穷大的，我们必须在它上面定义一个有限大小的窗口，作为显示之用。这是通过我们给出两个点的平面坐标得到的，即左下和右上方的点(umin,vmin)和(umax,vmax)。

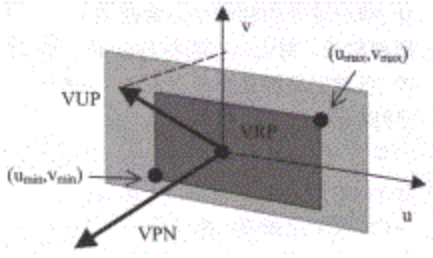


图 17-20 确定投影面(视平面)和投影面上的二维坐标系，以及窗口

需要特别指出的是：投影中心不一定在 VPN 所在的直线上，而 VRP 也不一定是窗口的中心。当我们用直线把窗口的四个顶点和投影中心连接起来后，我们就得到了一个金字塔状的空间，金字塔底是在无限远处无限大。从理论上说，只有这个空间里的三维物体才有可能被投影到视平面上的窗口里。如果一个物体位于这个金字塔之外，则它不会出现在窗口里。我们把这个金字塔状的空间称作可视空间(view volume)。当然，我们实际上不需要无限大的空间，因为当物体太远了，则整个物体的投影就是一个点，毫无意义。因此，我们要把这个可视空间截取一段最重要的有限大小的空间，这样计算会简单得多，效果也不会有太大影响。一般是在

视平面前后各取两个与视平面平行的面。这样我们就得到了一个台状的空间。在这个台状空间之外的物体，我们不予理会；在台状空间之内的，则可以投影到窗口里面。如图 17-21 所示。

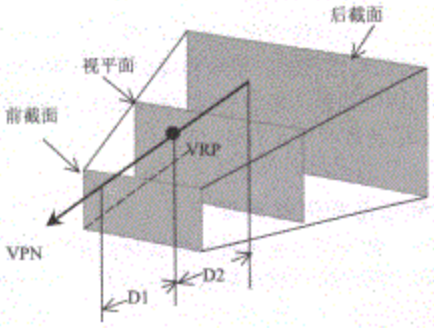


图 17-21 透视投影的台状可视空间。其中前截面和视平面距离为 D1，后截面和视平面距离为 D2

透视投影的数学表达

为了简化起见，我们假定视平面是 z=d，也就是说视平面垂直与 z 轴，位于 z=d 处。而投影中心在原点。这样的话，透视投影可以用一个 4x4 的矩阵来表示了。我们用图 17-22 来说明并推导出这个矩阵。

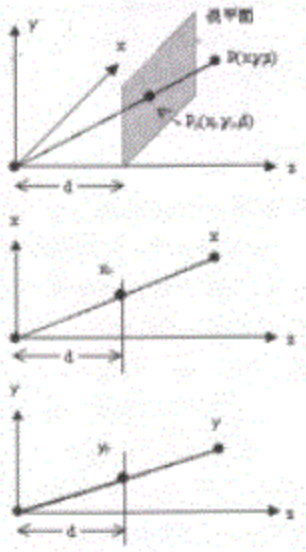


图 17-22 透视投影的数学计算

从图 17-22，根据三角公式，我们不难得出：

$$\frac{x_p}{d}=\frac{x}{z} \text{ 和 } \frac{y_p}{d}=\frac{y}{z}$$

于是我们得出：

$$x_p=\frac{d \bullet x}{z} \text{ 和 } y_p=\frac{d \bullet y}{z}$$

这样我们可以得出在视平面上的 XY 坐标了。如果用矩阵表示，则为：

$$M_{per}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 1/d & 0\end{bmatrix}$$

通过这个矩阵 Mper，我们就可以把台状可视空间里所有的三维的点，转换到二维的视平面上，得到其 XY 坐标值。

至于其他情况，比如说投影中心不在原点，或者视平面在其他地方，都可以推出其他的相应的公式和矩阵，和我们的例子大同小异。我们更可以做一系列变换，将投影中心变换到原点，再把视平面变换到和 Z 轴垂直。

三维空间裁剪

在介绍了透视投影的数学公式后，我们就可以来看看从三维空间到二维平面的转换的整

个过程了。这个过程由几个步骤组成，如图 17-23 所示。

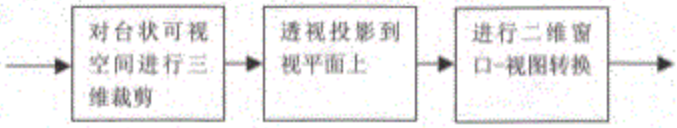


图 17-23 三维到二维的转换

这个过程具体地讲是这样的：我们首先在世界坐标系中放好各种三维物体，对游戏来说就是游戏中的山峦树木建筑敌人等。然后我们设定好投影中心和视平面，再设定好台状可视空间。然后把世界坐标系中所有物体和这个可视空间比较。如果这个物体是完全在可视空间里面，予以保留，因为这个物体可以被投影到视平面的窗口里。如果物体完全位于可视空间之外，则抛弃，因为它位于窗口外，玩家看不到。而如果物体有一部分在可视空间里，一部分在可视空间外面，这就比较麻烦，需要对物体进行裁剪，就是保留在可视空间里的那部分，抛弃在可视空间外的那部分。完成三维裁剪后，应用矩阵把三维的物体投影到视平面上。由于经过了裁剪，所有的图形都在视平面上的窗口里面。最后我们进行二维的窗口-视图转换，得到的是在屏幕坐标系下的图形，这样我们就可以在屏幕上显示了。三维游戏世界就这样被显示在玩家的面前了。

显而易见，这个过程的第一步，即三维裁剪，其计算难度是比较大的。首先这个台状的可视空间的形体就比较复杂，它有四个面都是斜面，这在计算的时候就很麻烦。于是程序员们想：有什么方法可以简化三维裁剪的运算呢?他们想到如果这个可视空间是立方体的话该多好!如果可视空间是立方体，则三维裁剪变得非常简单了。举例来说：如果可视空间取为一个长方体，其六个面为 x=-1,x=1,y=-l,y=l,z=0,z=1 的话，我们就不需要把物体和斜面相比较了，我们只需把一个物体和长方体的六个面做比较，这样就简单多了。我们更可以使用高效的 Cohen-Sutherland 算法。

因此，程序员们决定在所有步骤之前，加入另一个步骤，就是把一个台状可视空间转换为一个长方体可视空间。虽然在这一步我们需要做些额外的矩阵相乘的运算，但第二步，即三维裁剪的运算得到了相当大的简化。这个新的长方体的可视空间，叫做正规化可视空间(canonical view volume)。而这个变换，就叫做正规化变换(normalization transformation)。

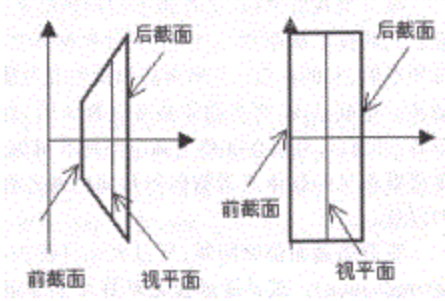


图 17-24 将一个台状可视空间转换为一个长方体可视空间，包括前后截面和视平面都被改变了。

而这个变换是通过一个矩阵相乘完成的。特别需要指出的是，这样的变换丝毫不会影响最后的二维图像，因为变换本身保持了物体的相对大小和位置关系。而且通过把台状可视空间转换为长方体可视空间，我们实际上已经完成了透视投影的一部分。新的三维空间到二维

图像的转换过程如图 17-25 所示。

图 17-25 三维到二维的转换

隐藏面消除

在三维裁剪之后，所有剩下的物体都是在可视空间之内了，也就是说它们的投影都在窗口内部。但这些物体之间的位置关系是很复杂的，有的在前，有的在后。在前面的物体就有可能部分或全部遮挡住后面的物体。也就是说，有些物体是可以看见的，有些则是隐藏在其他物体后面的。我们只希望在屏幕上画出那些看得见的物体或其表面，我们不希望把看不见的物体或者物体的背面也画上去。这就引入了隐藏面消除的问题。

所谓隐藏面消除问题，反过来也可称为可视面决定(visible-surface determination)，就是要通过某种算法来决定在可视空间内的物体的哪个部分是可以看见的，哪个部分是看不见的。在搞清楚了之后，就可以在屏幕上画出可以看见的那部分了。这样我们才能得到正确的二维图像。又因为三维物体一般是由多边形构成的(在最后处理的时候所有的多边形都要再细分成三角形)，因此我们的任务就是要搞清楚哪个多边形(也就是面)在前，哪个在后。如果一个多边形被遮住了一部分，我们还得把多边形再进一步分成两个小多边形，一个是可见的，一个是不可见的。

隐藏面消除算法主要有两种，一种被称为精确到点算法(imageprecision algorithms)，另一种被称为物体比较算法(object-precision algorithms)。第一种算法以 Z 缓存算法(z-buffer algorithm)为代表，其基本思路是对二维屏幕上的每一个像素，找到对应它的三维空间内的物体们(也就是在这个像素和投影中心的连线上)，距离最近的那个物体就是可视的物体了，其他物体为不可见。而物体比较算法，顾名思义，就是真正地去把一个物体和所有其他物体相比较，然后得出这个物体的哪个部分是没有被其他物体所遮挡，这个部分就是可见的。比较是在三维空间内进行的。

两种算法中，精确到点算法是根据屏幕分辨率来计算的。如果要放大一个图像，则需要全部重新计算。而物体比较算法和屏幕分辨率无关，所有的计算都是在原始的三维空间内进行，则不需要重新计算。但用物体比较算法得出结果后，还需要另外一步将计算结果显示到二维屏幕上。

Z 缓存算法

z 缓存算法，是一种最简单也是最流行的精确到点算法，并且在硬件上和软件上都比较容易实现。它的基本思想是除了在内存中保持一块帧缓存，用以存储各像素的颜色值外；还保持一个 Z 缓存。其大小和帧缓存相同，都是由屏幕分辨率决定。其中的每个单元，存储的是这个像素的 Z 值。

下面介绍一下这个 Z 值是什么，是如何得到的。我们知道在我们将一个三维空间里的三角形投影到视平面上，然后转换到屏幕坐标系下时，我们只需要对三角形的三个顶点进行变换。也就是说，在三维空间的三个点 P1(x1，y1，z1)，P2(x2，y2，z2)，P3(x3，y3，z3)分别对应屏幕坐标系下的 Pl(x1'，y1')，P2'(x2'，y2')，P3'(x3'，y3')三个像素。当我们把这三个点从三维转换到二维后，其 Z 坐标值就失去了。Z 缓存算法所做的，就是把这个 z 值先存在缓存里留作后用。而我们得到了屏幕坐标系下的三角 Pl'P2'P3'，就可以通过简单的方法得到这个三角上所有的像素(点)，因为屏幕分辨率是有限的。我们更可以根据这三个像素(点)的 XY 坐标和 Z 值，通过插值算出三角形上所有像素(点)的 Z 值。其计算方法如下：

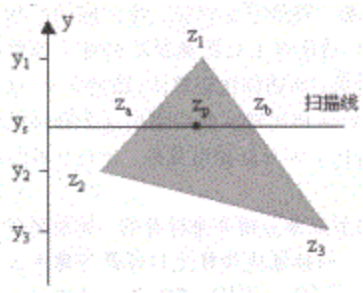


图 17-26 一个已经投影到屏幕坐标系下的三角形。扫描线是绘制二维图形时所用，也就是说当我们在屏幕上填充这个三角形时，是从 Z1 开始一行一行往下画的

图 17-26 所示为一个已经投影到屏幕上的三角形，我们知道其 P1'P2'P3'三点在屏幕坐标系下的 XY 坐标，我们也知道它们在三维空间里的 Z 值。这样我们可以应用插值公式，算出三角形上所有像素(点)的 Z 值。公式如下：

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

这个公式还有一个问题。就是每算一个像素(点)，都要进行除法运算。我们知道在程序中使用除法运算是最复杂也是最慢的。于是程序员们想：有什么方法可以回避除法运算呢？最后想到使用增量法(incremental calculations)可以解决这个问题。由于三角形是在一个平面上，而平面的公式为 Ax+By+Cz+D=O。因此我们可以得出：

$$z = \frac{-D - Ax - By}{C}$$

这样当我们沿着扫描线画第二个像素(点)，即(x+l,y)的时候，新的 Z 值就是：

$$z - \frac{A}{C}$$

而当我们画完了这个扫描线上所有的点后，要转到下一行画下一条扫描线。则同样可以算出 y 的增量是 B/C。这样我们只需进行加减，完全回避了乘除了(除了一开始要根据平面公式算 A/C 和 B/C)。

而当两个三角形都包含同一个像素(点)时，那么一定有一个三角形在前，一个在后。这时候 z 缓存就派上用场了。假设在三维空间里有一个红色的三角形和一个蓝色的三角形。我们先对红色的三角形转换到屏幕坐标系下，得出了三角形上所有像素(点)的 z 值，存入 z 缓存相应的单元中。然后再把红色存储到帧缓存的相应的单元中。然后我们对蓝色三角形做变换，得出另一些像素(点)的 z 值。然后把蓝色存储到帧缓存中。注意：在存储的过程中，当我们发现一个单元已经有了红色的时候，我们得把 Z 缓存中这个单元的 Z 值调出来和我们新得到的 Z 值比较，谁的 Z 值越大，说明谁在前面。如果是红色三角上点的 Z 值大，我们就不用进行任何操作。反之，则把蓝色存入帧缓存，并更新其对应的 Z 值。这样经过两次运算，就把两个三角形之间的先后关系搞清楚，并可以正确显示出来了。如图 17-27 所示。

```
0 0 0 0 0     5 5 5 5 5     5 5 5 5 5
0 0 0 0 0     5 5 5 5       5 5 5 5 0
0 0 0 0 0 +   5 5 5       =   5 5 5 0 0
0 0 0 0 0     5 5             5 5 0 0 0
0 0 0 0 0     5               5 0 0 0 0

5 5 5 5 5                     5 5 5 5 5
5 5 5 5 0       3             5 5 5 5 0
5 5 5 0 0 +     4 3       =   5 5 5 0 0
5 5 0 0 0       5 4 3         5 5 3 0 0
5 0 0 0 0       6 5 4 3       6 5 4 3 0
```

图 17-27　5x5 的矩阵代表 Z 缓存，先被初始化为 0。然后处理一个三角形，其中像素(点)的 Z 值为 5。然后把得到的 Z 缓存再和第二个三角形比较。最后得出的 Z 缓存正确反映了两个三角形之间的关系

而整个 Z 缓存算法的操作是这样的：首先根据屏幕分辨率确定缓存大小。帧缓存中存储每个像素的色彩值，如果用真彩的话，需要 24 位三个字节。而 Z 缓存中存储每个像素的 Z 值，一般用 16 位，两个字节。然后将帧缓存所有单元初始化成背景色值，然后将 Z 缓存所有单元初始化成 0。这样代表目前只有背景(即后截面，即最远的点)。然后对可视空间内所有的三角形，一个一个地依次投影到视平面上，再转换到屏幕坐标系上。然后再把三角形上所有的像素(点)的 Z 值求出来，和 Z 缓存中已经有的 Z 值比较。如果新的 Z 值比 Z 缓存中的 Z 值大，说明新的像素(点)在旧的前面，我们需要更新 Z 缓存中的 Z 值，并把新的色彩值存储到帧缓存中。Z 缓存中可能有的最大的 Z 值代表的是前截面上的点，因为不会有比前截面更近的了。

Z 缓存算法的伪码如下：

```
void zBuffer(void)
{
    int x, y;
    for (y=0; y<YMAX; y++)                          //屏幕上的所有行
{
        for (x=0; x<XMAX; x++)                      //屏幕上的所有列
{
            WfitePixel(x, y, BACKGROUND_VALUE);   //将屏幕设为背景色
            WriteZ(x, y, 0);                        //Z 缓存所有单元置 0
        }
    }
for (each polygon)      //对所有的多边形
    {
        for (each pixel in polygon's projection)
//多边形投影到视平面上后，转换到屏幕坐标系，对上面所有的点
        {
        double pz=polygon's z-value at pixel coords(x，y);      //计算相应的 Z 值
        if (pz>=ReadZ(x,y))
//取出此点目前的 Z 缓存中的 Z 值，如果新的 Z 值大
            {
            WriteZ(x,y,pz);        //更新 Z 缓存中的 Z 值
            WritePixel(x,y,polygon's color at pixel coords(x,y));
//更新帧缓存中的颜色值
            }
        }
    }
}
```

光线明暗处理

如果我们的游戏世界里没有任何光源和光线的话，那么所有的多边形都将是黑洞洞的，我们在屏幕上什么也看不见。因此，必须在三维空间里设置光源，照射到三维物体上，才能使得物体表面有明暗的效果。光线模型(illumination model)和明暗模型(shading model)这两个模型就应运而生了。其中，光线模型是根据物体表面上一个点的位置、方向、物体表面特性和光源特性来决定这点的色彩／光线强度。而明暗模型是根据已知的几个点的色彩／光线强度来计算出整个物体表面的明暗过渡关系。需要特别指出的是：光线明暗模型不是完全根据

自然世界的物理规律得出的，而只是一种纯数学上的模型，其目的是产生类似于自然界的光影效果。实际上也是某种视觉欺骗!

光线模型

●环境光(Ambient Light)

最简单的光线模型就是所谓环境光，也就是没有方向性，没有明确的光源，呈现漫反射效果的光。在现实世界中，当光线经过多个表面的重复多次反射后一般是这个效果。

要用数学表达某种光线模型，就是要建立一个光线公式(illumination equahon)，用这个公式来计算物体表面某点的光照强度。对环境光来说，其公式为：

$$I=I_aK_a$$

其中，I 是物体表面最后计算得出的光强(也就是亮度)。I_a 是环境光的强度，对所有的物体都是一样的。Ka 是环境光反射参数，完全由物体本身表面特性决定，其数值在 0 和 1 之间，使用环境光模型，对我们游戏世界中的各种物体进行打光的话，一个物体上的表面是同一个强度，也就是同一种颜色或灰度，其亮度由 IaKa 决定。

●点光源(Point Light Source)

显然，有了环境光模型，我们可以看到游戏世界中的各种物体了。但这些物体表面都是单一色彩均匀平涂的效果，没有高光和过渡，显得很不真实。于是我们引入了更复杂一点的“点光源”的概念。点光源就是在空间里有一点，光线从这一点向四面八方均匀射出。有了点光源，物体表面就可呈现出更复杂的明暗变化。

●漫反射(Diffuse Reflection)

粗糙的表面，如石壁、粉墙、砖面等，在光源照射下呈现漫反射特性。也就是说在光源照射下，表面向四周的反射是均匀的，向各个方向反射的光的强度是相同的。而这个强度取决与点光源的方向和表面的法矢量，如图 17-28 所示。

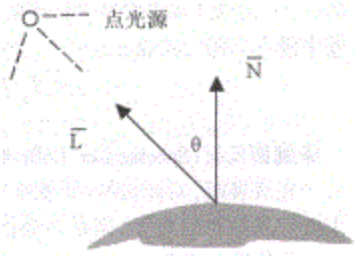


图 17-28 漫反射

$\overline{N}$ 是表面上一点的法矢量。$\overline{L}$ 是由表面上一点指向光源的矢量。两个矢量之间夹角为 θ。这一点的光强由以下光线公式确定：

$$I = I_p K_d \cos\theta$$

其中，I_p 是点光源的光强。K_d 是物体的漫反射参数，是由物体表面的属性决定的，其数值在 0 和 1 之间。如果我们知道点光源的位置，也知道物体表面的物理属性(K_d)和几何特征($\overline{N}$)，使用这个公式，就可以得出物体表面任何一点的光强了。

从上面的公式我们也可以看出，只有当在 θ 在 0° 和 90° 之间时，点光源才会对物体表

$\overline{N}$ $\overline{L}$ $\overline{N}$ $\overline{L}$

面有影响。也就是说物体背后的光源对物体不起作用。

为了计算 cos θ，我们可以使用　和　的矢量点乘。这需要先对　和　进行正规化处理，就是 normalization，使其长度为 1。这样公式就成为：

$$I = I_p K_d (\overline{N} \bullet \overline{L})$$

当一个点光源距离物体表面足够远时，我们可以认为物体表面所有的点都相同，这时的点光源就成了定向光源(directional light source)了。

前面的公式没有考虑点光源到物体表面的距离。一般来说，距离光源越远，光强越弱。为了体现这种效果，我们又引入了一个参数，这个参数称为光源衰减参数——f_{att}，用来体现距离对光强的影响。目前比较好的一个公式是：

$$f_{att} = \min(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1)$$

其中，C1，C2，C3 都是常量。d_L 是从物体表面上一点到光源的距离。这样整个漫反射的公式就变成：

$$I = f_{att} I_p K_d (\overline{N} \bullet \overline{L})$$

●镜面反射(Spectacular Reflection)

光滑表面，如玻璃等，呈现的是镜面反射的特性。比如我们把一个大红苹果放在强光下，苹果上会出现高光的效果，高光区域是白色的，不是物体的固有色。

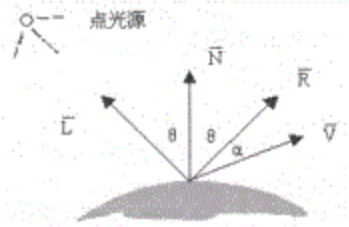


图 17-29 镜面反射

镜面反射如图 17-29 所示。

图中的 $\overline{R}$ 是 $\overline{L}$ 对 $\overline{N}$ 的镜像。对理想的光滑表面来说，入射的光线将只沿反射方向 R 传播，只有当观察者从 R 的反方向看过去才能观察到镜面反射。

而对一般的光滑表面来说，沿 R--方向反射的光强最强的(α=0--)，光强随着 α 角度的增加而迅速衰减。Phong 模型就是一种用来表示这种一般镜面反射的数学模型。其公式如下：

$$I = f_{att} I_p K_s \cos^n \alpha$$

其中，Ks 是物体的镜面反射参数，由物体本身的物理属性决定，其数值在 0 和 1 之间。n 的数值取从 1 到几百，一般是根据实际需要定。公式中 cosn α 的作用就是产生镜面反射的效果。当 n 取值接近 1 时，这个公式所产生的效果是沿方向的比较大范围的均匀的光强分布；而 n 的数值越大，则光强分布越紧密集中于的周围。

同样地，我们也可以使用矢量点乘来计算 cosn α，这样经过改进的公式如下：

$$I = f_{att} I_p K_s (\overline{R} \bullet \overline{V})^n$$

综合模型

根据前面我们所介绍的三种光线模型，我们就可以得到一个综合的模型，其公式如下：

$$I = I_a K_a + \sum \left[f_{att} I_p K_d (\overline{N} \bullet \overline{L}) + f_{att} I_p K_s (\overline{R} \bullet \overline{V})^n \right]$$

注意公式中的符号，它表明在游戏世界中我们可以设定多个在不同位置的点光源。它们和环境光配合，共同照亮了我们的游戏世界，并产生着漫反射和镜面反射的效果。

●明暗模型(Shading Models)

有了光线模型，知道了光源位置和强度，知道了物体表面的物理特性和几何特性，我们可以计算物体表面上的任何一点的光强和明暗了。但这么一个点一个点逐一计算无疑是很笨的法子，计算量也大到我们不能承受的地步。而明暗模型所解决的，就是如何根据物体表面上几个点的光强，来推算出整个表面上其他点的光强，从而形成某种明暗过渡的效果。

●平面明暗处理(Flat Shading)

显然，最简单的方法就是在一个多边形上任取一点，使用光线模型计算其光强，然后这个多边形上所有的点就都使用这一数值。从信号分析的角度讲，就是对一个多边形上的一点进行取样，并用这个样本来代替多边形上所有的点。这样的效果就是一个多边形一个颜色和亮度，没有细腻的过渡，多边形和多边形之间界限明显。效果如图 17-30 所示。

图 17-30 SQUARE 的《TOBAL 1》，使用的是最简单的平面明暗处理，使用单色涂满整个多边形。特别明显的是右边人物的衣服，多边形与多边形之间界限分明

插值明暗处理

为了弥补平面明暗处理的不足，人们又引入了插值明暗处理的方法，用以产生细腻的明暗过渡效果。其基本思想是对一个三角形(我们在前面介绍过：组成游戏世界的所有的多边形到最后都要被细分为三角形去处理)，计算其三个顶点的光强，然后使用线性插值，计算出三角形上所有其他点的光强值。

具体的插值计算方法的不同，决定了明暗处理效果的不同。两种比较常见的处理方法分别是 Gouraud 明暗处理和 Phong 明暗处理。

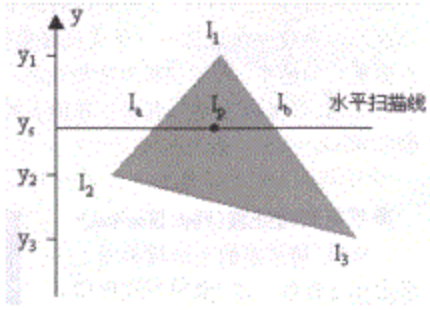


图 17-31 Gouraud 明暗处理

●Couraud 明暗处理(Gouraud Shading)

Gouraud 明暗处理，又称为强度插值明暗处理(intensity interpolation shading)。顾名思义，它是先根据三角形三个顶点的法矢量，和任意的光线模型，得出这三点的光强。然后，沿三

$$I_a = I_1 - (I_1 - I_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

角形的边和水平扫描线分别进行插值计算，得出这个三角形上的各点的光强。其示意图见图 17-31，公式如下：
其中的 I 是光强，x 和 y 是转换到二维视平面上的坐标。有了这个公式我们可以计算一个三角形上任意点的光强了。我们同时注意到这个线性插值公式和前面的 Z 缓存算法中使用的插值公式是一样的。

特别需要指出的是：使用 Gouraud 明暗处理，不仅一个三角形内部有了明暗过渡效果，邻近三角形之间的过渡也会比较柔和，从而整个物体的表面都显得更光滑了。这是通过计算三角形的三个顶点的法矢量实现的。

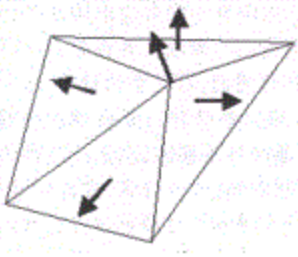


图 17-32 计算法矢量

在我们使用光线模型计算三个顶点的光强时，我们必须知道这三个点的法矢量。问题似乎很简单——直接用这个三角形所在平面的法矢量不就行了?在平面上的三个顶点，它们的法矢量当然是相同的。但是，我们不能忘记这一点：多边形模型是对复杂的曲面的近似!因此这个三角形实际上是一个复杂的曲面，这三个顶点的法矢量实际上是不一样的!那么有什么方法近似得出这三个点的法矢量呢?图 17-32 就提供了一种方法。

图中四个相邻三角形共有一个顶点，四个三角形的法矢量分别指向不同方向。共有的顶点的法矢量就是这四个法矢量的平均值。这样计算得出的顶点法矢量不仅保证一个三角形内部的明暗过渡，也保证了四个三角形相交处过渡柔和，不会出现明显的边界。

●Phong 明暗处理(Phong Shading)

Phong 明暗处理，又叫做法矢量插值明暗处理(normal-vector interpolation shading)。它比 Gouraud 明暗处理更进一步。在计算出三个顶点的法矢量后，我们先不计算这三点的光强，而是直接根据这三个法矢量来插值计算三角形上其他点的法矢量，有了这些法矢量后，再用光线模型来计算每一点的光强。

比较 Gouraud 明暗处理和 Phong 明暗处理的计算方法，我们可以看到使用 Gouraud 明暗处理的话，最亮的高光区不会出现在三角形内部，因为我们是对光强进行插值的，插值所得

到的三角形内部的点的光强不会超过三角形的三个顶点。而 Phong 明暗处理，由于是对法矢量插值，更为精确一些。计算出三角形内部的点的法矢量后，再根据光线模型计算光强，这样高光就可以出现在三角形内部，总体效果要比 Gouraud 明暗处理好一些。

表面材质

现在我们有了一个三维的游戏世界，里面有一些多边形构成的物体，几个光源，物体表面有了明暗过渡。但我们的游戏世界看起来还是太简单，不太自然!因为自然界中的各种物体表面，有复杂的材质，比如说树干的纹理、绿叶的脉络、土地的裂纹、墙壁的斑驳。要表现这些细节，靠增加多边形的数量和表面的复杂度是绝对不够的，也是计算量所无法承受的，最简单的解决办法，就是把预先画(或者合成，或者扫描)好的一张张纹理图，去“贴”到一个个多边形的表面。这样就可以达到乱真的效果。

二维材质

这种二维的纹理图，就称为材质图(texture map)，简称材质。一张材质自己定义了一个二维坐标系统，用 UV 坐标系来表示。材质上的点，叫材素点(texel)，用(u，v)来表示，其中 u 相当于 x，v 相当于 y。

材质处理(texture mapping)可以看成两步处理。第一步是把二维的材质坐标系上的点和三维空间内的三维物体上的顶点建立起对应关系。第二步是在把三维空间内的物体投影到二维屏幕上时，根据物体上的点和材质图上的点的对应关系去决定屏幕上的点的色彩和强度。这是一个二维到三维，再到二维的过程。如图 17-33 所示。

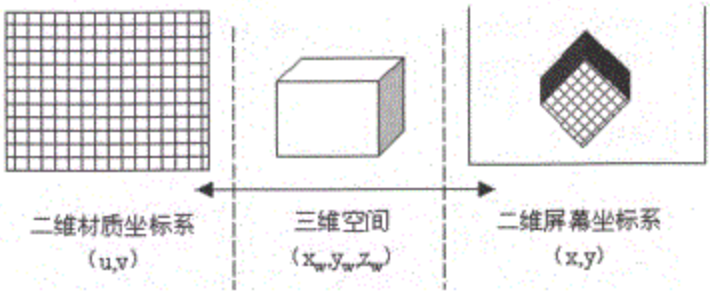


图 17-33 材质处理是一个由二维到三维，再由三维到二维的过程

材质处理有两种算法：前向算法(forward mapping)和逆向算法(inverse mapping)。其中逆向算法比较流行。所谓逆向算法，就是从二维屏幕坐标系开始，对屏幕上的每一个像素点(Pixel)，找到其对应的三维空间上的区域，再找到它对应的二维材质坐标系上的区域。二维屏幕上的像素点是正方形的，它所对应的材质坐标系下的区域是一个不规则的四边形。如图 17-34 所示。对这个区域内所有材素点的颜色进行加权平均，就得到像素点的颜色值。

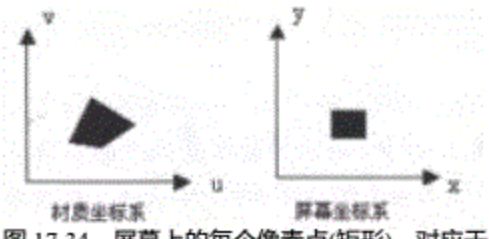


图 17-34 屏幕上的每个像素点(矩形)，对应于材质坐标系下的一个不规则四边形(甚至曲四边形)区域。矩形的四个顶点和四边形的四个顶点是一一对应关系

对于屏幕上像素点和材质坐标系下的材素点之间的对应关系，可以用一个简单的非线性变换来表示：

$$x = \frac{au + bv + c}{gu + hu + i}$$

$$y = \frac{du + ev + f}{gu + hv + i}$$

这样对屏幕上的每个点(x,y)都能找到对应的材质坐标系下的(u，v)了。上面的两个公式中有 9 个参数 a，b，c，d，e，f，g，h，i 需要事先计算出。一般是采用以下的方法：在建模的时候，手工设定三维模型上的多边形的顶点和材质坐标系上的点的关系。对于任一个四边形，如果有四个点的对应关系已知，就可以算出 9 个参数。然后可以根据公式得出四边形上其他点的对应关系了。

除了这种非线性变换的方法，一种更高效的方法是使用双线性插值。就是根据已知的顶点的对应关系，使用线性插值的方法，求出多边形上所有其他点的对应关系。关于这种方法，我们在前面的 Z 缓存算法和明暗模型中都介绍过其应用。

凸凹处理

凸凹处理(Bump Mapping)是 Blinn 在 1978 年设计出的一种很巧妙的方法。它通过对三维模型表面法矢量的处理使得表面出现坑坑洼洼的效果。对一个三维模型，如果只是用简单的二维材质处理，物体表面还会显得很平滑。如果要用增加多边形数量的方法来增加表面坑洼感，从计算处理上来说又太复杂。在前面介绍的光线模型和明暗模型中，我们看到物体表面的法矢量决定了表面各点的明暗。因此，Blinn 就想到在不改动物体表面多边形的基础上，可以通过修改物体表面法矢量，去“欺骗”明暗模型，在渲染时使得表面各点的明暗出现变化，从而形成坑坑洼洼的效果。

具体的算法简单介绍如下。首先需要一个二维凸凹材质图。这个凸凹材质图实际上定义了三维物体表面每个点的突起或者凹下的位移大小，用 B(u，v)表示。现有的三维物体表面上的点用 P(u，v)来表示。对表面上任意一点的法矢量可以用以下公式得到：

$$N = \frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v}$$

然后我们把点 P(u,v)沿 N 的方向，移动 B(u,v)的距离。

$$P'(u,v) = P(u,v) + B(u,v)N$$

有了 P'(u，v)，我们可以得到新的法矢量 N'。然后根据新的法矢量，使用明暗模型，

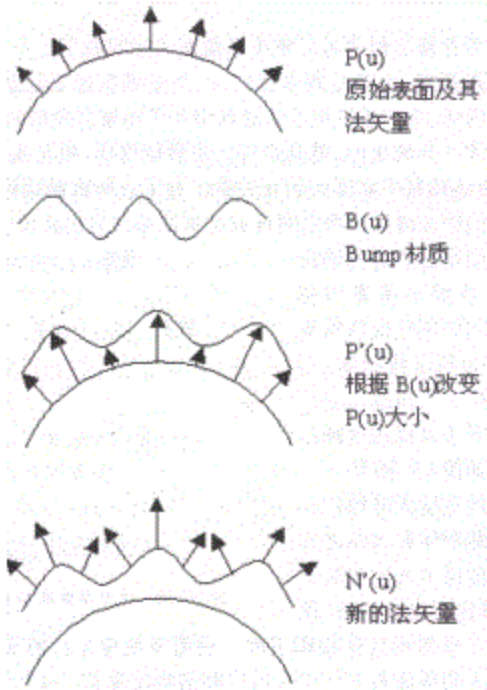


图 17-35　Bump Mapping 的一维示意图

得到的效果就是可以乱真的坑坑洼洼的表面效果。图 17-35 所示为凸凹处理的一维例子。

场景管理

一个游戏场景(scene)中，可能有成百上千个物体、上千上万个多边形，如何确定物体与物体之(多边形与多边形)间的关系成了一个大问题。要正确地显示游戏场景，必须知道哪个多边形在前，哪个多边形在后，谁把谁遮住了。这是我们前面介绍的隐藏面消除问题。而且游戏场景中的物体不全是静止的，而是不断运动的，这样就有了相互碰撞的可能。如何确定两个物体之间是否相互碰撞，属于碰撞检测(collision detection)问题的范畴。这两个问题相互关联，都很复杂棘手。我们曾经介绍过的 Z 缓存算法似乎可以解决隐藏面消除问题了。但实际上 Z 缓

存算法有其局限性，主要是效率比较差，如果场景过于复杂，并且实时性能要求高的话，就不敷使用了。这就引入了场景管理的问题。

场景管理所要解决的，就是设计一种数据结构，用它来表达复杂的游戏场景，表达场景中物体的相互关系。有了这种数据结构，我们处理隐藏面消除问题和碰撞检测的时候就会更简单并且更高效。

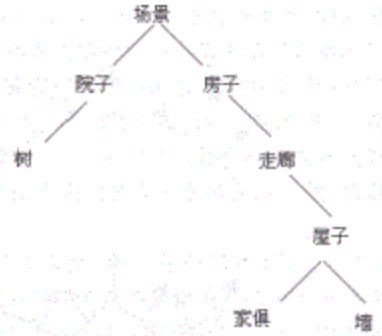


图 17-36　使用层级结构来表达场景

BSP 树

首先我们很自然地想到是否应该用层次结构来表达场景。因为我们曾用层次结构来表达一个复杂的三维物体，比如人体(包括头、四肢、躯干等)。一个场景也可以用这种方式来组织。如图 17-36 所示。

显然上面的层次结构可以帮助我们判断物体和物体之间的关系。比如说主人公在院子里行走时，要进行碰撞检测。我们可以沿这个场景的层次结构下溯，只需要处理左边的部分，整个右边部分所包含的物体和它们的多边形都不用处理了。这样就把碰撞检测的计算机量大大地降低了。

这种层次结构实际上是对空间的一种划分。也就是把空间分成很多小块，最终每块里只有一个物体或者物体的一部分。在层次结构的每一个结点上，都要包含以下信息：这个结点的空间区域多大，在区域内有哪个(些)物体或者其部分，指向具体物体三维模型数据结构的指针等。这样除了原来每个物体自己的三维模型数据结构外，我们就有了另一个全局的数据结构，即场景的层次数据结构。两者之间通过指针相连。

要形成这样的层次结构，需要进行预处理。也就是说，在把所有的模型放到一块形成场景的时候，要进行计算，得出这么一个层次结构，然后在游戏中进行实时计算时使用。

层次结构中被人们研究得最多的就是树状结构了。目前场景管理用得最多的层次结构是 BSP 树。最早使用 BSP 树的游戏就是大名鼎鼎的《DOOM》，此后业界最有名的三个游戏引擎系列 Quake II、Unreal、和 Lith Tech 都提供了对 BSP 树的广泛支持。

BSP 树

在介绍 BSP 树之前，先介绍一下基本的思想：为什么要用树来组织场景呢?因为使用树结构来表达场景的话，可视性问题和碰撞检测问题都最终归结为树的遍历和下溯，而这些操作都是 O(N)或者 O(NlogN)的，也就是效率比较高。这对游戏的实时画面处理非常重要。

为了更好地解释 BSP 树，我们先以二维平面的层次结构做例子，因为三维空间过于复杂，难于图示。

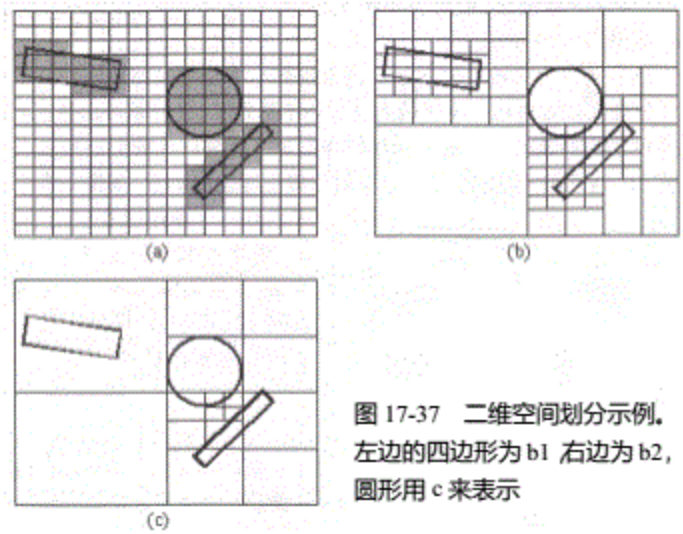


图 17-37　二维空间划分示例。左边的四边形为 b1 右边为 b2，圆形用 c 来表示

假设有一个二维空间里有三个物体。我们要把这个场景做空间上的划分(space subdivision)。最简单的方法就是把这个空间等距地分为更小的单元，如图 17-37(a)所示。每个单元赋予一个标签说明这个单元是空白的，还是有物体占据，是被哪个物体占据。这样如果有一个移动的物体，我们只需要知道目前它在哪个(些)单元内，然后根据这个(些)单元的标签就知道是否有可能和其他物体碰撞了。

显然这种空间划分的方法太过愚蠢，因为很多空间都是空白的，存贮资源和计算资源都被白白浪费了。一个简单的改进是使用四叉树(quadtree)。每次把空间分成四个单元，如果一个单元为空白或者达到最小单元，则停止；否则继续划分。其结果如图 17-37(b)所示。比(a)已经简化了很多。而且已经可以形成树结构了。每次划分四个单元，都是附加四个子树，从第 3 象限起，依次为子树 1,2,3,4。如图 17-38 所示。

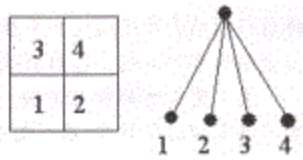


图 17-38 四叉树的形成

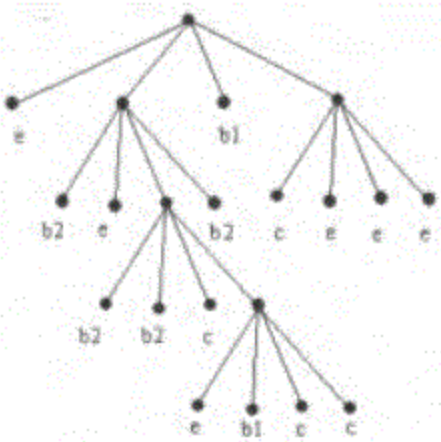


图 17-39 使用方法(c)生成的四叉树。e 代表空白的单元，b1 代表左边四边形的部分在这个单元内，b2 代表右边四边形的部分在单元内，c 代表圆形的一部分在单元内

(b)的方法有改进的余地。因为我们不需要每次都划分到最小单元。如果我们知道在一个区域内只有一个物体了，我们就可以停止。因为要做碰撞检测的话，如果运动物体在这个区域以外我们不用进一步计算了。采用这种思路，我们可以对二维空间划分如下：如果一个单元中只有一个物体或者空白，则停止；如果有两个以上物体，则继续划分，直到最小单元。其效果如图 17-37 中(c)所示。其对应的四叉树如图 17-39 所示。

使用方法(c)生成的四叉树比方法(b)结点少多了。但还有问题——其高度太大，不平衡(有的地方一级就结束了，有的地方深达四级。)由于树的遍历和下溯所需的计算时间由树的高度决定，树的高度越高，计算时间越慢，效率越差。这样窄而高的树显然不利于计算。为了使得树的高度降低，树形更平衡，BSP 树就被提出来了。

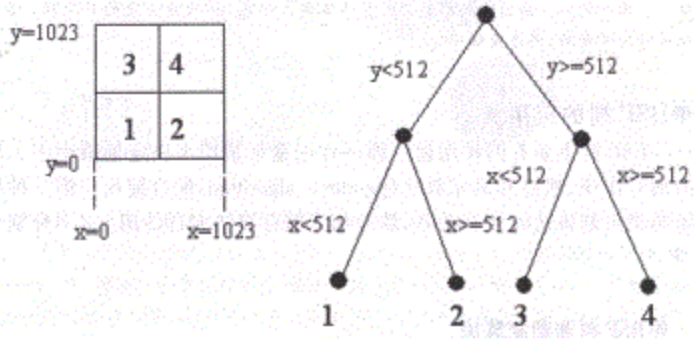


图 17-40 假定我们有一个正方形的二维空间，长宽都为 1024。第一次用 y=512 这条直线划分，把空间分为上下两块。然后对每一块再分别用 x=512 这条直线划分，划分出四个单元。这样就生成了一个 BSP 树

BSP 树(空间二分树，binary partitioning tree)每次把空间划分为两部分。在三维空间内，使用一个分割平面把三维空间划分成两块。在二维空间内，是用

一条分割线把空间分开。BSP 树的每一个非树叶结点代表一条分割直线或者平面。其每个树叶代表最小的空间单元，并有指针指向在这个单元内的三维物体(三维物体本身的几何模型的数据结构是单独存放的，通过指针和 BSP 树连接起来)。一个简单的二维示意图如图 17-40 所示。

仅从上面图例，还不能看出 BSP 树的优势。BSP 树的优势主要要靠两点体现出来。第一，分割平面和分割线不一定要横平竖直，可以是沿各种方向的。这样的话，就可以利用场景里三维物体本身的多边形所在的平面来进行分割。这方面的优势将在我们后面处理多边形时体现出来。第二，每次分割不一定要分成两个相等大小的块。因为物体在三维场景里的分布是不均匀的，肯定某些地方拥挤，某些地方空白。根据疏密程度进行分割，就可以使得 BSP 树的高度变矮，树形更平衡。

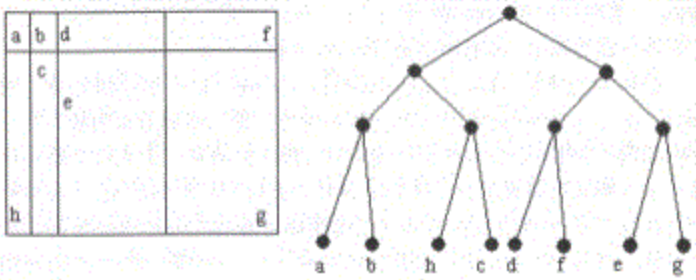


图 17-41　二堆平面上有 a,b,c,d,e,f,g,h 八个物体。呈不均匀分布。使用空间二分法，每次分割时保证分割出来的两块中所含物体数目相同。则生成右边的 BSP 树。这个树是均匀平衡的，高度最矮

BSP 树的应用

BSP 树主要有两种用途：第一种用途是被用来决定场景中多边形的前后次序，然后和画家算法(painter's algorithm)配合使用。第二种用途是决定物体之间相互关系，然后和 Z 缓存算法配合使用。以下分别介绍。

●BSP 树和画家算法

首先我们来看第一种用途。假定我们有一个游戏场景，其中有许多物体，每个物体由许多多边形构成。然后我们在场景中的某个位置安置了一个镜头(又称视点，view point)。通过镜头取景，使用投影的方法，我们可以把在三维空间内的每一个多边形都投影到二维视平面上。在屏幕上只要画出这些多边形，游戏场景就出现在玩家眼前了。但这就带来一个问题：多边形中有的在前，有的在后，有的相互遮掩。如何画呢?画家算法的基本思路就是把场景中所有多边形按和镜头的距离排序，然后由远到近一一画出。这样在前面的多边形后画，就可以遮住在后面的多边形，最后显示的场景是正确的场景。

这个排序的任务，就可以使用 BSP 树来完成。其基本方法是这样的。选定场景中一个多边形所在的平面作为分割平面，场景中所有其他多边形都要和这个分割平面做比较，看是在分割平面的哪一边，由此决定其在 BSP 树中的哪一个子树。如果一个多边形和这个分割

平面相交，则沿分割平面把多边形一分为二，分别置于两个子树中。重复以上操作直到每个单元中只有一个多边形(形成 BSP 树的树叶)。一个简单的图例如图 17-42 所示。

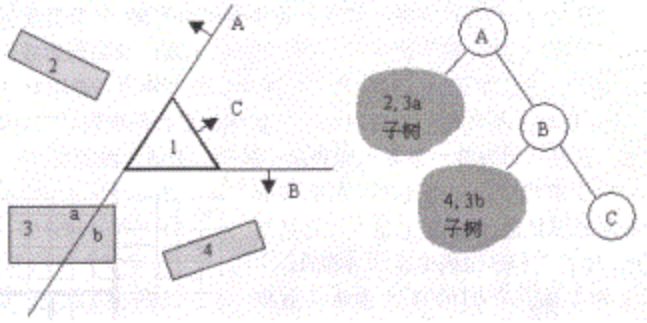


图 17-42　BSP 树示意图：三维空间中有物体 1,2,3,4。取物体 1 上的多边形为分割平面，首先用 A 平面进行分割，箭头表示法矢量方向。物体 2 在 A 的正面一侧，物体 4 在背面一侧。物体 3 和平面 A 相交，被分解成 3a 和 3b，分别置于 A 的两个子树中。然后再用 B 平面进行分割。就这么不停地分割下去，直到形成 BSP 树

有了 BSP 树，再加上镜头的位置，我们就可以对所有多边形进行排序了。具体方法如下：

1. 根据镜头的坐标数据，沿 B5P 树下溯；
2. 在每一个结点，判断镜头是在分割平面的前面还是背面；
3. 下溯距离镜头远的子树，并输出多边形；
4. 下溯距离镜头近的子树，并输出多边形。

这样得到的多边形序列，是按距离镜头由远到近的顺序排列的。下一步就可以使用画家算法了。

●BSP 树和地形

BSP 树最重要的用途还是在场景管理方面，也就是和 Z 缓存算法结合起来使用。也是分为两步，第一步生成一个 BSP 树，然后使用 Z 缓存算法。但这里生成 BSP 树和在使用画家算法前生成的 BSP 树不一样，画家算法借助 BSP 树来对所有多边形进行排序。而现在我们只是要用 BSP 树来表达场景内的大的地形信息，不用划分到多边形的级别。

具体操作如下：首先根据具体的场景生成 BSP 树，只需要细化到物体级，不一定要细化到多边形级。这样 BSP 树的树叶就是物体或者物体的一部分。然后根据镜头的位置，迅速决定哪些物体是在可视空间之外，将它们略去不处理。然后对剩下的物体的多边形使用 Z 缓存算法。这种方法是目前游戏中最常用的管理场景和渲染场景的方法。

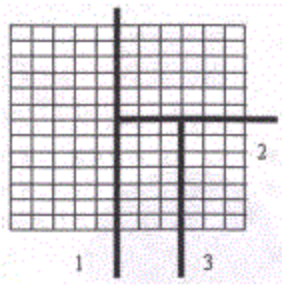


图 17-43　基于方格的场景，可以使用横平竖直的分割平面

对于游戏中的三维场景，虽然表面上看起来很复杂，但总还是基于一定的规则的。比如说 FPS 游戏中迷宫内部的许多屋

子和走廊，它们的规划是基于方格(grid)的，并且其墙壁和入口都是垂直于地面的，这样我们可以采用横平竖直的分割平面。其二维示意图如图 17-43 所示。

对于迷宫内部的场景管理，有一个很棘手的问题，就是门户(portal)问题。所谓门户，就是指墙上的门和窗，透过它们图 17-43　基于方格的场景，可以使用横平竖直的分割平面可以看到其他房间的内部或者外景的一部分。门户问题为什么棘手呢?因为如果是一个完全封闭的屋子的话，我们只需要用二个 BSP 树表达这个屋子和屋子里面的物体就行了。但现在我们有很多屋子，通过门户，各个屋子之间可以相互看到。这样 BSP 树就必须包括所有的屋子和它们里面的物体才行。但在一个屋子里，通过门和窗口所能看到的其他屋子里的物体实际上是十分有限的。为了这么几个物体，而把其他屋子里所有的物体都放到 BSP 树里，确实是极为浪费资源，也难于计算。

为了解决门户问题，人们提出了专门的方法。如图 17-44 所示。

图⑴显示了 5 间屋子，之间有门相连。图(2)中的阴影部分是在屋子 A 内(可以在 A 内任何地方)所可能看到的所有区域。图(3)显示了根据图(1)所得到的区域，对各个屋子里的物体进行判断。将不可见的物体忽略(白色物体)。经过这三步预处理，场景已经大大简化。我们就可以使用一般的算法进行场景的实时渲染了。

BSP 树除了被使用在场景管理中外，在碰撞检测方面也很重要，将在下面一节介绍。

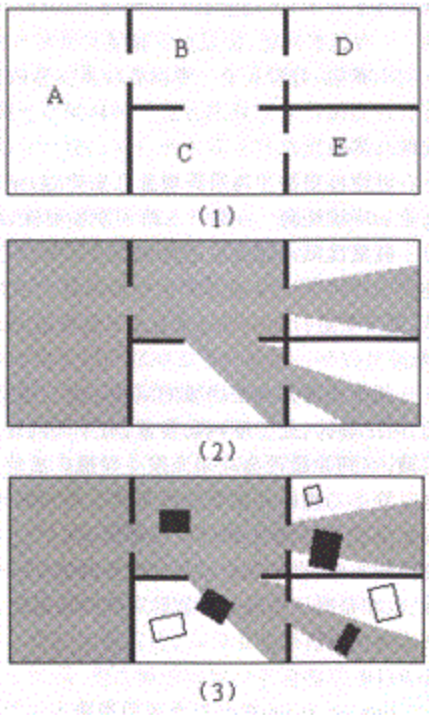


图 17-44 门户问题

碰撞检测

在这一节要介绍碰撞检测，基本的思路和方法是根据 Nick Bobick 在《Game Developer》上发表的两篇很流行的文章。碰撞检测是目前三游戏中最重要的课题之一。在很多游戏中我们看到如下情况的发生：NPC 挥舞着大刀，一不小心大刀就插入了墙壁中，但却丝毫没有任何反应，仿佛插入的是空气；主角走着走着，明明离一棵大树还有一段距离就是过不去了；飞机勇敢无畏地撞上山口，然后又没事地穿越后继续飞翔。所有这些问题，都是由于碰撞检测处理得不够精确。随着玩家的口味越来越高，对游戏中三维世界的真实感的要求也越来越挑剔。如果没有好的碰撞检测的处理方法，种种细节上的错误必然会极大地影响游戏的总体效果。

碰撞检测是和场景管理息息相关的。有了高效的场景管理，才能有高效的碰撞检测。在

设计三维引擎的时候，两者要统筹考虑。

碰撞检测问题的复杂度要取决于场景中的活动物体的总数。假设场景中物体数目为 n，如果使用最笨的方法对所有物体逐个进行比较的话，则需要进行 n(n-1)/2 次比较。也就是说这种算法是 O(n2)，是很低效的。

碰撞检测是一个四维的问题，即三维空间加上时间。在三维空间内，两个物体，处于不同的位置，有不同的速度和加速度，沿一定的轨迹运动，要判断是否会在未来发生碰撞。显然这样的计算太复杂了，因为这时静态的检测问题就变成了动态的预测问题。三维空间本来就已经很复杂了，再牵扯上时间就更麻烦。虽然在大学等研究机构里有人搞这方面的工作，也开发出相应的各种算法。但对于注重实时反应的游戏来说，实在是难于实现。于是程序员们就采取了简化的策略。这种策略如下面的伪码所示：

```
while(1){
    process_input();     ??处理用户输入
    update_objects();    ??更新物体位置和状态
    render_world();      ??渲染游戏场景
}
update_objects(){
    for(each_object)
    save_old_position();     ??保存物体现在所处的位置
    calc new_object_position       ??根据速度、加速度等信息，计算物体新的位置
    if(collide with_other_objects())     ??判断是否在新的位置上和其他物体碰撞
    new_oject_position = old_position();   ??更新物体的位置
```

这段伪码的第一部分是一个极度简化的游戏程序的主循环，第二部分是更新物体位置和处理碰撞检测的具体操作。它回避了物体移动轨迹的问题，只计算下一步物体的位置和近在眼前的可能的碰撞。如果有这种碰撞的可能的话，物体就停在原来位置。也就是说，它是离散的算法，而不是连续的算法。因为物体的下一个位置决定于最小的步距，也就是对空间的离散采样。而且它只考虑在下一个时刻的物体位置，也就是说把时间固定了。使得碰撞检测问题减少了时间这一维的复杂性。

这种离散的算法得到了很广泛的使用，但它也有很多问题。它的问题就在它的离散性上。因为不是连续地考虑时间和空间，有可能出现以下的情况。

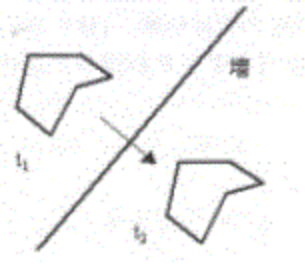


图 17-45 离散算法带来的问题

图 17-45 中所示物体在 t_1 的时候位于墙的左侧，根据其速度和加速度等数据，我们可以计算出在下一个时间 t：它将位

于墙壁右边。如果我们不考虑 $t_1<t<t_2$ 的连续时间内的情况，只考虑 t_2 的话，则在新的位置没有碰撞发生。我们更新物体的位置，物体穿墙而过。显然这是不正确的。

为了解决这个问题，最简单的想法就是把时间间隔定得小些，把步距定得小些。也可以从游戏场景和物体的三维模型入手。然而要完全杜绝离散算法的问题，需要引入连续的时间，一个思路就是把连续的时间转化为连续的空间。让物体从起始位置开始，到终止位置，移动的轨迹作为基准，所扫过的空间形成一个新的物体。然后比较这个新的物体和其他物体是否有可能碰撞，使用这种方法，则肯定可以发现物体和墙的碰状。如图 17-46 所示。

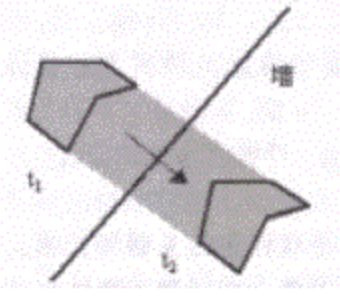


图 17-46 把时间转换为空间来进行碰撞检测

当然上面介绍的方法比较复杂。特别是有很多物体同时运动的时候，必须计算生成许多新的物体，这些物体的外形也不规则。在后面将要介绍的包容体等概念，将有助于提高这种方法的效率。通过上面的介绍，我们看到游戏中的碰撞检测都尽量回避了时间的问题，只是对一个固定时间下场景内所有物体的静止位置来进行判断。我们可以称之为静态碰撞检测问题。静态的碰撞检测主要有两种方法。一种叫两步法(broad phase / narrow phase)，一种叫一步法(single phase)。所谓两步法，就是分两步来检测可能的碰撞。第一步先处理物体与物体之间的关系，把根本不可能碰撞的物体剔除，然后第二步进行细致的碰撞检测，具体到多边形或者点和面的层次。而一步法是指通过使用 BSP 树进行空间和物体的划分，一步到位进行碰撞检测。

下面分别介绍两种方法。

两步法

●包容体的概念(Dounding Volumes)

碰撞检测的一个最基本的概念就是包容体(bounding volume)。所谓包容体，是指用一种规则的形状简单的“包容体”把所有物体都“包裹”起来。因为游戏中的物体千差万别，形态各异，要检测它们之间的关系实在是太困难了。而把它们用包容体“装”起来后，包容体和包容体之间是同样的形状，检测起来比较容易，是否碰撞比较容易判断。如果包容体之间没有碰撞的可能，则物体本身不可能能碰撞。这样就可以省略很多需要考虑的物体，大大地提高了碰撞检测的效率。

图 17-47 三种觉的包容体。阴影为物体

几种常见的包容体如图 17-47 所示。

●包容球

环绕球是最常见的一种包容体。它使用一个球体把三维物体包裹起来。然后检测球体和球体之间的碰撞。显然这种方法简单但不精确。有可能球体碰撞了但物体本身没有碰撞，如图 17-48 所示。

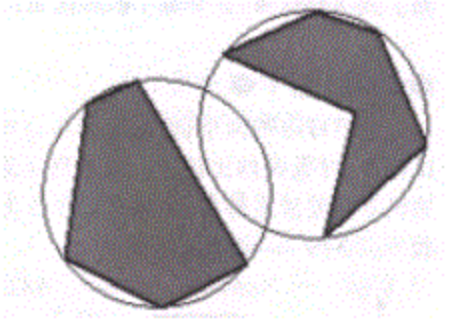


图 17-48 包容球相互碰撞但物体没有碰撞

为了解决包容球精确度不高的问题，人们又提出了球体树的方法。

球体树实际上是一种表达三维物体的层次结构。对一个形状复杂的三维物体，先用一个大球体包容整个物体，然后对物体的各个主要部分用小一点的球体来表示，然后对更小的细节用更小的包容球体。这些球体和它们之间的层次关系就形成了一个球体树。举例来说，对一个游戏中的人物角色，可以用一个大球来表示整个人，然后用中等大小的球体来表示四肢和躯干。然后用更小的球体来表示手脚等。这样在对两个物体进行碰撞检测的时候，先比较两个最大的球体。如果有重叠，则沿树结构下溯，对小一点的球体进行比较。直到没有任何球体重叠，或者到了最小的球体，这个最小的球体所包含的部分就是碰撞的部分。

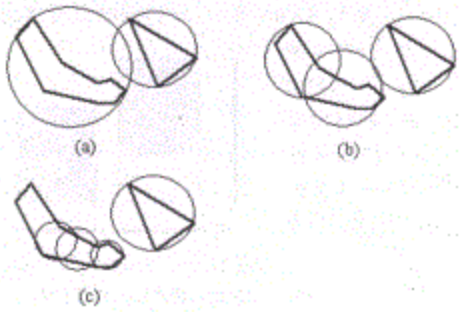


图 17-49 使用球体树对两个物体进行碰撞检测。图(a)中两个最大的球体有重叠，需要比较下一个层次的球体。图(b)中有两个小一点的球体，其中一个有重叠，还需要细分。图(c)中三个最小的球体，没有重叠发生。至此判定两个物体没有碰撞

AABB 盒子

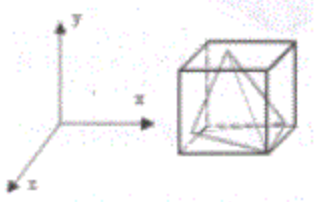


图 17-50 AABB 盒子。图中灰色为物体，黑色立方体为包容它的 AABB 盒子

包容球虽然简单，但用球体来包容一个三维物体，其内部空间浪费比较多，导致误判比较多。而使用立方体(盒子)包含三维物体，在很多情况下都要紧凑得多。有两种基于立方体的算法，即 AABB 盒子和 OBB 盒子。下面一一介绍。

AABB 盒子，即轴对齐包容盒子(axis-aligned bounding box，缩写为 AABB)是使用立方体把三维物体包裹起来。这个立方体的面和坐标系的轴垂直，因此是一个横平竖直摆放的盒子。

使用 AABB 盒子，我们也可以为复杂的三维物体构建 AABB 树。然后用前面介绍过的球体树的方法来进行碰撞检测。所不同的是，AABB 盒子之间的碰撞检测更容易。因为它是

一种降维的计算。其基本思想如图 17-51 所示。

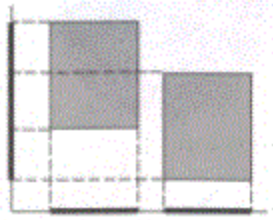


图 17-51 AABB 盒子之间的碰撞检测是一种降维计算

图 17-51 显示的是二维的情况。对两个二维 AABB 盒子，我们只需要进行两个一维的比较(降维)，看它们在坐标轴上的投影是否有重叠。如果在两个坐标轴上都有重叠，则盒子发生碰撞；若没有重叠或者只在一个坐标轴上有重叠，则没有碰撞发生。

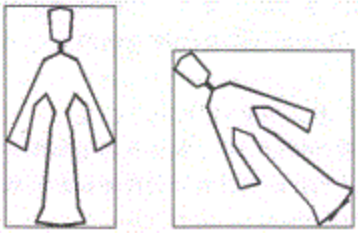


图 17-52 当物体倾斜时，新的 AABB 盒子体积变大，空白增多，导致碰撞检测精确度下降

AABB 盒子虽然简单，但有两个缺点。第一，当物体旋转时，需要重新计算 AABB 盒子。因为物体旋转后，如果直接把旋转矩阵变换应用到 AABB 盒子上，AABB 盒子就不是和坐标轴垂直的了。这时候必须根据物体的新的位置，重新计算出包容它的 AABB 盒子。第二，当物体倾斜时，AABB 盒子马上变大，内部无用空间增多，碰撞检测的精确度也随之下降。如图 17-52 所示。为了解决这两个问题，就引入了 OBB 盒子的概念。

OBB 盒子

OBB 盒子，即定向包容盒子（oriented bounding box，缩写为 OBB）。这种方法是根据物体本身的几何形状来决定盒子的大小和方向，盒子无需和坐标轴垂直。这样就可以选择最合适的最紧凑的包容盒子。

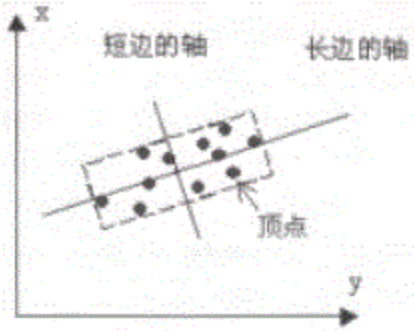


图 17-53 OBB 盒子生成的示意图

OBB 盒子的生成比较复杂。一般是考虑物体所有的顶点在空间的分布，通过一定的算法找到最好的方向(OBB 盒子的几个轴)。一个二维示意图如图 17-53 所示。

同球体树和 AABB 树一样，我们也可以用 OBB 树来进行碰撞检测。构建 OBB 树的示意图如图 17-54 所示。

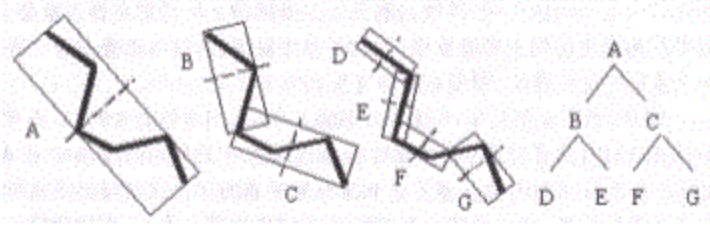


图 17-54 OBB 树的生成（粗线为物体）

使用 OBB 树进行碰撞检测，按如下步骤进行：两个三维物体，各有自己的 OBB 树。先对比两个树的根部，看是否有碰撞。如果有，则沿树下溯，直到没有碰撞或者到达树叶。

其间每一步都是两个 OBB 盒子之间的碰撞检测。

两个 OBB 盒子之间的碰撞检测，看似复杂，但实际上还是比较容易的。还是要用投影轴进行降维计算，但由于 OBB 盒子不和坐标轴垂直，无法使用坐标轴，我们必须找到其他合适的轴来进行投影。我们就仔细考察一下如何得到这些合适的投影轴。

首先，一个最直观的想法是把一个 OBB 盒子的每个边都和另一个盒子的所有面来比较，如果这个边穿过了另一个 OBB 盒子的一个面，则两个 OBB 盒子发生了碰撞。显然这种方法是很笨的。因为要进行 12×6×2=144 次边和面的比较。考察两个没有碰撞的 OBB 盒子，我们可以发现一些规律，来简化比较。我们发现：如果两个 OBB 盒子不互相接触，则我们或者(1)可以找到一个盒子上的一个面，这个面所在的平面可以把三维空间分为两部分，两个 OBB 盒子各在两边。(2)如果没有这样的表面存在，则一定可以在两个 OBB 盒子上各找出一条边，这两条边所在的平面可以把两个 OBB 盒子分在两边。有了这个平面，就可以找到垂直于它的分割轴(separating axis)。两个 OBB 盒子在这个分割轴上的投影，将是分离的。

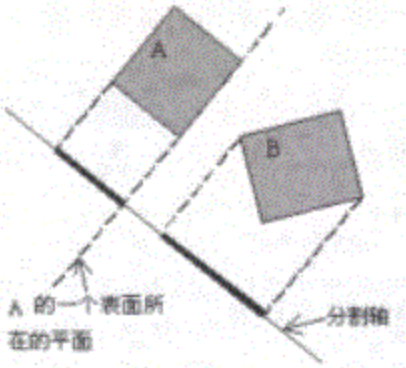


图 17-55 两个 OBB 盒子，和它们在分割轴上的投影

因此要判断是否两个 OBB 盒子碰撞，只需要看两个 OBB 盒子之间是否有这样的平面和分割轴存在。如果存在，则没有碰撞。如果不存在，则碰撞。对第一种情况，每个盒子有 6 个表面(其中每两个平行)，可以决定 3 个分割轴。两个 OBB 盒子一共有 6 个可能的分割轴需要考虑。对第二种情况，两个 OBB 盒子之间的边的组合可以有 3×3=9 种情况，也就是有 9 个可能的分割轴。这样对任意两个 OBB 盒子，我们只需要考察 15 个分割轴就可以了。如果在任一分割轴上的阴影不重合，则 OBB 盒子之间没有碰撞。

前面我们介绍了 3 种常用的包容体和它们之间碰撞检测的方法。包容体的碰撞检测是两步法中的第一步。当沿包容体树的进行比较，最终达到树叶的时候，就要进行第二步更细致的判断了。这包括对物体的实际几何形状的考察，对多边形和直线之间关系的判断，最终精确判定物体之间，而不是包容体之间，是否碰撞。这是在大学等研究机构里的方法。在游戏业界的实际操作中，这第二步是经常被省略的。原因是算法比较复杂，实时计算起来成本太高。因此游戏程序员们就为了实时性能，而牺牲了一定的精确度。由于碰撞检测到了树叶(最小包容体)就停止，而不是精确地根据物体的几何形状细节来判定是否碰撞，因此总能在游戏中看到一些小疏漏。当然随着硬件能力的提高，第二步处理会逐渐被重视起来。在此，由于篇幅限制就不介绍了。

BSP 树

一步法主要是使用 BSP 树来进行碰撞检测，不需要额外的包容体。这种方法的优越性在于 BSP 树既比较精确，实时性能又很好。作为一种空间划分和场景管理的方法，如果已经在隐藏面消除中使用(能够在隐藏面消除中使用本身就说明其较好的实时性能)，再用到碰撞检测中可以节省资源。对两个三维物体来说，如果各有其单独的 BSP 树(即所谓的本地 BSP 树，或者物体 BSP 树)。则碰撞检测就变成了两个 BSP 树的合并。其操作如图 17-56 所示。

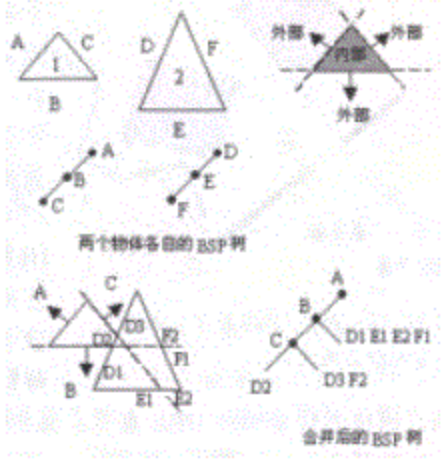


图 17-56 BSP 树的合并

如图 17-56 所示，两个物体 1 和 2，各由三个表面组成，这三个表面就是三个 BSP 分割平面，形成的本地 BSP 树如第二行所示。特别值得注意的是本地 BSP 树中包含了物体内部外部的空间信息。以物体 1 的 BSP 树为例，从树的根部 A 分割平面起，左边子树代表 A 分割平面的向内的方向，即物体 1 的内部。而右边子树则代表 A 分割平面的外部。当我们要考虑两个物体之间的相对关系时，我们以物体 1 的 BSP 树为基础，考察物体 2 的三个表面，看它们是否有部分插入了物体 1 的内部。如果是，则碰撞。首先看分割平面 A，物体 2 在 A 的向内的方向，所以要分到 A 的左边的子树。然后考察分割平面 B，我们发现 D 和 F 被 B 分割平面分割。其中 D1，Z1，E2，F1 位于 B 分割平面向外的部分。而 D2，D3，F2 位于其向内的部分。最后我们考察 C 分割平面，也就是到了物体 1 的 BSP 树的树叶。D2 位于 C 分割平面的向内的部分。这样我们得出结论，D2 位于物体 1 内部，也就是说物体 2 和物体 1 发生了碰撞。

镜头控制和旋转问题

在三维射击游戏和飞行模拟游戏中，镜头要随着玩家的操纵而不断变化，如何简单有效地定义并控制镜头，对三维游戏的总体效果起到很大作用。而除了镜头的拉进拉远外，最重要的效果就是镜头的旋转了。因此我们把在三维世界中物体的旋转和镜头控制放在一个小节里介绍。并且除了镜头的旋转之外，还有游戏中其他物体的旋转，比如说一个活动门的开和关。它们的数学基础是一样的。

我们知道每个三维模型在创建的时候都有自己的坐标系，叫做本地空间(local space)或者物体空间(object space)。当我们要把这些三维模型统一地摆放到游戏世界中时，我们要有一个世界坐标系，或者说世界空间(world space)。镜头的位置和方向是定义在世界坐标系里面的。在投影的时候，我们先要用一个矩阵把三维模型从本地空间转换到世界空间，这个矩阵就叫做 local-to-world matrix(本地空间到世界空间转换矩阵)。然后再根据镜头的位置和方向，形成另一个矩阵，叫做 world-to-camera matrix(世界空间到镜头转换矩阵)。用这个矩阵把世

界坐标系转换到镜头坐标系上。然后我们就可以完成投影等一系列转换了。

因此镜头的旋转，就体现在这个 world-to-camera matnx 上。而物体在三维空间内的旋转，则是体现在 local-to-world matrix 上。旋转的效果，就是通过改变这两个矩阵来实现的。但这种矩阵运算有很大的问题。因为这种矩阵实际是把好多种变换，包括平移、旋转、缩放都综合在一起了。这样的话难于理解，并且难于直接表达和具体地控制镜头的变换。而更麻烦的是，如果我们想通过插值法来计算位置的话，计算会非常复杂。一个简单的例子就是活动门，我们可以预先设定好起始两个位置，然后在游戏中让活动门自己一开一关。这就需要计算出中间的位置，也就是要进行插值计算。但对矩阵进行插值实在是太困难了！

为了解决这些问题，我们下面就介绍几种方法，包括 euler 角和四元数法(quaternions)。

Euler 角

在三维空间里的方向性(orientation)可以用三个 euler 角来表达，分别为：偏向角(yaw)、前倾角(pitch)、和翻滚角(roll)。它们代表了围绕 X，Y，Z 三个轴的旋转，示意图见图 17-57。

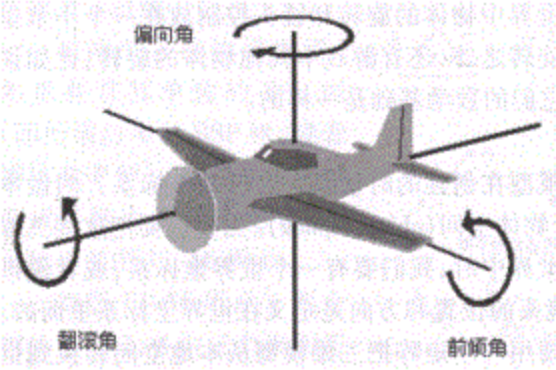


图 17-57 euler 角

使用 euler 角来表达旋转显然比矩阵要直观得多。正因为如此 euler 角的应用还是比较普遍的。但 euler 角的问题也还是很多，首先，euler 角很难表达一个复杂的复合旋转。其次，euler 角在插值方面的效果不是很好，会出现很生硬的效果。而更尴尬的一个问题就是所谓的“万向节锁定”(gimbal lock)。比如说我们先绕 X 轴转一个 α 角，然后绕 Y 轴转 90 度，然后绕 Z 轴转 β 角。这时我们就会发现，最后的效果和直接在 X 轴转 β 角的效果一样。因为绕 Y 轴的旋转使得 x 轴和 Z 轴重合了。这样我们实际上丢失了一维自由度。

在第一视角游戏中，玩家所见的就是游戏主角所见，也就是说镜头的位置就是主角的位置。玩家看不到主角的全身，特别是背影。从最早的《DOOM》到后来的《Quake》和《Unreal》系列都是第一视角的游戏的典型例子。第一视角的游戏中，目前比较通行的做法是使用鼠标来控制镜头。也就是说，三个 euler 角中，偏向角对应鼠标的 X 坐标，前倾角对应 Y 坐标。这样移动鼠标的话，镜头就有了二维自由度。翻滚角在游戏中使用比较少，因为在地上走的生物很少有这种能力。在飞行模拟游戏中，战斗机才有在空中翻滚的可能。

另外如果处理不好的话，euler 角的万向节锁问题定会在第一视角游戏中出现令人困惑的效果。比如我们在 x 方向移动鼠标使得主角的头转动 90 度，然后沿 Y 方向移动鼠标，这

时我们希望看到的是前倾角的变化，也就是抬头或者低头，但由于万向节锁定的问题，我们将要看到的是翻滚。也就是歪头的效果。当然这是我们不愿看到的。

四元数法

图 17-58 《Unreal》是标准的第一视角游戏

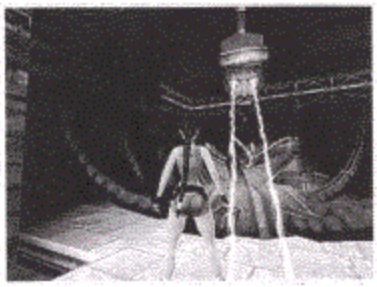


图 17-59《古墓丽影》是典型的第三视角游戏。其中所有的镜头控制都是基于四元数法（Quaternions）的

在第三视角的游戏中，镜头一般取在主角身后或侧后，这样玩家可以看到主角全身。镜头随着主角移动，并保持和主角的一定距离。《古墓丽影》《马里奥》等都是第三视角游戏。对于第三视角类型的游戏，镜头的控制稍微复杂一点。

四元数法（Quaternions）是第三视角游戏中使用比较多的旋转 / 镜头控制方法。它比矩阵和 euler 角都优越，而且它的应用不止是在镜头方面，在骨架动画(skeletal animation)，逆运动学(inverse kinematics)和物理编程等方面都很有用处，是三维图形程序员的一项利器。四元数有自成体系的运算规则，在这里我们只针对它在三维旋转中的应用简单介绍一下。

图 17-60 四元数示意图

顾名思义，四元数法就是用四个参数来代表旋转，一般写做(x,y,z,w)。假设我们在三维空间里，任意取一个轴 A，其矢量形式为(XA,YA,ZA)并绕它旋转 θ 度。如图 17-60 所示。

表达这个旋转的四元数就是：

$Q=(x,y,z,w)=(sX_A,sY_A,sZ_A,c)$

其中：

$$s=\sin(\theta/2)$$

$$c=\cos(\theta/2)$$

四元数法的优越性在于以下两点。第一是避免了矩阵相乘。比如说我们通过两个矩阵相乘达成一定的旋转效果，可以用相应的四元数相乘，则只需要进行 8 次乘法和 4 次除法，比 3×3 或 4×4 的两个矩阵相乘简单快速多了。

第二个优势，也是四元数法的真正威力所在，是使用四元数法易于插值，并且生成的插值效果好，可以形成更平滑柔和的动画效果。这一点无论是对镜头的控制，还是对游戏世界中一般物体的旋转，乃至物理编程等功能等，都是至关重要的。假设我们有两个四元数 p

和 q，要在它们之间插值。方法有很多种，其中比较常用的是 SLERP 插值(spherical linear interpolation，球面线性插值)，其公式如下：

$$SLERP(p,q,t)=\frac{p\sin((1-t)\theta)+q\sin(t\theta)}{\sin\theta}$$

其中：0≤t≤1

θ 是 p 和 q 之间的角度。其计算方法是使用 p 和 q 的点乘来计算其 cos 值，即 p●q=cos θ

这样找到的插值形成了一个在球面上的弧线。

四元数法在第三视角游戏中一般是这么应用的：镜头位于主角身后一定距离处，并且一般是高于主角头部，指向主角正对的方向。当主角在玩家的操纵下改变其面对方向(转身或者摇头)时，我们马上使用 SLERP 插值计算出镜头变换的轨迹，然后移动镜头并改变镜头的方向这样的效果是，既实现了镜头对主角的跟踪，又使得这种跟踪非常平滑和自然。

第十八章　人工智能技术

人工智能定义的双重标准

人工智能(artificial intelligence，缩写为 AI)是近几年来游戏业最炙手可热的研究领域，也最具争议性。争论的焦点就是很多人置疑游戏中使用的各种技术是否真的属于人工智能范畴。虽然现在很多游戏都把这个时髦名词当作卖点，但游戏中实际使用的技术也许并不是那么高妙和玄秘，有些实际上还是很“过时”的技术。美国各大学实验室中所做的 AI 研究和游戏业的实际应用之间相差十万八千里。刚毕业的大学生们会大吃一惊地发现他们在学校学的很多 AI 算法和研究思想在游戏业中根本找不着。学术意义上的 AI 和游戏业所谓的 AI 之间差别巨大。

那么什么是人工智能呢?各种各样的计算机教科书中对其有各种定义。透过那些字斟句酌的严谨圆滑的背后，我们看到它们可被分为四类(或其重叠)。

四类 AI 定义	人性	抽象的理性
思考	通过计算机系统模拟人类的思考能力	使计算机系统能够进行独立的理性的思考
行为	通过计算机系统模拟人类的行为能力	使计算机系统拥有独立的理性的行为能力

表 18-1 本表译自 Stuart J. Russell 和 Peter Norvig 所著的《Artificial Intelligence：A Modern Approach》一书

这四类定义实际上也预示了 AI 的四种不同的研究方向。由于研究侧重不同，由此又形成两大派别：人性派和理性派。其区别就是人性派更注重从研究人类自身出发。因为你要教会计算机像人一样思考，你先得搞明白人类是怎么思考的。而理性派则更注重抽象的逻辑和数学表达。四个研究方向的具体情况如下：

模拟人的思考能力：我们前面已经说过，要模拟人的思考，先得明白人类思考的内部机理。这一领域的主要研究包括人类视觉、自然语言、人脑记忆等。这一领域的发展标志，就是把构建计算机软件模型和传统心理学的动物人体实验相结合，形成了一门新的交叉学科，叫认知心理学(cognitive science)。它进一步形成了近年来蓬勃发展的另一个交叉学科——人机交互学的基础。

模拟人的行为能力：主要是指计算机能够和人正常而自然地交流，包括让计算机能够看到它的用户，能够听到用户的声音，能够理解人类的语言，并作出适当的反应，用语言或行动表达自己的反应。这一领域的主要研究包括计算机和机器人视觉、自然语言处理(包括识别人类语言语音，对其内容进行解释、存储和推理，获得结论，然后自动生成语音输出等一连串技术)、机器人技术等。

理性思考：我们知道人是一种理性的动物，但人并不完全按理性行事。因此，真正意义上的完全理性的思考，只存在于数学家的写满公式的纸上。这一领域的代表是逻辑系统，是从古希腊亚里士多德的学说发展而来的，像大家所熟悉的一阶逻辑多阶逻辑等。

理性行为：所谓计算机系统拥有独立的理性的行为能力，是指它可以有一定的目的，根据外部情况，决定所能所采取的行动。简言之，就是一个能自行其是的系统。最简单的例子，就是目前在网络方面最火爆的智能化代理技术(intelligent agents)。

在了解了 AI 在学术上的定义之后，我们在来看看游戏业所谓的 AI 的定义。在早期的游戏中是没有 AI 的说法的，在那些任天堂 8 位机上的横卷轴 ACT 游戏中，小妖们是固定在某个地点出现的，只要多玩几次就能够预先知道它们将在什么地方出现，何时出现。游戏没有任何 AI 可言，而 AI 技术真正发扬光大并获得普遍重视，则是在 FPS 和 RTS 这两种游戏类型出现以后。三维射击游戏使 NPC 的概念开始盛行。NPC 是具有一定自主活动能力，能够评估周围环境和敌我态势，并做出最优判断，不由玩家控制的游戏角色。显然这项定义和 AI 学术定义的第四条沾上边了。RTS 中，在战略层次，由计算机控制的敌方要完成资源创建和管理、生产协调、部队的集结调动等等复杂任务，这基本上是个最优化问题；而在战役战术层次，敌方部队又要完成寻径、队形组织、分配个人任务等任务。旧的编程方法显然无法胜任这些纷繁复杂的任务，程序员们像抓住救命稻草一样抓住了 AI 技术，希望 AI 能帮助他们解决问题。而 AI 也果然不负众望，出色地解决了 RTS 的一些基本问题。但 RTS 中的 AI 实际也只是局限在 AI 学术定义的第四条。

另外，学术上对 AI 的研究，注重的是内部机制。因为学术研究的目的是找出事物内部运行机制，不断地改进算法，使得内部结构更合理。而游戏业对 AI 的应用，则更重外部表象。如果一个新技术，从内部看技术前卫十分先进，让程序员们觉得自己很酷，但玩家在实际游戏中感受不到它和旧技术的区别，那这项技术对游戏就是毫无用处的。游戏业 AI 的指导思想，就是用最简单的方法，占用最少的资源，去愚弄玩家，造成假象，让他们觉得游戏 AI 水平高超。戏言之，就是能骗就骗。

从上面的分析我们可以看出，AI 在游戏界的实际应用和在学术上的研究有很大不同。游戏 AI 基本是在 AI 学术研究的第四条，即理性行为领域。游戏 AI 采用的技术简单。游戏 AI 没有或者很少触动 AI 学术研究的前三个领域。游戏 AI 目前还没有能力模拟人的思考和行为。

另外对游戏设计师和程序员来说，AI 的意义是完全不一样的。对游戏设计师来说，AI 是游戏规则的最高层——是游戏规则中最具有挑战性的，也是最模糊的部分。对程序员来说，AI 是对游戏设计师制定的复杂游戏规则的技术实现。本章中主要介绍的是后者，即从技术实现角度看 AI。

人工智能在游戏业的应用现状

谈到 AI 在游戏业的应用现状，我们一般都是谈美国游戏业的 AI 应用现状。有两方面原因：一方面，AI 应用最多的两种类型的游戏 FPS 和 RTS 都是在美国发展起来的，他们对这方面比较有经验，比较有发言权；另一方面，美国游戏业在 AI 技术上公开的交流比较多，比较容易了解业界的情况。

Steven Woodcock 曾连续几年在 GDC 上对业界 AI 技术应用现状进行了调查，他的发现被归纳于表 18-2 中。

	1997	1998	1999	2000
有专门负责 AI 的程序员的小组在业界的百分比	24%	46%	60%	80%
用于 AI 的 CPU 资源	5%	10%	10%	25%

表 18-2 AI 技术调查

从表 18-2 中我们可以看出，近几年游戏 AI 的发展确实迅猛。在 1997 年还只有 24%的制作组里有专职的 AI 程序员，而到了 2000 年 80%的制作组里都有 1 名以上的专职的 AI 程序员。CPU 资源也在向 AI 迅速开放(传统上这些资源都是给图像处理用的)。在 2000 年初一个专门研究游戏 AI 的教授在卡内基美隆大学演讲时还提到游戏中 AI 一般只占 CPU 资源的 15%，而 2001 年的报告中就已经提高到 25%了。

另外 Steven 还在他的网站 www.gameai.com 上对哪种游戏类型中更有可能实现 AI 技术的进步和飞跃做了一项调查。图 18-1 所示是统计的结果。

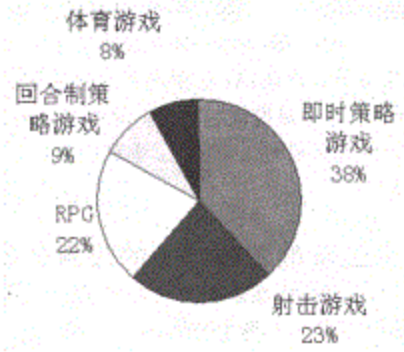


图 18-1 游戏 AI 技术统计

从这个统计结果我们也可以看出目前 AI 技术在不同类型游戏中的不同待遇。RTS 和 FPS 是绝对的主导，也是 AI 技术最早的发源地。RPG 目前也跟上来了，美式 RPG 中新技术应用得比日式 RPG 要多一些。回合制策略游戏和体育游戏，则由于自身的限制，AI 技术应用得不太普遍。

基本 AI 技术介绍

有限状态机

有限状态机(Finite State Machine，缩写为 FSM)是游戏业所使用的最古老的也是最普遍的技术，几乎所有的游戏都或多或少地采用了它。正因为其太简单和太古老了，有些人觉得它不配被归类为 AI 技术，说它更像是一种通用的程序组织形式和思维方法。这种说法也有一定道理。但 FSM 确实高效实用，是一切更高级的 AI 技术的基础。

简单地说，一个 FSM 就是一个拥有一系列可能状态的实体，其中的一个状态是当前状态。这个实体可以接受外部输入，然后根据输入和当前状态来决定下一步该转换到什么目标状态，转换完成后，目标状态就成为了新的当前状态。如此循环往复，实体和外部就这么交互下去，实体的状态就不停地改变着。

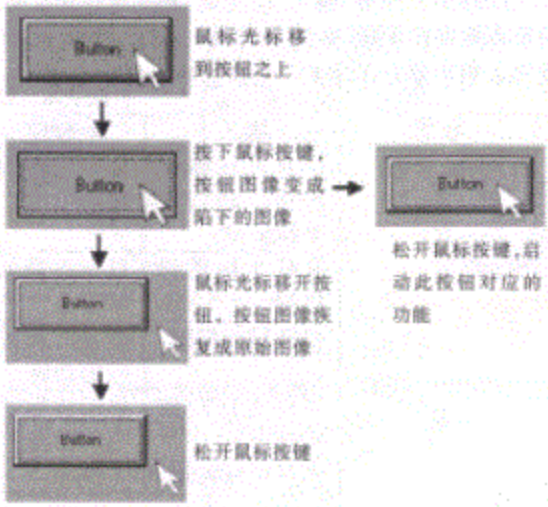


图 18-2 Windows 系统的一个标准按钮的行为

具体到应用上来看，大到整个软件程序，小到屏幕上的一个按钮，都可看成是 FSM 实体。FSM 可以表达它们的行为系统。我们熟悉的 Windows 系统的一个标准按钮的行为如图 18-2 所示（为了避免混淆，我们把屏幕上的 button 称为按钮，而将鼠标器上的实际 button 称为按键）。

与之对应的 FSM 图如图 18-3 所示。

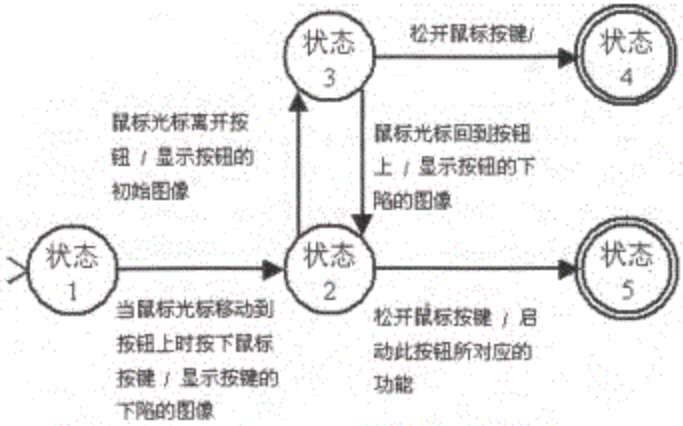


图 18-3 Windows 标准按钮的 FSM 图

FSM 由状态集、输入 / 反应集、和状态转换法则所组成。对 Windows 的标准按钮的 FSM 来说，共有 5 个可能状态，其中状态 1 是初始状态，状态 4 和 5 是终止状态。其状态转换法则如表 18-3 所示。

状态	输入	反应	目标状态
1(初始状态)	鼠标光标移到按钮上时按下鼠标按键	显示按钮的下陷的图像	2
2	鼠标光标离开按钮	显示按键的初始图像	3
2	松开鼠标按键	启动此按钮对应的功能	5
3	鼠标光标回到按钮上	显示按钮的下陷的图像	2
3	松开鼠标按键	无	4

表 18-3 Windows 标准按钮的 FSM 状态转换法则表

在 FSM 图中，圆圈代表状态，有尾巴的圆圈代表初始状态，双圆圈代表终止状态。带有箭头的实线代表状态转换的方向，它从当前状态出发，箭头指向目标状态。线上的标签由两部分组成，在“/”的前面是输入，也就是一定的外部事件。在“ / ”后面的是反应，也就是 FSM 在接受外部输入之后的行为。

那么 FSM 是如何应用到游戏中的呢?一个最简单的例子就是三维射击游戏中的 NPC。我们可以把一个 NPC 看成是一个 FSM 实体。

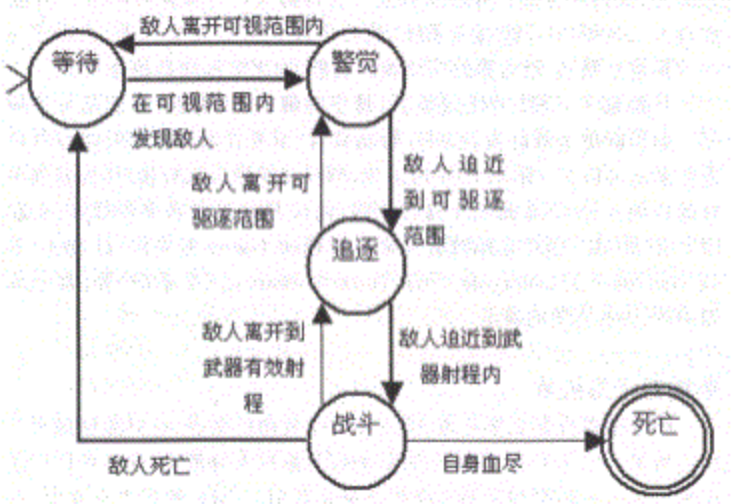


图 18-4 一个最简单的射击游戏 NPC 的 FSM 图

图 18-4 虽然很简单，但它向我们揭示了 FSM 的两点特性：第一，用 FSM 可以明确地表达 NPC 的行为系统，大部程序员，甚至非程序员都能毫无困难地理解；第二，一旦知道 NPC 的当前状态和输入，我们就可以判定其反应和目标状态。也就是说，我们可以准确预测 NPC 下一步的行为。因此，通过 FSM 所建立的是一种确定的行为系统，没有任何不确定因素。

目前，大多数游戏的 AI，特别是 FPS 类型的游戏，都是基于我们上面所介绍的 FSM 技术的。当然游戏中的 FSM 比上面的例子复杂得多，NPC 可能有几十个状态，状态转换法则也更严谨，共同构成了 NPC 的复杂的行为系统，使得玩家在和 NPC 对抗时觉得 NPC 的确有两下子，不可等闲视之。在编程时，用 C 语言的分支和循环语句就可实现上面的简单 FSM。但复杂的 FSM，一般要用 C++写一个通用的 FSM 类，然后根据不同的外部数据决定 NPC 的不同行为。也可以将 FSM 以矩阵的方式来实现，或将其存储在外部文件中。这样游戏设计师就可以用 FSM 编辑器自己编辑 NPC 的行为系统，然后将其 FSM 存在文件里，由程序自动读取运行测试，然后再进行修改和调整，而无需程序员的介入了。

总结起来，FSM 的优点如下：易于理解，易于编程，特别是易于纠错。如果测试游戏时发现 NPC 行动异常，只要在编译纠错时跟踪其状态变量就可以了。采用 FSM 的游戏，NPC 决策速度比较快(因为是确定性的行为系统)。正是由于这些原因，使得 FSM 这种古老

的技术，还在游戏 AI 领域老当益壮地挑着大梁。像 Epic Games 开发的《Unreal》系列，Activision 的《Interstate 76》，Valve Software 的《半条命》等，都是应用 FSM 技术成功的典范。

模糊状态机

FSM 虽然有那么多优点，但它还有个致命的弱点：它只能处理确定性的情况。使用 FSM 建立的 NPC 的行为系统太规范了，很容易被玩家识破。于是人们想到了是否能把不确定性引入 NPC 的行为系统中，这样一来 NPC 的行为就有更多变化了。于是另一种方法应运而生，这就是模糊状态机(fuzzy state machine，缩写为 FuSM)。

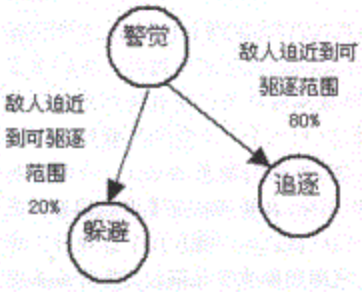


图 18-5 NPC 的 FuSM 图的一部分

FuSM 的基本思想就是在 FSM 基础上引入不确定性。在 FSM 中，只要知道了外部输入和当前状态，就可确定目标状态。而在 FuSM 中，即使知道了以上两点，也无法确定目标状态，而是有几个可能的目标状态。究竟转换到什么状态，则由概率决定。我们前面所示的 NPC 的 FSM 中警觉状态到追逐状态的转换，改成 FuSM 后，如图 18-5 所示。

我们可以看到，在 FuSM 中，当 NPC 处于警觉状态时，如果敌人迫近到可驱逐范围内，我们并不确定 NPC 是否转换到追逐状态还是躲避状态。而是根据概率，80%的情况下 NPC 会进入追逐状态，有 20%的情况 NPC 会躲避。这样一来，NPC 的行为就复杂多了，游戏性也更丰富了。

我们还可以看到，这个 NPC 是一个比较勇敢的 NPC。它在 80%的情况下都是勇往直前，只有 20%的情况会退缩，而我们只需要改变 FuSM 中概率的设定，NPC 就可以拥有不同的行为特征。比如说把 80%和 20%调换一下，则 NPC 会成为一个比较胆怯的 NPC。它在 80%的情况下会躲藏，只有 20%的情况会迎着敌人上去，而 NPC 行为特征的改变，是在不影响 FuSM 基本结构的条件下，简单地改变其概率设定而完成的。这是 FuSM 的一大优势!因为这样一来我们可以设计几个简单的通用的 FuSM，然后通过不同的概率设定(俗称的阈值)产生各种各样行为各异的 NPC。这可真是事半功倍!因为如果只用 FSM 的话，则每种 NPC 都要构造自己独有的 FSM，工作量太大了。

Activision 公司的《Call To Power》是大规模使用 FuSM 的典范。游戏中不同文明(种族)之间的差异就是 FuSM 的杰作!

AI 脚本和可扩展性 AI

前面我们介绍的 FSM 和 FuSM 都是所谓的“基于规则的 AI"(rule-base AI)。顾名思义，

基于规则的 AI 就是说事先要设计好容易理解的行为规则。然后在游戏中 NPC 必须遵循这些规则行事。而游戏设计师的主要任务，就是设计完善的行为规则，调节 FSM 和 FuSM 的各项参数，使得 NPC 的行为不致太过弱智。但游戏设计师们大多数编程能力有限，无法直接修改程序。程序员们就为他们设计了一些简单易用的工具，使得他们可以毫无困难地修改 NPC 的行为规则。在早期，大部分这样的工具是以脚本语言工具(script language)的形式出现的。它们不是像 Visual Studio 那样完整的复杂的集成编程环境，而是使用简化的编程语言，只有有数几个语句和数据类型，相当于一个复杂的编程语言的子集。使用这些工具所编制的小程序，被称为脚本(script)，用来控制 NPC 的行为。游戏设计师们使用脚本工具，真是得心应手春风得意。有一天，游戏设计师们突然开窍了!他们对自己说：“嘿!几个游戏设计师的力量和时间是有限的，但网上广大玩家的热情是无限的。为什么不把我们的脚本工具公开，让网上的玩家们自己设计 NPC 的行为规则呢?让他们自己去‘教’自己的怪兽如何去追逐敌人躲避敌人攻击敌人!那得出现多少种行为各异的怪兽啊!”于是一个伟大的思想诞生了（虽然从技术角度看没什么激动人心的内容），这就是可扩展性 AI（extensibleAIs）。

可扩展性 AI 的鼻祖是 QUAKEC。它是由 PC 游戏业里最有名的程序员 John Carmack 设计的一种脚本语言工具，几年前随着 QUAKE 游戏一同推出。它实际是 C 语言的一个简化版，是为了编写射击游戏中 NPC 的行为规则而量身订做的。使用 QUAKEC，玩家可以自己设计怪兽的行为规则和战斗策略，也可加入新的武器系统特性，经过编译，它们可以被游戏所采用，这样极大地丰富(延长)了原游戏的生命力。玩家并可把它们上载到网上去和别的玩家交流。这些用 QUAKEC 设计生成的怪兽统统被成为 bots。很快，各种各样的行为各异的 bots 就在网络上风行起来。其他的三维射击游戏也紧紧跟风，推出了自己的脚本语言工具。有代表性的要属《Unreal》和《半条命》了。两者之间又有不同：《Unreal》的工具更加简单一些，是基于指令的，输入指令序列和条件就可以了；而《半条命》的工具更类似于传统编程工具，如 Perl 和 JavaScript。

除了 FPS 类型的游戏外，令人吃惊的是连 RPG 也开始采用脚本语言工具了。被誉为美式 RPG 代表的《柏德之门》就使用了和《半条命》类似的脚本语言。玩家不仅可以改变敌方 NPC 的行为规则，也可以改变同伴 NPC 的行为规则，甚至可以产生新的 NPC 类型。一个简单的例子如下所示：

```
IF
        // If my nearest enemy is not within 3
!Range(NearestEnemyOf(Myself),3)
        // and is within 8
Range(NearestEnemyOf(Myself),8)
THEN
```

```
            //1/3 of the time
RESPONSE #40
            // Equip my best melee weapon
    EquipMostDamagingMelee()
            // and attack my nearest enemy, checking every 60 ticks
to
            // make sure he is still tLe nearest
    AttackReevalutate(NearestEnemyOf(Myself),60)
            // 2/3 of the time
RESPONSE #80
            // Equip a ranged weapon
    EquipRanged()
            // and attack my nearest enemy, checking every 30 ticks
to
            // make sure he is still the nearest
    AttackReevalutate(NearestEnemyOf(Myself),30)
END
```

以脚本语言工具为代表的可扩展性 AI 技术，其最大局限性是玩家需要有一定的编程能力。像 QUAKEC 这样的“东东”还是需要一番功夫才能掌握的。而且也不是所有类型的游戏都适用。提供对脚本语言工具的支持也不是一件简单的事，公司得起码在网上提供详细的接口信息和文档资料，还得提供一定的热线帮助和客户服务功能。这些都需要金钱和人力，并不是所有的公司都愿意付出的。

神经元网络

到了神经元网络，才真正有点 AI 技术的味道。因为从某种意义上说，神经元网络就是一个模拟人脑功能的数学模型。神经元，又称神经细胞，是人脑中处理信息的基本单元，也是所有神经器官的基本组成部分。神经元之间通过轴突(axon)连接在一起。信号通过电化学反应由一个神经元传导到另一个神经元。简单地说就是，从其他神经元传来的信号，使这个神经元本身产生一定

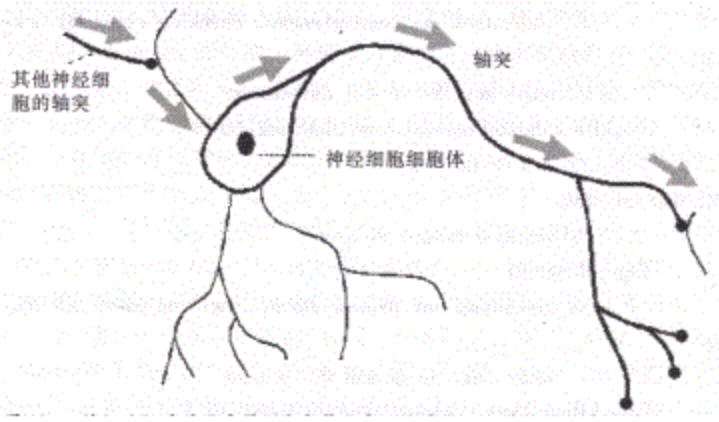


图 18-6 神经元细胞的示意图（灰色箭头表示信号由一个神经元细胞传送到另一个神经元细胞）

的反应，如果反应达到一定的阈值，则产生电脉冲，沿着轴突被传导到其他神经元。神经元的示意图如图 18-6 所示。

与人脑相对应，一个神经元网络是由一些节点(node)相互连接而成。这些节点就相当于人脑的神经元细胞。它们之间的连接被称为链，每条链(link)被赋予一个加权值(weight)。所谓的神经元网络的自学习就是通过一系列试验去更新调整这些加权值。有一些节点是和外部环境连接的，被称为输入 / 输出节点，另一些节点在网络内部。一个简单的两层神经元网络如图 18-7 所示。

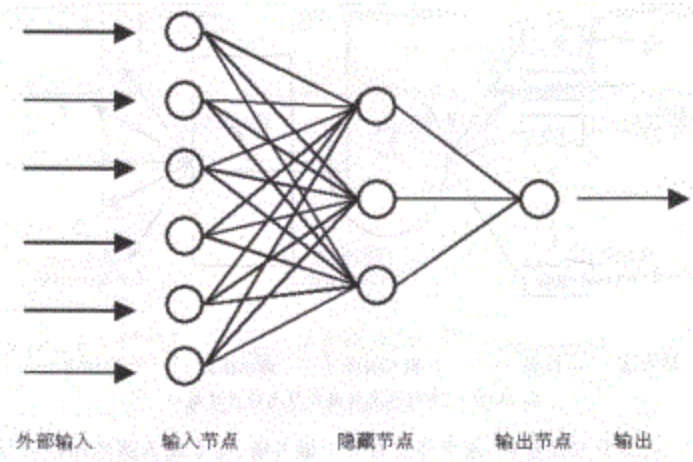


图 18-7 一个简单的两层神经元网络（由于输入节点只是接受外部输入并传给隐藏节点，不进行复杂的计算，因此不将它们单算一层）

图中的神经元网络有两层，包括 6 个输入节点、3 个隐藏节点(在网络内部)和 1 个输出节点。不同层节点之间是全连接，即所有输入节点和所有隐藏节点都有连接，所有隐藏节点和输出节点都有连接。输入节点的输出(链)就是隐藏节点的输入(链)，而隐藏节点的输出(链)又是输出节点的输入(链)。输入节点从外界接受输入信息，输出节点向外界发出输出信息。图中的神经元网络从外界接受 6 个输入信号，产生一个输出信号。

每个节点有一系列的输入链，可以是从其他节点来的，也可以从外部环境来。每个节点也有一系列的输出链，可以输出到其他节点，也可以是输出到外部环境。节点有一个最重要的属性是激活阈值(activation level)。当一个节点从其输入链接受外部环境或其他节点的输出信号时，它根据这些信号值和各条链的相应加权值，由一定的计算方法得出一个激活阈值，这个阈值就作为自身的输出值。一个神经元网络节点的内部结构如图 18-8 所示。

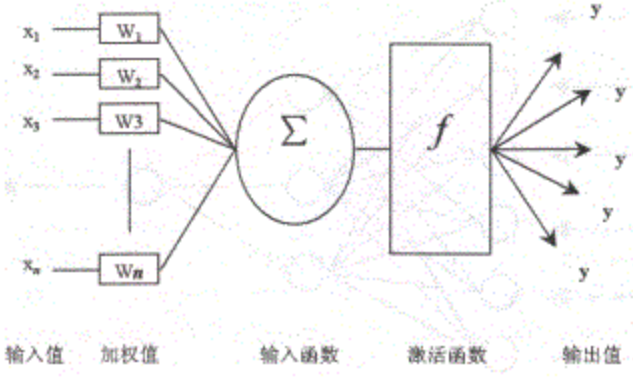


图 18-8 一个神经元网络的节点的内部结构

从图中可以看到：这个节点有 n 个输入链，每个输入链的加权值分别是从 W1 到 Wn，它从每个输入链接受的信号值为 X1 到 Xn。然后经过一个输入函数(input function)，实际上是个加权累加器，其计算结果为ΣXi Wi。而后再经过一个激活函数 f(activation funtion)，得出一个输出值 y。最后这个输出值 y 沿着节点的输出链被传给下一层的其他节点们。

激活函数的选择，对一个节点乃至一个神经元网络的行为起到至关重要的作用。目前常用的激活函数有三种，分别为阶跃函数(step function)、符号函数(sign function)和 S 曲线函数(sigmoid function)。其特性分别如图 18-9 所示。

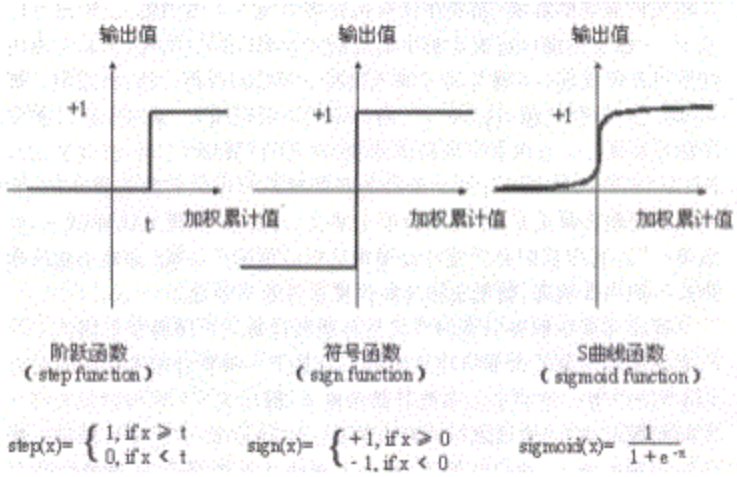


图 18-9 神经元网络常用的三种激活函数

需要指出的是，当使用阶跃函数时，为了简化计算，一般把阈值为 t 的阶跃函数转化成阈值为 0 的阶跃函数。为此需要给节点再外加一条输入链，其输入值永远是 1(也有人用负 1)，其加权值由 t 决定而来。经过这样转换后的神经元网络计算起来更方便。

神经元网络的组织结构有多种，其中最常用的是前向式(feed-for-ward)神经元网络和循环式(recurrent)神经元网络。循环式神经元网络中有闭环，更接近于人脑结构，但也更复杂且难于计算。在前向式神经元网络中，节点之间的链没有形成闭环。一般是将整个网络分成几层，本层节点之间无连接，本层节点只和下一层节点连接和前一层节点没有连接。

没有隐藏节点的前向式神经元网络被称为感知器(perceptron)。反之，则称为多层网络(multilayer network)。

前面简单地介绍了神经元网络的基本情况。那么，这么个乱糟糟的网络究竟有什么用呢?如何把它用于游戏中呢?它究竟有什么“魔力”值得人们对其大加青睐呢?

我们知道，用计算机解决实际问题，就是建立一个输入／输出对应关系的问题：我们输入一些数据到计算机里，然后希望计算机能够输出一些我们需要的数据，而要在计算机程序中建立正确的输入／输出对应关系，一般先要搞清楚需要解决的问题的逻辑(因果)结

构，然后才能用计算机程序实现。也就是说必须先搞明白问题的内部结构，然后用计算机编程去实现(模拟)这个结构，内部结构完成后输入／输出关系自然就能满足要求了。以古老的街机游戏《吃豆》(PACMAN)为例，游戏中追逐 PACMAN 的那些幽灵(ghost)的行为和策略是程序员们预先想好的，有明确的逻辑因果关系(用简单的中文来表示，就是“如果 PACMAN…，则幽灵…”)。程序员们必须设计好逻辑结构后再编程实现。如果不知道事物运行的内部机理，就无法用计算机模拟并解决问题。

神经元网络解决问题的方式与传统的计算机程序完全不同。它不需要把实际问题的内部结构搞明白，它甚至不需要针对不同的问题采用不同的方法。它只关心系统外部的输入／输出关系!它可以观察学习实际问题的输入／输出之间的对应关系。也就是说，由程序员提供一些实际问题的输入／输出的例子，神经元网络从这些例子中，调整优化各条链的加权值，使得神经元网络的输入／输出行为和实际问题的例子相吻合。这样神经元网络就在一定程度上实现了对现实世界的模拟。

同样以《吃豆》为例：我们可以用神经元网络来教计算机自己玩《吃豆》。具体方法是先设计好神经元网络的结构，包括输入／输出变量的选择，然后把这个神经元网络的各链的加权值赋初值。准备工作完成后，在《吃豆》的后台运行这个神经元网络，前台由一位游戏高手操作，玩上几十上百次。在高手玩的过程中，在后台的神经元网络采集输入／输出变量的数值，利用这些数值去一次次调整加权值。这个高手玩游戏的过程，就是神经元网络学习的过程。游戏高手就是神经元网络的“老师”。最后，经过长时间的学习训练，加权值调整得非常合适了。游戏高手可以歇歇了，由神经元网络顶替。这时就是计算机自己玩游戏了。这时的神经元网络实际上相当于一个 NPC。我们会发现，神经元网络有了自行其是的能力，而且它的行为特征和思想策略和它的“老师”十分类似。

由上面的例子可以看出，神经元网络实际上也可以被用来建立 NPC 的行为系统。只不过这个行为系统的内部运行规则和逻辑结构不像 FSM 和 FuSM 那么一目了然罢了。有人使用神经元网络的技术搞出来了针对《QUAKE II》的 bot，为了和用脚本工具生成的 bot 相区分，这种 bot 被称为 NeuralBot。而神经元网络的 NPC 的实现是通过一个训练过程完成的，并且需要大量的输入／输出数据。可以戏称为：从实践中来，到实践中去。更重要的是：由神经元网络所形成的行为系统不是死的，是可以通过不断的学习训练去动态地调整，去适应新的情况。这才真正是神经元网络的威力所在!这也是所谓的机器学习(machine learning)的范畴。

神经元网络正是由于具有学习和自适应能力而被游戏业寄予厚望。游戏设计师们希望可以通过神经元网络设计出更具竞争性的游戏。也就是说：游戏中和玩家对抗的计算机一方不应是“死”的，而是应该在和人的对抗过程中动态地根据对手的特点，去总结经验教训并调整策略战术。FSM 和 FuSM 对此是无能为力的，而我们前面介绍的神经元网络就具有这样的能力!

前面说了神经元网络的很多优点，但目前神经元网络在游戏业的应用还不普遍。是什么因素阻碍了这么有魅力的技术流行起来呢?主要有以下几个原因：第一，选择恰当的输入／输出变量是十分困难的。因为没有任何理论指导，要凭感觉和经验，而选择输入／输出变量是设计神经元网络的第一步，对其成功又起至关重要的作用。比如说那个《吃豆》游戏，输出变量比较简单，可以是代表左、右、上、下的四个变量。如果其值是 0 则代表不采取行动，如果是 1 则向所代表的方向行动一格。而输入变量就比较费心思了，可供考虑的有：幽灵的数量、各自的速度、各自距 PACMAN 的距离等等。究竟如何取舍，效果如何，则只有骑驴看唱本，走着瞧了。

第二个问题是训练过程。神经元网络的训练过程需要游戏高手，一般是游戏设计师，来充当“师傅”的角色，费时又费力。虽然神经元网络实际上可以不需要“老师”来教，而自己摸索学习，但效果一般没有保证(有“老师”指导的训练，称为 supervised learning；没有的，称为 unsuper-vised learning)。所以，一般游戏公司还是由游戏高手来训练神经元网络。但问题是，他们在经过一番训练得到了满意的神经元网络之后，就把网络的特性固定在游戏中了。也就是说游戏中的神经元网络是死的，并没有真正从玩家的反应来学习调整的能力。这样神经元网络的真正优势就没有发挥出来。

神经元网络更致命的弱点，则是无论训练得多么好的神经元网络，都有可能出现不可解释、不可理喻的行为。因为，虽然我们可以教它很多事情，但它终究会碰到以前从未出现的新情况。由于神经元网络本身是个黑箱，无法分析其内部逻辑关系，出了问题很难查找原因。反之，FSM 和 FuSM 出了问题很容易追本溯源查个究竟。这也是有些程序员在尝试过了神经元网络之后，又回归到原始的 FSM 和 FuSM 的原因。

总之，在游戏业发现神经元网络的巨大潜力的短暂惊喜之后，人们开始以一种切实的态度来对待它了。虽然其应用并非坦途，但作为一项已经发展得比较成熟的技术，在解决了一些实际操作上的困难之后，神经元网络有可能把游戏的智能提高到一个新的的高度。

遗传算法

谈到遗传算法，就要谈到大名鼎鼎的达尔文的《物种起源》一书。书中讲的就是大自然优胜劣汰。适合自然环境的物种可以得到更多的繁衍生息的机会，它们的后代部分继承了它们的优点，而不适应自然环境的物种则被慢慢淘汰掉。自然环境在缓慢地变化，而物种也在慢慢地进化。在进化过程中间或会有突变发生。大部分产生突变的个体都不能适应环境，短命夭折。但也有个别突变个体比变异前的物种更适应环境，从而繁衍了下去，并改良了旧的物种，或者产生了新物种。

人们从大自然中的优胜劣汰的现象得到启发，试图把这种机制引入到人工智能系统中，遗传算法由此应运而生。其基本思想如下：在一群个体中，通过突变(改变个体结构属性)、

交配(选择两个个体)和繁殖(合二为一)，再通过进化过程，最终获得更适合环境的个体。对游戏来说，个体可以是整个 NPC，也可以是 NPC 的一个属性或功能。比如说，一个游戏中，怪兽们(NPC)有不同的属性并有繁殖功能。那么，在和玩家的对抗中能够存活并成功逃走的怪兽们，将更有可能生育下一代。而它们的下一代，将更机智更难对付。这样游戏就更有竞争性。更广义地看遗传算法，它可以用来解决最优化问题，而 RTS 上的策略选择等问题，归根结底都是最优化问题。

在遗传算法中，一个个体由一系列字符排列组合来表示。每个字符被称为基因。对真正的 DNA 来说，有 AGTC(adenine,guanine,thymine,cytosine)四种，而遗传算法一般使用 0 和 1 就可以了。也就是说，个体用 0 和 1 的排列组合来表示，如 0100010111。个体对环境的适应能力，由适应函数(fitness function)来判定。将代表个体的 0 和 1 的字符串输入这个适应函数，得到的输出是一个数值，其大小代表此个体对环境的适应能力的高低。适应环境的个体将得到更多的机会去繁殖再生。繁衍再生的过程由两部分组成：交叉组合(cross-over)和突变(mutation)。经过繁衍再生后，我们得到的个体的新的一代，然后可以再进行新一轮进化。基本过程如图 18-10 所示。

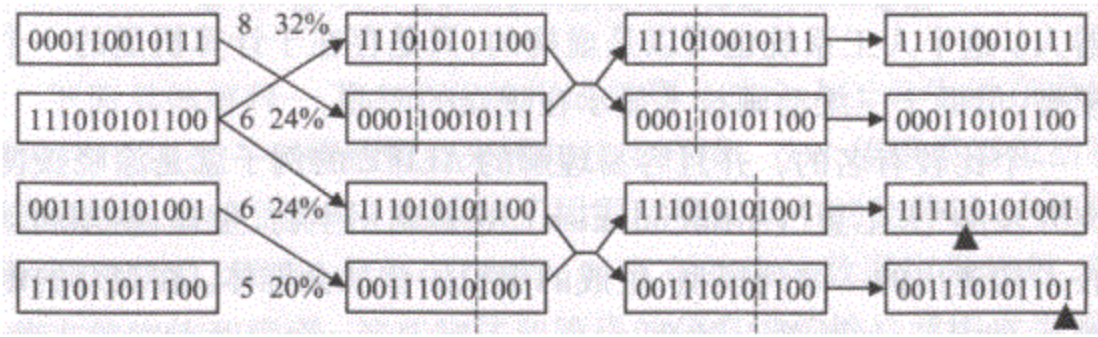


图 18-10 遗传算法的示意图。本例子出自 Stuart J. Russell 和 Peter Norvig 所著的《Artificial Intelligence: A Modern Approach》一书。

要用遗传算法解决实际问题，关键有两点：第一，如何将实际问题中的个体编码成 0 和 1 的序列；第二，如何决定适应函数。最简单的编码，以 RTS 类型的游戏为例，如果计算机一方要从 8 种不同建筑中选择决定当前应该修建哪种建筑，则 23=8，需要 3 位编码就可以了。更复杂的编码方法，则可以把复杂的策略选择考虑进去。

遗传算法的弱点是速度慢，占用 CPU 资源多。这些都影响了它在游戏中的使用。目前使用遗传算法比较多的，自然是虚拟宠物类型的游戏，如 PC 上的《Creatures》。

人工生命

人工生命(ALIFE)是一个极广阔的研究领域，它本身是一个综合性交叉学科，包含了生物学、心理学、计算机、精密机械、自动化等诸多领域的内容。正因为如此，很难对它确切定义。简单地说，ALIFE 研究的是如何使一个人工系统具有自然生命系统的特征。这个人工系统可以是有实体的。借助精密机械和自动控制技术，我们可以构造这样的 ALIFE 机器人。这个人工系统也可以是虚构的，只是存在于计算机里的一个数学模型，借助于三维动画技术显示在观众的面前。

图 18-11 观众所看到的人工鱼的图像

一个比较有名的，并且容易理解的 ALIFE 的例子就是涂晓媛博士研究开发的“人工鱼”(Artificial Fish)。这是利用生物、生理、物理和动画等技术构建出的一系列模型，生成的虚拟的鱼社会群体。在观众的眼中看来，就是一个屏幕上的鱼群。但这个人工鱼系统和动画最大的不同点就是动画中的鱼是人们事先设计好的，其每一个动作都是事先确定的。而这个系统中的人工鱼是具有意图、习性、感知、动作、行为的自主智能个体。

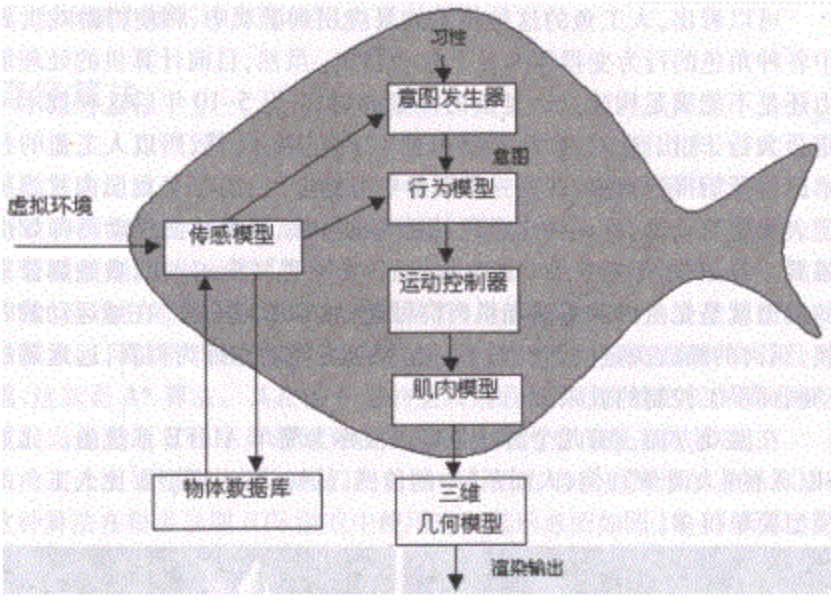


图 18-12 人工鱼的基本构成模块和信息传输示意图

下面具体解释一下人工鱼的构成机理。这个人工鱼生活在虚拟的环境中，它通过一个传感模型来感知这个环境。这个传感模型包括视觉感受器和温度感受器等。在感知了周围环境和其中的物体后，它把接受到的视觉信息(颜色、大小、距离等)和现有的物体数据库

做比较，辨认出前方的物体和障碍。这些物体和障碍的信息被传输给意图发生器，并根据这条鱼独特的习性，决定意图——可以是捕食、交配、闲逛等等。然后，意图被传输给行为模型来启动一定的行为。行为模型计算出控制参数，这些控制参数被传输到运动控制器。运动控制器类似于神经的作用。它向肌肉模型发出指令，收缩或者放松指定的肌肉群。人工鱼的三维模型内部有 23 个节点和 91 个“弹力—阻尼”单元。这些“弹力—阻尼”单元类似于肌肉。通过“弹力—阻尼”单元的收缩或放松，改变节点的位置，从而改变鱼的形体，产生运动(如摆尾)。

可以看出，人工鱼的这种技术如果应用到游戏中，将使得游戏世界中各种角色的行为变得极为复杂生动真实。虽然，目前计算机的处理能力还是不能满足构建这么复杂的模型的要求，但 5~10 年后这种技术一定会大行于世。在动画片领域，由于不需要实时计算，所以人工鱼的技术已经开始得到重视，并在一些影片中得到应用。特别是由肌肉放缩决定表面模型变形，从而决定角色的动作的方法，更是使三维动画师如虎添翼。在 2001 年的三维动画电影《怪物史莱克》(Shrek)中，费娥娜公主的模型就是先由内部骨骼和肌肉群做起，然后加上皮肤。在她运动的时候，肌肉的缩放决定动作的细节。结果是人物动作极为拟真，远远超过动画师手工控制的效果。

在游戏方面，有几个游戏是可以被称为简单 ALIFE 系统的。比如 SEGA 的《人面鱼》，但《人面鱼》中的传感、认知、行为模型要比人工鱼的模型简单得多。

图 18-13 世嘉的《Seaman》中使用了语音识别技术，玩家可以和 Seaman 对话，游戏中 Seaman 所生活的环境就是一个小 ALIFE 系统

图 18-14 《Creatures》系列是迄今为止使用 ALIFE、神经元网络、和遗传算法等 AI 技术最多的游戏了。也可以说是 AI 技术含量最高的游戏了

除了以上提到的各种技术外，其他如模糊逻辑(fuzzy logic)、专家系统(expert system)、决策树(decision tree)等技术也在逐渐被游戏业消化和吸收。比如说在《S.W.A.T》中就使用了模糊逻辑，而《帝国时代》的下一代产品《神话时代》就已经开始构建复杂的专家系统了。限于篇幅，在本章中就不涉及这些技术了。

寻径算法

很多人在谈到人工智能时，都愿意把寻径算法单列出来，由此可见寻径算法的重要性。在以前使用寻径算法较多的是 RTS 和 FPS，但现在 RPG 等类型的游戏也开始大规模使用这种技术了(在传统 RPG 中，玩家需要控制游戏主角的每一步；而新的 RPG 中，可以用鼠标点一下目的地，游戏主角自己就可以走到那里了)。值得欣喜的是经过了长时间的探索，目前游戏业已经基本上找到了行之有效的算法来解决寻径问题，这就是 A*算法。其在游戏业的受欢迎程度，不说放之四海皆准罢，但起码也是所有程序员遇到寻径问题时首先考虑的最有效算法。

在介绍 A*算法之前，我们先介绍一下更古老也更简单的寻径算法。这种算法在很多早期 RTS 游戏中使用过。其示意图如图 18-15 所示。

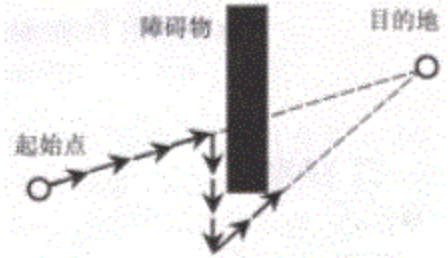


图 18-15 早期 RTS 游戏使用的寻径算法示意图

图中的黑色方块是障碍物，我们从起始点到目的地画一条直线，然后就沿这条直线走，直到碰到障碍物无法前进。这时再沿障碍物横向移动，直到能够找到指向目的地的直线，再沿直线移动。就这样沿直线移动和碰到障碍物横向移动交替进行，就可以走到目的地了。这种方法实际上没有寻径，也就是说在走之前并不知道从起始点到目的点之间是否有路，也不知道哪条路最短。虽然看起来很笨，但直接了当，容易编程，因此在第一代 RTS 游戏中被广泛使用(另外，当时第一代 RTS 的程序员们还没有开始认真研究寻径问题，自然不知道有更好的算法)。

实际上很久以来一个问题都在困扰着程序员们：是花大量时间先算出最优路径，然后再走好呢；还是先愣头愣脑地走一段，直到碰壁再修正好。头一种方法需要花大量的计算时间在寻找最优路径上；而后一种方法可以省去许多麻烦，但有可能走很远的路或者根本没有路也瞎走一气。目前来说，人们已经不满足于第二种方法了，人们希望计算机能够更加聪明地找到好的路径，这就引入了寻径问题。

A*算法简介

所谓寻径问题，就是在地图上的起始点 A 和目的点 B 之间找到一条可通行的道路。显然 A 和 B 之间可以有很多条道路，我们的目的是用最短的计算时间找到最优路径(距离最短或者最安全)。从大的方面看，寻径问题实际上属于算法研究中的搜索(search)问题。在计算机系开的算法课和人工智能课中，搜索问题都是重点，也有很多已经被搞得烂熟的算法可以用。比如广度优先搜索(breadth-first search)，深度优先搜索(depth-first search)，双向搜索(bidirectional search)等等。我们要介绍的 A*算法，也是这么一种搜索算法。

我们先以一个虚构的亚斯兰大陆的地图来介绍最佳优先搜索(best-first search)的概念。假设我们要找到从月影村到万念崖的最短路径。从月影村出发的话，我们有两条路可以走：

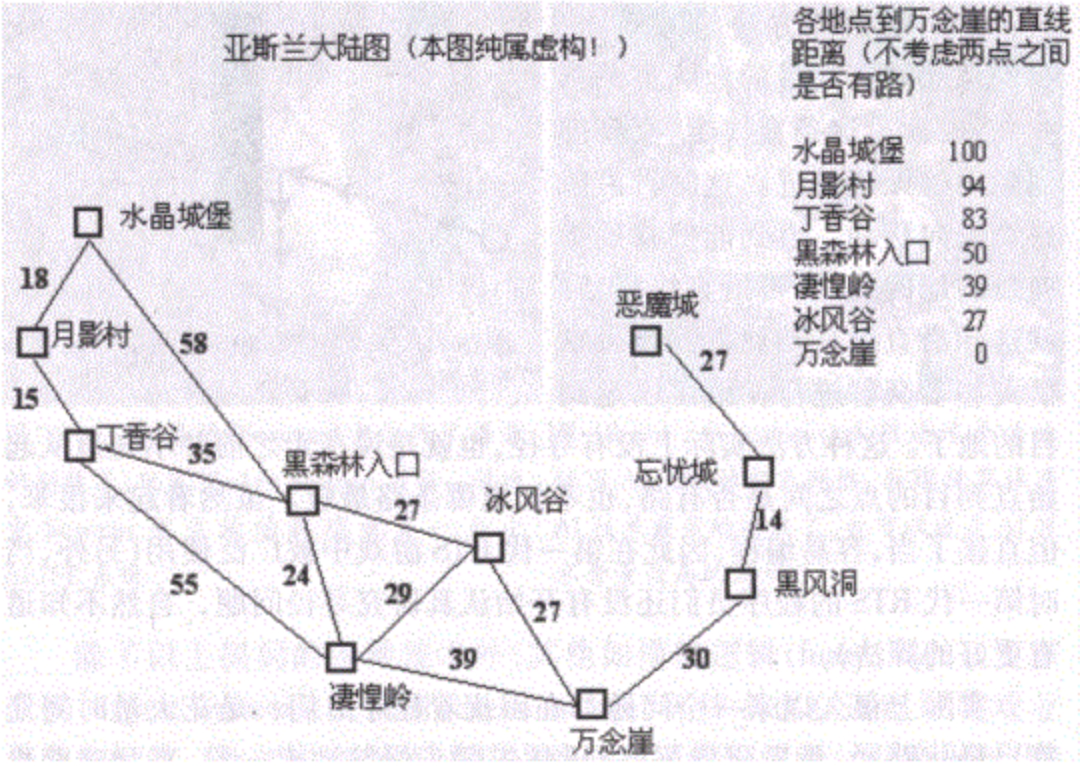


图 18-16 亚斯兰大陆地图。图中显示各地点和它们之间的道路，数字为道路长度

去水晶城堡，或者去丁香谷。如果我们随便选择走丁香谷这条路，那么到了丁香谷后我们又面临了两个选择：是去黑森林入口?还是去凄惶岭?我们看到，每一步都要面对诸多可能和诸多选择，都有很多不同的路径到不同的地点，如何做出选择呢?当然，究竟一条路好不好，是不是近路，得走一遍才能知道!但我们可以通过某种程度的预测来评估各条路径的好坏!这就引入了评估函数(evaluation function)的概念。评估函数的作用就是预测评估各可能路径的好坏，并返回一个数值。这个数值越小，则我们就越应该走这个数值所对应的路径。最佳优先搜索就是使用了评估函数的搜索算法。使用最佳优先搜索算法，每一步我们都选择评估函数返回值最小的路径，直到到达目的地。这是多么直截了当的方法啊!但是，问题远不是看起来的这么简单，光是这个评估函数就麻烦大了!

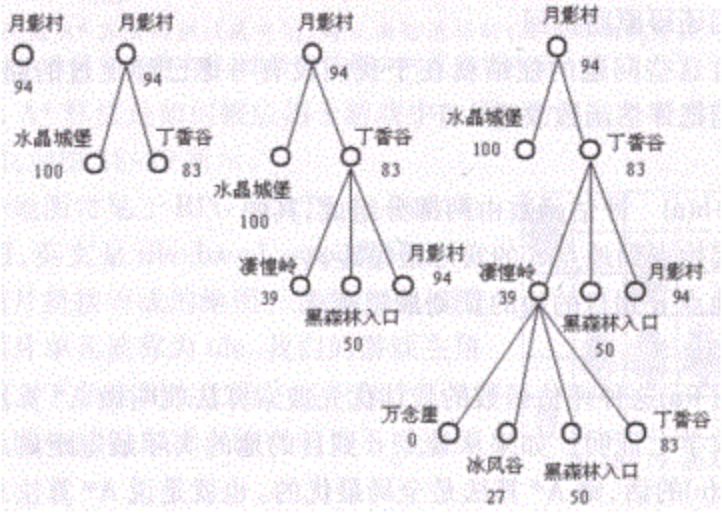


图 18-17 寻径问题实际上也就是搜索问题。这个树状展开图就显示了搜索的过程步骤。每次面临选择时，我们都把现阶段所有的树叶节点的评估函数值拿来做比较，选择数值最小的节点作为下一步

一个最简单的大家都能想到的评估函数就是：一个地点到目的地的直线距离。使用这个评估函数的示意图如图 18-17 所示。

图中的数字代表一个地点到目的地的直线距离(不考虑它们之间是否有路)，也就是评估函数的返回数值。在搜索时，每一步先找到与当前所在地点连接的所有邻近地点，然后使用评估函数，选择评估函数返回数值最小的地点作为下一步的起点。比如当我们走到丁香谷后，我们发现有三个邻近地点可以走，分别是黑森林入口、月影村和凄惶岭。它们所对应的评估函数值分别为 50、94 和 39，再加上另一条路上的水晶城堡，其评估函数值为 100。我们选择最小的数值 39，也就是凄惶岭，作为下一步的地点。这样一步一步，直到我们发现评估函数返回值为 0 了，这说明我们已经到了目的地了。

以上的算法似乎可以解决寻径问题了，但实际上它有严重的缺陷!首先它所找到的路径不是最短的。丁香谷的下一步应该选黑森林入口，而不是凄惶岭。因为这个算法只顾眼前利益，无法保证长远(全局)最优，因此人们又习惯地称之为“贪婪算法”。其次，贪婪算法有可能很慢。它有可能在一开始就沿着错误的路径走下去，直到醒悟回来再往回折。比如我们要从忘忧城走到水晶城堡，用贪婪算法的话第一步会走到恶魔城，因为恶魔城的直线距离距水晶城堡近。但实际上恶魔城是一条死路。我们还得原路折回。

所有这些问题的症结就在于我们没有考虑已经走过的路的距离。如果我们把评估函数改进一下。

f(n)=g(n)+h(n)　评估函数由两部分组成，其中

g(n)：从起始点到地点 n 的实际最短距离

h(n)：从地点 n 到目的地的预测最短距离

使用 f(n)这种评估函数的最佳优先搜索算法就叫做 A*算法。我们可以从数学上证明：如果从地点 n 到目的地的实际最短距离总是大于或等于 h(n)的话，则 A*算法是全局最优的。也就是说 A*算法总能找到最短的路径。当我们使用地点 n 到目的地的直线距离作为 h(n)时，显然这个条件是满足的。因为，地点 n 到目的地的实际最短距离总是大于或等于它们之间的直线距离的。而且我们

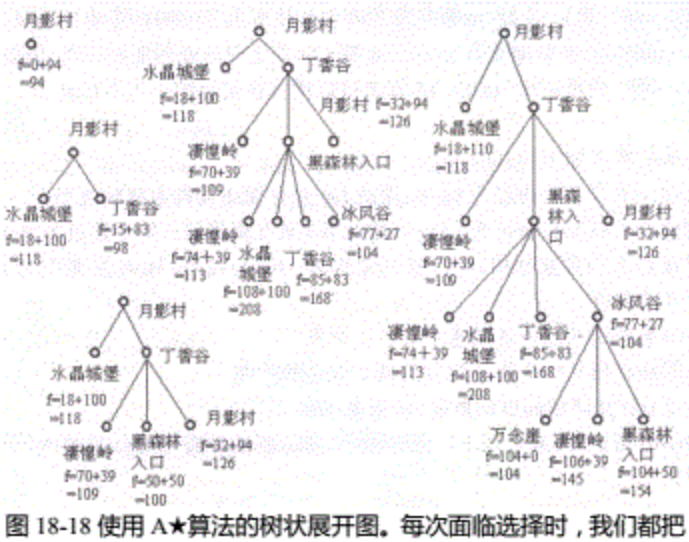


图 18-18 使用 A★算法的树状展开图。每次面临选择时，我们都把现阶段所有的树叶节点的评估函数值拿来做比较，选择数值最小的节点作为下一步

还可以证明 A*算法是效率最高的。使用 A*算法的搜索示意图如图 18-18 所示。

那么 A*算法是如何被应用于游戏中的呢?我们先来考虑最简单的二维地图,如图 18-19 所示。

这种地图常见于 RPG 和 RTS 中，被称为拼接地图，英文是 tile-based map，翻译过来就是由图片拼接而成的地图。这些大小相同的基本图片单元被称为 tile。我们的游戏主角要从地图左下角走到右上角，且只能上下左右移动。图中的树木所占的格子为不可逾越的。

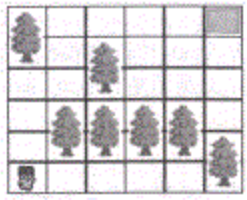


图 18-19 拼接地图

对拼接地图来说，显然不好用直线距离作为 h(n)，而一般采用曼哈顿距离(Manhattan distance)作为 h(n)。曼哈顿是美国纽约的一处地名，其地摩天高楼林立，道路横平竖直呈棋盘状，和我们的这种拼图一样。所谓两点之间的曼哈顿距离就是两点之间的水平距离(格子数)加垂直距离(格子数)。比如，从游戏角色现在的位置到目的地的曼哈顿距离是 10。显然，曼哈顿距离肯定小于或等于两点之间路径的实际距离，因此，使用曼哈顿距离作 h(n)的 A*算法可以找到拼接地图上的最优路径。

具体的算法如下代码所示：

创建一个空 list，命名为 Open，用此 list 来存储还需要考察的格子

创建一个空 list，命名为 Closed，用此 list 来存储我们已经考察过的格子

创建格子的数据结构，可以使用结构体(struct)或类(class)，需要包括以下几个属性：

g：用来存储从起始点到这个格子的距离

h：用来存储从这个格子到目的地的曼哈顿距离

f：用来存储评估函数的数值，也就是 g+h

parent：指针，用来记录上一步的格子，最后形成总路径时从目的地回溯时用

loc：可以使用结构体或类，用来存储 XY 坐标信息

将起始点的格子存人 Open，

计算起始点的 h,g,f,parent=null

循环直至 Open 为空

{

从 Open 中所存储的所有格子中，取出 f 值最小的格子

if 此格子是目的地 then 寻径成功，回溯输出路径

else

{

考察此格子上下左右的四个邻居

{

if 此邻居不是障碍物 then

{

计算其新 g 值

if 此邻居已经在 Open 或 Closed 中并且其新 g 值≥旧 g 值 then

{ }

else

{

更新其 g,h,f,parent 值

if 此邻居已经在 Closed 中 then 将其从 Closed 中删除

if 此邻居不在 Open 中，then 将其存入 Open 中

}

}

}

将此格子存入 Closed

}

}

没有找到可行路径，返回错误码

A*算法在更复杂情况下的应用

前面介绍了 A*算法在最简单的二维拼接地图上的应用。现在游戏越来越复杂，除了这种最原始的二维拼接地图外，还有《帝国时代》等 RTS 游戏中所惯用的 45 度角俯视的拼接地图，《坦克将军》等游戏中的菱形地图，更不要说完全三维的主视角射击游戏(FPS)了，那么 A*算法是如何对付这么多种不同的情况呢?值得程序员们庆幸的是，A*算法很容易根据实际情况进行改进和扩展。实际上目前 AI 程序员们在寻径方面主要的工作就是针对各种特殊的情况去改进和扩展 A*算法。

面对不同类型的地图，首要的任务是决定如何拆分地图。因为要用 A*算法，我们就得知道地图是怎么划分的，它的最基本单元是什么，位置怎么算，距离怎么算，等等等等。我们以图 18-20 所示的二维地图为例，来说明各种不同的划分地图的方法。

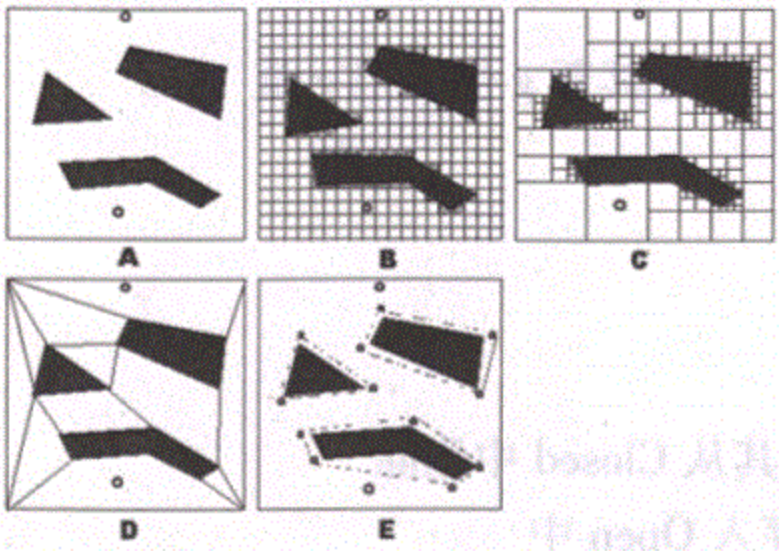


图 18-20 划分地图的几种方法（本图部分取自 Bryan Stout 的论文“Smart Moves：Intelligent Path-Finding"）

图 A 是基本地图，三个黑色区域为障碍物，底下的圆圈为起始点，最上面的圆圈为目的地。我们所要做的，就是找到有效的划分地图的方法使得我们能够使用 A*算法进行寻径。最简单的，同时也是最笨的划分方法就是把地图等距离地划分为大小相同的格子，如图 B。然后根据黑色在一个格子中所占面积的百分比来决定这个格子的属性。比如说我们决定黑色占一个格子

的 40%以上就认为这个格子是不可穿越的，如图 B 中阴影的部分。这样我们就把具体的地图划分成可以使用 A*算法搜索的地图了。这种方法直截了当，精确度也够，但问题是每个格子太小，造成从起始点到目的地要走的步骤多(所经过的格子数)，要做的选择也多。如果我们还记得本节最开始的例子的话，从 A*搜索的树状图上我们可以看到，如果格子多的话，树状结构的高度(也称深度，depth)越大。树的高度越大，则 A*算法的速度就越慢。

一种更有效的划分地图的方法则是四叉树(quadtree)，如图 C。它不把地图分成大小相同的格子，而是根据实际需要把地图分割成大小不同的格子。四叉树的划分方法是把一个大的正方形格子划分为四个大小相同的小的正方形格子，然后考察这四个小格子。如果黑色所占面积大或小到一定程度(比如大于 95%或小于 5%)，则不再分下去了，因为我们可以决定此格子的属性了(是否可以通过)；否则对小格子再分成四个小格。如此循环往复，直到所有的格子都满足条件。这种方法的优点显而易见，我们可以看到小格子基本都集中于障碍物的边沿，使用大格子则减少了格子的总数，使得 A*算法搜索起来更快。

另一种更复杂的方法是凸多面体法(convex polygons)，如图 D。它把地图划分为凸多面体。每个凸多面体都有自己的属性(全白或全黑)。这种方法比较适合于三维游戏，特别是 FPS 类型的游戏。关于如何把地图划分为凸多面体的组合，有各种不同的算法。其中一个比较流行的方法叫导航网(navigation mesh)，它是用来划分三维地图的(在三维里称做地图也许不太合适了，使用三维空间这个词似乎更恰当些)。我们知道 FPS 游戏中的关卡一般都是三维的，是由各种三维建筑物(走廊、柱子、屋子等等)和三维物品构成的。这些三维物体是由多边形构成的。游戏在处理时一般又把多边形再细分成三角形，三角形是游戏中三维空间的最基本的构成单元。这也就是说游戏中的三维空间是由一些大大小小的三角形拼接而成的。这些三角形中，有的是可以通过的道路，有的是无法通过的墙壁。导航网的基本思想就是给每个三角形一个属性，这个属性就反映这个三角形是否能够通过。走廊和地板所对应的三角形是可以通过的，而天花板和墙壁则不行。这样我们就可以对三维空间使用二维的 A*算法了。需要指出的是使用这种方法时不能像二维地图那样很容易地确认邻近的格子了。在二维地图上，如果现在位置为(X,Y)，则邻居为(X+1,Y)，(X,Y+1)，(X-1,Y)，(X,Y-1)。但在三维空间里，三角形之间显然无法这么容易地计算相邻关系。因此必须事先将三角形之间的相邻信息存储起来，在使用 A*算法时调出来使用。(之所以被称为“导航网”，导航比较容易理解，而网(mesh)是针对三维物体说的。因为三维物体由多边形构成，多边形的边相互连接形成一个中空的网，被称为 polygon mesh。)

最后再介绍一下可视点法(points of visibility)。可视点法采取了不同于上面几种方法的策略：它不硬性划分地图的空间结构，而是在障碍物外的邻近地方取点，用几个点把障碍物包起来。用寻径算法时只需把这些点连起来就可以了。这和本节一开始的例子近似。

分层寻径

一种比较有效的优化 A*算法的方法叫做分层寻径。其基本思想是把二维地图(或三维空间)按不同的复杂层次来划分。比如说，一个城堡可以先分成不同的房间，然后再考虑每个房间里面的细节，如图 18-21 所示的例子。

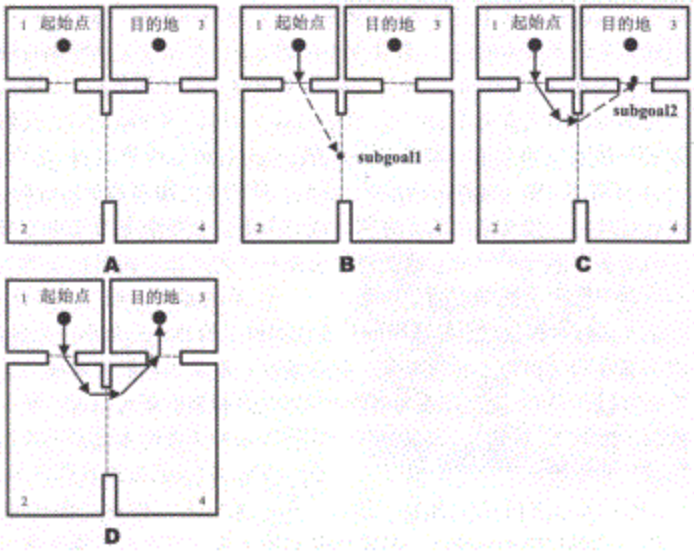


图 18-21 分层寻径的例子（本例子部分取自 Steve Rabin 的论文“A* AestheticOptimizations”）

图中有四个屋子，标号为 1、2、3、4。屋子之间有门，如虚线所示。我们要从 1 号屋子里的起始点走到 3 号屋子里的目的地。我们可以把地图分为两层：第一层只包括屋子和屋子之间的关系(门)，不考虑屋子里的细节。这层最简单。第二层是屋子内部的细节，比如桌椅等，这层比较复杂。然后我们采取以下步骤：

1．对屋子使用 A*算法，找到屋子 1 到屋子 3 的路径为 1-2-4-3。

2．设定中间目的地 subgoal1，它位于 2 号屋子和 4 号屋子之间的门上。对 1 号和 2 号屋子的内部细节使用 A*算法找到从起始点到 subgoal1 之间的路径，如图 B。

3．让游戏角色走到 2 号屋子的门外。

4．确定中间目的地 subgoal2，它位于 4 号屋子和 3 号屋子之间的门上。对 2 号和 4 号屋子的内部细节使用 A*算法找到从现在位置到 subgoal2 的路径，如图 C。

5．让游戏角色走到 4 号屋子的门外。

6．对 4 号和 3 号屋子的内部细节使用 A*算法找到从现在位置到最终目的地的路径。

7．走到目的地，如图 D。

使用这种方法，可以极大地简化 A*算法。因为，如果我们不分层处理的话，一开始我们就要面对四个屋子里面复杂的内部结构，如柱子、桌椅等。这些细节无疑会使得 A*算法手忙脚乱拙于应付，而分层次处理，第一步只考虑屋子和屋子之间的关系，不考虑屋子内部的细节，这样 A*算法可以很快地把大方向确定。当确定了全局最优的路径是 1-2-4-3 后，

再考虑每个屋子的内部结构，找出局部最优路径。

除了分层寻径的方法外，人们还想出了许多其他优化 A*的方法。所有这些努力，都是为了使得 A*算法在面对复杂情况时能够迅速快捷地找到最优路径。因为对 AI 程序员来说，他们所能获得的 CPU 资源总是不够的，提高一个算法的速度和效率永远都是他们孜孜以求的目标。

简单的地形分析

前面我们在考虑寻径问题和评估函数时，只考虑了空间的距离，而没有考虑不同地形之间的差异。比如趟水过河和利用桥梁对 NPC 的体力消耗不同，而沼泽地和干地对战车的行进速度也会有影响，更不用说如果大路边有个炮台必须回避了。这些因素都会影响路径的选择，都是应该考虑进来的。

一个简单的方法是把每种地形赋予一定的加权值，然后把各种地形的加权值和其图片数相乘，再取和，以此作为这条路径的成本，可以供 A*算法的评估函数使用。比如说，一条路径要通过 4 个沼泽地的图片单元和 5 个干地的图片单元，而沼泽地的加权值为 6，干地的加权值为 3，则整条路径的成本为 4x6+5x3=39。另一条路只经过干地，但要走 10 个图片单元，其成本为 3x10=30。虽然前一条路的实际距离短，但由于经过沼泽会造成游戏角色更疲劳，造成其成本略高，显然应该走后一条路。

使问题更加复杂的是在 RTS 中各种游戏角色的属性不同，它们对地形的依赖程度也不相同，比如轮式装甲车和履带式装甲车。不同游戏角色之间速度也不同，比如步兵和骑兵。这样的话，对各种不同的游戏角色，同一地形有不同的加权值。所以 A*算法不仅需要考虑地图的划分和属性，也得考虑不同游戏角色属性的差别。

更复杂的地形分析方法，则要考虑更多的战术性策略性的问题，例如什么地方敌人有可能设下埋伏，什么地方容易造成阻塞(桥梁和小路的路口)等等。目前使用比较多的两种技术是感应地图法(influence maps)和可视图法(visibility graphs)。在《帝国时代 II：The Age of Kings》中就使用了感应地图法，其具体细节略去不表。

移动问题

我们使用 A*算法解决了寻径问题之后，似乎可以往椅背上一靠双手枕于脑后长舒一口气了。但是且慢!找到最优路径是一个问题，而如何让游戏角色走这条路径则又是一个问题，而且这问题远非那么简单!我们将之称为移动问题，以和寻径问题相区别。

更加自然地移动

首先，在现实世界中的人和动物面对视野中可见的目标一般走直线，不会频繁地拐小弯，而 A*算法有时给出的路径是由很多的小弯组成的，如图 18-22 中 A 所示。这样看起来就觉得很别扭，非常不自然。因此，有必要对 A*算出的路径进行一定程度的修正，使其看起来比较舒服一点，弯弯绕绕少一点。目前有专门的算法来处理这个问题。而被“拉”直了一点的新的路径如图 18-22 中 B 所示。

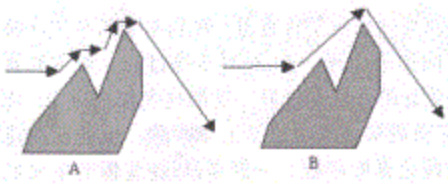


图 18-22 将 A★算法得到的路径拉直

单元协调移动

除了修改具体的路径使得其更自然合理外，当有两个以上的个体在地图上移动时，很有可能它们的路径交叉，从而相互碰撞到一起。这显然是我们所不愿意看到的，因此我们需要某种方法来协调调度，预测并避免碰撞。

首先定义单元(unit)的概念。单元，指任何类型的能够独立寻径并移动的个体，如一个步兵、一个骑兵、一辆投石车等。单元是组成团队和复杂队形的基础。

单元协调移动是指两个以上单元同时寻径和移动时，如何避免相互碰撞造成阻塞的问题。

●路径锁定(Path Locking)

最简单的想法就是当一个单元找到路径后，把这条路径上所有的位置都做上标记，使得第二个单元寻径时不能穿过这些位置。这种方法叫做路径锁定。因为每个单元寻径后就把自己的路径上的位置给锁住了，其他单元不能使用。这样碰撞自然不会发生了，因为它已经被消灭于寻径阶段了。但这个想法显然十分愚蠢，因为这样找到的路径是千奇百怪的，而且单元个数一多，很可能连可行的路径都找不到。但是，愚蠢的想法改进改进，往往成为很好的想法。我们可以不把一条路径上所有的位置都锁定，而仅仅是锁定单元前方的 3~5 个位置。这样的后果是短期的碰撞不会发生了。然后走 3-5 个位置之后，再进行寻径和位置锁定。这样进行下去，可以避免大部分碰撞，又能基本保证路径的合理。

●优先权系境(priority system)

另一种思想方法是不锁定路径，先对所有的单元寻径，然后开始移动，每走一步前都要考察是否下一步会造成碰撞。当出现碰撞时，我们不希望当事双方重新寻径，因为这加倍了计算量。我们希望两个实体可以礼让一下，一个停下或者向旁边让开一步，让对方先过去。双方究竟谁让谁可以随机决定，当然更好的办法是计算一定的优先权来决定。优先权(priority)的设定对单元协调移动来说是至关重要的。优先权决定了当碰撞发生时，当事双方谁让谁的问题。一般来说，速度慢的单元应该让着速度快的单元。而优先权也随时间

变化，当一个单元等待时间较长时，可以把它的优先权提高。这样可以避免有些优先权低的单元被堵在一个地方很长时间或者总是在给别人让路。

团队移动和队形

在解决了几个个体之间的协调和碰撞预防问题后，更复杂的问题是当一大群人移动时的协调控制问题。如果处理不好的话，很容易造成局部阻塞。玩过《帝国时代Ⅱ》的玩家都知道，当你编成好几支强大的队伍，意气风发地向它们发出指令让它们向敌人目标挺进，然后却发现它们在一个狭窄的路口上挤作一团，欲进不得欲退不能，白白成为敌人的炮灰时，真是欲哭无泪了。这就引入了团队、队形和协调移动等问题。

在单元的基础上，我们再引入几个概念。

●团队(group)：一些单元被组合成一个团队，可以一起向同一个目标移动或攻击，但单元之间的相对位置不固定。

●队形(formation)：在团队基础上，有一定的整体形状，最重要的是有了面的概念，包括正面、背面、和两个侧翼。无论是在移动中还是在战斗中，每个单元都要试图最大程度地保持自己在队形中的相对位置。队形是古代战争中最重要的部分，其目的是为了更合理地配置并发挥远近火力，保护自己。历史上有很多著名的队形(阵形)，如罗马军团的方阵、滑铁卢之战英军为了对付法国骑兵而采用的圆形队形、中国古代南北朝时著名的方圆大阵和半月阵，明朝戚继光的鸳鸯阵等。(也可用阵形这个词，但目前的 RTS 游戏中团队一般最多不超过 100 个单元，称为阵有点不够规模。)

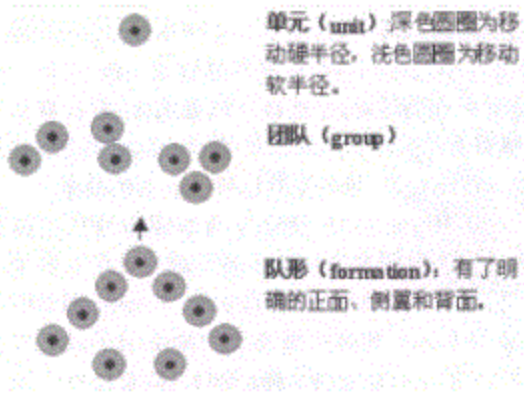


图 18-23 单元、团队、队形

对团队来说，由于队员之间不要求保持相对位置，所以整个团队也没有固定的形状，但是各队员之间的距离还是要保持在一定范围之内。也就是说团队还是有一个中心点(centroid)的，各队员和中心点的距离不能太远。由于不同的队员移动速度不同，当它们组合成团队后，一般整个团队使用最慢队员的最快速度移动。这和现实生活中差不多。有的程序员提议当一个比较慢的队员加入到一个团队后，适当提高他的移动速度，以免他影响整个团队的速度。比速度问题更重要的就是寻径问题，摆在我们面前有几种选择：第一种方法，以中心点为出发点寻径，然后所有的队员都沿这条路径移动；第二种方法，最靠前的单元寻径，后面的单元都跟着它走。

畜群算法

"… and the thousands Of fishes moved as a huge beast，piercing the water． They appeared united，inexorably bound to a common fate．How comes this unity?"

——Anonymous，17th century

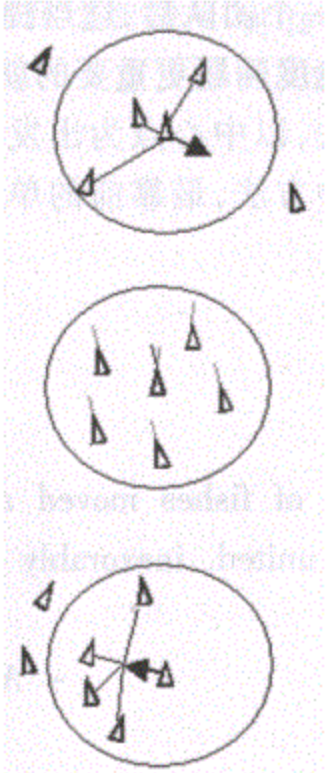
上面的英文是一位 17 世纪无名作家的感叹。他说“上千条鱼聚集，如巨大猛兽般移动，击破海水!如此一致，如此无情地命运与共。是什么使得它们如此团结?”他所观察到的，就是自然界中常见的群体行为。我们知道在自然界里动物们成群结队移动时，遵守一定的规律，绝对不会乱作一团。虽然群体内有老有小，有强有弱，但都能够协调一致，速度控制适中，大方向正确。移动时壮观的景象，往往让人叹为观止。畜群算法(flocking algorithms)，又称蜂群(swarming)或者牧群(herding)算法，就是一种试图在计算机里模拟群体行为，特别是群体移动行为的技术。

这种技术最早是由 Craig Reynolds 在 1987 年的 SIGGRAPH 会议上的一篇论文中提出来的。他当时研究这种技术的目的是要在三维动画中模拟复杂的群体行为，比如说鱼群。在那篇名叫“Flocks，Herds，and Schools：A Distributed Behavior Model”的论文中他提出了以下的想法：如果有一群能够自行其事自由移动的个体，我们只需要使每个个体服从于三个附加的法则，就可以使由个体组成的集合具有像自然界的鱼群和鸟群那样的群体行为能力。这三个附加的法则，又被称为群体行为的三个特性，就是：

- 间距性(separation)
- 齐向性(alignment)
- 内聚性(cohesion)

其具体图示和说明如图 18-24 所示。

请注意图中的圆圈所圈定的范围是一个个体的可视范围。也就是说一个个体无需知道这个群体的整体情况，它只需知道自己圆圈中的邻近队友的情况就可以了。正因为如此，这种模型被称为基于个体的模型(individual-based



间距性，就是说个体应当避免和自己的队友距离太近，造成阻塞．如右图中中间的个体发现自己和一个队友太近，因此需要向外移动。

齐向性，就是说个体应当和自己附近的大多数队友保持相同的方向和速度。右图中中间的个体发现自己方向偏离，因此需要修正．修正的根据是附近的队友们的平均速度和方向(矢量)。

内聚性，就是说个体也不能离自己附近的队友太远。这是通过计算附近队友的平均位置并向其靠拢而实现的。

图 18-24 群体行为的三个特性。中心的个体是我们目前要处理的个体。圆圈是其可视范围。圆圈中的其他个体是其邻近队友

model)。其特点是个体只需处理局部信息，无需掌握全局信息。

畜群算法的实现并不太困难，一般来说用 C++构造几个类就可以了。目前 PC 游戏中有很多都借鉴了这一思路。比如《Unreal》和《半条命》中都用这种方法来控制大批 NPC 的行动。Westward Studios 的《Enemy Nations》中也用了畜群算法来控制小组队形。使用畜群算法最合适的场合是《帝国时代》类型游戏的地图上的动物群了(虽然《帝国时代》里动物太少根本称不上群)，另外专门有一类放牧类的模拟宠物的游戏。

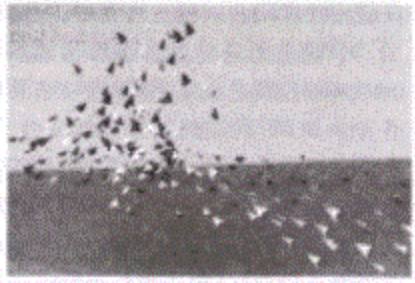


图 18-25 使用畜群算法实现的动画(本图出自 Craig Reynolds 的论文)

AI 技术应用专题

三维射击游戏的地形分析技术

之所以把三维射击游戏单拿出来作为一节，来介绍它们所采用的地形分析技术，是因为对三维射击游戏来说，AI 是其成功的最重要的因素。可以说 AI 基本上是在三维射击游戏上发展壮大起来的。三维射击游戏的 AI 最主要的体现就是 NPC 是否可以依托不同的地形地貌来和玩家对抗，或者由玩家控制的小组是否能够机智地执行玩家所发出的指令，根据地形来展开行动。

首先我们来谈谈如何来表达一定的地形信息。当关卡设计师们(level designer)设计三维地图时，他们要构建三维迷宫的内部模型，包括各个房间、走廊、大厅等。这些三维模型是以多边形的形式存在的。而由这些多边形构成的三维地图显然过于复杂，复杂到无法进行地形分析的程度。面对成百上千个多边形，如何高屋建瓴地把握地形信息呢?目前比较常用的一种方法是中继点图(waypoint graph)。

中继点图是对由多边形构成的三维地图的简化和浓缩。中继点(waypoint)，顾名思义，就是在一条路的中途的休息点。比如说从电梯到本关卡 BOSS 所在的控制室的中途，可以有几个地方有补充能量的物品。显然这几个地点对进攻路径的选择和进攻策略的采取都是非常重要的。AI 中的中继点图，则是把这种概念扩展。简单地说，就是在三维迷宫里，在成千上万多边形中，只找出对于地形地貌最重要的点，然后把这些点连接起来，形成图(graph)。有了这个图，无论是寻径还是地形分析，都会简单多了。图 18-26 就是中继点图的例子。

图 18-26 所示的中继点图是高度概括的。比如整个一面墙，用头尾两个点就表示了。这样简化是有道理的：首先，它不影响实际寻径算法的应用，因为我们在寻径过程中遇到这堵墙的话，我们最后也要归结到这堵墙的两个边上。其次，它极大地加速了寻径算法的运行时间。由于中继点数很少，寻径算法的搜索树比较矮，很快就可以找到路径。而最重要的，我们可以把不同的中继点赋以不同的属性，也就是说我们可以定义不同类型的中继点。比如说图中就可以有这么几种不同的中继点：窗口，看得到里面，但无法逾越；门，既看得到里面，又可以穿越等等。图中不同颜色的圆圈就代表了不同的中继点。

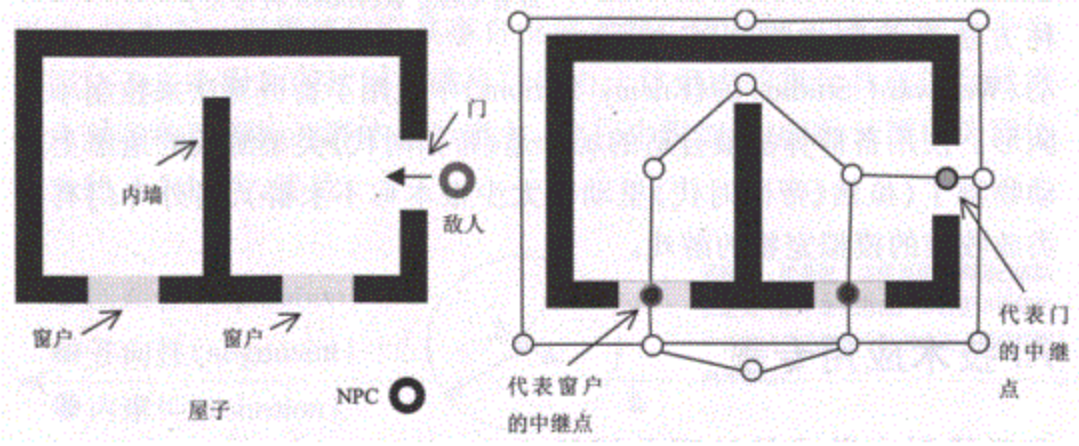


图 18-26 左图是一个屋子，有内墙、窗户和门。一个敌人正进入屋子。我方的 NPC 在下方。右图是《半条命》中使用的中继点图的示意图，由一些中继点和它们之间的连线组成。我们可以看到这个图十分简单浓缩，但基本上表达了有关地形的信息

我们更可以为中继点选定复杂的数据结构。这样关卡设计师们在设计关卡时就可以计算设定好各种信息，比如说这点的灯光和明暗程度(对射击游戏来说很重要)，这点代表何种地形地貌，这点到其他点的距离，周围环境概况(是否靠近隐蔽所或者周围有补血的物品)，从这个中继点可以看到哪些中继点等(是否是比较好的打冷枪的地点)。把这些信息存储到文件中后，在游戏时就可以取出来应用，使得计算量减轻。这样中继点就成了我们分析地形并决定策略的重要工具。

前面的中继点图的例子是一种最简化的版本。我们也可以用更多的点来更精确地表达地形地貌信息。如图 18-27。

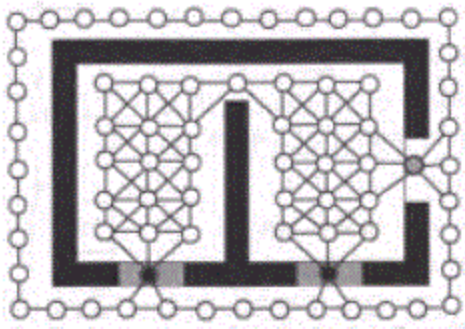


图 18-27 更加细致的中继点图。点数多了，能够更精确地表达地形地貌特征，但同时也会加大寻径的难度和计算时间。因此中继点图的复杂程度，最终要根据实际情况调试决定

在我们确切地知道了周围的地形地貌后，就可以用一系列办法根据地形来决定战术策略了。可以用很简单的方法，当然也有很繁琐的方法。比如说有人专门搞了个公式，来作为一个点是否是好的狙击点的判断标志。数值越高，则这个点越好。然后在游戏中反复试验，来修正这个数值，最后得到比较好的方案。

在此限于篇幅，就不深入介绍这些具体方法了。

三维射击游戏的小组 AI

所谓小组 AI(team AI)，指的是一个小组中不同的组员有不同的分工，肩负不同的职责，他们相互配合，共同完成一定的战术动作，达成一定的战术目标。小组 AI 最重要的一点是不同的组员要有不同的身份和职责，从而体现更高层次更具多样性的战术策略。比如说当一个小组面对正面屋子里的敌人时，可以由狙击手负责掩护，使用远程武器封锁住对方的窗户；由少量队员负责正面佯攻；而真正的突击队员，则等待时机迂回到侧后使用大威力近距离武器突击。这样的游戏将更紧张刺激，也更有挑战性。

早期的三维射击游戏多为单人游戏，在《DOOM》中玩家要一个人单枪匹马地面对所有的敌人和挑战。而发展到现在，多人组队已经越来越流行了，比如《Soldiers of Fortune》等以小组(squad)为基础的游戏。无论是玩家和友方 NPC 组合，还是几个玩家在网上组合，要完成一定的战斗任务都迫切需要更高级的小组 AI。这个问题也已经越来越被程序员们所重视。

一般来说实现小组 AI 要使用分层结构(hierarchical architecture)。而这个分层结构由三层组成，如图 18-28 所示。

更加具体地解释一下图中的几个概念。任务(goal)，就是由更高一级传达给小组的命令，表明了需要小组达到的行动目标和效果。举例来说，任务可以有以下几种：

- 沿指定路线巡逻
- 守卫某个建筑或单元
- 移动到一个新的地点
- 向某处发起进攻

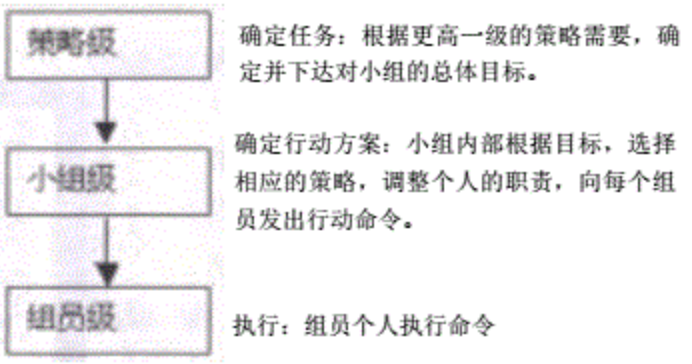


图 18-28 小组 AI 三层结构示意图。使用这种结构来组织 AI 比较有条理

上面这些任务是一些简单任务。我们也可以把任务加上比较复杂的规则，比如说巡逻可以有两种，一种尽量避免离开巡逻线。这样如果碰到敌人后，只是简单地驱逐。另一种则要把碰到的敌人追到天涯海角赶尽杀绝后才回到巡逻线。

行动方案(action plan)：是由一系列分工和具体行动步骤组成，用来完成任务。一个简单的行动方案(有所省略)如图 18-29 所示。

我们可看到行动方案是由总任务(goal)、分任务(sub-goal)和具体行动(action)组成。总任务是小组要达成的目标，总任务可以细分成一系列分任务，分任务再细分成了最小的步骤，也就是具体的行动，由组员来执行。

身份(role，也可翻成职责)：身份决定了一个小组成员采取什么样的行动。比如埋伏、掩护、攻击、移动等。

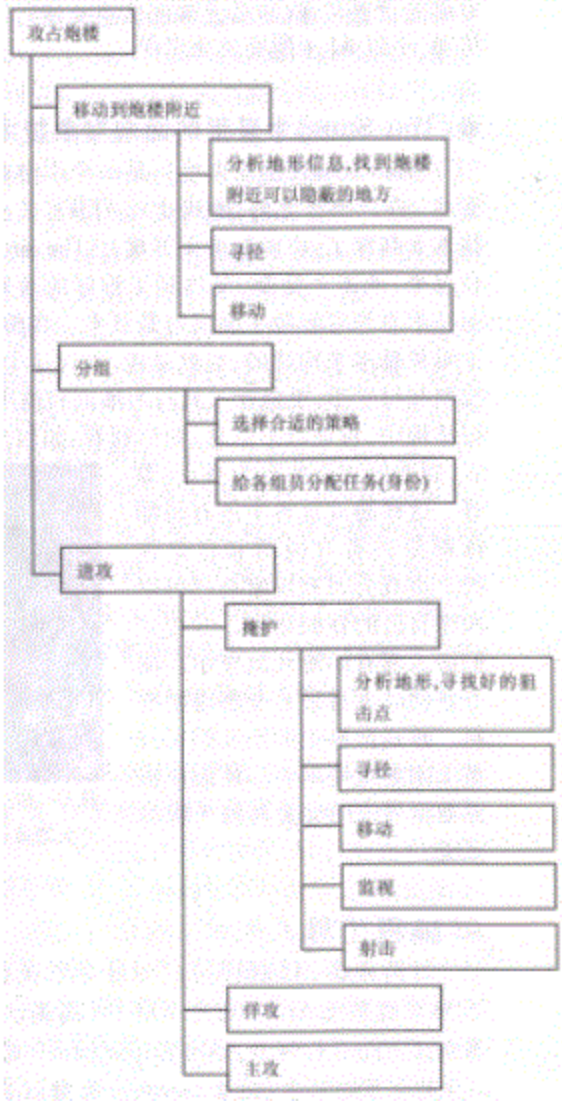


图 18-29 行动方案

前面介绍的三层结构，是一种自顶向下的结构，其优点是决策过程简单，作为组员的个体只要执行命令就可以了。我们可以很容易地用C++的类来实现这种结构，比如说我们可以建立一个小组类(squad)，再建立一个士兵类(soldier)，然后在分别构造对应于任务、行动方案和身份的类。这时候要注意的是这些类之间的信息传递。一般来说信息传递是通过类指针实现的。在小组 AI 的结构中，既有纵向信息传递(自顶向下下达命令)，也有横向信息传递(组员之间的信息交流)。只有很好地表达这两种信息传递，小组 AI 才能更灵活多样。

《The Sims》中采用的面向对象技术

《The Sims》(虚拟人生)是一个在游戏 AI 的发展历程上很重要的游戏。在其推出之前，游戏业对 ALIFE 技术有点失去了耐心——ALIFE 技术太高深了，应用起来太困难。《The Sims》通过把 FuSM 和 ALIFE 结合起来，既不太复杂，又达到了很好的效果，而在《The Sims》中最基本的技术思想实际源于面向对象技术。在游戏中所有的物品都是使用基于对象技术来构建的，也就是说，不仅这个物品的数据结构中有其三维模型和材质图，更包含了其行为准则和使用说明。比如一台电视机的数据结构中，包含以下信息：如何观看，如何开和关，在何种情况下可以观看，观看时观众的行为举止，等等。这样游戏世界中所有的物体都是具有自说明

图 18-30 《The Sims》中使用了面向对象技术使得游戏具有无限的扩展性

和自足性的。游戏公司和玩家自己可以构建自己的新的物品，并把它们加入现有的游戏世界中。由于新物品本身带有全部说明资料，游戏世界中的小人们可以毫无困难地使用它，从而使得游戏世界中的交互具有无限的可能。

AI 编程工具

目前来说，AI 程序员们还是处于独自为战的状况，AI 编程工具(SDK)，或者说 AI 引擎的应用还不普遍。在近两年有几个公司曾经试图在这方面进行开发，做出商业化的 AI 编程工具供 AI 程序员们使用。其中最具代表性的，就是 DirectIA 和 Motivate 了。

法国 MASA 公司的 DirectIA 是一个基于代理(agent)技术的 SDK。这套 SDK 可以用来生成自主代理(autonomous agents，或者称之为自主智能单元)或自主代理群体(agent group)，从而使游戏中的角色具有一定的自主学习性和适应性，使得它们可以具有感知能力和反应能力。值得一提的是 DirectIA 使用的不是 FSM 和 FuSM 等基于规则的 AI 技术，而是基于生物和认知科学(cognitive science)的相关模型的。DirectIA 的另一个附带产品是 DirectIA 嵌入系统。这是个简化的系统，可以集成到硬件上，使用到玩具中，使得这种玩具具有一定智能性。DirectIA 的功能单一，适用面比较狭窄，它的主要卖点是其自主学习性的功能。

另外一个很有名的产品就是 Motion Factory 公司的 Motivate。这是一个从机器人技术和实时控制技术研究成果中转化来的 AI SDK。使用这套 SDK，游戏角色的动作不是事先定得死死的动画的回放，而是根据环境和物理法则而决定的更加真实的动作。游戏角色的行为不是一堆 if-then-else 控制语句，而是一个复杂的层次 FSM 系统(hierarchical finite-state-machine)。使用这个层次 FSM 系统，AI 程序员和游戏设计师们可以给游戏角色设计复杂的行为规则。这个系统还提供了类似于 Java Script 的脚本语言工具。在《波斯王子 3D》中就使用了这套系统，当时作为一大卖点。但遗憾的是，由于价格的原因和大多数 AI 程序员的抵制，这套系统卖得不好。现在 Motion Factory 公司已经停止了开发工作。

总的来看，AI 引擎的开发和推广遇到了不少阻力。但是可以肯定地说，随着游戏越来越复杂，玩家对 AI 的要求越来越苛刻，AI 程序员们再这么闭门造车各自为战是不行了。专业化的高性能的 AI 引擎必将会得到业界的重视。其实我们仔细想想 3D 引擎的推广和流行，也是经历了同样的曲折。一开始大家都不愿意用别人的引擎，都愿意自己内部开发一套自己用，到后来终于不行了，3D 技术越来越复杂，独立开发 3D 引擎成本越来越昂贵，这时 QUAKE 的 3D 引擎开始流行。到现在很多公司都愿意使用商业引擎来开发了，都把使用高性能的商业 3D 引擎看成天经地义的事。作为游戏编程两大支柱之一的 AI，也会走到这一步的。

机器人技术和智能化玩具

一说起机器人和智能玩具，人们自然会想起最近炒得很热的 SONY 的机器狗 AIBO。作为售价 1500 美元以上限量发售的超级“玩具”，AIBO 确实代表了智能化玩具发展的趋势。1999 年推出的第一代产品 5000 只在 4 天之内全部售完，而加强版全球限量发售 1 万只，收到的预定却是 13 万，结果不得不进行抽签。

AIBO 本身的功能是十分强大的。主要包括了机械功能：可以行走、追逐、摇头摆尾，认知功能：通过摄像镜头、触觉传感器、听觉传感器，感知周围环境和事物；情感功能：可以通过 LED 显示、声音和特定动作表达自己的心情；学习功能：通过训练，可以学会一些技巧；人机交流：所有前面的功能，都要组合起来，为机器狗能够更好地和人进行交流服务。

比 AIBO 机器狗本身更重要的是 SONY 为 AIBO 所开发的一整套硬软件开发工具和平台——OPEN-R 体系结构。一般的工业机器人都是为了生产线上某项单一任务而设计的，因此其硬件软件相对来说极为专门化，适用性窄，不具有扩展性。娱乐机器人则恰恰向反，需要有很好的适用性和扩展性，才能满足人们娱乐的需要。OPEN-R 就是为了满足娱乐机器人的独特要求而设计的基于部件技术(component)的模块化结构，开发人员可以根据需要对机器人的硬件(物理属性)和软件(行为规则)进行调整。硬件模块之间可以相互自由组合。比如说，要设计一个履带底盘机器人，只需要把 AIBO 的四足拆下，换上履带模块就可以了。

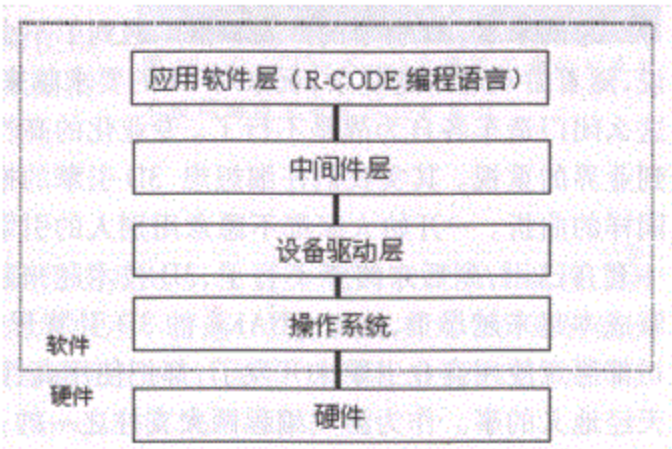


图 18-31 OPEN-R 的软件结构

OPEN-R 的软件结构如图 18-31 所示。

其中 R-CODE 是一种简单的编程语言，可以用来对机器人的行为特征进行调整和控制。

通过 AIBO 的例子，我们看到未来的高档娱乐机器人一般要具有以下特性：

●具有感知周围环境的能力：通过使用镜头和各种类型的传感器，使得玩具能够感知周围环境。对周围环境的感知能力是赋予玩具更复杂的行为功能的基础。

●具有和人交流的能力：通过语音识别、语音合成和姿势识别等技术，使得玩具能够更

容易懂得人的意图和情感状态，并能用人类的语言来和人交流。

●具有和其他玩具交流的能力：通过红外或者无线技术，使得玩具和玩具之间能够相互交流，并产生一定的交互。

●具有一定的自主智能：包括判断、归纳、学习、适应等。

●具有一定的情感：通过情感模型的建立，处理并表达一定简单的情感。

●具有复杂的机械活动能力：通过机械部分，实现复杂的行走、平衡等功能。

除了高档的娱乐机器人，一般的智能化玩具目前也越来越流行。MIT 的媒体实验室(MIT Media Lab)在这一领域做了很多研究工作。他们有一个很大的研究项目叫做“明天的玩具”(Toys of Tomorrow)。他们的很多研究成果都已经商品化了。比如说名为“真真宝贝”(My Real Baby)的洋娃娃。她是一个具有一定的人工智能，能够和小主人进行情感交流的机器人偶。并且随着小主人对她的呵护，她本身也会渐渐地“长大”，逐渐具有独立的意志和好恶。

图18-32由MIT和玩具公司联合开发的“真真宝贝”玩偶

以上介绍的各种拟人拟动物的娱乐机器人，其主要目的是培养起作为宠物的机器人和其主人之间的亲密关系，这只是玩具的一个功能。玩具的另一项主要功能是培养孩子们的创造力和想象力。传统的积木和 LEGO 玩具属于此类——通过组合产生无穷的变化。MIT 的另一个研究人员想到：为什么不把 LEGO 玩具智能化呢？把计算机芯片加到积木里面，这样 LEGO 玩具本身就变成了可以通过组合具有各种功能的娱乐机器人。在 MIT 和 LEGO 公司的合作下，MindStorms 诞生了。这套玩具的成功之处在于它把对一般人来说很复杂很难理解的人工智能和机器人技术(如机器人视觉)转换为一个个易于理解易于组合的模块，从而使得玩家可以从大的方面掌握这些功能模块，并设计出自己独特的娱乐机器人系统。

图 18-33 使用 MindStorms 模块构建的机器人

管理篇

第十九章　软件工程基础

本章主要介绍软件工程的基本概念和方法，它们是游戏开发具体实践中使用的各种方法的理论基础。

软件工程

软件工程(software engineering)和计算机领域所有其他概念一样，众说纷纭并且没有一个确切的定义，各种“权威”的定义起码有 10 种以上。简单地说，软件工程就是试图以成熟的工程化的方法来开发软件。我们知道，人类自古以来就有各种大型的建筑工程，比如古埃及金字塔和古罗马的大竞技场，都是著名例子。在这么浩大的工程中，成千上万的员，千头万绪的任务，必然要使用工程化的方法来组织协调规划，否则是无法完成的。工程化方法要处理的，不外乎以下几个问题：人员、过程、方法。首先是人员，需要驱使奴隶或者雇佣城市平民，而且要把他们组织起来，分配不同的任务。过程：罗马非一日建成，众多任务肯定有先有后，如何确定这种次序?如何保证前面的任务被按时按质完成，不会对后面的任务产生影响?方法：使用什么方法来运石头?用什么方法来保证石头的质量(包括大小、形状、重量、强度)?如何协调、指挥、通信?等等问题。

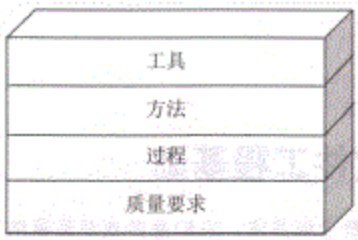


图 19-1 软件工程四层结构示意图

软件工程，就是在借鉴了从古到今的其他领域的工程方法实践的基础上，根据软件开发的特殊需要而提出的一系列思想方法和框架结构。Roger S．Pressman 在解释软件工程的定义和范畴的时候，用了一个四层示意图。这个四层示意图非常直观，有助于我们从大的方面理解并掌握软件工程，见图 19-1。

图 19-1 中最下一层是质量要求。对软件质量的要求和重视是软件工程的前提和基础，任何公司都是在有了提高软件质量的迫切要求后才开始重视软件工程的。第二层是过程。这是软件工程最重要的组成部分。过程就是一个由一系列任务组成的序列。每个任务包括目标、资源、功能、监控方法、完成标志和其他任务的关系等。这些任务所组成的就是一个框架结构。在开发软件时遵循这个框架结构，就能够基本保证项目的完成。第三个层次是方法。方法指的是对过程中各项任务和其细节如何实现。比如说质量控制的方法、进度预测的

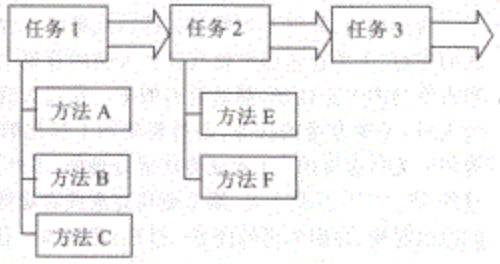


图 19-2 过程和方法的关系形成一种二维的结构。过程是横向的主干，方法是纵向的枝叶

方法等等。同一个任务，可以有很多种不同的方法来实现，要根据具体情况选择使用。因此过程和方法的关系，实际上是树枝和树叶的关系，也可以用图 19-2 这样的示意图来表示。

最后一个层次是工具。对软件工程来说，就是可以帮助软件开发人员来实现过程和方法的各种有形的和无形的、自动的和手工的工具，比如说 UML、项目管理软件、版本控制软件、CSCW 软件等。

上述四个层次把软件工程所涵盖的各层面上的内容都非常有条有理地解释了。可以说这个四层模型本身就是对软件工程的最好的定义。下面我们就以先过程，后方法的顺序来详细介绍。

软件过程

前面我们已经解释过，软件过程(software process)实际上是一种结构性的框架，它所体现的是对软件开发的一种高级的策略性的解决方案。软件过程模型(software process models)是在研究和实践基础上，对软件过程的正规化的定义和表述。这些模型被大家普遍接受并理解，各具特色，可以在开发工作中根据实际需要选择使用。

我们先介绍三种偏理论的过程模型，然后介绍一种实用的过程模型，最后介绍未来游戏开发中可以采用的一种模型。

经典瀑布模型

经典瀑布模型，又称线性模型(linear sequential model)，是软件过程模型中最古老的一种，也是应用最广泛的一种。它的思想是把基本的设计开发任务按线性排列，各任务相互之间界限分明，前面一项完成后才能进入下一项。其示意图如图 19-3 所示。但这种经典瀑布模型对游戏开发来说有三个致命的弱点，分别是 ：

图 19-3 瀑布模型的简单示意图

1．这种经典瀑布模型是以代码为中心展开的。也就是说瀑布模型表达的是一个程序(代码)从设计到编写的过程，我们称之为代码路径(code path)。对一般软件来说，除了代码和文档，以及少量界面设计内容外，所包含的其他媒体的内容较少。而游戏开发不仅仅是围绕着代码进行的，还有各种其他复杂的媒体，要制作各种图像、动画、声音以及更重要的游戏设计。也就是说不只是有这么一条路径。还有美工的路径、游戏设计的路径，这些和代码的路径有重合、有交错，也可并行展开。这些是经典瀑布模型所不能表达的。

2．游戏设计师很难在项目开始阶段就搞定游戏的所有设计细节，也就是说他们经常要

在开发过程中修改她们的设计。瀑布模型没有考虑到项目早期的这种普遍的不确定性，反而要求在一开始分析阶段必须解决所有分析问题，设计阶段必须解决所有设计问题。这些要求在实际工作中是不现实的。

3．使用瀑布模型，只有到项目的最后阶段我们才能看到可以运行的程序。这对游戏开发来说是不能容忍的。原因有二：第一是经济原因，现在游戏业中通用的第三方开发模式，是由游戏开发公司提出一个游戏设计方案，然后由游戏发行商对这个方案进行评估。如果通过，则采取分期付款的方式，分阶段付给游戏开发公司制作费用。每到下一次付款时间，游戏开发公司都必须提供阶段性的演示版(demo)，让发行商进行评估，看看进度是否进展顺利，技术是否成熟。如果这个阶段性的演示不能满足游戏发行商的要求，他们可以停止资金投入，如果演示可以满足他们的要求，则可以继续投资，继续开发工作。一般一个游戏，起码要有3-4 个这样的阶段性演示版。软件发行商采用这种分期付款方式的主要原因是为了降低投资风险——如果游戏开发确实不顺利，则可以迅速撤资减少损失。如果游戏开发采用经典瀑布模型的话，投资者只有等到项目最后才能看到产品，这时候想撤资也晚了。这种风险是显然是作为投资者的游戏发行商所不愿意承担的。第二个是游戏设计方面的原因，如果只有到项目最后阶段才能有可以运行的程序，那么游戏设计方面的测试和修改将十分困难。因为游戏设计方面的问题，很多只有通过对可运行的程序的测试才能发现并改进。

正因为以上的三个原因，经典瀑布模型必须经过改进才能适应游戏开发的需要。人们同时也在思考其他的方法和模型。其中的一种和前面第三项中的分析息息相关，这就是原型法。

原型法和复进式模型

原型法，又叫：快速原型法。它的提出是为了解决软件开发初期的不确定性问题。其基本思路就是当我们对设计拿不准(比如说这个游戏要使用一种全新的游戏规则)，或者对新技术没信心的时候，通过构造一些短小精悍的原型(prototype)，去测试一下这些设计想法是否可行，技术是否容易实现。这样做省钱省时，可以及时得到反馈，对设计和开发做出相应的调整。因此，对游戏来说，原型就是在某种程度上对最终游戏的一个缩减／模拟版本，其目的就是为了测试构想中的游戏的某一方面的内容。

原型法的示意图如图 19-4 所示。

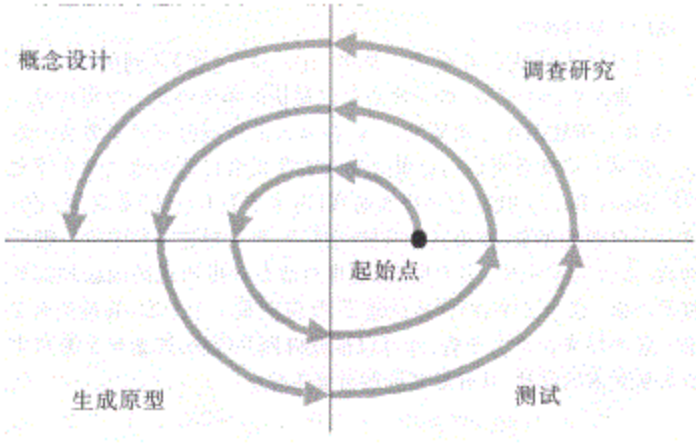


图 19-4 原型法和复进式设计示意图

原型法一般分四步进行，分别对应图中的四个象限。从右上角开始，第一步是调查研究，使用多种手段了解用户的需求，搞

明白有哪些不确定的因素需要用原型法来获得印证——是新的游戏规则?还是别出心裁的用户界面?或者是某项从未使用过的技术?然后进入概念设计阶段，就是在确定了要验证的目标后，在一个有限的范围内，设计一个小系统的功能和界面。然后进入开发阶段，要建造一个可以“运行”的原型。可以用最简单的方法，如纸上原型、故事板；或者用比较复杂的方法，使用 Director、Visual Basic 等原型开发工具；甚至使用 C++开发一个很强的技术原型。然后把这个原型交给用户，让他们去测试评估。根据获得的反馈，我们可以决定是否这个设计可行，是否技术上达到要求。

如果仅仅是一个循环的话，原型法的优势还不明显。如果我们多次循环，就可以将现有的原型同时向深度和广度展开。所谓广度，就是说经过循环，新的原型可以包括更多的功能，而深度是说新的原型可以使现有的功能更加细化和深入，使其更逼真，和系统的最终形态接近。这种原型法的多次循环，也被称为复进式设计(iterative design)，在人机界面设计上应用比较多。

原型法在游戏业有很多支持者，比如说《文明》系列的设计师 SidMeier。他在采访中透露：他每次都要以最快的速度构建一个很粗糙的原型，基本不考虑图像等表象的东西，但基本的游戏内容要素要有。然后上手测试，不断改进。马里奥之父宫本茂提出过著名的“宫本茂盒子(Miyamoto Box)”，就是对动作类游戏(由多个关卡组成)来说，先完整构建一个单独的关卡，使得其非常接近成品，然后对这个关卡进行彻底的测试，通过这个关卡，小中见大，发现游戏设计中出现的问题和需要改进的方面。这个关卡，即宫本茂盒子，实际上是一个原型，有深度而无广度。在评估完这个关卡后，就可以根据得到的信息，调整整个游戏中所有其他关卡的设计，并开始实际的开发工作。

渐增模型

渐增模型和原型法有某种程度的类似，都是基于多次循环往复的思路。但它和原型法最大的不同点是：原型法中建造的原型，不是最终游戏的组成部分，在帮助我们获得反馈和辅助决策后就没有用了。而渐增模型中所构建的部件，是最终游戏的组成部分。因此，原型法一般使用于设计阶段，而渐增法则被用于开发阶段。

目前使用的渐增模型中，比较流行的是自顶向下渐增模型。具体是这样实现的：第一次循环，我们完成所谓的游戏核心(core game)。这个游戏核心是整个游戏的大框架，而把游戏的细节部分规划好，但不实现。对 C++程序来说，就是类结构在那里了，但具体的成员函数没有真正实现，都是空的。然后第二次循环，实现一些中层功能，再一次循环，实现底层功能。这样每次循环，我们都有可以运行的程序，并且这个程序随着循环次数的增多发展壮大，直到达到最终产品的水平。

渐增模型的示意图如图 19-5 所示。

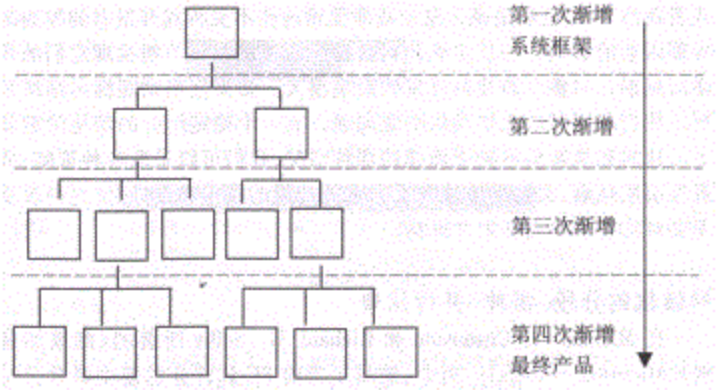


图 19-5 渐增模型示意图

图中左边是自顶向下设计的模块结构，第一次渐增先构建整个系统的框架。然后第二次渐增细化实现第二层的模块，第三次渐增实现第三层的模块，最后到最底层，所有的底层模块都完成后，我们得到的就是最终产品。

上面介绍的是自顶向下的纵向渐增模型，也可以先构建系统框架，然后按功能划分来横向渐增。比如说，一个 RPG 游戏，使用 C++语言，先把类结构和文件模块划分定义好后，从主循环开始，先实现图像部分，然后实现基本的操作(移动、对话等)，然后加入事件序列，最后加入战斗系统。使用这种策略，游戏的第一个版本中，主角可以在地图上到处走并和人对话；第二个版本中，可以触发事件了；第三个版本中，可以随机引发战斗了。

显而易见，渐增模型是一种非常稳健的方法。其主要目的是力图最快地使得我们拥有一个可以运行的系统，而且这个系统是最终系统的基础。和原型法比，渐增模型做的无用功少，因为中间“产品”要被集成到最终产品中的。另外，我们知道在游戏开发中经常有这种危险：所开发的游戏要用到现在没有的，预定于几个月甚至半年后推出的新主机或者新的 API。也就是说开发游戏要受到这些不受游戏开发者的控制的外部因素的影响，一旦主机公司或提供 API 的微软不能实现它们的承诺而延期，对整个游戏的开发的影响很大。如果使用功能横向渐增模型，我们起码可以先把其他功能完成，有一个稳定运行的简化的游戏了。这时如果我们遇到上面说的那种情况，我们可以采取两种策略：或者推出简化版，或者迅速改变受影响的功能的技术和设计，使这种改变不会对已有的游戏有太大影响。

微软的分段—缓冲—并行法

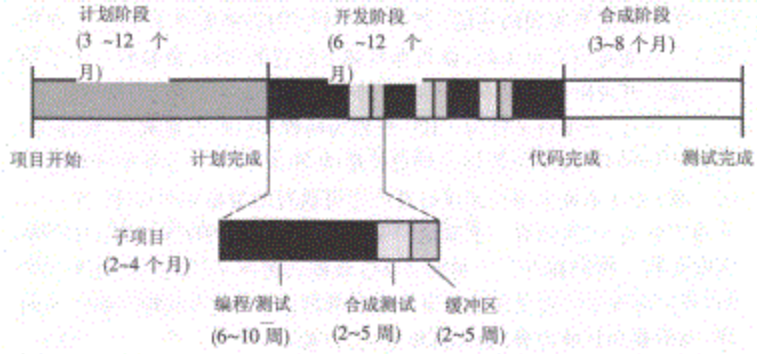
在 Michael A．Cusumano 和 Richard W．Selby 所著的《微软的秘密》(Microsoft Secrets)一书中，披露了当前 PC 软件业的霸主微软公司在软件开发方面所采取的方法和经验。微软所采用的软件开发过程模型是各种模型的综合，其中有瀑布模型，也有渐增模型，还有原型法。综合起来，其主要特点有三：

1．分段：把一个大项目，分成 3~4 个相对独立的子项目(subpro－ject)。每个子项目是

麻雀虽小，五脏俱全。这样易于控制，易于管理。

2．缓冲：在项目计划中插人多处缓冲区，以应付突发情况。因为对未来谁也说不准，无论是多么高级的预测方法，还是多么有经验的项目经理，都无法保证自己的计划是万无一失的。总会出现很多意外情况，总会有很多事先考虑不周的地方。因此，在进度计划中加入缓冲区，能够反应这种不确定性，从而使得计划能得到更加合理的执行。

3．并行：不同功能部门齐头并进，并在一定阶段互相协调。微软的软件过程模型如图19-6 所示。



	计划阶段	开发阶段	测试合成阶段
开发组	功能设计，技术可行性评估	编写/优化代码，编译，测试，去除 bug	总合成测试，去除 bug
可用性(usability)测试组	确定可用性目标	反复各种测试原型	使用精心设计的案例，实地测试
人机界面设计组	研究借鉴现有 UI 标准，开始初始设计	使用快速原型法和复进式设计，设计界面，最后完成各种图像和标志	最后的设计评估

图 19-6 微软的软件开发过程

这个图比较复杂，需要具体说明一下。在上半部分，是微软软件过程的图解。我们看到大的框架还是瀑布模型的，分三大阶段，分别为计划阶段、开发阶段、合成阶段。

计划阶段：实际包括计划和设计两方面。其成果是完整的开发计划和要实现的功能规格说明(functional specification)。计划阶段一般为 3~12 个月。

开发阶段：就是一般我们所说的编程阶段，但微软把这一个阶段分成了 3-4 个相对独立的子项目来进行，类似于渐增模型。首先需要把从计划阶段得到的功能规格说明中的所有需要实现的功能按重要性进行的排列，取出最重要的，必须实现的 1/3 功能。然后把这些功能作为第一个子项目需要实现的功能。然后第二个子项目再实现 1/3 功能，最后的 1 / 3 功能是可有可无的，就是那些锦上添花的功能，由最后一个子项目完成。开发阶段一般 6~12 个月，每个子项目 2~4 个月。

子项目：子项目又分为三段。分别为编程 / 测试、合成测试、缓冲区。据微软的经验，

缓冲区最好占到总长度的 20%~30%。每个子项目都自成一体，也就是说子项目完成后要产生可执行的很稳定的程序。第一个子项目完成后，我们有一个实现了 1/3 核心功能的程序了，第二个子项目完成后，我们拥有一个实现了 2/3 功能的程序了。最后第三个子项目，我们得到了实现了所有设计功能的程序。当然这个程序还有一些问题，也需要和其他内容，如人机界面，去合成。

测试合成阶段：这部分基本和其他公司的做法差不多，惟一的区别是也要插入一些缓冲区。

在图 19-6 的下部是用三个小组为例(实际有 10 个以上不同功能的组)，说明在图上部的框架下，各组是如何齐头并进的。在平行前进的同时，各组之间是需要不断地交流协调的，比如说可用性测试组需要人机界面组的原型去测试。人机界面组需要可用性测试组的测试结果作为改进设计的依据。

总的来看，微软的分段—缓冲—并行法是以瀑布模型为框架，在开发阶段同时采用了渐增模型(对开发组)和原型法(对设计组)。微软所采用的方法有两大优点：

1．把大的项目分成 3-4 个子项目，每个项目实现部分功能。先实现主要功能，再实现次要功能。这样如果在项目最后阶段出现拖期，可以牺牲一些不太重要的功能，保证软件按时上市。我们知道软件拖期是个非常严重的问题，没有什么好的解决方法。Fred Brooks 的软件工程经典著作《神秘的人月》(The Mythical Man-Month)中对软件拖期问题有精妙的阐述。他的观点是如果软件拖期，加入额外的人力资源是没有用的，反而会火上浇油。当软件拖期时只有两个选择：偷工减料，或者延期交付。在现实世界中，人们一般选择前者。微软的方法也是其一。但它保证了核心系统的完成，并保证了核心系统的质量，省略的功能也可以到下一个版本中再加入。

2．缓冲区的引人，给了开发人员更大的自由度和安全感，使得他们在以后的开发中不至于焦头烂额。缓冲区的存在，无论是对开发计划的实现和对进度的有效管理，还是对软件的质量，都起到至关重要的作用。

专门针对游戏开发的 Triptych 模型

前面所介绍的几种过程模型，各有其优劣。但它们都是在大的层面上的普遍模型，更不是为游戏开发量身定裁的。微软的方法，则主要是为开发 Office 这类商用软件而设计的，对游戏开发也不一定适用。众所周知，游戏开发和一般商用软件的开发有很大不同，其不同点有四：

1．商用软件以功能为主，游戏软件以游戏性为主：对 Word 和 Excel 这种商用软件来说，

功能是决定其成功的主要因素。因此在软件设计阶段，主要是定义软件的功能。当然人机界面目前也越来越重要。对游戏来说，功能不是决定游戏成败的关键，对游戏来说，也没有功能一说。决定游戏成败的关键是游戏性，而游戏性又是一个很复杂的没有确切定义的概念。这也就决定了游戏的设计，要比商用软件的设计(仅仅是决定功能和人机界面设计)复杂得多。因此，游戏项目中投入设计阶段的时间和资源，应该要比商用软件项目中投入设计阶段的多。

2．测试方面的不同：商用软件的测试主要是代码测试；游戏软件的测试既包括代码测试，也包括游戏性的测试和调整。因此，游戏项目中用于测试的时间应该比一般商用软件项目多。

3．投资方面的不同：游戏项目一般是游戏发行商和开发商共同投资，商用软件是开发商自己投资。显然微软在自己开发 Office 的时候不需要听别人说三道四。但游戏开发商在开发的时候却必须按期向发行商证明开发的成效——因为发行商是投资的大头。只有拿出令人信服的成果后，发行商才会分期付款资金到位。因此，游戏项目必须分阶段地拿出可以评估的成果。

4．游戏软件是严格的节日经济的产物：和一般商用软件不同，游戏软件的时效性很强，其销售完全为节假日所左右。在美国，就是圣诞节消费高峰期。据统计，美国人在圣诞节期间的消费是全年消费的 1／3～1／6。因此，游戏项目的最终底线就是圣诞节，如果不能在圣诞节前发售，那么游戏公司将血本无归。因此，对游戏项目来说，拖期的后果将是致命的，游戏项目比之商用软件项目更加不能容忍拖期!

这四个主要不同点，以及通过分析得出的四项结论，必然会影响到游戏开发中过程模型的使用。因此有必要把前面各种模型根据游戏开发的特殊需要而修改综合。

这里提出一种专门针对游戏开发的组合式过程模型，姑且叫做 Triptych 模型。这个模型的基本结构如图 19-7 所示。

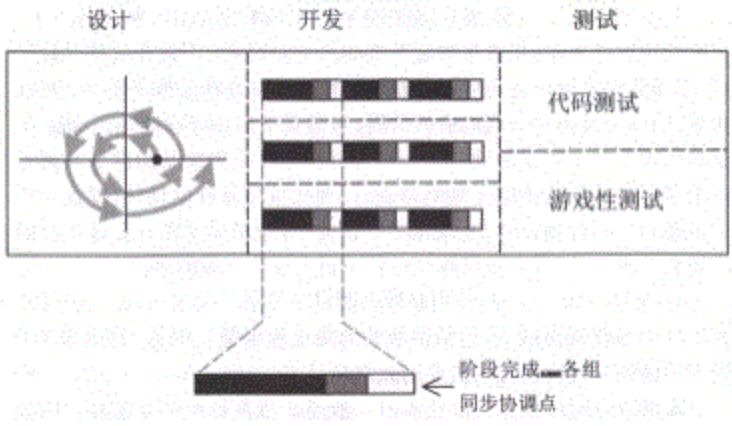


图 19-7 比较适合于游戏开发的 Triptych 模型

之所以起名为 Triptych 模型，是因为这个模型中设计、开发、测试这三个部分各占整个项目的 1/3 时间。在设计阶段，使用原型法和复进式模型；在开发阶段，采用的是微软的并行和分阶段子项目的方法，也就是渐增模型；最后是测试阶段。

那么这种模型的理论依据何在呢?其根据就是我们在前面分析的四点不同。这个模型与其他模型相比，其设计和测试所占的比重是很大的。之所以要加大设计和测试的比重，就是根据第一点和第二点分析。首先我们来看设计部分。

目前，美国游戏公司中对一个 12~18 个月的游戏项目，一般用 3 个月进行设计，然后用 6~9 个月左右时间编程测试合成。但实际上短短三个月要完成复杂的游戏设计，包括基本的游戏规则、人机界面、基本 AI 的考虑等，实在是勉为其难了。设计不完善就仓促上阵编程的后果就是程序员们写好一段程序后被设计师告知设计需要改变，于是他们的心血已然白费，必须推倒重来。如果把设计部分延长到 1/3，即 4~6 个月左右时间，则有可能得到一个比较成熟的设计。这样在开发阶段就可以少做无用功。与此同时，程序员们并不是闲着，他们需要进行技术上的准备，具体地说，就是要搞出一个技术演示(tech demo)来，用以测试新技术的可行性。

测试部分同样扩张到 1/3，其原因之一就是前面的第二点分析，游戏不仅需要代码测试，还需要游戏性的测试和调节。第二点原因是从软件工程的更广阔的背景上考虑的。在 Fred Brooks 的经典著作《神秘的人月》中，他所给出的理想的设计—开发—测试三者的比重是 1/3，1/6，1/2。也就是说，纯编程只占项目总长度的 1/6，测试合成占 1/2。听起来似乎耸人听闻，现实中也很少有项目经理是按这种教科书方法去制定计划的，但实际上很多项目最后都走到了这一步。1/3 也算是 1/2 向现实的一种妥协吧。

在开发阶段，基本上借用了微软的并行—分阶段子项目法，根据是前面的第三和第四点分析。分阶段子项目的渐增模型使得我们可以保证定期给发行商以可执行的程序，取悦于它们。在实在万不得以的情况下，我们也可以果断地做出决定，削减部分次要功能，保证游戏的按时发售。

总之，这个 Triptych 模型从理论上来说比较适合与游戏开发，至于实践中具体的成效如何，则有待时日来证明。

软件过程成熟度评估和 CMM 模型

卡内基美隆大学(Carnegie Mellon University)的软件工程学院(SEI)的 CMM 模型(Capability Maturity Model)，是一种用来评估超大规模软件公司里软件开发过程的成熟度的模型。对游戏公司来说，它们一般也就算中小规模软件公司，也许永远也不会达到那么大的规模和复杂程度，因此 CMM 模型不会真正被游戏公司去实行，但 CMM 的思想方法，对游

戏公司来说还是有借鉴意义的。

CMM 模型共分 5 级。具体说明如表 19-1 所示。

级数	含义
1 级：初始级	手工作坊式的软件开发，仅凭经验和直觉，没有完整的过程控制，产品的质量完全取决于个人的努力。如果核心人员离开，则他的经验也随之被带走。
2 级：可重复级	基本的过程控制的实现，对成本，进度和功能有了监测和评估。在过去项目经验的基础上，建立基本的规则。这些规则的确立，可以使我们把成功项目的经验延续下来，保证下一个类似项目的成功。这也就是可重复的含义。
3 级：已定义级	软件开发过程文档化，标准化，并且集成到整个公司的策略体系中。也就是建立公司统一的标准化软件过程。然后针对不同项目的不同要求，对这个统一标准软件过程进行具体化和细化并实话。所谓已定义就是说定义了标准软件过程。
4 级：已管理级	对整个软件开发过程的质量和软件本身的质量做量化的检测和评估，从而达到理解，控制，并改进开发过程的质量和软件质量的目的。已管理的意思就是实现了量化管理。
5 级：优化级	通过量化的测量和反馈，不断持续地改进软件开发过程，并引入新技术和新思想。

表 19-1 CMM 模型

我们看到，第四级需要对整个过程的质量和软件的质量做量化的检测和评估，并建立相应的数据库，这种量化的努力需要极大的财力为后盾，并需要长期不懈的投入，对游戏公司来说不太现实。起码在近几年，游戏公司们学习 CMM 第三级就可以了。学习了解 CMM 模型的目的，并不是去要真的去搞 CMM 评估。那东西对游戏公司绝对是华而不实的。我们也没听说微软通过了几级 CMM 论证，人家照样是 PC 软件的老大。了解 CMM 模型的目的，是理解它的思想和思路，而非其条条框框。择其有用者，择其精练者用之。

软件项目管理方法

软件项目管理基本上属于软件工程四层中的方法和工具层面。其内容包含软件度量(software metrics)、制定计划、风险分析、进度监控、质量保证和版本控制等。有些方法，如质量保证和软件度量，要横跨软件过程的几个阶段。

制定计划

制定计划是软件项目管理的第一步。用通俗的话来说，就是要在动手之前，搞清楚 5

个问题：需要做什么，如何做，什么时候去做，谁去做，怎么才算是做完了。下面的各项步骤就是围绕着这 5 点展开的。

●目标确定

制定计划的目的自然是为了执行计划，但是世界上没有任何计划是得到 100%执行的。这就出现了一个悖论——既然事先制定的计划肯定会被现实证明是不准确的，为什么还要花费大量时间精力去这么做呢?回答很简单：计划虽然永远不准确，有时偏差很大，但有计划总比没计划好。因此，制定计划的最终目标不是要使这个计划 100%贯彻执行，因为这是不现实的。制定计划的目标应该是建立一个框架，使得由上到下的监控成为可能，从而在实际项目过程中能够不断地动态调整和修正计划。

●任务细分

制定计划最重要的就是要对整个项目所需时间做出预测。但人类缺乏对未来的复杂任务的预测能力，即使他很有经验。相反地，人类对于细小的可控制的任务的预测能力还是比较准确的。正如没人可以一下子告诉你盖一栋房子需要多少时间，但人们可以准确地告诉你打一个钉子需要多少时间，劈一块木头需要多少时间，码一块砖需要多少时间。基于这种原因，预测一般要采取自顶向下的方法把大的任务细分成小的任务，直到我们能够对这个小的任务做出准确的预测，然后再反过来预测整个项目所需的时间。WBS(work breakdown structure，工作拆分结构图)图就是一种自顶向下进行任务细分的方法。一个简单的例子如图 19-8 所示。

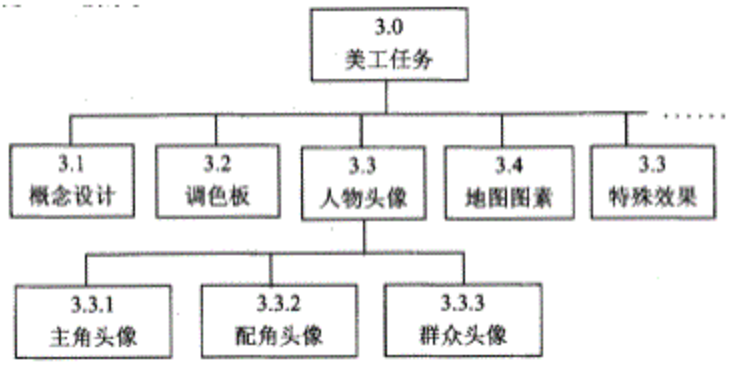


图 19-8 WBS 示意图

从图 19-8 我们可以看出 WBS 是一种层次化的，自顶向下逐渐细分的树状结构。顶层的是大的任务，如图中的美工任务，包括了游戏美工的一切工作。然后往下一层层细化，第二层把美工任务细分成人物头像、地图图素、特殊效果、物品器件等各项。然后再往下把人物头像细分为主角头像、配角头像、群众头像等。一般来说，细分到一个任务可以由 1 到 2 人在 1 周内完成就不用再细分下去了。

WBS 图最底层的树叶们就是基本的任务单元(task)。也就是说这些任务已经细分到最小程度，是我们可以精确预测的，是我们对整个项目所需时间进行预测的基础，同时也是我们

按人头分派任务的基础。一般来说，对每一个任务单元的具体信息要用一种类似于卡片的格式组织起来，叫做工作包(work package)。一个简单的例子如图 19-9 所示。

我们可以看到这个工作包卡片包括了关于任务单元的最重要的信息。其中几项重要条目的解释如下：

工作包编号	3.3.1
工作包名称	主角头像
具体内容	根据概念设计和调色版设定，完成二维主角头像的位图(.bmp 文件)
所需时间预测	1 周
所需资源	人员：1 美工 特殊工具：扫描仪
最终产品	位图(.bmp)文件
前提条件	3.1(概念设计) 3.2(调色版确定)
后继关系	11.0(合成测试)
完成标志	通过艺术指导的评审
风险	无
具体实行负责人员：_____	
起始日期：_____	完成日期：_____
实际成本：_____	

图 19-9 一个 WBS 工作包卡片

●所需时间预测：由于任务单元已经经过了细化，我们从而可以根据 经验比较正确地估计出完成这项任务单元所需的时间。这是后面 建立计划 / 进度表的基础。

●所需资源：为了完成这个任务单元所需要的各种资源，包括人员和工具。在后面的步骤中我们会看到如何用这些信息去调整计划 / 进度表，使得资源分配合理，避免出现对资源的争夺和撞车现象。

●前提条件：在进行这项任务单元之前必须完成的其他任务单元。

●后继关系：和这项任务单元有直接关系的任务单元，它们的前提条件是本项任务单元顺利完成。前提条件和后继关系是后面任务排序的基础。

●完成标志：决定本项任务怎么才算是完成了。

●风险：要完成本项任务单元所可能涉及的风险。在风险分析中需要用到这项信息。

另一方面，要建立 WBS 图，必须对整个项目的范畴和软件的内有足够的认识和了解。以上面的例子来说，就是必须对整个美工的工作有大致的认识，对需要多少头像、地图多大、游戏中有多少物品出现等等，都必须清楚了才能使用 WBS。因此 WBS 一般是在游戏的基本设计有了眉目后才使用的。

任务排序

在我们把一个大项目分解到 WBS 的各种任务单元后，这些任务单元之间有从属和先后关系。我们就可以开始采用某种方法，来对它们进行排序和安排了。有几种方法可供使用，甘特图(Gantt chart)，关键路径法(CPM)和 PERT 图。以下逐一介绍。

●甘特图

甘特图是一种很直观的图解方法，也是制定计划和调节进度时应用最广泛的一个工具。

如图 19-10 所示。

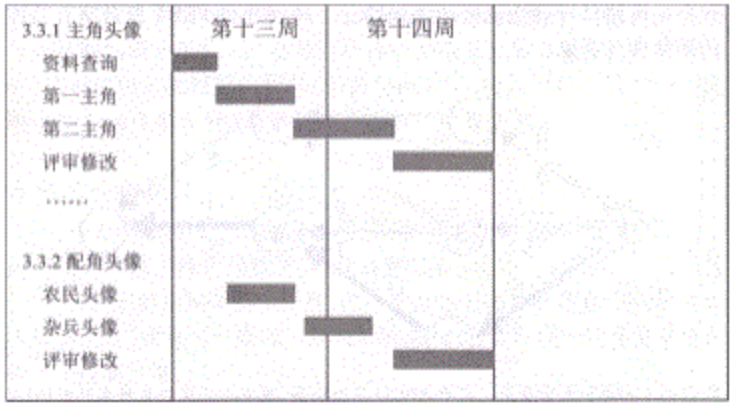


图 19-10 简单的甘特图

从图 19-10 中我们可以看出，由于甘特图是沿时间轴展开的，因此它可以明确表示各任务单元在时间上的重叠，因此有助于我们分配资源，避免在同一时间段多个任务单元使用同一个资源造成撞车，但甘特图不能明确表示各任务单元的前后依赖关系(谁是谁的前提条件)。一般来说，我们先制定 PERT 图或者 CPM 图，然后将 PERT 图或 CPM 图自动地转换成甘特图，然后再细调。

●CPM 方法和 PERT 图

之所以把这两种方法合起来介绍，是因为它们有很多共同点。CPM(critical path method，关键路径图)/PERT(program evaluation and re-view technique，计划评审技术)基本上是一种网状结构，由结点和箭头构成。在 CPM 中，结点代表任务单元，箭头代表任务单元之间前后顺序和依存关系；而在 PERT 中，结点代表事件，箭头代表事件之间的过程。任务单元和事件之间的关系是线和点的关系——任务单元代表了一段时间内所从事的工作，是一个过程；而事件则用来代表这个过程的起始两端。

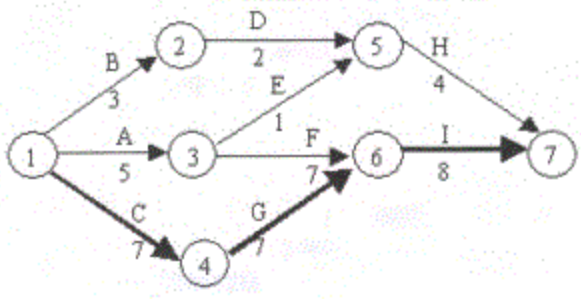


图 19-11 PERT 图示意图。图中 ABCD 等为任务，数字为完成任务所需的时间（天数）。1 为开始点，7 为结束点。箭头显示前后次序关系。通过计算，C-G-I=7+7+8=22 天，为最长。所以 C-G-I 为关键路径

有了 CPM/PERT 图后，根据每个事件最早可能发生的时间和可以容忍的最晚发生时间，我们可以计算出整个网络从头到尾所需的最长时间。具体的计算方法这里不介绍了，在软件工程的教科书中有更详细的解释。从始到终所需时间最长的路径就被称为关键路径(criticalpath)，如图 19-11 中黑线所示。

任务预测

在所有软件工程的任务中，预测是难度最大的。目前有两种方法，一种是采用量化分析的方法，比如从软件设计中的功能点(functionpoints)，推算出所需要的代码行数(LOC)，然后根据程序员的工作效率推算出项目需要的时间。这种方法是经典教科书的方法，有很多复杂的数学模型，但对游戏来说不适用。另一种方法就是把项目任务自顶向下，细分到比较容易预测的 WBS 工作包，然后由几个有经验的高手，一般来说一个主程和一个美术指导，分别估计每个程序或者美工 WBS 工作包的所需时间。

进度表细调

在估计完这些工作包后，再建立 PERT 图，找到关键路径，把位于关键路径上的 WBS 工作包填入甘特图，然后根据资源共享和人员配置的实际情况，把其他 WBS 工作包加上去，这样一个基本的计划就成形了。在制定甘特图时，主要考虑资源问题和人员配置要均衡，另外必须加入足够的缓冲区。一个在项目经理中不成文的规矩是在制定计划的时候，只使用现有资源的 85%(资源包括人力、时间、资金、技术等)，把剩下的 15%资源留给处理突发事件和项目拖期时用。

风险分析

风险分析是制定计划时的一个必需的部分，也是十分重要的部分。因为未来充满不确定性因素，充满各种风险，如果不事先想一想，准备一下的话，到时后难免措手不及，甚至导致整个项目的失败。所谓“不打无准备之仗”就是这个道理。战场上瞬息万变，对各种可能情况要准备好多个应对之策。

风险种类

软件工程中的风险，是指所有可能危及项目成功的可能发生的事件。可分为几类：进度风险、资金风险、人员风险、质量风险、技术风险等。

●进度风险

进度方面最容易出现问题的就是关键路径上面的各种任务了，如果不能按期完成，则肯定影响整个项目进度。另外，在 PERT 图上，扇入扇出度大的结点也是容易出问题的地方(扇出度：一个结点后面的直接连接的后继结点个数；扇入度：一个结点前面的直接连接的结点个数)。扇出度大的结点，在这个结点的任务完成前下面所有的任务都无法展开。扇人度大的结点，必需等待前面所有任务完成后才能开始。因此这些结点是风险最大的地方。

●资金风险

对高科技企业来说，很多都是借钱过日子。投资人分几步投资，如果他们觉得前景不明

朗，就会中途撤资。这时候就面临这本应到手的资金没有到，如何维持开发的问题。另外还有开发进度受挫时，面临拖期的危险。拖期必然导致需要更多的钱来维持开发。如何评估需要多少附加资金，如何取得这些附加资金，如何进一步说服投资者等问题，都在资金风险的范畴。

●技术风险

比如说新技术的采用，经验的不足，或者要用到某公司一个将在未来几个月内发售的API，都有可能引发不同程度的风险。以前曾经有过游戏公司过分依赖某公司的承诺——该公司在他们的项目开始前承诺了将在 6 个月后提供某软件的诸多功能，但后来该公司没有完成那些功能，结果这个游戏公司搞得焦头烂额，不得已自己重新写了有关部分，从而严重影响了游戏的发售。

风险预测

风险预测和控制，就是以周或者月为基础，将可能的风险因素列出来。每个风险因素要包括以下几项：可能发生的概率，产生后果的严重性(一般换算成金额)，近期还是远期，导致风险产生的主要原因。然后计算一个 RE 指数。RE=风险发生的概率 x 产生后果的严重性，然后按 RE 指数将风险排序，这样可以形成一个表。这个表是随时间(每周或者每月)经常变动的。就是风险预测的基本方法。风险控制是指根据这个表，去做准备，采取多种策略。比如说可以回避风险，转移风险，准备应急措施等等。

总之，风险预测和控制是个很重要但又难以把握的问题。在某种程度上，经验和直觉更重要。正如一个优秀的将领，在战场上知道在什么时候和在什么地方自己的军队达到临界状态了，并在这时果断地投入预备队。同理，一个优秀的项目经理，应该具有这样的素质：他知道什么时候这个项目有危险了，并果断地提出应对措施。

进度监控

有了计划，还要对计划的实际执行情况和进度进行监控。否则的话，计划就毫无用处了。

●进度监控环

进度监控有三个步骤：

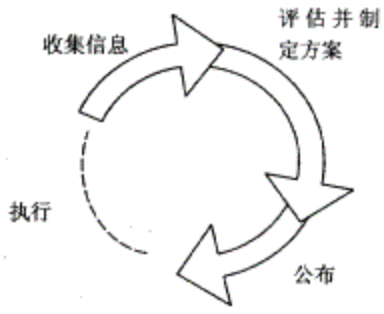


图 19-12 进度监控环

1．收集项目进度信息：收集关于项目进度的信息，包括各项工作的完成情况、资源的使用情况等。

2．评估并制定应急方案：根据收集到的信息，分析项目的现状。如果发现问题，则制定相应的应急方案。

3．公布项目进度信息和新的方案：公布项目进度信息和新的方案，使得组员们可以调整自己的工作。

随着项目的进行，这三步不断地被重复，形成了所谓的进度监控环。如图 19-12 所示。

●使用过程模型和 WBS 包进行监控

我们介绍过的过程模型和 WBS 包是我们对软件项目进行监控的两大基础。我们前面说过，过程模型是一个大的框架，这个框架由几个大块和它们之间的拓扑结构组成。这个框架是我们进行监控的结构性指示，但仅有大的框架是不够的。在 20 年前，Fred Brooks 在他的软件工程经典著作《神秘的人月》中这样写到："一个软件项目究竟是怎么搞到拖期一年还不能完成的焦头烂额的地步的呢?答案很简单：因为每天都延误了一点点。"他的观点得到了大多数软件工程师的赞同——进度监控必须从最基本的最微小的细节着手。所谓从细节来监控，就是说当项目延误了一天，一周，或者两周的时候，我们可以及时发现并有所动作，去把项目拉回到正轨上来。WBS 包就是这样的最基本的控制单元。通过对 WBS 包的监控，我们就可以从细节上把握项目的进程。

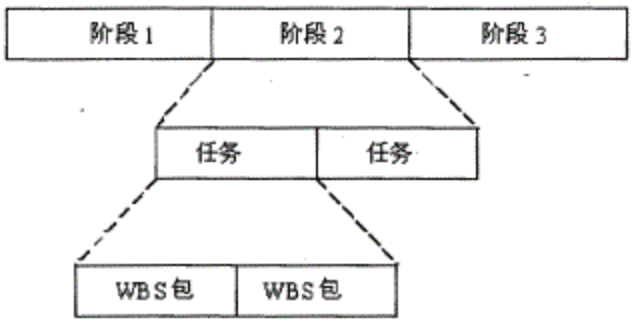


图 19-13 过程模型的各阶段，经过分阶段细分，最后形成的 WBS 包。这些 WBS 包是进度监控的基本单元。

对图 19-13 中的每一个单元，无论是最上层的阶段，还是中层的任务，还是底层的 WBS 包，每个单元都要有明确的启动条件(entry criteria)和完成标志(exit criteria)。启动条件是可以开始执行这项任务的必要条件，比如物质的条件和前面任务的完成。完成标志是考核这项任务完成的标准，如是否通过质量验收、是否完成了预定的产品和文档等等。

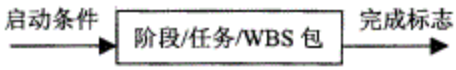


图 19-14 启动条件和完成标志是细节监控最重要的两项

最简单的进度监控，就是列一个大表，包含所有 WBS 包，然后定时检查它们的完成情况。所谓的二进制监控法(binary reporting)，就是指一个 WBS 包只能有两个状态，完成，或者还未完成，分别用 0 和 1 表示。在那个大表里，完成的 WBS 包填上 1 或者画钩。更复杂一些的监控方法，有 EV(Earned Value，赢值分析方法)等。对 EV 方法是否适用于游戏软件的开发项目，目前业界还有争论，尚无定论。

改动控制

正如人生是在不断的激荡变动中一样，软件开发的过程也是充满了种种变动。原因很简

单：大家都想把事情做得更好。随着项目的进行，以前不太明确的东西更清晰了，更好的方案出现了，人们自然希望改进以前的工作。这就带来一个重大的问题——如果老是这么返工，对已经完成的东西进行优化，工作永远也不会完成，产品也永远不能定型。另外，如果多人同时以一个文档为基础开展工作的话，其中一个人中途把文档改了，而其他几个人不知情，还在按旧的指标工作，这样的结果将是系统无法合成。所有这些，都是属于改动控制(configuration manage—ment)要解决的范畴。

●基准

要对改动进行监控，就需要让所有组员知道什么是可以随便改的，什么是不能随便改的，这样才能保持项目的基础不动摇。基准(baseline)这个概念就被提出来了。所谓基准，就是指软件开发过程中所产生的文档或产品，它们已经通过了严格的审核，被所有组员所接受并理解，从而成为了以后工作的基础，不能随便被修改。

一般来说，基准是和软件项目开发过程的框架结构息息相关的。我们所介绍的过程模型中，当每个大的阶段完成时，就产生相应的文档和产品，这些文档和产品就成为了基准。这个基准是对前一阶段工作的肯定，是下一阶段工作的基础。

●控制过程

基本的改动控制过程如图 19-15 所示。

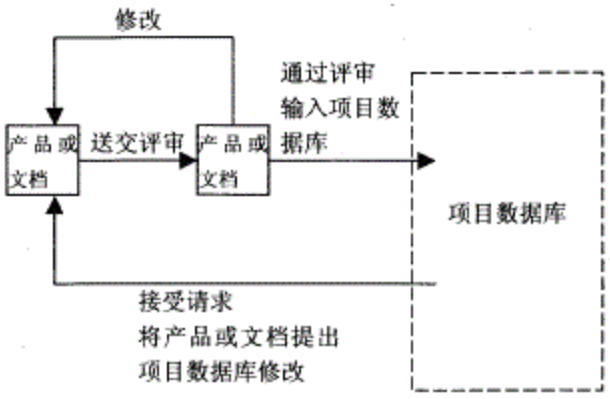


图 19-15 基本的改动控制过程

具体解释如下：一项任务完成后，必将产生一定的产品或者相关文档，这时就要送交评审，以确定它们是否达到了完成的要求。如果否，则还得返工修改。如果符合要求，则产品和相关文档被存入项目数据库，成为基准。进入项目数据库的产品或文档就不能随便修改了，如果实在有修改的需要，当事人应该向项目数据库管理者提出正式请求，然后由全体人员权衡利弊，做出决定。如果同意修改，则从项目数据库调出这个产品及相关文档，进行修改。

软件工程工具

软件工程工具，位于软件工程四层模型的最上一层。包括各种手工的(报表)和自动的(软

件)工具，用来帮助项目管理人员工作。下面简单介绍几个在游戏业应用比较广泛的工具软件。

UML

UML(Unified Modeling Language)语言是由 Rational 公司的 GradyBooch 等人开发出的一套图形化建模工具。主要用途有二：一是去构建和表达商业模型。一个商用软件，必须是要“嵌入”到一个组织的日常商业活动中的，才能发挥其作用。而在开发软件之前，必须先搞明白这个商业组织是如何运作的，要开发的软件在这个商业运作中的角色——也就是软件在这个商业运作的嵌入点是在哪儿。UML 提供了一系列工具，比如说使用案例建模(use-case modeling)，去图形化地表达商业的运作——即商业模型。这样软件开发人员可以更好地从大的层面理解商业运作和其需要，以此为基础去进行需求分析。

对游戏来说，由于是娱乐软件，显然没有这个商业模型的问题。因此，UML 在游戏业中的应用，主要是第二个用途，即作为一种图形化的工具来分析、建立、表达软件的系统结构。UML 是基于面向对象技术的，也就是 OOA 和 OOD(object oriented analysis and design)的工具。它的基本组成部分如下：

- 类建模(class modeling)
- 状态转换建模(state transition modeling)
- 模块建模(component modeling)

类建模所处理的是静态的抽象的类的定义，包括每个类的属性和行为，以及它们之间的关系。如图 19-16 所示，方块代表类，方块中列出了类的属性(数据)和行为(成员函数)，方

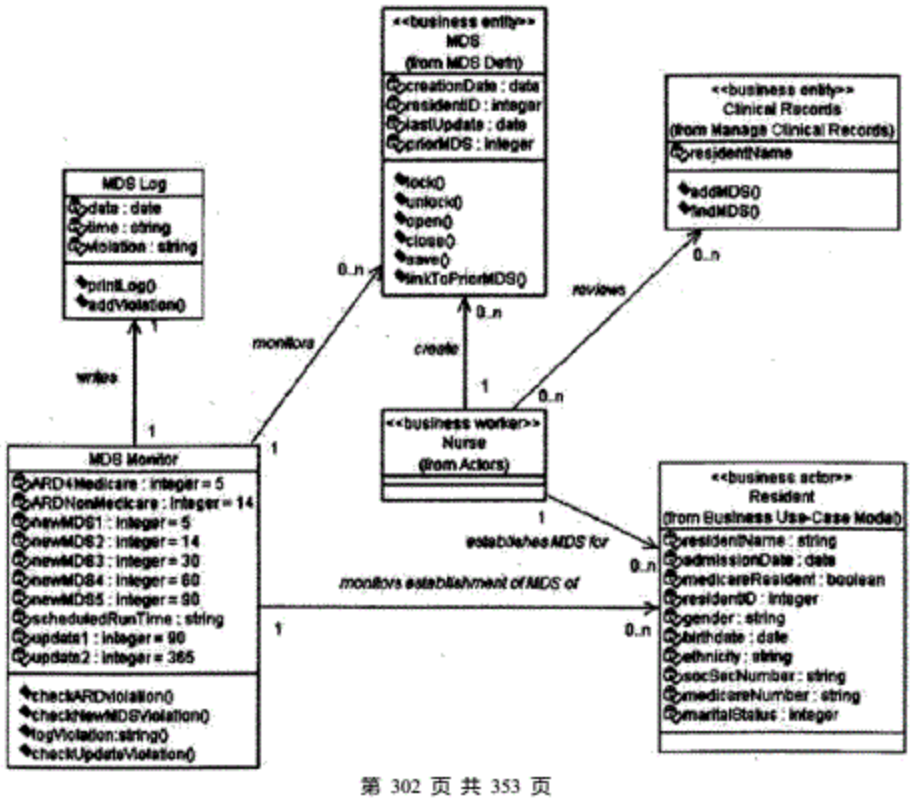


图 19-16 类建模的简单例子

块之间的线段代表它们之间的关系(1 对 1，1 对 N 关系，继承关系等等)。

状态转换建模所处理的是动态的问题，它所表达的是类和它们的实体——对象，在系统中的动态行为和相互的影响。

模块建模处理的是软件的具体实体，即编程模块、动态库和各种源文件之间的关系问题。

有了这三类建模，整个程序的大的框架就可以表达得非常清楚，有助于程序员之间的沟通。UML 可以说是程序员们之间的共同语言。目前使用 UML 比较多的是美国的 PC 游戏开发公司。

除了 UML 以外，Rational 公司还开发了很多软件工程工具。工具之繁多，覆盖了软件工程过程和工具的各个可能的方面。单项工具中有专门帮助测试的工具(Rational Suite Test Studio)，帮助进行改动控制的工具(Unified Change Management)。大的方面，有对整个软件开发过程进行规划的 Rational Unified Process。

Rational 公司的各种软件工程工具总的来说都是不错的。只有一个缺陷一它们是为商用软件的开发项目设计的。要应用到游戏开发项目中，需要做一定的调整和取舍。

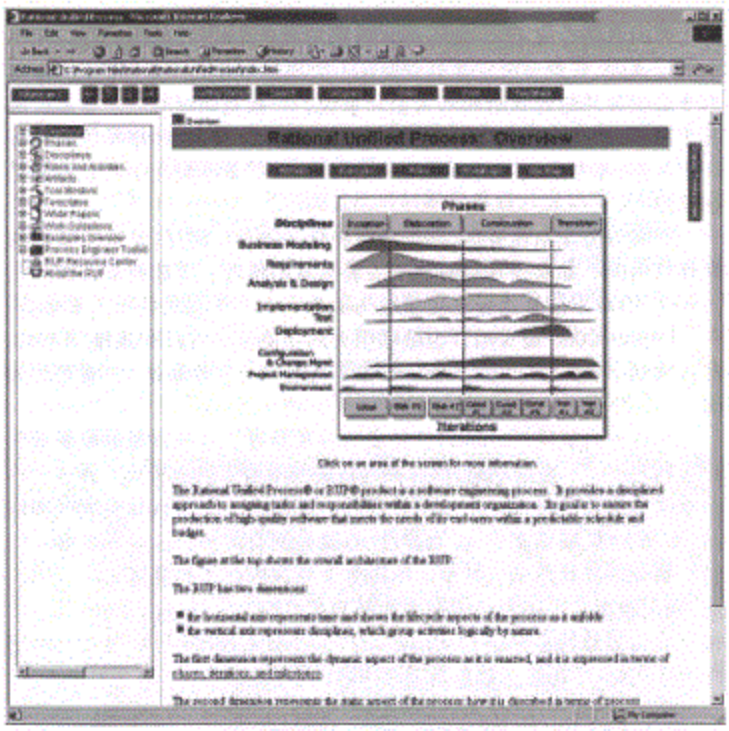


图 19-17 Rational Unified Process 是一套基于互联网的软件过程辅助软件。顾名思义，它着眼的是软件过程，也就是软件工程中比较宏观的层次。它给项目组提供了一个大的框架性的结构，整个软件开发的各种活动可以构建在这个框架上

Microsoft Project 2000

微软的 Project 2000 是一套比较实用的项目管理软件。其中比较重要的功能是软件计划和资源管理。软件计划方面，主要是进行任务分配、时间估计、确定任务之间前后依存关系，然后由前向后或者由后向前(由项目截止日期向前)生成并细调甘特图。

资源管理方面，分为人力资源和物质资源。人力资源体现在员工每周工作时数的分配上。一般是把人员的每周工作时数按任务分配。对小型公司来说，资源分配主要是资源在一个项目中各项任务中的分配。而对于那些采用矩阵式结构的大公司来说，一般是同时给一个员工分配几个项目，每个项目每周 10 几个小时的工作量。在这种情况下资源分配更复杂，要考虑到不同项目之间的关系，要尽量避免几个项目同时达到临界点。

把资源分配给任务后，还需要监控任务的执行情况，比较资源的实际使用情况，如果发现资源闲置或者紧缺的情况，需要调整计划。在 Project 2000 下这是通过比较基准(baseline)和实际执行情况而实现的。

Project 2000 最实用，也最吸引人的功能是其可以和其他 OFFICE 部件集成，特别是可以用 Outlook 来分配任务、召集会议、传阅文件等等。

近几年内联网(Intranet)已经成为很多游戏公司项目管理的最基本的工具之一了。最简单地，内联网可以起到项目信息公告板的作用。每一个项目组可以设立一个属于自己的内部网站，把项目有关的各种信息，大到文档、基准文件、计划进度，小到每次会议的会议议程和会议记录，都可以放到网站上共享，使得整个项目的计划和进展情况一目了然，极大地改善了项目组内部的沟通和交流能力。Project 2000 的一个很重要的组件就是 Project Portal。使用 Project Portal 可以有选择地把 Proiect 2000 中的项目信息发布到内联网上。

总的来看，微软的 Project 2000 是一个比较适合中小规模公司使用的项目管理软件，适合各种类型的软件的开发。

AlienBrain

现在游戏越做越大，游戏开发也越来越复杂。在 1995 年，开发一个一般一点儿的 PC 游戏，需要 3 个人工作 6 个月，产生 1000 个左右文件。现在开发一个 PC 游戏，则需要 15 个人工作 12~18 个月，产生文件近 50 万个。在游戏机上的某些大作，其开发人数已经达到几百。这么多人同时使用这么多文件，由此带来文件共享、版本控制、备份恢复等各种问题。特别是现在国际合作开发，有可能制作人员在不同的城市甚至跨洋越洲，他们之间如何协调工作更是难上加难。除了上面各点，传统的软件开发项目管理问题，如人员、进度、代码控制，都在困扰着游戏开发人员。为了帮助游戏公司们面对这些挑战，NxN 公司开发出了 AlienBrain。

AlienBrain 是一套适用于大型软件开发的项目管理软件。它一开始的侧重点是在数据媒体文件的管理(digital media management)方面。后来经过不断充实改进，其功能已经可以涵盖软件开发的方方面面。其基本模块包括媒体文件管理、项目进度管理、用户管理、工作流程管理、异地开发协调管理等。它采用了客户／服务器结构，用中央式数据库存储文件，在服务器端提供了完备的文件共享和用户管理功能。在其客户端，通过嵌入件技术，使得 AlienBrain 和各种美工和编程工具有机组合，实现了命名原则和滚动恢复等功能。客户端和服务器端合在一起，真正能够使得开发人员能够达到对项目的有效控制，可以说是项目经理的有力武器。

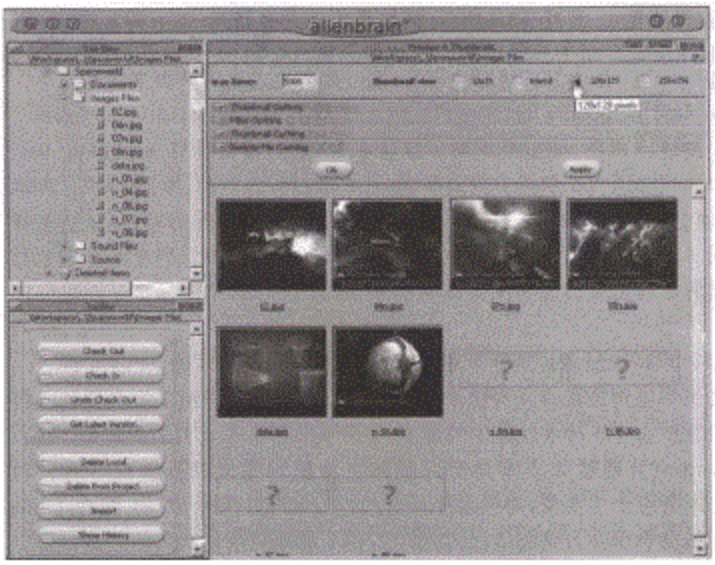


图 19-18 AlienBrain 给美工提供了随便的工具，使得他们可以轻松地管理各种媒体文件，不至于引起冲突

最近 NxN 公司已经进一步拓展了 AlienBrain 的功能，提供了几个很有用的附件。其中最重要的是 SMX(Studio Management Extension)。它比项目管理软件更进一步，扩展到整个公司的管理和多个项目的控制，借鉴了目前最流行的企业内联门户网站(portal)概念(所谓企业内联门户网，就是指大中型企业内部有大量的信息需要让员工知晓，员工之间亦需要沟通。借用门户网站的概念和内联网的技术，可以提供一种工具去帮助实现信息在企业内部更有效地流通共享。信息更有效地流通共享，将带来企业生产效率的提高)。这种工具可以为公司的管理层提供最及时和最准确的信息，使得他们对公司的运行状况，各项目的执行情况都胸有成竹。

总之，AlienBrain 的各种工具，一开始就是针对游戏开发的特殊需要量身定做的，比较适合游戏公司使用。但要使用其工具，公司的规模不能太小，项目规模也不能太小，否则其优势发挥不出来(本身其软件就很昂贵)。另外 AlienBrain 的汉化工作已经展开，马上将有简体中文版本面世。对于已经达到一定规模的国内公司，此乃佳音，可以通过采用其软件，把游戏软件的开发进一步规范化和制度化。

第二十章　游戏测试与质量保证

广义的软件测试

长期以来，计算机业界对软件测试的看法都存在着极大的局限性。众所周知，软件测试与质量保证概念的兴起，是 20 世纪 70 年代软件危机的直接后果之一。70 年代以前，程序员们写程序时毫无章法，随心所欲。他们的程序结构混乱，不加注释，不留相关文档。随着社会对软件的需求的增加，再这样写程序肯定不行了。于是有了所谓的“软件危机”。软件工程师们才开始思考软件工程的有关问题。其中最主要的一个方面，就是软件测试和质量控制。为了保证程序的质量，他们想出了种种方法，如代码审核、通读法等。经过 20 多年的发展，作为大框架的软件工程理论和实践日趋成熟，作为具体操作的软件测试和质量保证也日趋正规化，软件本身的质量也越来越高。业界的人士发现一个问题：软件质量越来越高，BUG 越来越少，系统越来越稳定，但为什么用户还是不喜欢我们的软件呢?原来，他们的软件虽然质量高，BUC 少，系统稳定，但不好用!问题的症结是：软件不是孤立的，软件的质量并不是仅由其本身就能决定，而是由软件+用户这个大系统来决定的，脱离了用户而奢谈软件质量是毫无意义的!具体到游戏来说，就是一个 PC 游戏，仅仅做到易于安装，从不死机，各项功能运行无误是不够的，因为前面说的那些都是 PC 游戏软件本身的质量，而对 PC 游戏软件成功起决定性作用的，是其是否好玩，游戏和玩家是否能构成一个和谐的系统。

在明白过来之后，计算机业界开始了新的研究和思考。正如 30 年前软件危机导致软件工程的诞生一样，这次业界把目光投向了计算机研究的另一个前沿——人机交互。正如软件工程的各种方法的目的是 前面介绍了一般商用软件的软件测试，已经从单一的代码测试拓展到包括代码测试和用户测试。对于游戏软件来说，则情况更为特殊。为了提高软件本身的质量一样，人机交互的各种方法和思想是围绕着如何提高软件+用户这个大系统的质量而展开的。人机交互中最主要的一个方面，是通过认知心理学的方法，去观察用户，去理解用户，并从用户获得反馈，然后去改进系统的设计。因此，一个崭新的概念提出了，这就是用户测试(user test)，即测试用户在使用软件时的行为，以此来作为用户+软件这个大系统质量的指示。90 年代末期以来业界谈到软件测试，基本上就指广义的软件测试了，包括两个方面：一是代码测试(codetest)，也就是传统意义上的软件测试，主要目的是找程序中的 BUG，看系统是否稳定，另一个是用户测试(user test)，主要目的是看软件是否好用，用户在使用软件过程中有什么问题，从用户那里获得反馈，再改进软件的界面(interface)和交互(interaction)。两个方面相辅相成，缺一不可。但两者既然侧重不同，实施的方法也不尽相同。

游戏软件测试

前面介绍了一般商用软件的软件测试，已经从单一的代码测试拓展到包括代码测试和用户测试。对于游戏软件来说，则情况更为特殊。首先，游戏有一个游戏性调节的问题。在游

戏性一章中，我们介绍过游戏性是游戏的最重要的属性，同时又是最模糊的概念，需要最细微的调节，才能达到很好的效果。游戏性的调节，包括两方面：一是经过测试发现原有的游戏规则需要改进，从而改变游戏规则的某个方面；二是在大的游戏规则不变的情况下，仅仅进行数据的调节，从而改变细微的游戏性。这两方面中，后者，即数据的调节是最繁重的，因为它是一个手动求最优的过程。RPG 和 RTS 游戏中都用到庞大的数据表格，表格中成百上千个数字。这些数字之间的大小关系就影响着细微的游戏性(例如对 RTS 游戏来说就是平衡性)。当游戏设计师面对一张空白的大表往里填数的时候，他第一次就把所有的数填得恰到好处的可能是 0.000000000001 惟其如此，游戏设计师必须通过不断的游戏性调节测试，去调整表格中的各种数字，以求达到最优效果。这种努力，必须由游戏经验丰富的人手工完成，所以旷日持久，难度很大。

其次，由于游戏的用户就是玩家，所以我们前面介绍的用户测试称为玩家测试(gamer test)更合适。玩家测试的目的，是把游戏(包括最初的设计草图、原型、半成品、成品)拿出来给一部分有代表性的玩家试玩。通过试玩，发现设计中的问题，改善游戏性，力求最终的产品被广大的玩家群所接受。

对于游戏性调节测试和玩家测试之间的关系，很多业界人士都有疑问：既然我们有了那么多经验丰富的游戏设计师进行游戏性调节测试，为什么还需要使用玩家来进行测试呢？有以下几个原因：第一，任何设计师都有所谓“盲点”。所谓盲点，是指设计师自己设计东西出来，如果自己评测，肯定在感情上对自己的设计有所袒护，对某些设计缺陷和错误会不可避免地视而不见，此乃人之常情。第二，设计师作为一个群体有自己的“偏见”。所谓偏见，是指设计师这个群体，由于个人经历教育程度社会经验等等方面有一定的同一性，使得他们对某个事物的看法具有自己的特殊性。这种特殊性必然在设计的时候会不自觉地体现出来，这种特殊性是否能够被大众所接受则是一个问题。比如一个深受武侠小说影响的游戏设计师，在做出游戏设计的时候，会认定所有玩家都会和自己一样是武侠小说迷，从而做出各种离奇的设计。这样设计出来的游戏，对非武侠小说迷来也许毫无兴趣可言。又如几乎所有的游戏设计师都是从资深玩家过来的，他们对游戏的理解比初级玩家深刻，他们的游戏技巧比初级玩家娴熟，这些反而成为障碍，使得游戏设计师无法设身处地地为初级玩家着想，这样他们设计的游戏很可能不能满足初级玩家的需要，游戏难度过大，技巧要求过高。

个人的盲点，可以通过其他设计师的参与来加以修正，但游戏设计师群体的偏见，则无法在游戏设计师内部解决。只有通过引入第三方——玩家来对游戏设计进行评测，才能保证获得宝贵的第一手材料，保证公正和无偏见的评估。所获得的反馈信息，可用于改善游戏设计，调节游戏性，甚至引入更具创造性的内容。所谓集思广益，也就是这个道理。

游戏设计师们必须认识到：游戏设计师设计的游戏不是给自己玩的，而是给千千万万游戏水平不如他们的一般玩家玩的。玩家，只有玩家，他们的好恶爱憎才是决定游戏成功与否的惟一标准。这一点，在商用软件设计界已经达成共识。普遍认为，软

件开发设计在前 40 年经历了三个阶段，即：以工程师为主导的软件设计(Engineer-Centered Design)，以设计师为主导的软件设计(Designer-Centered Design)，以用户为主导的软件设计(又称基于用户的设计，User-Centered Design)。以工程师为主导的软件设计的时间范围为 20 世纪 60 年代到 80 年代初期，当时所有的计算机软件系统都是由程序员设计的，用户也都是程序员，也就是说程序员们自己设计程序给自己用。80 年代到 90 年代，则为设计师为主导的软件设计时期，其标志是计算机软件渗入到人类生活，特别是商业生活的各部分。商业的需求驱动软件设计，纯技术类型的程序员们显然无法胜任软件设计了。这时则有商业分析员和软件结构设计师共同分析商业需求，决定软件的设计。90 年代以来，则逐渐过渡到以用户为主导的软件设计，即在分析商业需求的同时，将强对软件未来用户的观察和分析，力求使得软件在实现功能的同时，保证效率和可用性，真正做到一切以用户为中心。

除了游戏性调节测试和玩家测试外，传统的代码测试也必不可少。代码测试主要是由公司雇员做的。他们是真正训练有素的测试人员，并且具有一定的软件开发的知识。他们使用各种复杂的报表，将每一个 BUG 记录在案。

表 20-1 显示了三种测试的异同。

	执行者	测试的目的
游戏性调节测试	游戏设计师，专业测试人员	从设计者的角度细调游戏性
玩家测试	玩家	从玩家的角度验证游戏是否好玩，从玩家直接获得关于游戏设计的反馈信息，从而改进游戏设计
代码测试	程序员，专业测试人员	去除 BUG，使得游戏运行稳定

表 20-1 游戏开发所用到的三种类型的测试

游戏性调节测试

游戏性调节测试一般在游戏基本完成后进行。显然只有等到有了基本能够稳定运行的游戏，才能着手进行游戏性的调节测试。目前几乎所有的游戏在开发阶段都是数据库驱动的，也就是说游戏中各项参数由数据库中的表所提供的。游戏设计师在进行游戏性调节测试的时候，主要是改变各种表中的各项数字。

游戏性调节测试耗费时日，有时人们觉得其费效比不突出--游戏已经能玩了，已经稳定了，为什么要再费时间调整那几个参数呢?对这个问题，需要这么看：任何类型的测试的费效比，都可以用图 20-1 中的曲线来表示。

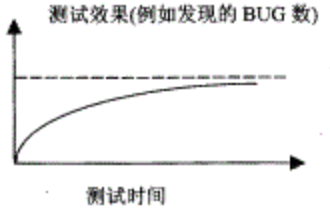


图 20-1 测试曲线

这个曲线是这么解释的：一开始测试的时候，立竿见影，马上就找到很多 BUG。随着测试的进行和深入，发现 BUG 就

会越来越困难。因为浮在表面上的 BUG 比较容易发现，在测试的开始阶段都发现了，越到后来剩下的都是隐藏很深的 BUG，需要极大的精力和时间去发现并纠正。所以上面的曲线一开始很陡，也就是单位时间发现和处理的 BUG 数多。越到后来曲线越趋于平坦，也就是单位时间发现和处理的 BUG 数越来越少。最后曲线和一个固定数值无限接近，表明测试永远无法把所有的 BUG 找出。

游戏性的调节测试也基本上符合这条曲线。即一开始调节测试时，容易发现很多游戏规则和游戏性方面的疏漏和问题并改进之。过一段时间后，问题更加隐蔽和细腻，需要的努力更大，耗费时间更多。从表面上看，测试一段时间后费效比必然下降，改进愈发困难(曲线趋缓)。但这些努力一般会得到补偿——一个泛泛地看来比较好的游戏，也许是在曲线上距渐近线 70%那点，而一个真正的近乎完美的杰作，也许仅仅是前进了一小步，在曲线上是距渐近线 80%的地方那个点。这两个点在纵轴上相差不远，在横轴上则相差极大。这意味着在基本游戏规则成熟后，细微游戏性的小小改进，耗费时日和精力(金钱)。这种细微的游戏性上的差别，也许就是一个一般性的好游戏和一个真正的杰作的差距所在。在业界更有这么一种说法：区别一个杰出游戏和一个平庸游戏的，也许就是几个月的游戏性调节测试。

玩家测试

玩家测试主要由公司以外的广大玩家来完成。世界上玩家成千上万，找什么样的玩家来测试呢?这个问题是在进行玩家测试之前要首先解决的。

找什么样的玩家来是由游戏设计的总体定位来决定的——这个游戏是给谁设计的?是铁杆玩家还是初级玩家?是 RPG 玩家还是 RTS 玩家?是动作型玩家还是感受型玩家?游戏设计的第一步就是确定这款游戏的定位(这是游戏业的商业化运作所无法避免的后果)。只有在游戏的总体定位确定后，才能知道需要如何设计而投其定位的观众群的所好，更进一步知道需要雇用何种玩家来进行测试。比如说我们开发一款铁杆动作游戏，自然首先要邀请铁杆的动作游戏爱好者来测试，因为他们是潜在消费群的主体。如果我们想突破动作游戏类型的局限，看看是否一般玩家或者其他类型游戏的玩家能够被这款游戏吸引，我们也可以有意识地筛选一些一般玩家，看看他们对这款高难的铁杆游戏是否能够接受。

玩家测试具体实行方式有两种，一种是所谓的实地测试(in-house test)，即把一些玩家请到公司里，安排一间干净简单的屋子，让玩家玩游戏的测试版，然后观察其反应，记录下各种问题。另一种就是在网络上散布 Beta 版，让所有有兴趣的玩家去下载，然后他们可以给公司发信写 Email 谈自己的看法和心得。下面就这两种测试细细道来。

对于 Beta 测试，由于不是在公司实地进行，公司对其具体操作无法过多监控，主要需要注意的是是否要给玩家在网上建立专门的网站，让他们提交并讨论发现的问题。另外如何收集他们发现的问题，是否需要强制使用一定的报表格式等。实地测试，由于是在公司进行

的，就有很多讲究。目前主要有两种方法：玩家单独测试和宣泄法测试。

●玩家单独测试

这种测试是比较规范化的，需要专用的“软件可用性实验室”(usability lab)，实际上就是一个屋子，内外两间，用一个玻璃墙隔开。注意这不是一般的玻璃墙，而是我们在侦探电影中常见的单向可见的玻璃墙。从里间看得见外边，从外间看不见里面。外间是给玩家玩游戏的地方，墙壁上装有摄像头，一般要两部，一部对着电视或计算机屏幕，一部对着玩家。里间是公司人员观察玩家的地方，包括计算机设备、录像／录音设备等。具体使用时，请玩家到公司来，确认其自愿进行玩家测试并被摄像，填写表格后，领其进入外间。公司员工向其介绍硬件和游戏的基本操作；然后离开。玩家一人在外间玩游戏，公司员工从里间观察玩家什么地方遇到问题，什么地方引起兴趣，什么时候感到疲惫并无法继续游戏。这种方法由于没有公司员工在场，玩家比较放松自然，但缺点是由于没有公司员工在旁帮助，会出现玩家卡在一个地方出不来的情况，另外玩家也不可能实行太复杂的任务。因此持续时间不能太长，一般以 1 个关卡或者 1 小时为限。

●宣泄法测试(Think-aloud Protocols)

另一种方法是大名鼎鼎的宣泄法(Think-aloud Protocols)。这种方法被认为是 HCI 诸多方法中最基本也是最有效的一种，其应用日见普遍。其基本思想就是：玩家玩游戏时，让公司员工坐在玩家旁边，并且让玩家大声地把自己心中所想的和情感的好恶宣泄出来。我们不是一直说游戏设计师要想玩家所想，游戏设计师不能代替玩家吗?宣泄法向游戏设计师们提供了一个有力的武器去真正了解玩家心中所想，他们真正喜欢什么，厌恶什么!当玩家一边玩游戏，一边大声表达出他们的愤懑、迷惑和喜悦时，在一旁坐着的游戏设计师可以说是真正搞到了关于这个游戏的游戏性的第一手材料。他们这时需要把引起玩家表达感情最激烈的“事件”一一记录下来。另外，让玩家说出心中所想的每一步骤，可以帮助我们了解玩家在玩游戏时所采取的策略和游戏的设计相对照，我们可以发现其中不匹配的地方并加以改进。

在讨论代码测试之前，我们先谈谈代码审核。它和代码测试是息息相关的。

代码审核

审核并不是实际测试，但比实际测试更重要。代码审核的概念很早就有了。我们知道早期的计算机，没有文本编辑器，没有键盘，其输人设备是打孔机。程序员把程序写好后，用打孔机在专用的计算机纸上打孔，然后输入计算机。如果程序错误很多的话，程序员得重复很多次这样的操作，费时并且费钱。这就逼着程序员们必须写出很高质量的程序，争取一次打孔成功，而要在打孔输入之前发现程序中的问题，必须靠代码审核，不能靠编译器。因此，他们每次把程序写出来后，都自己反复检查其逻辑和拼写是否有错，并且和同事共同通读好几遍。久而久之，这种操作成了业界的标准。进入 20 世纪 80 年代以来，随着文本编辑器和集成编译环境(如 Visual Studio 等)的出现和流行，程序员们越来越忽视代码审核，越来越多

地依靠编译器了。他们写完程序后，不仔细检查，不先纠错，就急急忙忙去编译，然后等着一大堆错误出现，再去修改程序。这种方法事倍功半，是极其错误的。因为出现了编译错误，再反回去找原因是一个逆向思维的过程。即透过现象，反求本质。这是非常困难的!有时候，程序中一个简单的字符打错了，却引起极其复杂的毫无规律的系统错误。程序员们耗费许多时间和精力，最后发现原来栽在这么简单的一个小错误上，他们那时真是要欲哭无泪了。很多问题，如果细心些，在编译之前就能解决。通过本质，推理出现象也相对容易些。

另外，代码审核也提供了一个机会，让程序员静下心来，通读一遍自己的程序，真正形成清醒的认识，构建所谓的 mental model。更重要的是，如果是采用小组审核的话，除了具体写这段程序的程序员外，其他程序员也能对其程序有所认识，相互交流取经，有助于公司整体编程水平的进步。如果公司有一个非常优秀的高级程序员的话，对他写的程序进行小组审核和通读，则能够使其他的初级程序员们获益匪浅。以后即使这位高级程序员跳槽了，他的基本编程思想和套路已经被其同事所理解，那么他剩下的工作可以被别人较顺利地完成。

代码审核基本上以三种形式进行：小组审核(inspection)、通读法(walk-through)和个人审核(personal reviews)。

●小组审校

小组审核是一种比较正规的审核方法。需要小组内各成员的共同参与，每个人有其确定的职责。小组审核包括三个阶段：准备阶段、审核阶段、修改并复审阶段。在准备阶段，进行一次简单的例会，通知大家要审核哪一段程序。然后大家回去分头准备，去阅读程序并且提出问题。然后举行正式的审核。在会上，由会议主持者保证时间的有效利用，由与会者们提出自己发现的问题，作者回答这些问题。然后大家讨论解决的办法。最后，确定由谁修改，如何复审。会议结束，大家回去修改。改好后再进行一次复审，合格后就可以通过了。这种方法的缺点是费时最多，另外也比较适合于组员水平比较平均的情况，对其他参与审核的程序员的要求比较高。其优点是能最有效地发现错误。

●通读法

通读法(walk-throush)是一种随意性比较大的审核方法，更像是一段程序的作者个人面向小组其他成员的演示。在会上，作者使用投影仪将他所写的程序显示在大屏幕上，然后向与会者介绍其逻辑、界面、算法等。听众一边听，一边发现问题并提问。由于是由作者把这段程序从头到尾过一遍，所以笔者将其翻译成“通读法”。显而易见，这种方法对听众来说收益最大。一般来说，由一个经验丰富的高级程序员带几个初级程序员进行通读法，能够起到“传帮带”的作用，迅速提高初级程序员们的水平。

●个人审核

个人审核(personal reviews)是最简单的审核方法。顾名思义，就是在编译之前，由程序员个人把程序看几遍，力求发现一些显而易见的错误。举凡业界最顶尖的程序员们，无不是

在个人审核方面很有一套的。他们在编译之前，一般都能发现找出大部份错误，因此才能够满怀信心地去编译。

审核的目的，包括：检查程序逻辑，程序结构，拼写，本模块和其他模块之间的界面是否一致，是否遵守了本公司的代码标准，是否有易于理解的注释。审核时最重要的工具就是审核明细表(checklist)了。我们知道，飞机起飞着陆时，主驾驶和副驾驶都要一项一项地对照这种明细表来检查是否所有规定的操作都被执行了。明细表的作用，就是使我们清楚地知道应该审核些什么，并保证我们不漏掉任何一项。对于代码审核来说，明细表的内容要包括各种常见的容易出错的地方，比如说 C++语言的程序内存是否释放干净、是否文件在打开用完后被关闭了等等。这个表格更可以根据公司里面各种实际项目的经验(统计数据)加以修订删改。

代码测试

代码测试本身是个极复杂的话题。鉴于已经有许多书籍专门地深层次地探讨代码测试的策略、计划、测试案例(test case)的编写等，这里只是简单介绍一下几种概念。

代码测试有各种分类方法，比如它可以分为白盒测试(white-box testing)和黑盒测试(black-box testing)两种。所谓白盒测试，是指已经知道程序的内部结构，并以此为依据设计测试案例，力求覆盖所有可能的逻辑路径和分支。黑盒测试是指在不知道程序内部结构的情况下测试程序的输入输出对应关系。白盒和黑盒，一为以局内人眼光审视程序，着眼其内部错综复杂之结构；一为以局外人眼光审视程序，观其外部输入输出的大略。是看同样的东西，但侧重不同，角度不同。

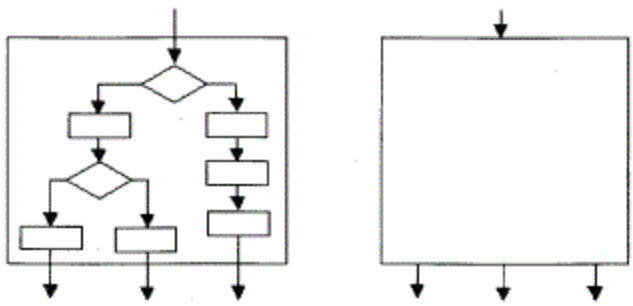


图 20-2 白盒测试（左）和黑盒测试（右）的示意图

代码测试又可以分为单元测试(unit test)、集成测试(integration test)和系统测试(system test)。单元测试是在一个模块完成后进行，其目的是保证一个模块内部功能运行无误。集成测试则测试多个模块组合起来后是否能够稳定运行，模块之间的界面和信息传输是否正确。系统测试则是把整个系统所有模块集成起来后测试整个系统的功能是否达到要求。

单元测试一般为白盒测试，是由程序员在编制好一个模块后进行的针对其内部逻辑结构所测试。系统测试，则主要为黑盒测试，测试人员并不知道程序内部结构，只是测试其外部功能，并将 BUG 记录在案，反馈给知道程序内部结构的程序员去改正。

在游戏业，人们对代码测试有种种误解和偏见。其一为轻视测试和测试人员。有人认为

测试人员为整个游戏业食物链的最底层，任何人员都可胜任。实际上测试人员也有严密的组织结构，需要一定的专业知识。主测试人员一般要有很深的理论水平(特别是对于软件结构设计方面)和长期的经验。像微软等大公司里主持测试的人员都要有硕士以上学位，而在主测试人员手下的长期雇用的一般测试人员和临时雇用的临时测试人员，也都要经过一定的培训后才能上岗。

其二，代码测试也不仅仅是一些人在那里玩游戏找 BUG。对代码测试来说，其测试的策略(何种游戏，需要何种的测试，多长时间的测试，等等问题)和测试的计划(测试由谁完成，如何执行，如何汇报发现的 BUG，如何监督改正，等等)极为重要，它们决定了代码测试的大框架，更决定了代码测试的效果。

其三，代码测试不是等到代码完成后才开始的，测试策略的确定、计划的制定和测试案例的编写，都是越早越好，一般是在软件的结构设几阶段就开始了。有经验的主测试人员，可以不必等到软件编写完毕，而是根据其结构设计来决定如何测试，并预先编写出完善的测试计划和案例来。

目前来看，在中国游戏公司里面，比较缺乏高水平的能够独挡一面统领全局的主测试人员，进而导致测试策略和测试计划方面比较薄弱，极大地影响了代码测试的效率，最终导致软件质量的低劣。

游戏机游戏和 PC 游戏测试之异同

许多玩家都抱怨为何 PC 游戏的问题那么多，安装的时候安装不上，装上之后死机，玩起来后莫名其妙地退出。似乎游戏机游戏从来就没有这些问题，对游戏机游戏来说，开机放碟然后开玩，什么问题都没有(当然盗版碟问题除外)。是否是说 PC 游戏的程序员就不如游戏机游戏的程序员呢?答案是否定的。因为游戏机，比如说 PS，全世界几千万台全是一个模子里出来的，硬件上没有任何差异(除了美版、日版的问题)。因此测试时，只需要一台 PS 就可以了，而 PC 由于各组件的生产厂商成千上万，由此构成的可能的组合是天文数字，没有任何公司可以说能够完全测试这么多组合。这种类型的测试，是代码测试的一种，叫做适应性测试，所测试的是游戏在不同硬软件设置环境下的运行稳定性。PC 游戏的适应性测试难度要比游戏机大得不知道多多少倍，甚至达到游戏公司本身都无法承受的地步。我们看到，开发游戏机软件的公司，如 Square，有很庞大的测试部门，专门负责自己游戏的测试。他们不需要把游戏拿出去测试，自己就能胜任。PC 游戏公司，再大的公司，也往往把游戏放到网上让全世界的玩家去下载，这样他们能够让玩家们去测试尽可能多的硬件设置。

第二十一章　游戏公司组成结构

本章要介绍美国游戏公司所普遍采用的比较成熟的组织结构、人员分工和职责划分，这些对中国游戏业和游戏公司很有借鉴意义。我们看到在很长的一段时间内，中国游戏公司内部职责不明、分工不清的问题十分严重，比如说没有把制片人单分出来，造成游戏设计师负担过重，不能全力投入游戏设计，为琐事所分心。

公司组织结构

目前比较常见的公司组织形式，不外三种：基于功能划分的组织结构和矩阵式结构。下面一一介绍。

基本功能划分的结构

基于功能划分的组织结构是公司里最常见的一种结构，如图 21-1 所示。

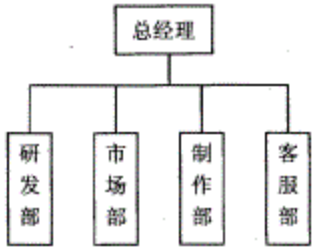


图 21-1 基于功能划分的组织结构

图 21-1 所显示的是公司里常见的基于功能划分的组织结构图。以总经理为金字塔顶部，然后按专业功能分成不同功能部门，每个部门由一个副总经理领导。

这种组织结构是基于以下的考量：(1)按专业分工是大型企业日常管理必须的；(2)相同专业的人员集中起来比较好管理，他们技能相同，有共同语言，可以互相帮助互相学习；(3)按功能划分可以最大程度地集中利用相似的资源。

这种组织结构的缺点是只能处理日常事务，无法胜任项目开发。因为一个项目必然需要不同背景和技能的人在一起工作。比如游戏，既需要美工也需要编程，在一个项目内部需要协调和合作，使用这种基于功能划分的结构无法达成这一点。因此，制作部一般都要采用另外一种组织形式，即基于项目的结构。

基于项目的结构

所谓基于项目的结构，就是以一个个项目为中心来组织人员和资源，不同背景和技能的人加入到一个项目组中，每个项目由一位项目经理带队，有足够的自主权，如图 21-2 所示。

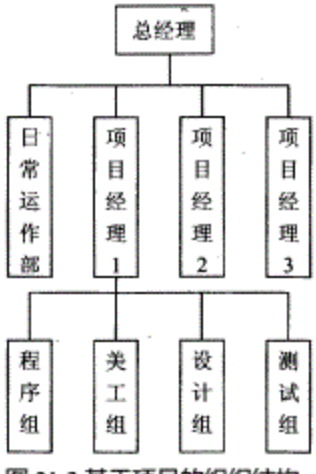


图 21-2 基于项目的组织结构

这样，一个大的公司就细分成相互独立的，临时性的(对应于项目周期)，专门针对某个特定项目的小的项目组。值得注意的是项目组内部还是使用了基于功能划分的结构。另外，公司日常运作部分和开发部分是分离的。

这样的组织结构的优点是每个项目组目标明确，人员职责明晰，项目经理有足够的权威性和对资源的独占。但缺点是当项目很多时，不同项目组的美工和美工之间，程序和程序之间缺乏沟通交流，这样不利于个人水平的提高和整个公司的进步。另外，这种组织结构是着眼于短期行为的，即一个短期项目的成败，而并不能反应一个公司长期的策略。

矩阵式结构

既然人们认识到基于功能划分的结构和基于项目的结构各有优缺点，于是就想：是否能够把它们结合起来，综合两方优点，弥补各自缺点呢?矩阵式结构的概念就这么产生了。

矩阵式结构，是一种多维的组织结构。前面介绍的基于功能划分的组织结构和基于项目的组织结构都是直线一维的，自顶向下的多层结构。矩阵式结构是多维的。它先按功能划分来纵向组建基本的部门，比如说美工部、程序部、测试部。然后对具体的项目，设立项目经理和多个项目协调员，来横向组建项目组。

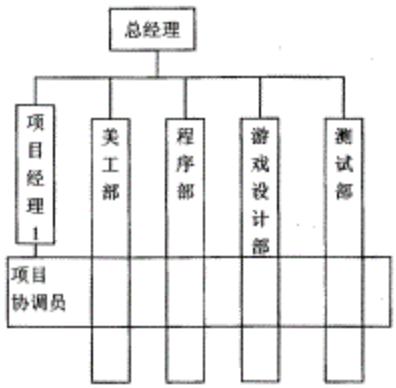


图 21-3 矩阵式结构

这种组织结构的优越性在于综合了基于功能划分和基于项目的组织结构的优点。一方面，相同技能的人有一个集中的部门，有利于统一管理。比如说要在程序部推广一种新的代码标准，或者在美工部引入一种新的 3D 软件所带来的培训任务。有统一的按功能划分的部门有助于这些任务的成功实施。另一方面，通过项目经理和项目协调员，横向地管理一个项目，保证了跨部门之间的协调。一般来说，一个项目需要数个协调员代表各个部门，比如美工协调员、测试协调员。具体的运作如下：假设六个美工加入了一个项目，则要由一个美工协调员(美术指导)作为他们的头。美工协调员作为中介，去从项目经理接受任务传达到各个美工。另一方面，也从各个美工接受反馈，传达给项目经理。因此，在矩阵式结构中，信息传输和反馈既有纵向也有横向。

当然矩阵式结构也必然有其缺点。简单地举一个例子：对一个美工来说，他可能要面对两个直接领导——项目经理和部门经理。有时候两个经理对这个美工的要求是会有冲突的，包括时间方面的冲突，不同的指标等等。这时候这个美工就会处于一种非常尴尬为难的地位——究竟听谁的呢?显然两个经理都是不好惹的。为了解决这些棘手问题，需要公司内部的规章订得比较详细，把这些问题的方方面面考虑到，需要投入一定资源建立这些规章制度。显然对小公司来说，要达到这一点比较困难。因此，矩阵式结构一般被中大型公

司所采用。

前面介绍的三种组织结构，要根据公司实际情况来选用。世界知名大公司中，康柏(COMPAQ)使用的就是基于项目的组织结构，而惠普(HP)使用的是矩阵结构。对游戏公司来说，一般小公司用基于项目的结构就足够了，因为人员少，交流可以通过人际交流等非正规的途径进行。如果有 5 个以上项目组同时工作、员工人数 100 多人以上的公司，就必须考虑矩阵式结构了。在国内，Ubisoft 中国分公司采用的就是矩阵式结构。

游戏项目组所采用的三驾马车机制

所谓三驾马车机制，是指游戏业中所通行的一种项目管理机制。它是游戏业经过多年发展，并借鉴了好莱坞电影业的经验,所制定的一种行而有效的项目组织结构。这种机制代表的是一个项目组的组织结构，而非整个公司的组织结构。如图 21-4 所示。

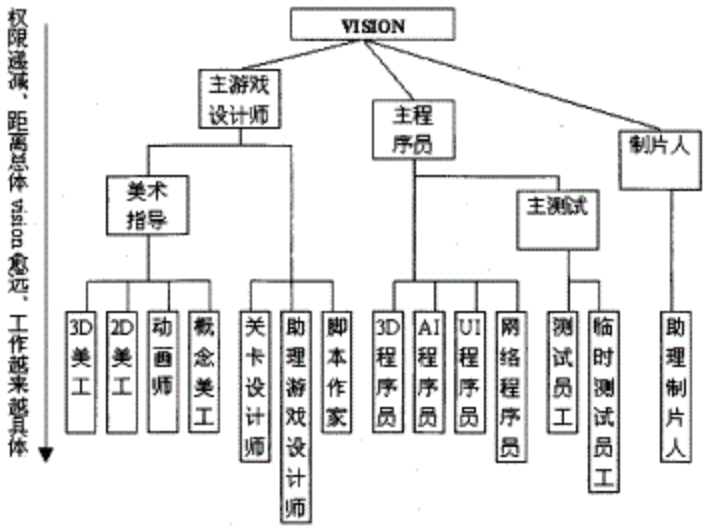


图 21-4 游戏项目组的组织结构

图 21-4 中显示的是三驾马车的机制。最上面的一个方块里是 vision。这个词很难用中文解释，因为中文中没有与之对应的能够恰当解释它的词汇。这里尝试着简单解释一下：游戏一开始肯定是在某个人，或者某几个人脑子里想出来的——这个游戏是什么样的，如何玩法，等等。也就是说关于最后展现在玩家面前的游戏将是什么样的，游戏的设计者们肯定是在脑子里先形成想法。然后他们要做的，就是把这个虚无飘渺的想法，表达出来，并通过自己和其他人的工作，把这个东西物质化实体化。在具体执行的过程中，必然会由于交流的原因或者技术的局限，使得想法和实际效果出现偏差。在这时候游戏设计者就必须有那种从总体上掌握的能力，知道什么地方要变通修改，什么地方要坚持强调，什么地方要进一步说明解释。同时，他还需要考虑到策略性的问题，就是开发过程的大框架和协调问题。因此 vision 可以解释为对一个现在还未具体实现的事物的总体上的掌握、前瞻性的理解和策略性的考量。

我们看到在图 21-4 中，离 vision 最近的是主游戏设计师(lead designer)。这是因为游戏一般是由主设计师主持设计的，是他们的梦想和冲动造成最初的 vision。这并不是说他们是 vision 的惟一贡献者。其他人，如美术指导和主程，也必然从其他专业角度对 vision 去进行细化和调整。主设计师要指导关卡设计师和其他助理设计师的工作，他们的工作比较具体。比如关卡设计师主要负责关卡的搭建和物品的安置，而助理设计师可能要进行平衡性的微调和其他历史政治文化背景知识的研究工作等。另外，美工指导是向主设计师报到并负责的，因为整个游戏世界的外观是要由主设计师和美工指导共同完成的。主设计师提供设想，美工指导提供具体的解决方案。

图 21-4 中，主程序员(lead programmer)的地位仅次于主游戏设计师。原因是主游戏设计师的设想和 vision 必须得到主程序员技术上的首肯。主程序员就像主游戏设计师的技术顾问，他的责任是把主游戏设计师的设想从艺术角度转换到技术实现角度来评估。他手下为一般程序员，按专业划分，有人工智能程序员、3D 程序员等。另外，主程序员手下的另一支重要力量是测试人员。主测试员是向主程序员报到并负责的，因为测试结果影响最大的将是程序部分。

在图 12-4 中的右边是制片人(producer)。制片人是除了主设计和主程序外对一个游戏项目组最举足轻重的人。制片人概念是从好莱坞电影业借鉴来的，并且和游戏开发的具体实践结合。制片人，在有的游戏公司被称为项目经理(project manager)，或者软件工程师(software engineer)。他所负责的主要是软件工程过程方法的管理和实施、对外联络，以及日常事务性工作。特别需要指出的是事务性工作。它是制片人十分重要的责任之一。事务性工作包括一切和游戏有关的琐事，比如后勤工作。把事务性工作和游戏设计开发工作分离，从而保证整个项目组工作起来没有后顾之忧，是美国游戏业的成功经验之一，也是目前中国游戏公司普遍做得不太够的地方。比较制片人、主程和主设计师，前者主要侧重于软件开发的过程，后两者则更关注软件产品本身。

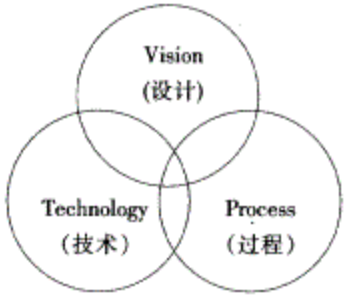


图 21-5 三驾马车机制体现的是游戏的三大基石：vision, technology, processs

从上面的分析，我们看出，三驾马车机制中的三巨头：主游戏设计师、主程序员和制片人，所代表的是 vision(设计)、technology(技术)和 process(过程)。这三点恰恰是游戏本身和游戏的设计与开发的基石。

人员分工和职责

下面重要介绍一下美国公司中经过长期实践形成的各种职务和其职责。需要说明的是有些职务是和美国业界的商业运作息息相关的，也许并不符合中国国情。但大部分职务的

设立和职责的划分还是十分科学的，并经过了时间的检验。另外有些职务，如人机界面设计师(user interface designer)，目前在游戏业并不普遍，但其重要性必将越来越受到重视，因此在此一并介绍。

游戏设计师

●主设计师(lead designer)

前面说过主设计师是距离 vision 最近的人。他对于要开发的游戏有总体上的掌握、前瞻性的理解和策略性的考量。形象一点说，主设计师就是在所有人看到一部电影之前，在心里把电影过了一遍的人。然后他告诉开发组其他人员他心目中的电影是什么样的，之后大家去实现。在实现过程中，由于技术资金等的限制，必然要不断做出妥协和修改。主设计师是游戏设计方面这些妥协和修改的最后决策者。当然他在做出决策之前必须听取其他两巨头——主程和制片人的意见。主程和制片人会从技术和资金进度人力等角度评估这些改变的要求。

主设计师必须有能够听取各方意见，并冷静分析的能力。因为游戏实际上是一个各方不断妥协的产物。妥协是必须的，主游戏师不能过于顽固地坚持不切实际的所谓理想。主设计师同时必须有很好的平衡能力，可以指导协调各组员。特别是和主程序和制片人的合作十分重要。

最后需要特别强调的一点：主设计师不是，也永远不会是游戏的惟一作者和成功的惟一原因。游戏是一个项目组所有开发人员的心血的结晶。当然有时候媒体会罔顾这一点而对主设计师大肆吹捧。这时候作为主设计师本身应该清醒并保持和自己和组员的关系。处理得不好的话，整个项目组的士气就会受到影响。

●助理设计师(assistant designer)

助理游戏设计师的主要职责是帮助主设计师进行细节实现，包括对游戏规则的细化、平衡性的调节、历史地理政治文化背景知识的研究。另外很重要的一条就是助理设计师要负责所有设计文档的管理和更新。

●关卡设计师(level designer)

关卡设计师是最近几年随着三维游戏，特别是三维射击游戏的盛行才出现的一种职务，关卡设计师的工作是设计关卡内部结构，包括建筑、光影、物品摆放、敌人的配置；然后把设计蓝图交给美工建模；美工做出各种 3D 模型后，关卡设计师要把它们集成到一起。并且不断测试，直到符合要求。关卡设计师的工作比较具体，一般来说需要玩很多游戏，对某种类型的游戏十分精通，这样在关卡的设计和测试中就有可能凭直觉做出正确的判断。

关卡设计师所使用的工具主要是关卡编辑器(level editor)。关卡编辑器可以是公司自己

开发的，也可以是买来的 3D 软件包所包含的。

●剧本作家(writer)

现在的游戏越来越重视故事情节了，不仅传统的 RPG 如此，连射击游戏和 RTS 都要有引人入胜的背景故事才行。有的公司甚至要请专业小说家来执笔，比如《彩虹 6 号》等游戏的故事背景，就是由《追捕红十月潜艇》等畅销小说的作者 Tom Clancy 编写的。剧本作家就是这些背景故事和对话的作者，他们的任务是通过对故事情节的整理和人物的刻画，使得游戏增加一些深度。

图 21-6 Tom Clancy 是美国畅销小说家，其作品《追捕红十月潜艇》《爱国者游戏》等多次被搬上电影屏幕。近年有很多部游戏的故事情节都是由他执笔，如《彩虹 6 号》等

程序员

●主程序员(lead programmer)

主程序员是主设计师的左膀右臂，他帮助从技术实现角度对主设计师的 vision 进行评估和细化，并提出技术解决方案。他的主要职责是对游戏开发进行技术评估，包括游戏平台的评估、新技术的评估、开发环境的评估等。在此基础上，设计软件的系统结构，也就是一个游戏程序的主体框架结构，包括各模块的功能和相互之间的关系。系统结构的设计很有点像手艺活——没有教科书式的完全"正确"的答案，需要对这个领域有很多感性的认识，再加上一点灵感和大胆的决定，有了一个好的系统结构，其他各部分模块才能各司其职，充分发挥作用。如果系统结构上有大问题，不管单个模块的质量如何高，整个游戏还是会有很大隐患的。

主程序员的另一项重要任务是和制片人合作，共同处理软件工程的管理。制片人主要负责事务性工作，但当遇到重大问题时，比如要推翻原有代码重来，这时制片人必须和主程序员在一起评估可能的后果，比如在进度和资金上的影响等，然后做出决策。

主程序员同时是其他程序员的导师(mentor)。他制定了代码标准和接口规范，让所有程序员来遵守。他要监督相互评估(peer review)的工作，并主持代码审核。

主程序员同时作为所有技术人员的代表，负责向组内其他非技术人员解释自己和其他程序员的工作，并回答技术性问题，比如说向美工解释技术上的限制。

●一般程序员(programmer)

对程序员来说，现在业界的惯例是按专业领域分到很细的程度。比如说 AI 程序员就专攻 AI 部分，3D 程序员就专攻 3D 部分。这么做有利有弊。其利在于术有专攻。现在游戏这么复杂，以一人之聪明和精力，也只能掌握一个领域的最新成果并在这个领域做出成

绩。但分得这么细的弊端是对程序员们来说，长期在一个狭小孤立的领域工作，其适应性和创造性肯定会受到影响。

下面就是基本的专业领域划分。

●基本游戏规则程序员(game logic programmer)：实现游戏的基本游戏规则。

●AI 程序员(AI programmer)：负责人工智能部分。人工智能可以看作游戏规则的高级部分。另外如果使用脚本语言(script language)来表达敌人的智能的话，则需要定义脚本语言的语法规则，并编写相应的编译器。

●3D 程序员(3D programmer)：负责图形图像部分，主要是 3D 引擎(3D engine)的编写。

●UI 程序员(UI programmer)：负责界面部分，包括窗口菜单的实现和键盘鼠标手柄的控制等。

●物理程序员(physics programmer)：物理程序员是近年来新出现的职务，他们的任务是实现重力加速度等功能，使得 3D 世界更加真实生动。

●网络程序员(network programmer)：实现网络对战功能，包括网络通讯协议、纠错功能、数据包的设计等。

●工具程序员(tools programmer)：编制游戏开发所需要的各种辅助工具，比如关卡编辑器、人工智能脚本语言编辑器、数据库支持工具等。

另外，程序员包括初级(entry-level programmer)和高级程序员(senior programmer)。一个游戏中最重要的部分显然要交给有经验的高级程序员完成，这样才能保证质量和进度。一些比较不重要的部分就交给初级程序员了。一般来说，AI 和 3D 是最重要的，要由有经验的程序员完成。工具和 UI 部分，可以由初级程序员完成。

美工

●艺术指导(lead artist，又称 art director)

艺术指导在主设计师直接领导下工作。他最重要的工作是帮助确立整个游戏的统一的视觉风格(visual style)，也就是游戏的外观，包括色彩配置、文化背景、特殊图案、角色设计风格、建筑设计风格、界面设计风格等。最后要编写一部风格指导手册(style guide)，美国人又叫它 the art bible。风格指导手册的作用是使得所有美工在进行工作时有所依据。当他们进行具体工作时可以对照风格指导手册，根据其要求调整自己的作品，从而保证整个游戏画面风格一致。

艺术指导的另一个重要职责就是评审美工的作品。一般来说，美工作品经过艺术指导和主设计师的评审通过后，就入库不再轻易修改了。

所以艺术指导的工作基本上是大面上的，是指导性的而非实践性的，是概括性的而非具体的。

●概念美工(concept artist)

概念美工是艺术指导最重要的助手。他(们)帮助艺术指导把主设计师的 vision 转换成画纸上有型的东西。概念美工要根据主设计师和艺术指导的解释，简单迅速地画出各种可能的方案，包括人物设计、场景设计、基本物品设计等。概念美工的作品，被称为概念速写(concept sketch)。可以是一般的速写草图，也可以是比较细致的钢笔淡彩，或者马克笔. 概念速写经过和主设计师和艺术指导多次反复评审通过后，就将成为风格指导手册的一部分，被拿给下面的 3D 美工去建模，2D 美工去绘制材质。

一般来说在一个游戏公司中，概念美工都是经过了比较严格的传统美术训练的。他们的造型能力、绘画技巧、色彩感觉等都是最强的。这是由他们的工作性质决定的。其他美工这方面的要求就不那么高。比如说 3D 美工不一定要求画得好，他们只需要根据画稿去建模就行了。

概念美工的另一项任务是为过场动画(cut scenes)绘制故事板。显然，绘制故事板需要很高的绘画技巧和对电影构图及镜头控制的一定认识。一般电影制作，特别是动画片制作中，需要专业的故事板画家(storyboard artist)。他们的作品虽然很少能够最终展现在观众面前，但他们的工作是一部电影的基石。他们所绘制的故事板，线条老道、构图精妙、言简意赅，其高超的技巧令人叹为观止。

对游戏公司来说，如果不能找到专业故事板画家，则最有能力担当这个职务的就是概念美工了。所以在大部分游戏公司中是由概念美工兼任故事板画家的。

●3D 美工(3D Artist)

3D 美工的任务是参照概念美工的设计草图，利用 3D 软件，构建游戏中的人物、生物、建筑、物品的三维模型。特别需要指出的是：3D 美工中又有不同的侧重和分工。比如说人物建模和建筑建模需要不同的技巧，用于过场动画的高分辨率模型和用于游戏中的实时低分辨率模型也有很大不同，这些必然导致在 3D 美工中更细的分工。

●2D 美工(2D Artist)

早期的游戏都是 2D 的，所以那时候的美工都是 2D 美工。他们的任务是使用 2D 绘图软件在计算机上绘制低分辨率的图像，包括背景和 sprite。3D 游戏流行后，2D 美工的任务改变了。他们现在的任务主要是为三维模型绘制材质(texture)，以及界面上使用的背景等图像。

●动画师(animator)

如果一个游戏中使用大量的片头和过场动画的话，专业的动画师将是必不可少的。这里需要强调二点：动画有其自己独特的规律和法则，没有经过专业培训的人是无法胜任的，随便找一个美工来凑数是不能制作出高水平的动画的。动画师一般要对传统手绘动画有很深刻的理解，包括 Disney 动画的 12 条经典法则，以及相应的镜头和构图方面的知识。没有这些知识，盲目追求 3D 动画效果只能是缘木求鱼。

制片人

制片人(producer)，又称项目经理(project manager)。在制片人中，又有内部制片人和外部制片人之分。内部制片人为游戏开发公司(de veloper)工作，外部制片人为游戏发行公司(Publisher)工作。以下分别介绍。

●内部制片人(internal producer)

内部制片人就是一般意义上的项目经理，是游戏开发公司的雇员，为游戏开发公司工作。他负责游戏开发项目的管理，包括前面章节中介绍过的各种软件工程和项目管理方法的具体实施，其中最重要的是计划的制定、执行和监控。

内部制片人是一个项目组对外联系的桥梁。他负责向公司上层管理者汇报开发的进度和遇到的问题。他维持和媒体的联系，不断发布关于游戏的新消息。他更要和来自游戏发行公司的外部制作人打交道。

内部制片人的很重要的一项任务是人员的招聘和管理。因为那些软件工程和项目管理方法，最后都要靠人来制定，遵守和执行。招聘到合适的组员是个艰巨的任务。他们不仅需要满足基本的技术性要求，更重要的是新组员能够比较适应整个项目组的氛围，大到所谓的“企业文化”，小到同事的衣着举止。如果不能适应的话，则会带来很多棘手的问题，如果组里两个人谁看谁都别扭，那这个项目一定要出问题。一个和谐的工作环境和人际关系对开发工作是至关重要的。

●外部制片人(external producer)

外部制片人不是游戏开发公司的一员，他们为游戏发行公司工作。美国游戏业界的运作是这样的：游戏开发公司开发游戏，但并不是 100%用自己的钱，而是靠游戏发行公司以分期付款的方式来支持。游戏发行公司要拿出钱来，自然要承担风险，必然需要人去监督钱的使用。外部制片人就是担当这种监督的角色。他的任务是作为投资方的代表来掌握项目开发的情况，并根据开发进度和效果，决定是否追加下一步投资。

外部制片人最重要的能力是交流沟通的能力--他必须成为投资方和制作方之间的桥梁，保证双方交流沟通管道的畅通。对投资方来说，他们需要了解游戏开发是否按计划进行了，质量是否达到了要求。对制作方来说，他们最关心的是下一步钱什么时候到位，下

月的工资是否能发出去。双方都迫切需要了解对方的最新动态，而外部制片人就是他们的通信员。

对外部制片人来说，虽然他有控制资金的生杀予夺的大权，但这并不意味着他可以对游戏和游戏开发公司随便说三道四。由于其特殊地位，外部制片人在和游戏开发项目组成员在非正式场合接触时发言必须十分谨慎，否则不经意的批评或者吹捧都可能极大地影响游戏开发公司的工作。但同时，在游戏发行公司内部分阶段评审了游戏开发公司的工作之后，外部制片人必须向游戏开发公司明确传达发行公司的最新要求和看法，并严格监督其贯彻实施。因为最终决定游戏的命运的，是游戏发行公司而不是开发公司。

一般来说，外部制片人同时兼管几个外部项目。但根据业界的经验，超过三个以上的话，外部制片人就忙不过来了。对内部制片人来说，一般来说一个人只能管一个项目。

●助理制片人(assistant producer)

助理制片人是内部制片人的助手，他们的工作包括保证游戏开发顺利进行的一切琐事。从维护系统数据库和各种文档和文件，维护游戏开发专用的内部网站，到每日备份更新，接待媒体，为媒体准备游戏画面(screenshot)和录像，甚至包括收发邮件和为大家订午餐等琐事。

测试员

●主测试员(the test lead)

主测试员在主程序员指导下工作。他的任务是根据游戏程序的特点，参与制定一整套测试，报告，修改，审核机制，编制测试报表，制定测试计划，指导协调一般测试员的工作。

主测试员绝对需要对软件工程和程序设计有足够的了解。实际上，在很多美国公司里，主测试员都是既有很高的计算机学历又有很多工作经验的。他们是游戏制作流程中最后一道把门的，对游戏的成败起关键性的作用。以前业界曾有一种倾向把测试员们当成“食物链”的最下层的，认为他们从事的不是创造性的工作，所以他们是可有可无的。现在事实已经无情地驳斥了这种看法。在中国游戏公司中，这方面做得不如美国公司。我们迫切需要提高测试员，特别是主测试员在公司中的地位。

●一般测试员(tester)

这是一群以玩游戏为工作，以玩游戏为生的人。人们肯定会说：整天一边玩游戏一边挣钱，这是多么理想的职业啊!但与人们的普遍误解相反，以玩游戏为生不是一件轻松并且愉快的事情。对游戏测试员来说，玩游戏已经不是一项乐趣，而是一项必须完成的任务。这项任务必须在严格的时间限制内完成，测试员必须找到程序中的 bug，完成细致的记录

工作，并及时将报告呈交给主测试员。

一般来说，游戏公司内有长期雇用的专业测试员，也有临时雇用的短期测试员。长期测试员参与一个游戏自始至终的测试，临时测试员是在项目后期测试任务加重的时候加入的，测试完成后离去。长期测试员对公司内部已经确立的规章制度和测试流程比较清楚，在临时人员加入时必须培训并指导这些新手。

其他人员

目前有一些新的职务随着游戏设计的发展和技术的进步应运而生。比如说人机界面设计师(user interface designer)。人机界面设计师专门负责游戏中人机界面的设计和测试。他们在主设计师指导下工作。其任务是设计出好学、好用、高效、并且容易记忆的界面和交互。在本书人机界面一章中，对人机界面设计师的工作的性质和范畴有详细的介绍。

制作游戏还需要很多其他人员，比如说公司的 PC 及网络维护人员、秘书等。他们都是十分重要的。作为公司的领导者，要把平等的观念灌输给公司内的每一个员工。因为大家在一起开发游戏，每个人没有贵贱之分，只是工作性质不同。每个人都被尊重，都开心才能开发出好游戏来。一个著名游戏设计师曾说过：任何一个成功游戏的背后，都是一个和谐的大家庭一样的项目组。同样，一个失败的游戏背后，很可能就是一个职责不明、相互轻视、内耗严重的项目组。这一点是必须引起足够重视的!

教育篇

第二十二章　游戏的教育

本章所要谈的是一个成熟的游戏市场和游戏产业所必需的基础教育问题，基本上以美国市场和学校为例，希望能够对国内有所启示。

从手工作坊式教育到学院式教育——新兴产业必由之路

一个新型产业发展之初，必然是从其他传统相关产业中吸收人才，然后通过实际工作使得他们获得足够的技能和经验，成功地转型到这个新兴产业上来。这种模式，实际上是一种师傅带徒弟的手工作坊的教育方式。其缺点是时效性差，像以前的学徒一样，要几年才能出师；培养面狭窄，因为受公司规模和雇佣人员的限制，这种培训只能针对极少数人；更严重的问题是培养出来的人才过于实用而缺乏前瞻；而其优点是通过这种途径培养出来的人实战能力强，因为是在实际工作中锻炼出来的；并且和本公司或者整个产业的文化比较协调，不像学校里出来的会感到不适应。目前的游戏业所采用的也是这种手工作坊的教育方式。以游戏设计师这一职业为例，在美国没有任何公司会直接从大学毕业生中招聘高级游戏设计师(就是能够负责整个游戏策划的人)，都是从年轻人中招聘初级的关卡设计师，然后在实际工作中让他们逐渐提高技能并筛选，最后提拔到高级游戏设计师。这种内部提拔培训机制在以前是比较有效的。

随着产业发展壮大，人才需求越来越急迫，对人才的素质要求也越来越高。游戏产业面临着人才的饥荒——到什么地方去找人才，如何找到高素质的人才呢?如果无法保证满足人才需求的话，产业的发展速度必然受到影响。游戏产业这时意识到单靠产业自己是解决不了人才问题的，于是施行学院式教育，或者说求助于学院式教育的可能性就被提上议事日程了，以电影为例，20 世纪 50 年代以前的好莱坞电影导演都不是从学校出来的，因为那时学校里也没有电影系和电影课。后来大学里普遍设立了电影系，对各项技术的总结和研究也日渐深入，70 年代以后的导演，如乔治·卢卡斯(南加州大学)等都是大学电影系科班出身。

从精英教育到职业培训——学院式教育的发展

从历史角度看，近现代西方的学院式教育是在中世纪的神学院(monastery)和行会组织(guild)这两个基础上发展而来。学院式教育主要体现在以下几方面：相对封闭，与社会一定程度上分离(神学院的影响)；教授(知识分子)是一个具有自己特殊准则的独立群体，有自己的等级和升级系统(行会组织的影响)；在一定时间里学生们集中精力全时学习，有严格的课程设置和考核规则，培养知识的广度重于深度，培养前瞻性重于实用性，等等。学校，特别是大学，在西方人眼里是极为神圣的所在。在 20 世纪以前的西方社会，基本上都是施行的

这种精英教育。但进入 20 世纪，随着教育的普及，学院教育也改变了。这种改变体现在美国的州立大学和社区大学系统，上大学的人越来越多，精英教育显然不能满足大多数人的需要，因为不是人人都可能去当精英的。你哈佛、耶鲁可以说我们培养的是美国未来领导人，但其他大学显然不能以这种思想办学。大部分学校开始注重学生工作技能和专业知识的培养，也就是说，把学院看成学生走向社会参加工作的准备，这样从整体来看，学院式教育从一种精英教育演变成了工作培训的性质了。最显著的例子就是计算机，现在是市场上迫切需要什么新技术，或者科学家群体预测将来市场需要什么新技术，然后 ACM(美国计算机协会)针对这一领域，制定颁布指导性教学大纲，以便各学校的教授们在学校里开讲这方面的内容，因此，市场对学校教什么，设定什么课程，都起决定性的作用了。实际上这也体现了业界和学校的人才供需关系。

美国游戏教育的现状

既然游戏产业和游戏市场有这个需求，必然会影响到学校的教育。我们看到美国很多大学中已经开始设立了很多关于游戏的课程，以卡内基美隆(Carnegie Mellon)大学为例，先后开的课有“游戏的戏剧结构”，“游戏文化评论”，“游戏编程实验”，“虚拟世界的构建”等课程。斯坦福大学则有“游戏设计的历史：技术，文化和商业”等课程。其他大学的各种有关游戏的计算机、行为科学、社会科学、文学等方面的课程也有很多。可以说，学校的教授们是从多角度、全方位地看待游戏的。但这些课程目前没有系统化起来形成一个统一的连贯的教学方案，因此只是给学生们过过瘾，并不能起到培训的作用。只有两个大学现在有比较成熟的专业设置，一个是卡内基美隆大学的娱乐技术中心(Entertainment Technology Center)的娱乐技术专业，另一个是南加州大学的电影电视学院设立的游戏及交互娱乐专业(Games and Interactive Entertainment)。这两个学校，一个是计算机名校，在软件技术、人工智能、人机交互等方面赫赫有名；另一个是电影名校，自 70 年代以来培养出一批大牌导演。他们的这两个专业，体现了从技术和艺术两个角度对游戏的重视。

游戏教育的难题

仔细研究这两个专业的课程设置，我们会发现很大问题：这两个专业是以其他专业为基础发展来的，有很大的模仿的影子。比如说卡内基美隆大学的娱乐技术专业是从人机交互专业(HCl)里分出去的，因此其课程中很多照抄 HCI 的课程，然后加上其他系已经有的课程，关于游戏本身独特的艺术 / 技术规律的课程还是很薄弱。南加州大学的专业，则是从电影和三维动画为基础发展起来，这两方面比较强，游戏本身的内容也比较弱。

这个问题实际和游戏的现状有关，游戏基本上有三个根基。第一是技术根基，包括计算机软件，人工智能、图形学、界面、软件工程等方面，一般大学的计算机系的课程都包括这些内容。第二是艺术根基，包括绘画、三维建模、动画、视频处理剪辑、电影技术等等，一

般大学的艺术系或者专门艺术院校都可以教授这些内容。第三，游戏本身的艺术规律和技术特点。这方面在目前来看是没有任何大学有能力教授的。因为无论是业界还是教授们都还没有能够真正了解游戏本身的法则和规律，游戏设计本身还只是一种盲动。因此，游戏的教育自然出现前两项强，最后一项弱的情况。

游戏教育的另外一个问题是缺乏合格的教师。大部分开游戏课的教授们是把游戏当成比较抽象的东西来研究，比如说，研究游戏和社会的关系、游戏中的性别角色问题等等。他们本身没有游戏开发经验，对高新技术的掌握很多，但具体的实践不在他们兴趣之内。而游戏界的游戏设计师们，大多数是手工作坊内培训出来的，在业界摸爬滚打可以，但上讲台对他们来说太困难了。他们的学历和水平，包括总结归纳的能力和知识的广度甚至表达能力都不够。形象地说就是开水壶里煮饺子——有东西倒不出来。当然极少数顶级设计师的水平是很高的，但他们都在忙着做游戏，不可能走上讲台传授他们的知识。

Digipen 理工学院

既然大学目前无力满足游戏业的人才需求，专门的游戏学校就出现了。在日本这样的学校很多，但在美国，有名的只有一家。那就是由任天堂协办的 Digipen 理工学院。这个学院位于西雅图，学制为 8 学期，每学期 15 周；授予大学本科或者专科学位。其教师不是学术上有成就的教授，而是有游戏开发经验的游戏开发人员。其课程设置可以说完全是针对游戏业的需要，游戏需要什么就有什么，比如说基础课，除了基本的数学和物理就没有其他的。然后是编程、动画等课程。再后是游戏设计的课程，包括理论和实践课。最后学生们可以从头到尾做出一个比较像样的游戏，还有全部文档资料。这样的毕业生，游戏公司们可以雇佣后马上就用。

但这样的教育有严重的问题，就是毕业生视野狭窄，只知道游戏，而不知道其他东西。而业界真正顶级的游戏设计师不是那种只知道玩游戏的人，而是能够从更广大的历史文化角度把握游戏本质的人。因为电子游戏只有 20 年左右的历史，而人类的各种娱乐可是有上千年历史了，而且人类本身的进化是很缓慢的。在这个技术发展如脱缰野马的时代，只有把握住历史和人性的根本，才能驾驭技术而不是被技术所驾驭。笔者曾经听过《虚拟人生》系列的主设计师 Will Wright 的讲座，深为其知识广博、精深所折服。在那个讲座上，Will Wright 不只讲了《虚拟人生》的设计，而是从日本园林，一直讲到工业设计中的椅子，令人耳目一新。如果毕业生只知道游戏的话，最终他们只能达到业界的中下层，而不可能真正成功。对这一点，中国古人也有“汝果欲学诗，工夫在诗外!”的精辟见解。因此 Digipen 这种游戏学校，实际上是为游戏业培养廉价初级劳动力的场所，并不是游戏教育的最好的解决方案。

走笔至此，本书也要告一段落。对于如何在中国进行游戏教育这个问题，笔者并没有在书中给出答案，因为它超出了本书的目的和范畴。本书是在游戏教育方面进行的一点微薄的

尝试，而要真正实现成功的游戏教育，则还有很长路要走。

术语对照

2D artist	2D 美工
3D artist	3D 美工
3D digitizing pen	三维数字笔
3D engine	3D 引擎
3D morphing	3D 变形动画
3D programmer	3D 程序员

A

AABB(axis-aligned bounding box)	轴对齐包容盒子
AAVG(Action Adventure Game)	动作冒险类
ACM(Association for Computing Machinery)	美国计算机协会
ACT(Action Game)	动作游戏
action sequence	指令序列
activation function	激活函数
activation level	激活阈值
affordance	自解释性
Age of Empires	《帝国时代》
agent	代理
AI（Artificial Intelligence）	人工智能
AI programmer	人工智能程序员
AI script	AI 脚本
ALIFE	人工生命
alignment	齐向性
Alone in the Dark	《鬼屋魔影》
ambient light	环境光
ambient light	环境光
animator	动画师
anticipation	预示
appeal	吸引力
archetype	人物原型
arcs	弧线运动
area light	区域光
ARPG(Action RPG)	动作 RPG
art bible	美工指导手册
art director	艺术指导
Artificial Fish	人工鱼
asistant producer	助理制片人
assistant designer	助理设计师

atmospheric effects	空气效果
autonomous agent	自主代理，自主智能单元
AVG(Adventure Game)	冒险类游戏

B

back light	背面光源
Baldur's Gate	《柏德之门》
baseline	基准
best-first search	最佳优先搜索
beveling	削斜角操作
bidireetional search	双向搜索
Black & White	《黑与白》
black-box testing	黑盒测试
blend shapes	融合外形
blending	融合
blobby surface	密度球体技术
Boolean operations	布尔运算
bouncing ball	跳跃小球
boundary patches	边界面片
bounding box	包容立方体
bounding plane	包容面
bounding sphere	包容球
bounding volume	包容体
breadth-first search	广度优先搜索
broad phase / narrow phase collision detection	两步法碰撞检测
BSP tree(binary space partitioning tree)	空间二分树
Bulge	凸凹
bump mapping	凸凹处理
bump texture mapping	凸凹贴图

C

camera animation	镜头动画
canonical view volume	正规化可视空间
CAVE(Cave Automatic Virtual Environment)	洞穴状自动虚拟系统
centroid	中心点
chain	链结构
checklist	审核明细表
chiaroscuro	明暗效果
Chrono Trigger	《时空之匙》
Civilization	《文明》
close shot	近景镜头

close up	特写镜头
CMM(Capability Maturity Model)	CMM 模型
code path	代码路径
code test	代码测试
cognitive modeling	认知模型
cognitive science	认知科学
Cognitive Walkthrough	CW 方法，一种测试软件的可学习性（learnability）的方法
cohesion	内聚性
collision detection	碰撞检测
color system	色彩系统
Command & Conquer	《命令与征服》
common visual axis	普通视觉轴
concept artist	概念美工
concept sketch	概念速写
cone	圆锥体
cone of vision	眼睛视锥
configuration management	改动控制
Contextual Inquiry	CI 方法，一种研究用户需求并建模的方法
control point	控制点
convex polygon	凸多面体
copy	复制
coverage	覆盖率
CPM(critical path method)	关键路径法
cross-over	交叉组合
CUBE(Computer-driven Upper Body Environment)	计算机驱动上半身显示环境
cube	立方体
cube projection mapping	立方体投影贴图
cubic reflection mapping	立方体反射贴图
curve editing	曲线编辑
curved surface	曲面
cut	剪切
cut scenes	过场动画
CW 方法	Cognitive Walkthrough 方法
cylinder	圆柱体
cylindrical projection mapping	圆柱投影贴图

D

data visualization	数据信息的可视化
DDR(Dance Dance Revolution)	跳舞机
decay / fall off	衰减

deceptive visual match	视觉相似欺骗
decision tree	决策树
deform	变形
degree of freedom	自由度
density	密度
density map	密度图
depth of field	景深
depth-first search	深度优先搜索
design methods	设计方法
design process	设计流程
desktop	桌面
developer	游戏开发公司
difference / subtraction	差集
diffuse reflection	漫反射
diffuseness	发散率
direct manipulation	直接操作
directional light source	定向光源
displacement texture mapping	位移贴图
dissolve	溶解效果
divide and conquer	分而击之
Dragon Quest	勇者斗恶龙
drop off	边缘淡化
DTP	桌面排版
duck	压铁
Dune	《沙丘》

E

earned value	赢值分析方法
ease in / ease out	渐入渐出
effector	受动器
efficiency	效率
elasticity	弹性
entry-level programmer	初级程序员
environmental density	环境密度
environmental reflection mapping	环境反射贴图
evaluation function	评估函数
exaggeration	夸张
experimental animation	试验型动画短片
expert system	专家系统
extensible AI	可扩展性 AI
external producer	外部制片人
external reverse angles	外部相反角度

extreme close up	极度特写镜头
extrude	伸展
eye tracking	眼球跟踪

F

face tracker	面部表情捕捉
faceted shading	碎面描影法
facial expression	表情
fade out and fade in	淡人淡出
fan	风扇
FC	任天堂 8 位游戏机
feed-forward	前向式(神经元网络)
fidelity	真实度
field-of-view	取景域
field-of-view annie	取景角度
fill light	弥漫性光源
film noir	黑色电影
Final Fantasy	最终幻想
fitness function	适应函数
Fitt's Law	费茨定理
FK(forward kinematics)	前向运动学
flat shading	平面明暗处理
flocking	群体移动
flocking algorithm	畜群算法
fog light	雾灯
follow	跟进运动
follow through and overlapping action	跟进与重叠运动
force	力量来源
formation	队形
forward mapping	材质处理前向算法
FPS(first-person Shooter)	第一视角射击游戏
fractal geometry	不规则碎片几何体
frame per second	每秒钟帧数
free-form modeling	全手工建模
FSM(Finite State Machine)	有限状态机
FTG	格斗游戏
full shot	全景镜头
function points	功能点
functional organization	基于功能划分的结构
functional specification	功能规格说明书
FuSM(Fuzzy State Machine)	模糊状态机
fuzzy logic	模糊逻辑

G

game genre	游戏类型
game logic programmer	基本游戏规则程序员
gameplay	游戏性
gamer test	玩家测试
Gantt chart	甘特图
GDC(game developers conference)	游戏开发者大会
genetic algorithms	遗传算法
genre	类型
genre conventions	类型范式
genre cycle	类型环
genre game	类型游戏
geometrical transformation	几何变换
geometry	三维模型
gesture control	姿势控制技术
gesture recognition	姿势识别
gimbal lock	万向节锁定
glow	自发光
goal oriented animation system	目的驱动动画系统
God Game	上帝模拟游戏
Gouraud shading	Gouraud 明暗处理
gravity	重力
grid system	排版格子系统
GUI(Craphical User Interface)	图形用户界面

H

Half-Life	《半条命》
HCI(Human-Computer Interaction)	人机交互理论
Heroes of Might and Magic	《魔法门英雄传说》
Hero's Journey	英雄之旅理论
Heuristics Evaluation	HE 方法，一种测试软件可用性的方法
hidden surface removal	隐藏面消除
hierarchical architecture	分层结构
hierarchical finite-state-machine	层次 FSM
hierarchical pathfinding	分层寻径
highliSht decay	高光衰减
HMD(hcad mounted display)	头盔式显示器
hot key	热键
hull	外壳
human-computer interface	人机界面
hypermedia	超媒体

hypertext	超文本

I

icon	图标
IK (inverse kinematics)	反向运动学
illumination equation	光线公式
illumination model	光线模型
image-precision algorithms	精确到点算法
immersion	置人感
inbetween	中间帧
inbetweening	在关键帧中间加入中间帧
incandescence	炽热
incandescence mapping	自发光贴图
incremental calculations	增量计算法
incremental model	渐增模型
individual-based model	基于个体的模型
infinite / directional light	方向光源
influence maps	感应地图法
information architecture	信息结构学
In-game UI	游戏中界面
in-house test	实地测试
input function	输入函数
inspection	小组审核
integration test	集成测试
intelligent agent	智能化代理技术
intensity interpolation shading	强度插值明暗处理
interaction	交互
interactive cyle	人机交互环
interactive storytelling	交互式故事
interactivity	交互性
interface agent	界面代理
internal producer	内部制片人
internal reverse angles	内部相反角度
internationalization	国际化
interpolating	插补操作
interpolation shading	插值明暗处理
intersection	交集
Intranet	内联网
inverse kinematics	逆运动学
inverse mapping	材质处理逆向算法
iris	彩虹效果
iterative design	复进式设计

K

key light	主光源
key shape	关键外形
keyframe	关键帧
keyframed acceleration	关键帧加速度
key-framed animation	关键帧动画
kinetic friction	动摩擦力
King of Fighters	《格斗之王》
knee shot	膝盖镜头
Kohan	《可汗》

L

laser scan	激光扫描
lathed surface	纺梭面建模(车床旋转面建模)
lattice animation	网格动画
lattices	网格操作
lead	主导运动
lead artist	主美
lead designer	主设计师
lead programmer	主程序员
learnability	可学习性
level	关卡
level design	关卡设计
level designer	关卡设计师
level editor	关卡编辑器
life span	生命周期
lighting	照明
lighting links	照明连接
lighting links	镜头光晕
lights	照明
Lindenmayer systems	L 系统(用于模拟植物的生长)
line	线
line of action	动作基线
line of interest	兴趣线
linear approximation	线性近似
linear interpolation	线性插值
linear sequential model	线性模型
LOC (line of code)	代码行数
local space	本地空间
local-to-world matrix	本地空间到世界空间转换矩阵
lofted surface	梯形面

long shot	长镜头
look-up table	速查表

M

machine learning	机器学习
Manhattan distance	曼哈顿距离
mass	质量
master shot	主场景
matrix organization	矩阵式结构
medium close up	极度特写镜头
medium shot	中景镜头
memorability	可记忆性
menu	菜单
merging	缝合操作
mesh	网
Metal Gear Solid	《合金装备》
MFC(Microsoft Foundation Class)	微软基础类库
Microsoft Flight Simulator	《微软飞行模拟》
Miyamoto Box	宫本茂盒子
MMORPG(Massively Multiplayer Online Role-Playing Game)	大规模多用户网上 RPG
model sheet	模型板
motion capture	动作捕捉技术
motion dynamics	动力学模拟
motion path	运动轨迹
motion preview	动态预览
muhilayer network	多层网络
muhi-modal	多通道交互技术
mutation	突变
MVC(Model-View-Controller)	一种软件模式
MYST	《神秘岛》

N

navigation mesh	导航网
network programmer	网络程序员
neural network	神经元网络
Neverwinter Nights	《无冬之夜》
non key-farmed animation	非关键帧动画
noncommand interface	非指令性界面
normal vector	法矢量
normalization	正规化处理

normalization transformation	正规化变换
normal-vector interpolation shading	法矢量插值明暗处理
NPC(None-Player Character)	非玩家控制的角色

O

OBB(OrienteD bounding box)	定向包容盒子
object space	物体空间
object-precision algorithms	物体比较算法
Onimusha	(鬼武者)
orientation	方向性
orthographic reconstruction	三视图重建法

P

PACMAN	《吃豆》
painter's algorithm	画家算法
panning	摇镜头
Paiazer General	《坦克将军》
parallel positions	平行位置
parallel projection	平行投影
parameter curve editing	参数曲线编辑
parameterized texture mapping	参数贴图
particle system	粒子系统
particle-like system	类粒子系统
panick-system modeling	粒子系统建模
paste	粘贴
patch	面片
path deformation	路径变形
path locking	路径锁定
pathfinding algorithm	寻径算法
perceptron	感知器
persistence of vision	视觉停留原理
personal reviews	个人审核
perspective projection	透视投影
PERT（program evahtion and review technique）	计划评审技术（PERT 图）
Phong shading	Phong 明暗处理
physical simulations	物理模拟建模
physics programmer	物理程序员
pitch	前倾角
pixel	像素点
placement	贴图位置

planar projection mapping	平面投影贴图
plane	面
plant generators	植物生成器
playtest	游戏性调节测试
point	点
point light	点光源
point lighti source	点光源
points of visibility	可视点法
polygon	多边形
polygon mesh	多边形网
polygon meshes	多边形模型
polygonal approximation	多边形近似
polygonal modeling	多边形建模
portal	门户网站
portal	门户
Prince of Persia	《波斯王子》
procedure animation	过程动画
process	过程
producer	制片人
project manager	项目经理
project organization	基于项目的结构
projection	投影
projection texture mapping	投影贴图
props	道具
prototype	原型
publisher	游戏发行公司
pull-down menu	下拉菜单
Puzzle	解谜

Q

quadtree	四叉树
quatemions	元数法
question and answer	虚假的因果关系

R

RAC（Racing Game）	赛车游戏
rapid prototyping	快速原型法
ray tracing	光线跟踪算法
recurrent	循环式(神经元网络)
re-establishing shots	重新建立对话场景
reflection mapping	反射贴图

reflectivity / mirror	反射
refractivity / bend	折射率
rendering algorithm	渲染算法
rending	渲染
right angle positions	右角度位置
roll	翻滚角
root	根结点
rotoscoping	一种动画技术，使用摄像机把人物动作拍下来，再使用特殊仪器将影像投影到动画师的玻璃画板背后，供动画师参考
rounding	边缘制图
RPG(Role-Playing Game)	角色扮演类游戏
RTS(Real-Time Strategy Game)	实时策略游戏
rubber hose animation	橡皮管动画
rule-based AI	基于规则的 AI

S

scale	缩放比例
scene	场景
scene file	场景文件
scene matching	场景匹配
screenshot	屏幕截图(游戏画面)
script editor	编程语句编辑器
script language	脚本语言
seamless	无缝连接
secondary action	辅助动作
senior programmer	高级程序员
separating axis	分割轴
separation	间距性
shaded-face rendering	阴影表面渲染
shader	物体表面属性的集合
shading	描影法
shading language	描影法语言
shading model	明暗模型
shape changing animation	外形变化动画
shear	斜拉
Shell UI	游戏初始及设置界面
Shogun：total war	《幕府将军》
Shrek	《怪物史莱克》
side light	侧面光源
sigmoid function	S 曲线函数
sign function	符号函数

SIM(Simulation Game)	美式模拟游戏
Simcity	《模拟城市》
single phase collision detection	一步法碰撞检测
skeletal animation	骨架动画
SLERP(spherical linear interpolation)	球面线性插值
SLG(Simulation Game)	日式模拟游戏
slow in and slow out	慢进慢出
smooth shading	柔和描影法
software architect	软件结构设计师
software engineer	软件工程师
software engineering	软件工程
software metrics	软件度量
software pattern	软件模式
software process	软件过程
software process models	软件过程模型
solid drawing	立体感
solid modeling	实体建模
solid texture mapping	实体材质贴图
South Park	《南方公园》
space subdivision	空间划分
space-oriented procedural techniques	基于空间的模拟技术
spectacular reflection	镜面反射
specular shading	反射描影法
specularity	高光
specularity mapping	高光贴图
speech recognition	语音识别
sphere	球体
spherical projection mapping	球体投影贴图
spherical reflection mapping	球体反射贴图
spot light	局部光源
sprite	活动块
SPT(Sports Game)	体育类游戏
squad	小组
squash and stretch	挤压和拉伸
staging	展示
static friction	静摩擦力
step function	阶越函数
STL(Standard Template Library)	标准模板类库
storyboard artist	故事板画家
storytelling	叙事方法
straieht ahead action&pose-to-pose action	非关键帧动画和关键帧动画
Street Fighter	《街霸》
structure-oriented procedural techniques	基于结构的过程模拟技术

style guide	美工指导手册
subdivision	面细分建模
subieetive view	主观视角
subproject	子项目
surface characteristics	物体表面属性
surface modeling	表面模型(表面建模)
system test	系统测试

T

Tamagochi	宠物蛋
tangent vector	切线矢量
taper	锥形放缩
TBS(Turn-Based Strategy Game)	回合制策略游戏
team AI	小组智能
tech demo	技术演示
technology	技术手段
Tekken	《铁拳》
terrain analysis	地形分析
test case	测试案例
test lead	主测试员
tester	测试员
TETRIS	俄罗斯方块
TETRIS	《俄罗斯方块》
texel	材素点
texture	材质
texture map	材质
texture mapping	材质处理
texture mapping	材质贴图
The Sims	《虚拟人生》
the triangle principle	三角形法则
the Wizard of OZ	《绿野仙踪》
think-aloud protocols	宣泄法测试
tile	图片单元
tile-based map	拼接地图
time props	时间道具
timing	时间控制
timing curve	时间曲线
Tomb Raider	《古墓丽影》
tools programmer	工具程序员
torus	圆环面
Toy Story	《玩具总动员》
tracking	跟踪镜头

transparency mapping	透明贴图
transparent mapping	透明贴图
traveling	平移镜头
truncated cone	去顶圆锥体

U

UCD(User-Centered Design)	基于用户的设计
UI(user interface)	用户界面
UI programmer	界面程序员
UML(Unified Modeling Language)	统一建模语言
union / addition	并集
unit test	单元测试
UNREAL	《虚幻》
usability	可用性
usability lab	软件可用性实验室
use of dark areas	使用阴暗的区域
user interface designer	人机界面设计师
user test	用户测试

V

vertex	顶点
view	视图
view point	视点
view volume	可视空间
Virtual Fighter	《VR 战士》
virtual sculpting	模拟雕塑
visibiUty graphs	可视图法
vision	对现在还未具体实现的事物的总体上的掌握、前瞻性的理解和策略上的考量
visual punctuation	视觉标点法
visualization	可视化
VR(Virtual Reality)	虚拟现实

W

waist shot	中度特写镜头
walk cycle	行走动作循环
walk-through	通读法
Warcraft	《魔兽争霸》
waterfall model	经典瀑布模型
waypoint graph	中继点图

WBS work breakdown structure	工作拆分结构图
white-box testing	白盒测试
white-outs and color fades	白屏和彩色淡入淡出
WIMP(window-icon-menu-poiming device)	WIMP 界面(窗口、图标、菜单、鼠标)
wind	风力
wipe	擦抹效果
wire-frame rendering	线框渲染
work package	WBS 工作包
world space	世界空间
world-to-camera matrix	世界空间到镜头转换矩阵
wrap	重复
writer	剧本作家
WYSIWYG(What You See ls What You Get)	所见即所得

X

XEROX PARC	施乐研究中心

Y

yaw	偏向角

Z

z-buffer algorithm	Z 缓存算法
zooming	调焦镜头

参考书目

1 Dennis Jones, What is a CAVE? http://www.sv.vt.edu/future/vtcave/whatis/
2 Joseph Dauben, The Art of Renaissance Science: Galileo and Perspective, http://www.crs4.it/Ars/arshtml/arstitle.html.
3 《艺术原理》，[英]罗宾.乔治.科林伍德，中国社会科学出版社，1985 年。
4 《音乐的情感与意义》，[美]伦纳德·迈尔，北京大学出版社，1991 年。
5 《软件工程导论》，清华大学出版社。
6 《数据结构》第二版，严蔚敏，清华大学出版社。
7 Ben Shneiderman, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, 1995.
8 David Bordwell, Kristin Thompson, Film Art: An IntroductiOn, 5th ed., McGraw Hill, 1997.
9 Pam Cook, Mieke Bernink, The Cinema Book, 2nd ed., British Film Institute publishing, 1999.
10 Marc Saltzman, Game Design: Secrets of the Sages, Macmillan Publishing, 2000.
11 Jakob Nielsen, Usability Engineering, Morgan Kaufmann, 1993
12 Jakob Nielsen, Robert L. Mack, Usability Inspection Methods, John Wiley & Sons, Inc, 1994.
13 Hugh Beyer, Karen Holtzblatt, Contextual Design： Defining Customer-Centered Systems, Morgan Kaufmann, 1998.
14 Pierre Alexandre Garneau, Fourteen Forms of Fun, Gamasutra.com, 2001.
15 Chuck Clanton, An Interpreted Demonstration of Computer Game Design, in the Proceedings of CHI 98.
16 Randy Pausch, Rich Gold, Tim Skelly and David Thiel, What HCI Designers Can Learn From Video Game Designers, in the Proceedings of CHI 94.
17 Christopher Vogler, The Writer's Journey 2nd Edition, Michael Wiese Productions, 1998
18 V. Propp, Morphology of the Folktale, University of Texas Press, 1968.
19 Chris Crawford, Understanding Interactivity, 2002.
20 Chris Crawford, Assumptions underlying the Erasmatron interactive storytelling engine.
21 Tito Pagan, Where's the Design in Level Design? Gamasutra.com, 2001.
22 Ivan Beram, Levels of Complexity: A Level-Design Methodology, Gamasutra.com, 2001.
23 Paul Warne, Three Inspirations for Creative Level Designing, Gamasutra.com, 2001.
24 Jim Molinets, Level Design Strategies and Pitfalls, in Proceedings of Game Developers Conference 1999.
25 Archer Jones, The Art of War in the Western World, University of Illinois Press, 1987.
26 Greg Street, Sandy Peterson, How to Balance a Real Time Strategy Game: Lessons from the Age of Empires Series, in Proceedings of Game Developers Conference 2001.
27 Andrew Rollings and Dave Morris, Game Architecture and Design, The Coriolis Group, 2000.
28 Donald A.Norman, The Design of Everyday Things, Doubleday, 1988.

29 Dan R. Olsen, Jr., Developing User Interfaces, Morgan Kaufmann Publishers, 1998.
30 Michael Johnson, Andrew Wilson, Bruce Blumberg, Christopher Kline and Aaron Bobick, Sympathetic Interfaces: Using a Plush Toy to Direct Synthetic Characters, in Proceedings of CHI 99.
31 Jakob Nielsen, Noncommand User Interfaces, in the Communications of the ACM 36, 4, 83-99.
32 Randy Pausch, Dennis Proffitt, George Williams, Quantifying Immersion in Virtual Reality.
33 Eric Paulos, John Canny, Eduardo Kac, Ken Goldberg and Mark Pauline, Interfacing Reality: Exploring Emerging Trends between Humans and Machines.
34 Preston Blair, Cartoon Animation, Walter Foster Publishing, 1994.
35 Toby Gard, Building Character, Gamasutra.com, 2000.
36 Kevin Mullet, Darrell Sano, Designing Visual Interfaces: Communication Oriented Techniques, SunSoft Press, 1995.
37 Virginia Howlett, Visual Interface Design for Windows, Wiley Computer Publishing, 1996.
38 Craig Stitt, John Fiorito, Lessons in Color Theory for Spyro the Dragon, Gamasutra.com, 2000.
39 John Lasseter, Principles of Traditional Animation Applied To 3D Computer Animation, Computer Graphics, Volume 21, Number 4, July 1987.
40 Whitaker, Harold and Halas, John, Timing for Animation, Focal Press, London, 1981.
41 Thomas, Frank and Johnston, Ollie, Disney Animation-The Illusion of Life, Abbeville Press, New York, 1981.
42 Tom Porter, Galyn Susman, Creating Lifelike Characters in Pixar Movies, in the Communications of the ACM, January 2000/Vol. 43, No.1.
43 Michael O'Rourke, Principles of Three Dimensional Computer Animation, W. W. Norton & Company 1995.
44 Isaac Victor Kerlow, The Art of 3-D Computer Animation and Imaging, 2nd Edition, John Wiley & Sons, Inc. 2000.
45 Mary Shaw and David Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.
46 Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal, Pattern-Oriented Software Architecture, Volume1: A System of Patterns, John Wiley & Son Ltd, 1996.
47 David Gries, The Science of Programming, Springer, 1981.
48 Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, Introduction to Algorithms, The MIT Press, 1997.
49 Andre LaMothe, Tricks of the Windows Game Programming Gurus, 1999, Sams.
50 David Eppstein, ICS 161: Design and Analysis of Algorithms lecture notes for January 9, 1996.
51 Rick Decker, Working classes: data structure and algorithms in C++, 1996, PWS Publishing Company.
52 James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley Publishing Company, 1997.
53 Alan Watt and Fabio Policarpo, 3D Games: Real-time Rendering and Software Technology,

Addison-Wesley Publishing Company, 2001.
54 Mark A. DeLoura, Game Programming Gems, Charles River Media, 2000.
55 Nick Bobick, Advanced Collision Detection Techniques, Gamasutra. com, 2000.
56 Nick Bobick, Rotating Objects Using Quaternions, Gamasutra.com, 1998.
57 Michael Abrash, Graphics Programming Black Book, The Coriolis Group, 1997.
58 X. Tu and D. Terzopoulos, Artificial fishes: Physics, locomotion, perception, behavior, ACM Distinguished Dissertation Series, Springer-Verlag, 1999.
59 John Funge, Xiaoyuan Tu and Demetri Terzopoulos, Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters, SIGGRAPH 1999.
60 Mark A. DeLoura, Game Programming Gems, Charles River Media, 2000.
61 Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
62 Dave C. Pottinger, Implementing Coordinated Movement, Gamasutra. com, 1999.
63 Marco Pinter, Toward More Realistic Pathfinding, Gamasutra.com, 2001.
64 Eric Dybsand, Game Developers Conference 2001: An Al Perspective, Gamasutra.com, 2001.
65 Craig W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model, Computer Graphics, 1987.
66 Bryan Stout, Smart Moves: Intelligent Path-finding, Gamasutra.com, 1999.
67 Greg James, Using Generic Algorithms for Game AI, GIGNews.com, 2002.
68 Steven Woodcock, Game AI: The State of the Industry, Gamasutra. com, 2000.
69 Richard H. Thayer, Software Engineering Project Management, IEEE, 1997.
70 Watts S. Humphrey, A Discipline for Software Engineering, Addison-Wesley, 1995.
71 NxN 公司产品目录, 2002.
72 Lisa A. Bucki, Managing with Microsoft Project 2000, Prima Tech, 2000.
73 Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, 1997.
74 Steve McConnell, Rapid Development, Microsoft Press, 1996.
75 Frederick P. Brooks, The Mythical Man-Month, Addison-Wesley, 1995.
76 Grady Booch, Object-Oriented Analysis and Design, Addison-Wesley, 1994.
77 Michael A. Cusumano, Richard W. Selby, Microsoft Secrets, Touchstone, 1995.
78 Bob Bates, Game Design: The Art & Business of Creating Games, Prima Tech, 2001.

内容提要

本书是一本综合性的讲解游戏设计与开发的图书。

如果我们把设计和开发游戏所需的知识结构体系，沿纵向和横向分别表示的话，则有以下两个示意图。

纵向的三个层面

横向的三个单元

本书以这两个框架结构为基础，从设计、美术、技术、管理、文化等诸多角度介绍了游戏设计与开发的一系列方法和原理。

本书中所介绍的技术和思路，一部分是西方游戏界所广泛使用的方法，另一部分是刚刚走出大学实验室的理论和技术。前者经过了时间和实际工作的考验，已经非常成熟；后者虽然还比较单薄，但包含有很多新颖的思路和新奇的想法。了解前者，可以帮助读者打下扎实的基础；了解后者，则可以帮助读者拓宽思路。

本书兼顾了游戏设计与开发中的技术层面和艺术层面。介绍技术层面时，着根于技术发展的总体框架和脉络，力求使得读者们了解某种技术为何而来、如何应用、将来趋势如何；谈论艺术层面时，则着重可控制的可过程化的部分，避免虚化。

本书中你可看到一种全新的视角。你将会认识到迪斯尼的动画法则和戚继光兵书中的格斗要义相合，算法评估的思路和美国国父们的开国理念一致，分类法对游戏性定义的启发，时髦的 VR 技术原只不过是几个世纪人类复现现实努力的延续。你会感悟到文理可以互通，历史知识可以驱动技术发展。这种写法，突破了一般计算机技术书籍的局限性，增加了可读性。

正如前人所说：“汝果欲学诗，功夫在诗外！”这本书将给对游戏具有的无比热忱的你以“诗外功夫”的启发。

本书针对的读者群是游戏业的从业人员和大中学生中的游戏爱好者，也适合作为游戏业界的培训教材及相关高等院校的参考教材。

前　言

如何设计和开发游戏，是一项复杂的系统工程。它所涉及的问题错综复杂，其广度和深度都是笔者个人的绵薄才力和有限视野无法完全涵盖的。因此，本书的写作目的绝对不是说要通过一本书教会读者如何设计和开发游戏。 这是不可能的，也是笔者不能企及的。故此，本书写作的一开始，就是谨慎地定位于仅求比较系统地介绍一些国外游戏业所通行的思想方法和工具。中国古代神怪小说中的神仙每当遇到危难之时，就会从百宝囊中掏出一物，望空抛去，口中念念有词。这本书也有点像百宝囊，内容既多且杂。如果我们的游戏设计师、美工和程序员们遇到实际问题解决不了的时候，能够从本书中找到些许灵感和帮助；或者当我们的有志于游戏的青年们，空有热情而苦于无人指引时，本书能给他们指点一些路数，并将他们的视野拓展一些，若此两点实现，则本书的写作目的也就基本达到了。

如众周知，专业书大抵可分为艰深型和广博型两类。前者针对的是具有一定专业水平的读者，专注于某一专题，务求精辟专深；而后者则针对刚刚入门还未得要领的普通读者，力求给他们一一个关于整个行业或技术领域的全景式的介绍，并不要求把问题发掘得很深。就美国来说，最近一两年出了很多游戏书籍，基本属于广博型，如 Marc Saltzman 的《Game Design：Secrets of the Sages》，而真正艰深型的游戏书在过去比较少见，最近半年才渐渐多起来了，如 David H. Eberly 的《3D Game Engine Design：A Practical Approach to Real-Time Computer Graphics》。而对中国游戏业来说，则是两者都缺。经过反复掂量权衡，考虑到我国游戏业的现状和实际需要，笔者决定这本书走广博型的路子，仅在某些重要问题上有限地深入。而本书的读者群，则定位于对游戏开发感兴趣的青年人（广大中学生和大学生），以及游戏开发公司中的年青人，力求他们都能从中汲取营养。

本书共计六篇二十二章，其中基础篇阐述了基本游戏理论和游戏性等问题；设计篇列出几个游戏设计方面的专题；美工篇集中介绍了二维和三维美工和动画技术；编程篇主要围绕三维图形和人工智能展开；管理篇则探讨软件工程在游戏开发中的应用；最后以教育篇来总结全书。内容基本上涵盖了游戏设计开发的所有主要阶段和层面，力求给读者一个对于游戏设计开发的全景式的描述。其中第五、十二至十五章为叶丁所主笔，其余为叶展主笔，即：叶丁所负责的主要为美工和设计部分，叶展则负责技术和管理部分。

游戏的设计与开发为一广大主题。本书虽经作者灌注不少时间心血，而且两人联手，取长补短，使得本书比较全面，文理兼容，但和任何书一样，疏漏恐怕难免，敬请读者鉴

谅并指正。记得有位外国科学家这么说过：“一本书要被称为好书，并不需要书中的硬性内容 100%正确，或者读者 100%认同其观点。这是不现实的!现在的理论，将来大多要被推倒或者修正！一本书要成为好书，主要是看它是否能够给读者以软性的启发，并从这启发里诞生新的思想和实践。”这也是本书作者的共识。笔者并不能够保证书中内容 100%正确或者 100%有用，但笔者坚信本书能够给您以足够的启发和拓展。

最后要感谢参与本书出版工作的<电子游戏软件>编辑部的同仁。笔者二人远在万里之外的大洋彼岸，若无他们的热情帮助，此书无法如此及时地与国内读者见面。

自序：游戏与人生与梦

扉页上印的是清华时期的老友 tjww 想出来的一段话。他说“人生就是一场游戏，只是不能存盘罢了。”当日他在宿舍里说出这话后，一室绝倒，因为此话确实含义隽永。

人生如游戏，游戏亦如人生。只是在人生中你不能喊停，不能重来，错误不能弥补，伤痛无法挽回。在游戏中则可以存盘，可以反悔，可以重新开始。人生之路上的可能无穷无尽，但路最终只有一条，走过以后绝无反悔之可能，绝无回溯之便利，更不可能再做旁的选择。而游戏之路可以重走许多遍，每次尝试不同的路径，看不同的风景。

十几年前玩《侠客英雄传》，游戏中可结识不同类型的女子并娶之为妻，于是每到选择之前必定存盘，看过结果后再把存档调出来，再去进行下一段旅程，和下一位女子邂逅。现在人们玩《最终幻想》，每次和大魔头对决之前都要找到存储点，以防被打死后前功尽弃。趋利避害，人类本性如此，游戏则利用这点予以发掘。用专业一点的话来说，游戏是在有限的资源下实现了选择代价的无穷小。

在人生中显然没有这样的好事。少年犯了错误，做错了选择，一时痛心疾首，总要痴想：“倘若当时如何如何……”。成年人一般不做如是想，盖因他们已习惯了人生中选择的残酷性。

游戏亦如做梦。叶丁在《电软》上的连载叫作“一场游戏一场梦”，是借一首老歌的题目，实在是贴切。一个孱弱的男孩子，在街机厅里用隆把对手打得满地找牙，他做的是强者的梦；一个小女孩，对自己手机里的虚拟宠物百般呵护，她做的，是慈母的梦；毕业前夕，有同学裹一件军大衣，揣一瓶可乐（2 公升的大瓶），到黑漆漆冰冷冷的清华主楼里的实验室去打通宵，沉湎于李逍遥的风花雪月和悲欢离合中，他做的，是绮色的梦。

我们的少年时代都做过很多梦，很多温馨的梦。在梦里梦见所爱但现实中得不到的东西，和所爱但现实中怯于亲近的人。但每次从梦中醒来时，又要忍受那种得到后又失去的怅惘和失落。但少年的希望总是有，少年总是坚信他的梦将来可以成为现实。于是虽然生活本身并不如意，少年的梦还是做下去，不断地做下去。直到随着年龄的增长，少年明白梦已经不太可能成为现实了。他的梦也就逐渐减少，乃至消亡了。而游戏是少年的梦的延续，它保持我们梦想的能力，短暂延缓它的消亡。

当然梦境与现实会时有冲突。上大学时有一次一连三天躲在家里打智冠的《三国演义》，

玩得天昏地暗，除吃方便面和方便的时间外，眼不离屏幕手不离鼠标。当时的感觉，也只有一个“爽”字可以形容。更专业一点的说法就是达到了绝对的体验感和置入感，这也是游戏设计师们所希望玩家们达到的境界。但等三天疯狂过后，跨上自行车赶到清华，发现课旷数节，作业欠数篇，搞得心情沮丧，灰头土脸。这就是梦与现实的差距和割裂感。

因此，游戏与人生与梦，有矛盾，有冲突，也互补，也调剂，三者缺一不可。人生的苦涩和艰辛，需要梦想来抚慰和鼓励，而梦想的翅膀，需要游戏为之打开。

本书是一本技术类书籍，似乎与人生与梦搭不上太大关系。但这只是表象，实际上任何貌似冰冷的技术的底下都是涌动着人生的激越和梦想的碰撞。

因此本书的副题起名为“梦开始的地方”。因为一部游戏，无论对于编者还是玩者来说，都是梦想的开始。

作者的话