计算机动画与游戏

计算机图形学——原理、方法及应用（第4版），潘云鹤、童若锋、耿卫东、唐敏、童欣，高等教育出版社，2022。

计算机动画

计算机动画简介（1/3）

- 动画的定义：
  - 就是一组连续的图像序列。当按一定的速率显示的时候，能传递一种运动的感觉
- 动画的技术要求：
  - 每一幅图像或者每一动画帧，都必须有机地、无缝地和其他的图像融合在一起，这样才能随着时间的变化，产生平滑的、连续的运动

计算机动画简介（2/3）

- 传统的动画制作方法
  - 将人物运动的所有帧序列手工地绘制出来
- 计算机在动画制作中的作用
  - 自动产生某些中间帧序列
  - 模仿手工动画，将多层绘制的场景由计算机来进行合成
- 制作方式的改变
  - 从“绘制出每一帧”到“使用计算机工具来制定图像序列如何变化”

计算机动画简介（3/3）

- 动画的分类：
  - 二维动画
    - 二维动画集中于图像的处理和操作，如
      - sprite-based animation, blending or morphing between images, embedding graphical objects in video footages, or creating abstract patterns from mathematical equations
  - 三维动画
    - 三维场景和人物的建模以及他们之间的交互
      - 建模
      - 运动的指定 (√)
      - 绘制

三维动画概述（1/4）

- 三种主要的运动方式
  - 场景内容不变，视点在移动
    - 漫游等
  - 场景内容在动态改变，视点不变
    - 人物表演动画等
  - 混合方式
    - 场景内容和视点都在运动

三维动画概述（2/4）

- 最简单的情形
  - 使用标准的绘制程序，仅移动物体或视点.
- 技术核心
  - 各个运动物体的平移和旋转量确定。
- 关键问题
  - 如何指定和控制物体在场景中的运动.
  - 相机或者物体移动，或者两者同时移动.

三维动画概述（3/4）

- 相机的任意外部参数都可以形成动画。这类动画一般在第一人称游戏中最常用。
- 用户可以控制相机位置（三个自由度）和相机的朝向（两个自由度）.
  - 在两个相邻的关键点上需要插值相机参数，在此过程中最重要的是保持相机的up向量。
- 此外，用户也可以控制视点的运动，而视线则始终跟踪一个目标点。
  - 该目标点可以是静止的，也可以移动。

游戏中的三维动画概述（4/4）

- 所涉及的关键技术
  - 漫游
    - 碰撞检测（collision detection）√
  - 人物动画
    - 骨架驱动的动画√
    - 运动的生成 √
    - 面部表情
  - 粒子系统 √

人物动画概述（1/3）

- 从绘制流水线的角度看
  - 在某个瞬间（某一帧），只要知道此时的物体和场景的空间位置的信息，就可以绘制出画面
- 关键：运动指定
  - 一帧一帧指定：每秒25到30帧，。。。
- 人物动画的难点：
  - 动画人物的身体的各部分的和谐的、自然的运动。

人物动画概述（2/3）

- 人的视觉系统（人类动画师）并不明显区分建模信息、运动信息和视觉效果信息
- 基于骨架的人物动画
  - 抽象出骨架：建模和运动分开，在绘制时进行骨架和模型的绑定。
    - 方便运动的指定
    - 模型和运动都可以重用

人物动画概述（3/3）

- 什么是关节?
  - 它由一系列刚体、它们之间的连接关系组成，这使得物体的不同部分可以相关联地运动。
- 整个物体的运动受全局的连接关系限制
  - 整个结构由一条联结关系组成，某个关节的运动造成它的邻接关节运动
  - 并且限制了邻接关节的运动方式，如人类的骨架关节无法做完全旋转。

动作（motion）获取的三种主要方式（1/3）

- 关键帧：指定每一关键时刻的物体的运动位置和方向
  - 优点：简单，易理解
  - 缺点：
    - 它要求动画师经验丰富，并且耗时
    - 难以模仿动作的某些细微特征或者特定的动作

动作获取的三种主要方式（2/3）

- 动态（过程）模拟 (物理建模)
  - 根据物理规律和指定的目标，使用算法或者过程来模拟产生运动或者动作序列
- 优点：
  - 很容易根据运动的要求配置相应的参数，初学者也能产生高质量的动画序列
- 缺点：
  - 对简单的运动比较有效
  - 没有系统性的方法来描述一个复杂的运动，或者一个具有细微特性的运动

动作获取的三种主要方式（3/3）

- 基于动作捕捉设备
  - 将表演者的现场表演的运动数据实时纪录下来，并映射到计算机的动画人物中
- 优点：
  - 实时，直观，可视
    - 运动数据可以被实时地、直观地可视化。
    - 演员、导演和动画师可以在一起工作，来获取所期望的动作，便于交流，
  - 高质量动作数据
    - 可以产生特定人物的特定运动数据，这是手工指定所无法比拟的
    - 产生的运动数据比较真实、自然，并可表现出表演者的人物个性。很多细微的运动元素已经自然地体现在捕捉下来的数据中，不需要通过领域知识来添加。

我们的mocap设备（1/2）

- 浙江大学计算机学院动作捕捉实验室
  - 曹光彪东楼313-314
- 生产厂家：
  - Motion analysis 公司
    - 如果只用于游戏和动画，Vicon更佳

我们的mocap设备（2/2）

- 性能
  - 10Hawk摄像头
  - 每秒200帧
  - 精度小于0.5mm

关键帧技术

- 关键帧系统的应用背景
  - 在电影和动画界，这是一个最常用的技术
  - 为了节省动画片制作中的巨大工作量，动画公司采用层次的系统结构，即高级动画师首先手绘一段关键帧构成的动画序列。
  - 然后这些序列由动画师(inbetweener)完成帧间动画的绘制。最后，由动画师(inker)将这些序列着色。

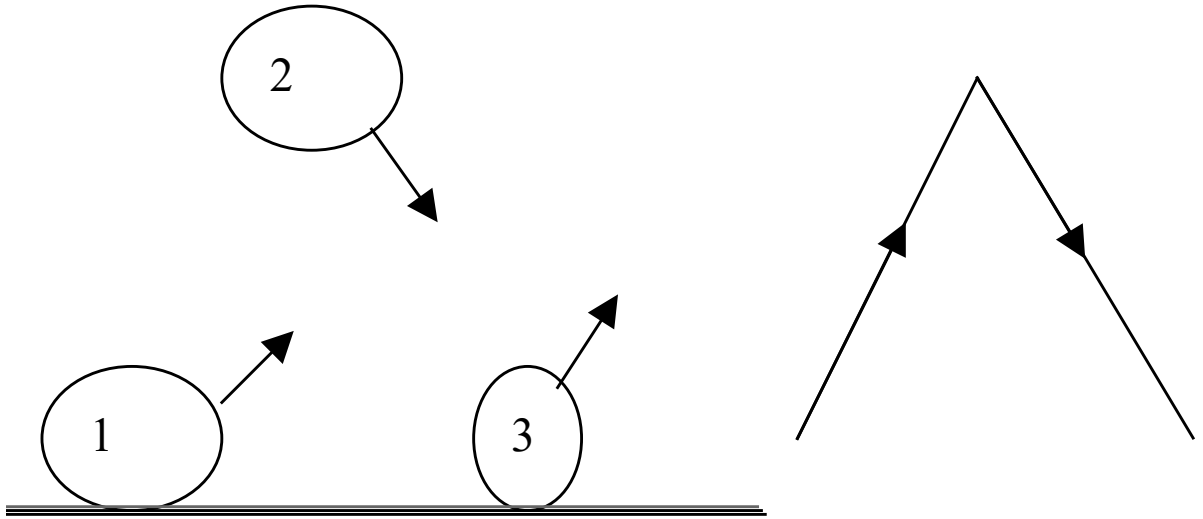
关键帧技术(续)

- 扩充到三维动画
  - 三维空间中的物体也可以由关键帧定义
  - 并由计算机完成中间帧的生成。

- 相比手绘动画，在计算机动画中需要更多的关键帧以获得完美的效果。

关键帧技术(续)

- 以一个弹性球为例. 下图展示了三个关键帧。



指定关键帧

- 指定骨架关键帧有多种方法，比如：
  - 最简单的情形：通过空间中的一系列点确定—关键帧点

    定义一个物体所经过的点序列，并将这些点列用三次曲线拟合生成路径。

指定关键帧信息

- 比较好的方法
  - 允许动画师指定更多的关键帧之间的运动信息
    - 例如直接定义一条运动曲线.
  - 此外，可以定义运动路径上的速度变换函数.
- 一般地，为了正确控制运动，必须显式地定义
  - 位置变换与时间的函数关系
  - 指定路径上的动态行为.

指定关键帧信息

- 参数
  - 位置
  - 关节角
  - 形状－变形物体
  - 材质
  - 相机运动
  - 光照
- 如果我们仅考虑位置，那么我们可以应用一个与时间相关的4x4位置矩阵于场景物体。

关键帧插值

$$M(t)=\begin{bmatrix} a_{11}(t) & a_{12}(t) & a_{13}(t) & t_x(t) \\ a_{21}(t) & a_{22}(t) & a_{23}(t) & t_y(t) \\ a_{31}(t) & a_{32}(t) & a_{33}(t) & t_z(t) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 为了使物体运动真实，矩阵的二阶导数必须连续

- 矩阵中的位置部分的三个分量可以独立插值，但旋转部分不行。

关键帧控制器

```
Void KeyFrameController::Update(float time)
{
    InterpolateTranslation(time, localTranslate);
    InterpolateRotation(time, localRotate)
}
```

- 每个插值进程以某一时刻为准，找到与之最近的两个关键帧时刻点。这两个关键帧用下面描述的方法进行插值。

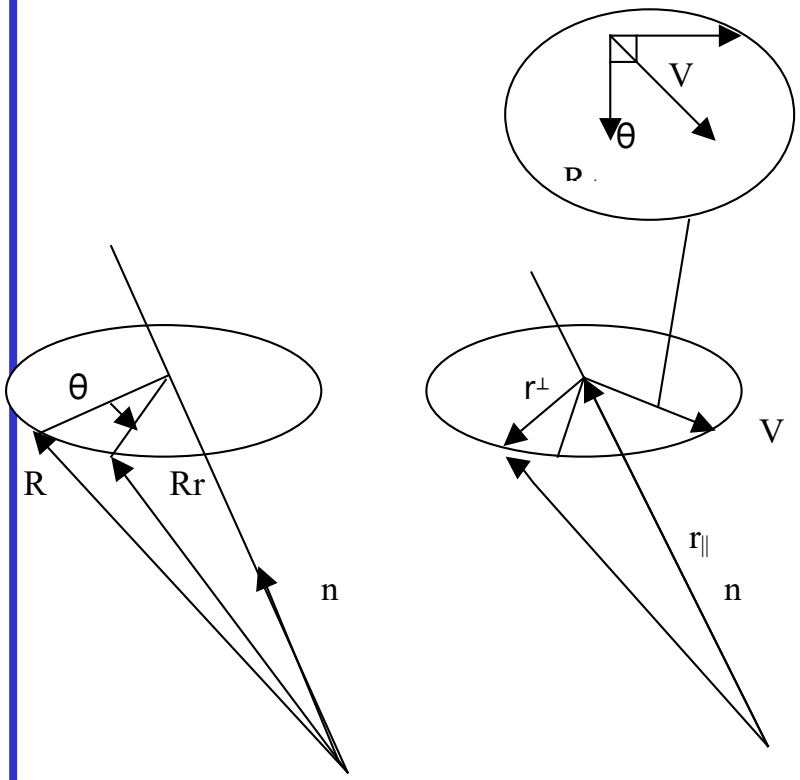
旋转部分插值（1）

- 基本思想:
  - 用四元数表示旋转
  - 将旋转矩阵变换到四元数空间，然后在四元数空间进行插值
  - 插值后的四元数变回到三维空间并应用到物体上。
- 四元数
  - 可视作一个类似矩阵概念的算子，它能将一个向量变到另外一个，但是四元数的选择是唯一的.

旋转部分插值（2）

- 4-元数是什么？
  - 将旋转定义为绕轴n旋转角度θ的一对关系：$R(\theta, n)$.
    - 绕三个轴向变成绕一个轴向
- 为什么使用4-元数来插值？
  - 基于欧拉角的旋转具有轴相关性

旋转部分插值（3）



$$r_{\parallel} = (n \bullet r)n$$

$$r_{\perp} = r - (n \bullet r)n$$

$$V = n \times r_{\perp} = n \times r$$

$$Rr_{\perp} = (\cos\theta)r_{\perp} + (\sin\theta)V$$

$$Rr = Rr_{\parallel} + Rr_{\perp}$$

$$= (n \bullet r)n + (\cos\theta)(r - (n \bullet r)n) + (\sin\theta)(n \times r)$$

$$= (\cos\theta)r + (1 - \cos\theta)(n \bullet r)n + (\sin\theta)(n \times r)$$

旋转部分插值（4）

- 考察四元数的旋转功能。 r 可表示为四元数 p=(0,r)，因此该操作可视为：

$$q = (\cos\theta, \sin\theta \quad n) \qquad where \qquad |n| = 1$$

$$R_p(p) = qpq^{-1} = (0, r\cos 2\theta + (1-\cos 2\theta)n(n \bullet r) + \sin 2\theta(n \times r))$$

- 因此，将向量r旋转一个 角度(θ ,n) 相当于利用角度位移的概念，即将它提升到四元数空间，然后表示为单位四元数

(cos (θ /2), sin(θ /2)n)

旋转部分插值（5）

- 对四元数 (0, r)执行操作$q()q^{-1}$，可将方向参数化为以下四项：

$$\cos(\theta/2),\ \sin(\theta/2)\,n_x,\ \sin(\theta/2)\,n_y,\ \sin(\theta/2)\,n_z$$

- 利用四元数代数操作每个分量

旋转部分插值（6）

- 假设动画师设置了一系列旋转的关键帧序列，那么
  - 每一帧可由单个旋转矩阵决定。这些矩阵序列将被转换到一系列四元数。
  - 在关键帧四元数之间进行插值，产生一系列连续的四元数，再将它们转换到旋转矩阵。
- 这些矩阵在应用到物体上。
  - 一般的，四元数的使用对动画师来说是透明的。

旋转部分插值（7）:四元数到旋转矩阵

- 将一个向量P 旋转一个四元数q :

$$q(0, P)q^{-1}$$

q 又可表示为:

$$(\cos(\theta/2), \sin(\theta/2)n)=(s, x,y,z)$$

- 它等价于下面的旋转矩阵：

$$M = \begin{bmatrix} 1-2(y^2+z^2) & 2xy-2sz & 2sy+2xz & 0 \\ 2xy+2sz & 1-2(x^2+z^2) & -2sx+2yz & 0 \\ -2sy+2xz & 2sx+2yz & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

旋转部分插值（8）:旋转矩阵到四元数

- 若旋转矩阵

$$M = \begin{bmatrix} M_{00} & M_{01} & M_{02} & 0 \\ M_{10} & M_{11} & M_{12} & 0 \\ M_{20} & M_{21} & M_{22} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 则

$$q = (s, x, y, z)$$

where

$$s = \pm \frac{1}{2} \sqrt{M_{00} + M_{11} + M_{22} + 1}$$

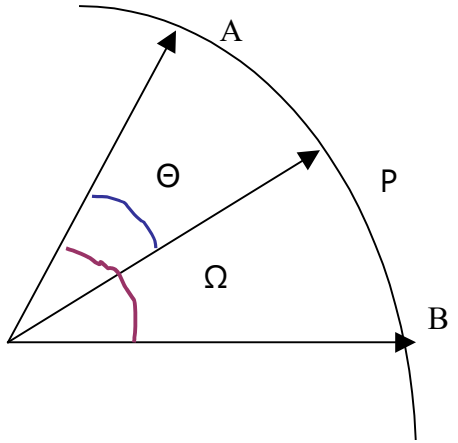
$$x = \frac{M_{21} - M_{12}}{4s}$$

$$y = \frac{M_{02} - M_{20}}{4s}$$

$$z = \frac{M_{10} - M_{01}}{4s}$$

四元数的球面线性插值（1）

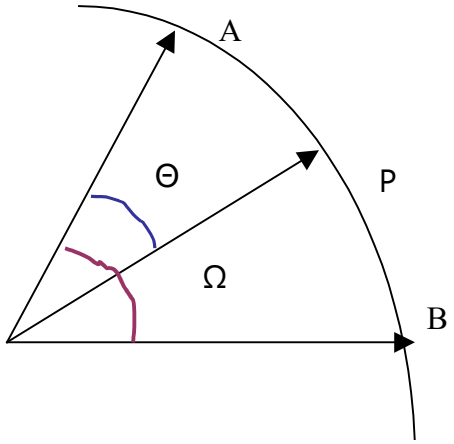
- 球面线性插值的公式是:
  - 考虑两个二维向量 A 和 B，它们之间的夹角是Ω，其中 向量 P 与A成θ角度
  - P是A和B的球面线性插值的结果:
    - $P=\alpha A+\beta B$



球面线性插值(2)

- 一般地， α, β由下式给出:

  |P|=1, A·B=cosΩ, A·P=cosθ



$$P = A\frac{\sin(\Omega-\theta)}{\sin\Omega} + B\frac{\sin\theta}{\sin\Omega}$$

球面线性插值(3)

- 两个单位四元数 q1 和q2之间的夹角为Ω:

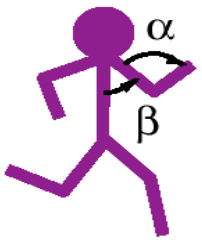
  $q1 \cdot q2 = \cos\Omega$

- 它们之间的球面线性插值可将前面的公式推广到四维： ($u \in [0,1]$)

$$\mathrm{slerp}(q1, q2, u) = q_1 \frac{\sin(1-u)\Omega}{\sin\Omega} + q_2 \frac{\sin \Omega u}{\sin \Omega}$$

人物动画中的运动生成（1）

- 三个主要的运动动方式
  - 前向动力学
  - 逆向动力学
  - 运动捕获



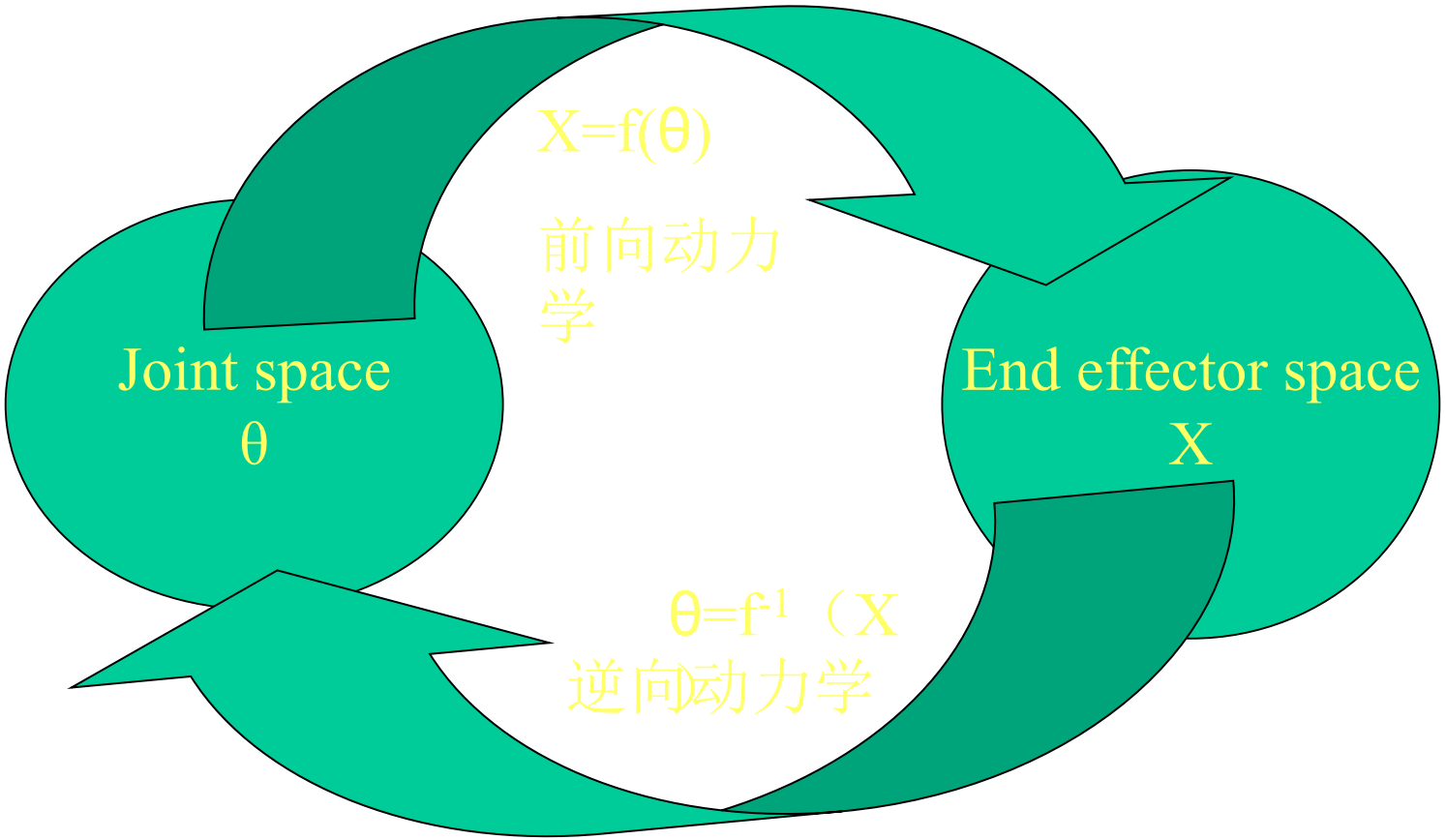
Forward: $A = f(\alpha,\beta)$ **Inverse:** $\alpha,\beta = f^{-1}(A)$

人物动画中的运动生成（2）

- Joint space：各个关节点和关节的DOF
- End-effector space: end effector 的m-维空间
- World space: 人物场景空间
  - 骨架/关节点的位置是各个关节夹角（用户指定，前向动力学）的一个函数
  - 用户也可以直接指定骨架的姿势和关节点的位置（逆向动力学）

人物动画中的运动生成（3）

- 动画编程人员眼中的前向动力学和逆向动力学问题描述

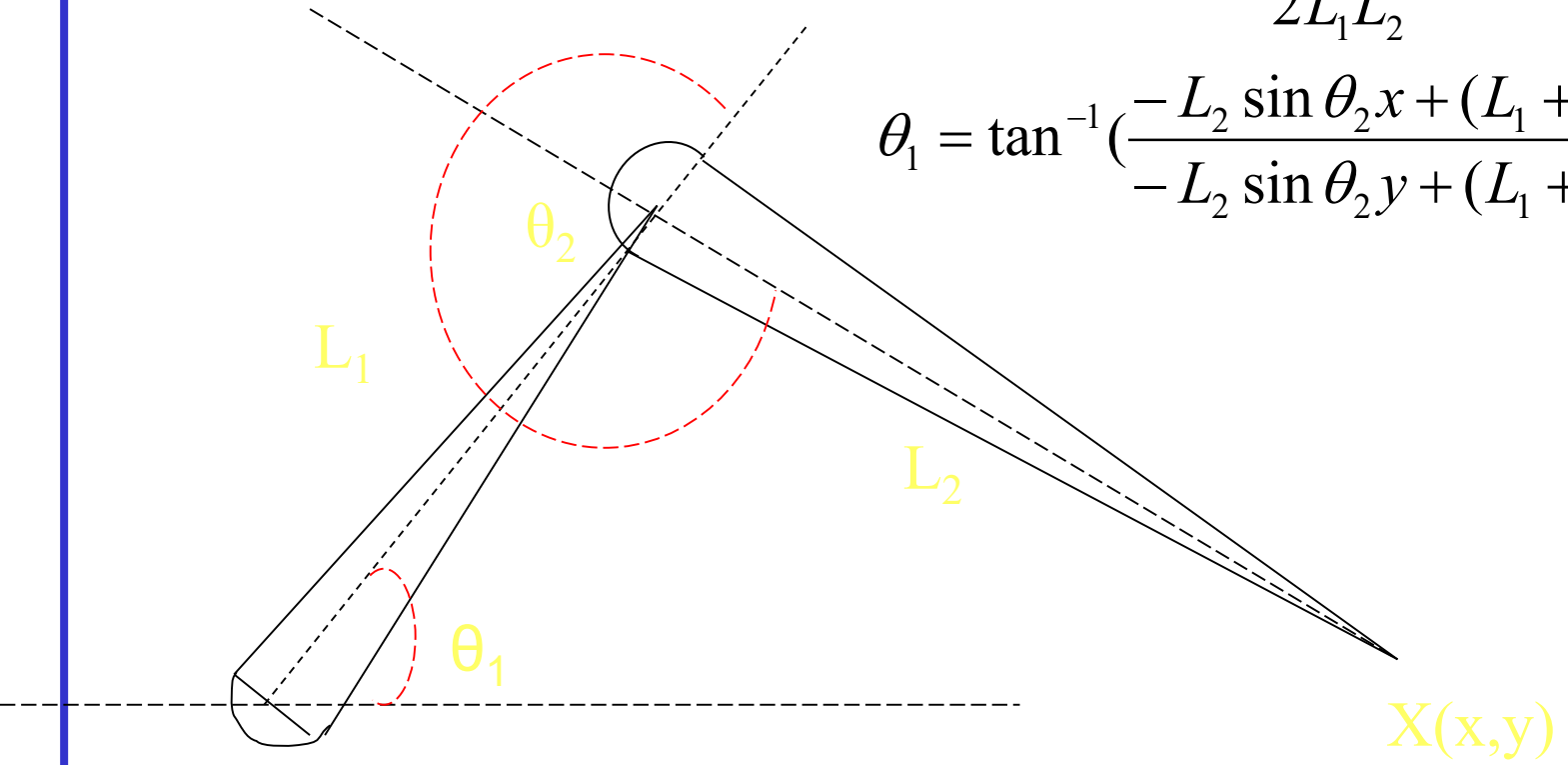


人物动画中的运动生成（4）

- 两个骨架Link的例子：

$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1L_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{-L_2 \sin\theta_2 x + (L_1 + L_2\cos\theta_2)y}{-L_2 \sin\theta_2 y + (L_1 + L_2\cos\theta_2)x}\right)$$



人物动画中的运动生成（5）

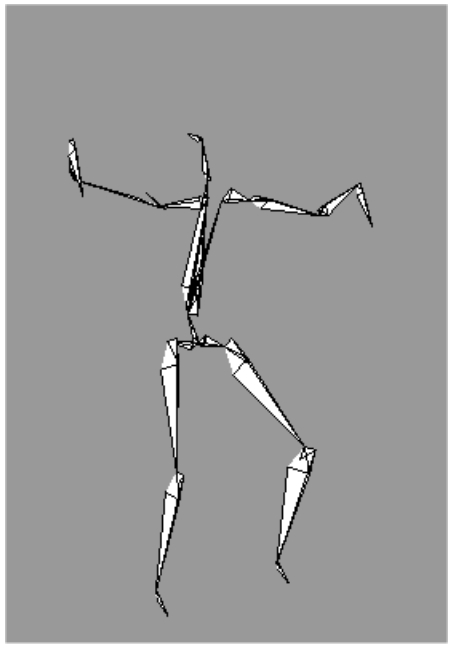
- IK的解决方案：
  - 解析解
  - 差分算法
  - 优化
  - 非线性规划
  - 。。。。。。。
- IK的演示
  - IK demo

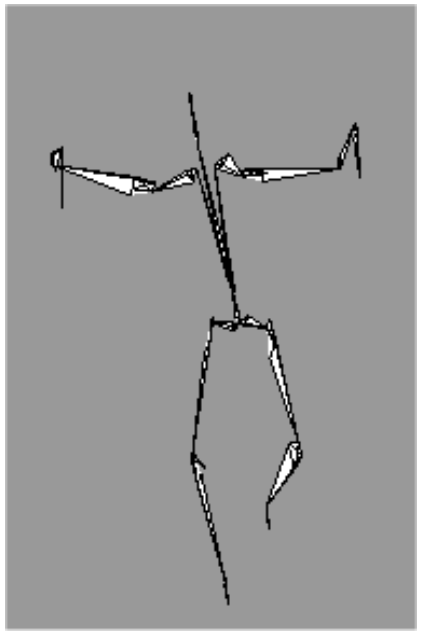
人物动画中的运动生成（6）

- 动作捕捉数据驱动
  - 前向或者逆向动力学？
- 动作捕捉数据的格式
  - TRC
  - HTR
  - BVH
  - BVA
  - C3D
  - ….

人物动画中的运动生成（7）

- 文件格式实例
  - 骨架定义
  - 各帧的数据
- BVH 文件解剖
- BVH demo





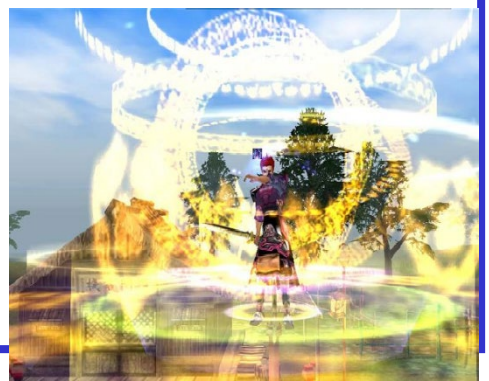
计算机游戏

什么是游戏？（1/6）

- 游戏的最初方式是非对抗性的、友好的体力与技巧比赛。
  - 英文中的单词是Game，译意是“比赛、竞赛、游戏”
- 游戏从单纯的体力活动逐渐向体力与脑力结合的方向发展，甚至出现了一些纯粹的脑力活动。
  - 比如棋类和牌类游戏的发明。
- 进入科技时代以后，越来越多的高科技手段被运用到了娱乐行业中，最终能够以计算机的运算代替原来必须由人来承担的角色，此时的游戏更多的是一种为了娱乐的活动
  - 在电视机普及到家庭以后，游戏通过电视游戏机进入了家庭。
    - Console game
  - 随着家用计算机的发展和普及，游戏又进入了计算机。
    - PC game
  - 随着网络的发展，游戏进入了网络世界
    - Online game

什么是游戏？（2/6）

- 游戏的内容
  - 游戏和戏剧、电影一样，是一种综合性艺术，一种融合了技术的、更高层次的综合艺术。
    - 游戏被称为继绘画、雕刻、建筑、音乐、诗歌（文学）、舞蹈、戏剧、电影（影视艺术）之后的人类历史上的第9种艺术。
    - 高雅
  - 计算机游戏是一个让玩家追求某种目标,并且让玩家可以获得某种“胜利”体验的娱乐性文化产品。
    - 通俗
  - 狭义理解，游戏构建了一个虚拟的世界，既要有内涵，又要好玩。
    - 虚拟的世界具有“价值“
    - 游戏世界的“秩序”

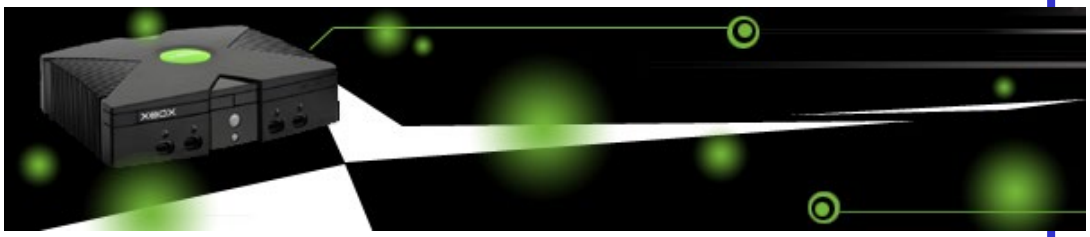


什么是游戏？（3/6）

- 游戏的技术
  - “以计算机为操作平台，通过人机互动形式实现的、能够体现当前计算机技术较高水平的一种新形式的娱乐方式。”
    - Dos, Windows, X-box, PS2,Brew等
    -
  - 计算机游戏也是一种软件。
    - 如易用性、稳定性等
  - 游戏必须具有高度的互动性
    - 技术实现

什么是游戏？（4/6）

- 游戏的运行平台
  - PC
    - Window Series + Direct-X
  - 独立游戏平台
    - PS2
    - Xbox
    - ……
  - 网络游戏平台
    - 点对点
    - 局域网
    - 服务器—客户端
    - 。。。。。。

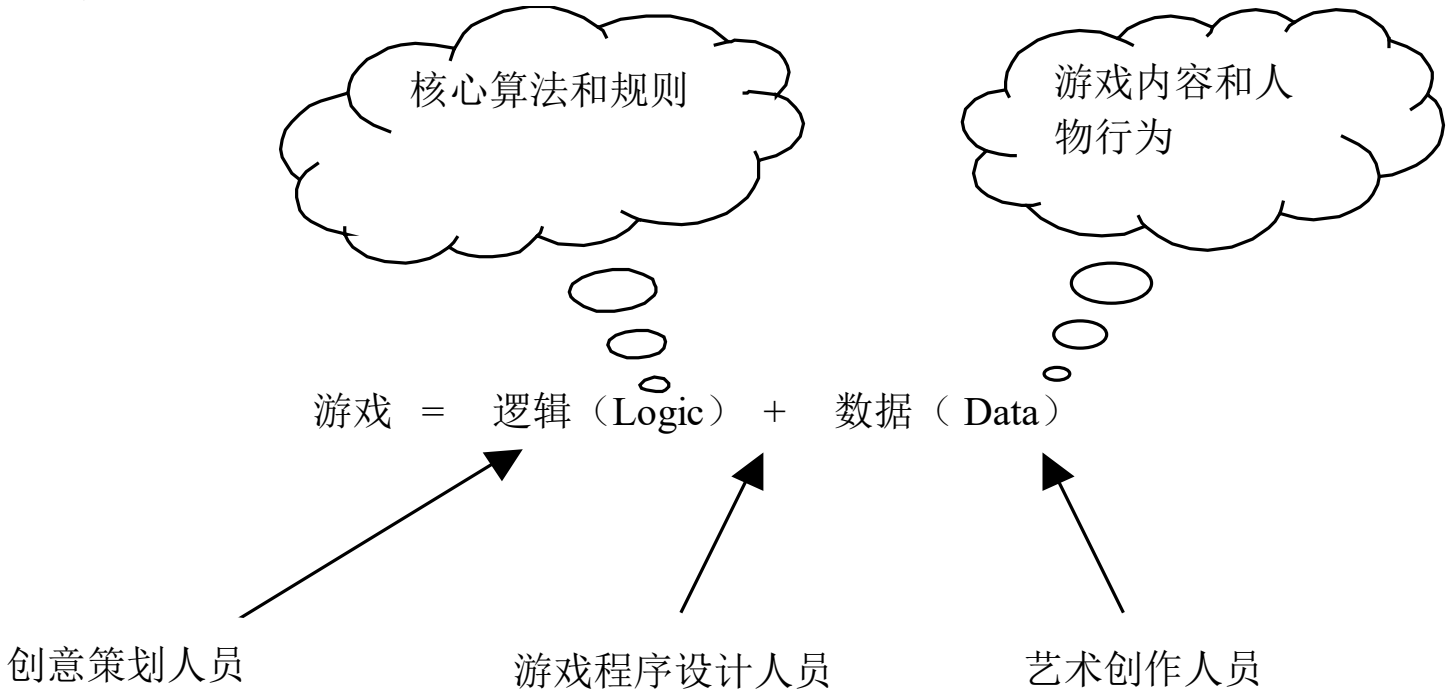


什么是游戏？（5/6）

- 游戏的玩家
  - 计算机游戏提供了挑战的机会和场所
  - 计算机游戏具有虚拟的社会性
  - 计算机游戏提供玩家独处的经历
  - 计算机游戏能提供满足感
  - 计算机游戏能提供情感的体验
  - 计算机游戏能提供幻想
  - 。。。。。。

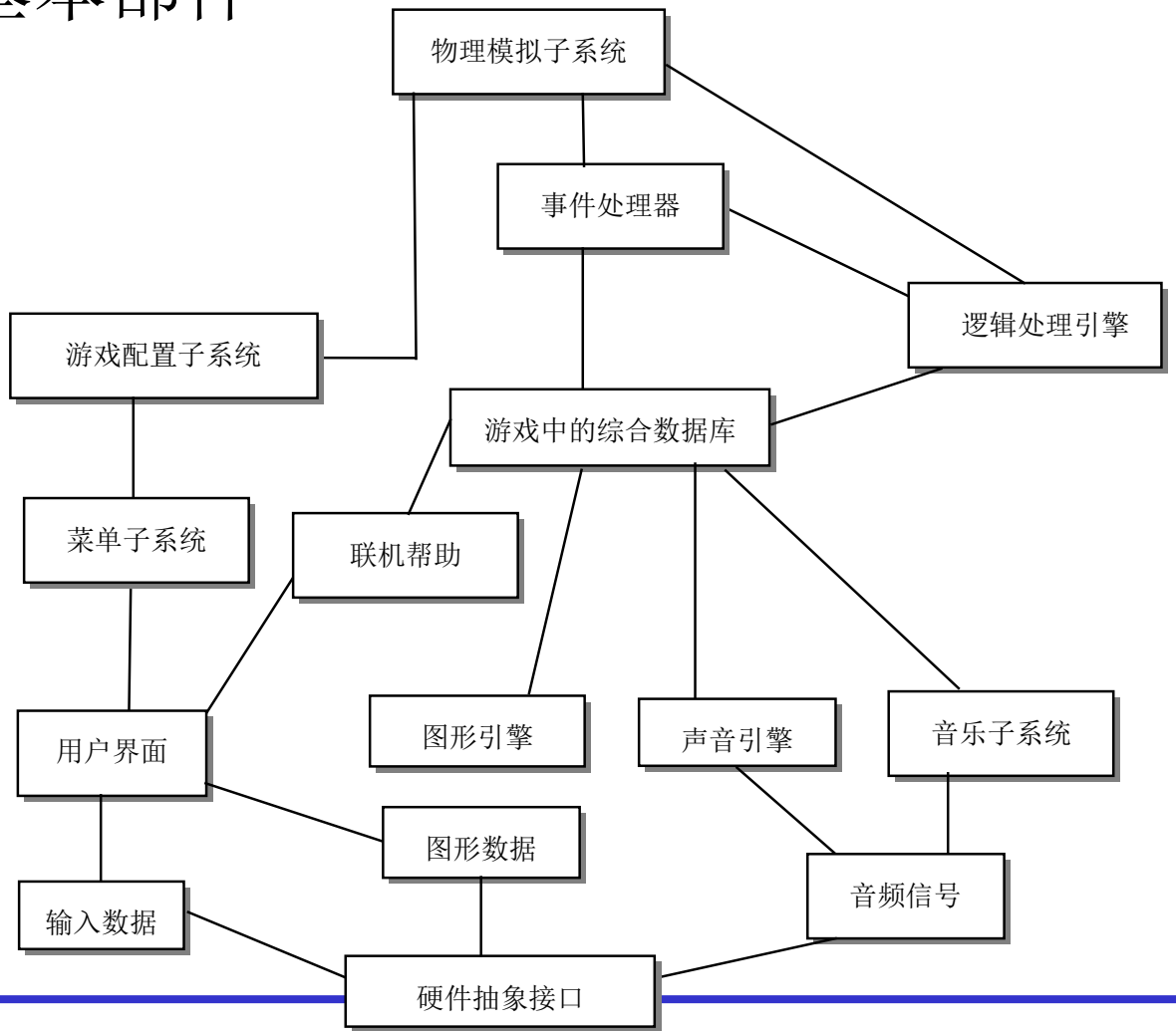
什么是游戏？（6/6）

- 游戏程序员看游戏
  - 游戏”只是一个具有某种“逻辑”和某些“数据”的结合体



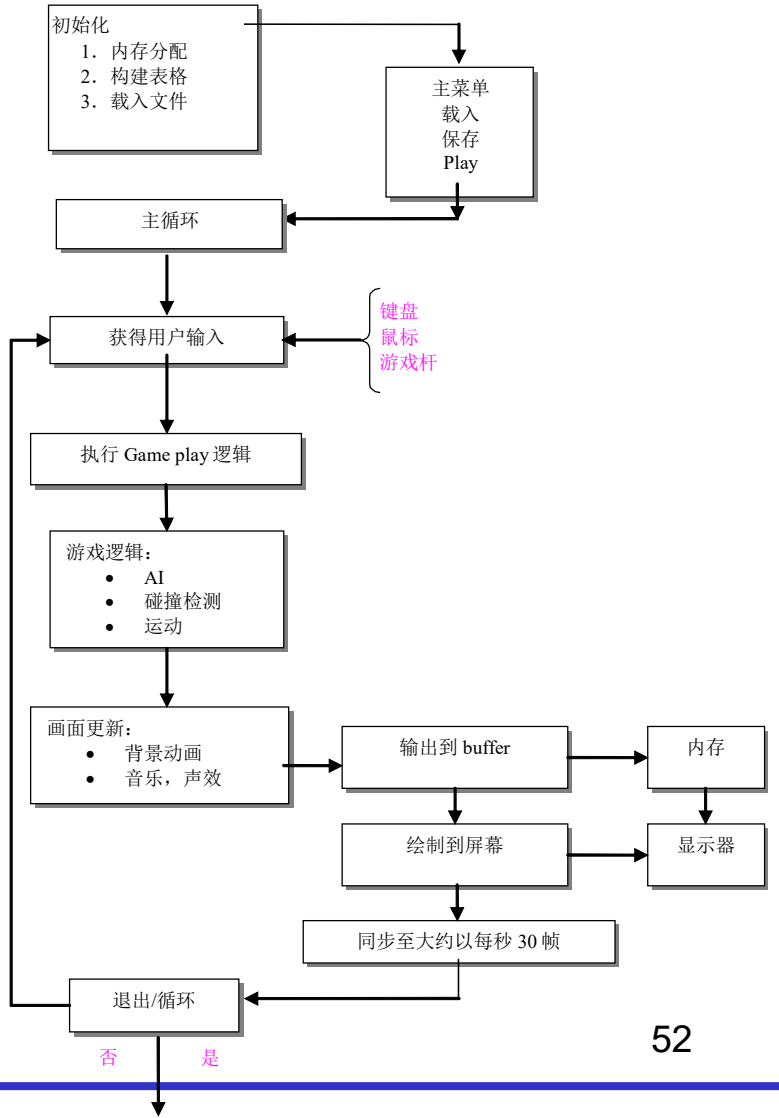
游戏的组成

- 游戏的基本部件



游戏的运行流程

- 游戏其实就是一个不断按某种逻辑更新各种数据（画面、声音等）的过程。
  - 游戏的基本流程只是一个连续的循环，它不断地按某种逻辑来绘制新的图像，并刷新画面
  - Dave Roderick曾形象地把游戏类比为一个带有前置终端的实时数据库，该终端实时地接受用户（玩家）输入的各种交互指令，取出相应的数据，并“优雅”地将这些数据以各种形式（视觉、听觉等）展现给用户。



游戏引擎技术（1）

- 游戏引擎技术的出现是游戏程序设计技术发展的里程碑之一，并已成为当前计算机游戏开发的关键技术和核心平台。
  - 它也是软件工程、专业化分工和游戏产品的独特文化性要求在游戏开发的综合体现，对游戏产业的发展起了巨大的推动作用

- 游戏引擎的意义
  - 游戏编程人员就不需要从头做起，而是可以直接调用游戏引擎提供的强大功能，高质量地在很短的周期内开发出新游戏，适应游戏产业的激烈市场竞争
  - 游戏引擎促使游戏编程人员进行更为专业化的分工
    - 高水平的编程人员注重于性能要求很高的游戏引擎的开发，一般水平的游戏编程人员则利用游戏引擎进行具体的游戏产品的开发

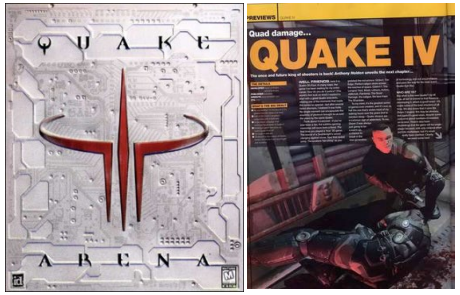
游戏引擎技术（2）

- 游戏引擎相当于游戏的底层框架平台。
  - 框架平台搭好后，只要往里填充内容就可以了。
  - 如果把游戏引擎比拟为一个“游戏操作系统”，那么最终的游戏产品则可比拟为一个个具体地运行在“游戏操作系统”上的应用程序。
- 游戏引擎已经发展为一套由多个子系统共同构成的复杂系统
  - 从建模、动画到光影和粒子特效，从物理系统、碰撞检测到文件管理、网络流量控制等，包括专业的编辑工具和插件，几乎涵盖了游戏程序设计过程中的所有重要环节

游戏引擎技术（3）

- 游戏引擎的终极目标
  - 游戏编程的透明化，让游戏的创意人员也能直接实现游戏
    - 根据创意，直接生成代码

- 著名的商用游戏引擎
  - Doom
  - Quake
  - Unreal
  - LithTech
  - ……



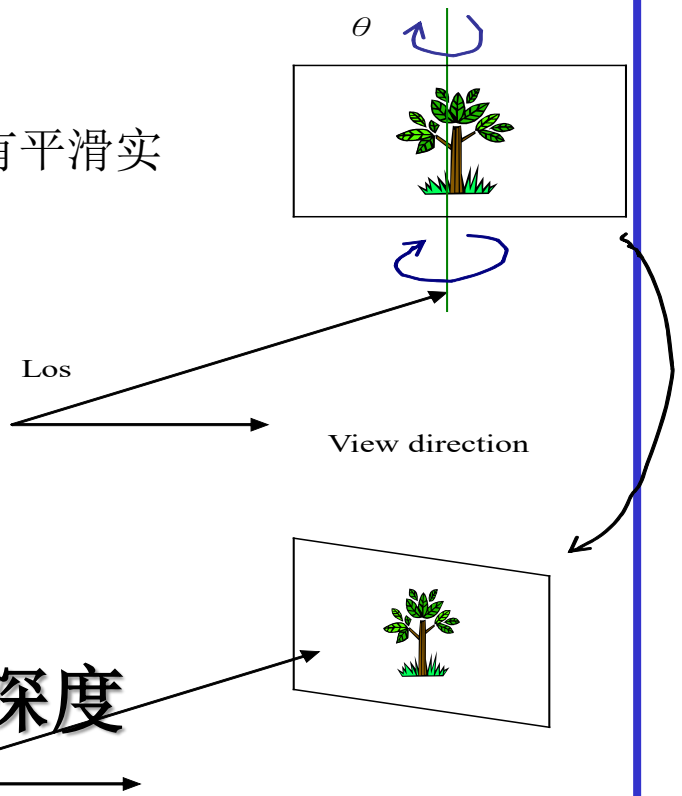
游戏引擎技术（4）

- Ogre引擎
  - 开源 http://sourceforge.n
  - 面向对象
  - 抽象底层图形库（D3D，OpenGL)
  - 灵活的渲染引擎 （不是一个完整的游戏引擎）
    - 大量采用C＋＋设计模式
    - 以插件的形式方便扩展OGRE的功能
    - 提供了抽象的渲染API，封装了底层的图形库。

基于公告板的快速绘制

- Billboard mapping （公告板技术）
  - 与alpha透明处理和动画技术结合，可创建不具有平滑实体表面的景象，比如烟雾，云朵，爆炸效果等

- 本质上是基于图像的绘制
  - Image-based rendering
    - 与图像中的像素数量有关，与场景的复杂度无关



- **对精灵纹理加上一个深度值，深度精灵。**

https://blog.csdn.net/ZJU_fish1996/article/details/84035220

Screen-aligned billboard
随着相机旋转总是面向相机，
且大小不随着镜头远近变化。

World-aligned billboard：
无论怎么旋转视角，它都面向摄像机；
但大小会随着远近而变化

Axis-aligned billboard：
适用于距离比较远的树木之类的物体。
在某一轴旋转固定，其它轴自由，
通常限制上下旋转的角度

- **朝向视平面的公告板**
- **朝向视点的公告板**
- **轴对称公告板**

经典类型

公告板对齐

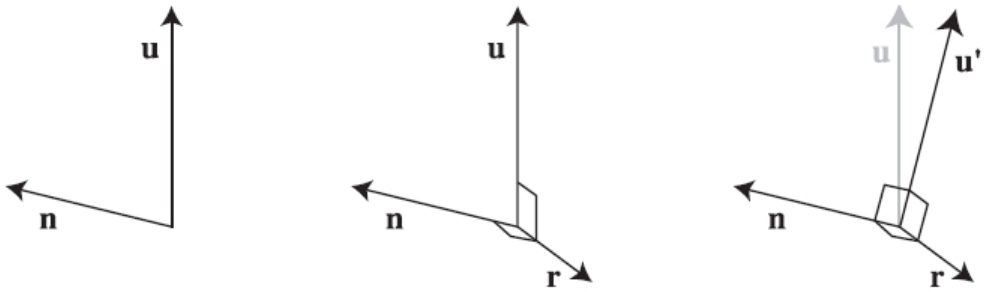
“对齐”：转化为坐标系变换

给定公告板表面法线向量n和近似向上向量u，创建一组两两相互垂直的向量便可以确定公告牌的朝向。

$r = n \times u$

$u' = n \times r$

$n' = r \times u$



公告板坐标系的定义

经典公告板类型（一）

- Screen-aligned billboard
  - 信息提示类

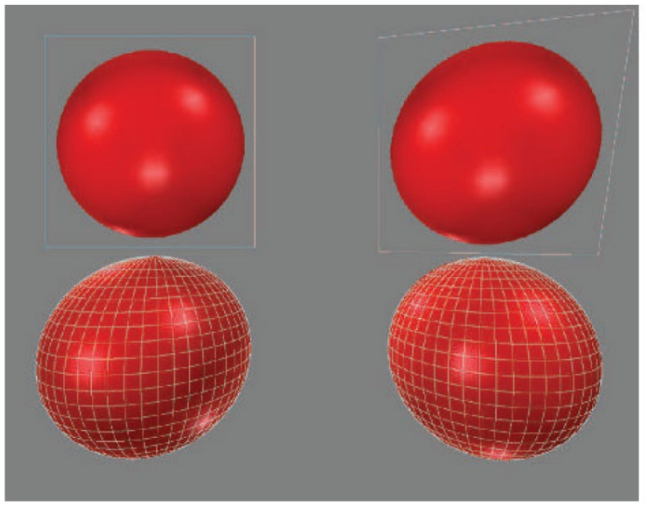
图像永远平行于屏幕，有一个固定的向上的值，它的n是镜头视平面法线的逆方向，u是镜头的up（有时也采用观察坐标系的up向量）。

n和u对于同一个镜头来说是一个定值，因此，场景中所有这种类型的billboard都可以用这一个变换矩阵

《艾迪芬奇的记忆》中的动态字幕效

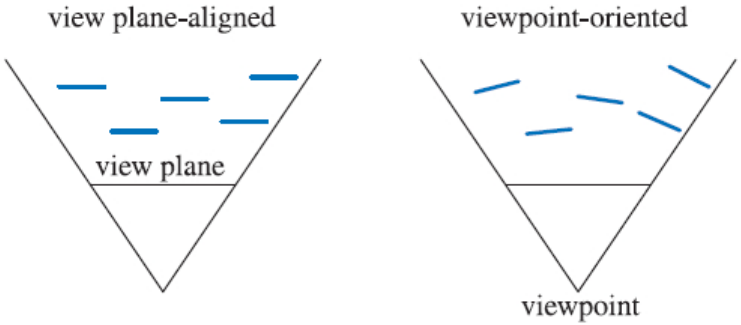
经典公告板类型（二）

- World-oriented billboard



可带有物理属性

n是视平面法线的逆方向，但是新的u是通过世界坐标的up来计算获得,或者和Screen-Aligned一样，是反计算得到



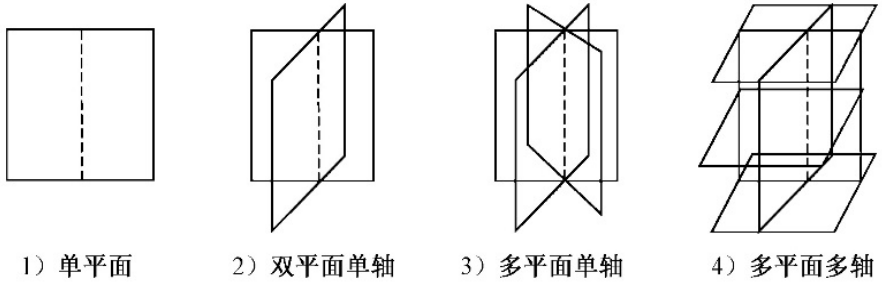
当物体很小或者FOV很小时，这种扭曲不会有太大影响。

可以采用Viewpoint-oriented Billboard的方法来代替

经典公告板类型（三）

- Axixal billboard

物体并不只是简单的面朝着视点，它能在一定的范围内沿着世界坐标轴旋转使自己能面朝着观察者。通常这种技术用于场景

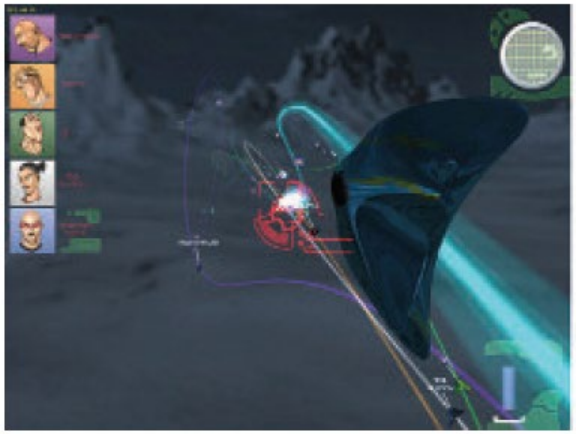


在Axial公告板中，世界的up向量决定了坐标轴，公告牌只能在这上面移动

当观察者从上方观察物体时，由于无法在观察轴旋转，看起来就是面片一样。

游戏中的使用技巧(一)

- Screen-aligned适合用于对称的球面物体，Axial适合表现柱状的对称物体。
  - 镭射线效果从各个角度都很一致，它就很适合用Axial billboard来表现



游戏中的使用技巧(二)

- **对精灵纹理加上一个深度值，深度精灵。**
  - 可以改变大小和形状。
  - 一组精灵也可以用来表示来自不同视图的对象。

图像的Alpha通道可以为sprite的各种像素提供全部或部分透明度

Ubisoft 雷曼大冒险

优点

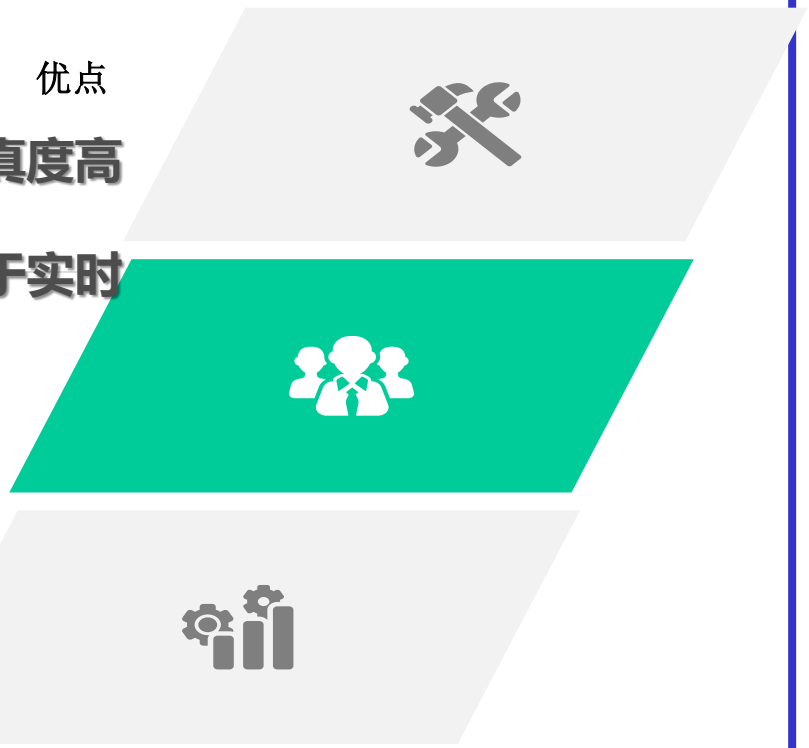
基于实景图像，所构造的虚拟环境逼真度高；

场景的绘制时间与其复杂度无关，利于实时

不足

缺乏统一的空间坐标体系；

难以实现任意方式的空间漫游与交互



改进

Billboard Cloud (云团，族)

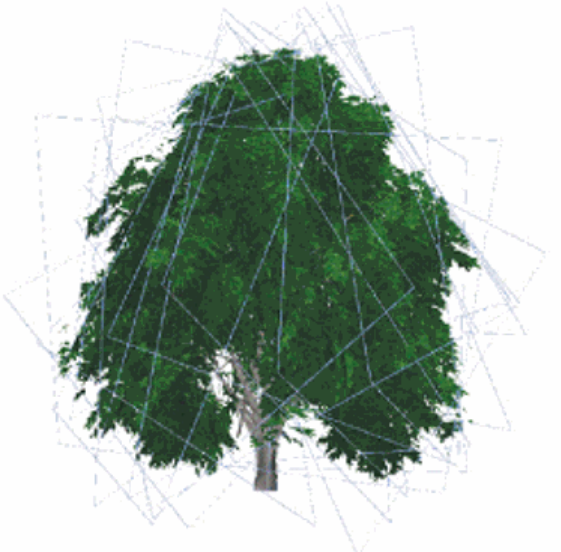
Imposters技术（伪装者，替代）

Billboard Clouds（云团，族）

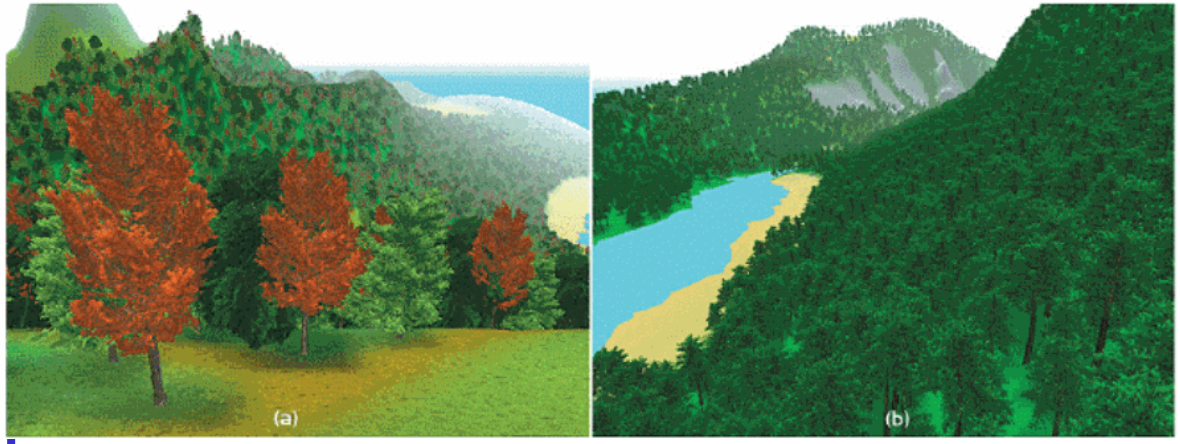
D´ecoret等人提出了公告板云（Billboard Clouds）的想法，即**一个复杂的模型通常可以通过一系列的公告板集合相互交叉重叠进行表示**

而D´ecoret等人也提出了一种在给定误差容限内对给定模型进行自动查找和拟合平面的方法。

公告板云可以添加一些额外的信息，如法线贴图、位移贴图和不同的表面材质。
裂纹沿裂纹面上的投影也可以由公告板进行处理。



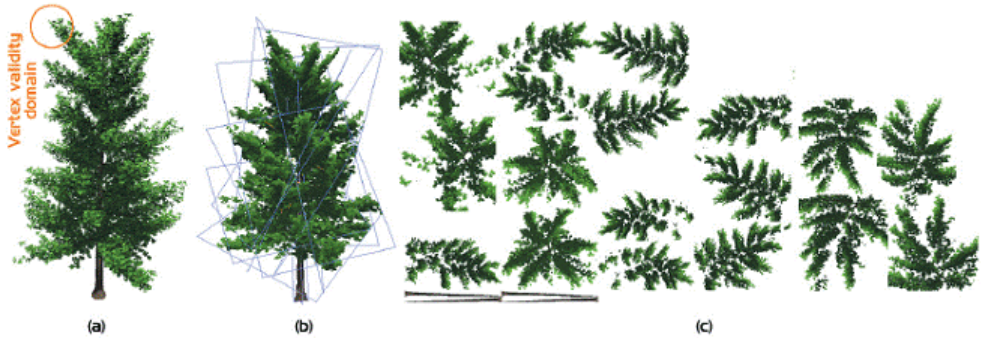
图a包含80000颗树，图b包含50000颗树



树由25公告牌（50个三角形）构成。
原始模型包含159853个三角形

Billboard Clouds（云团，族）

简化处理过程的想法是在一个误差控制范围内把高多边形、贴有纹理的输入模型投影到一系列近似原来几何体的平面

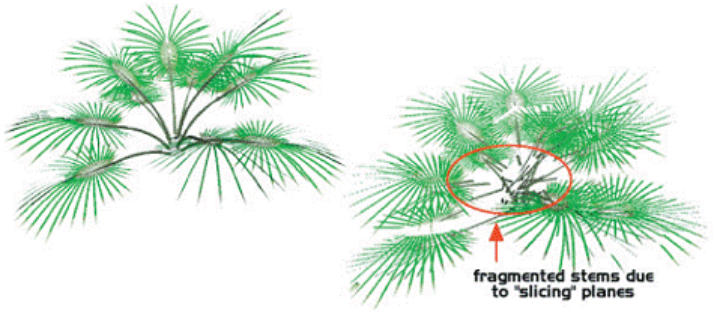


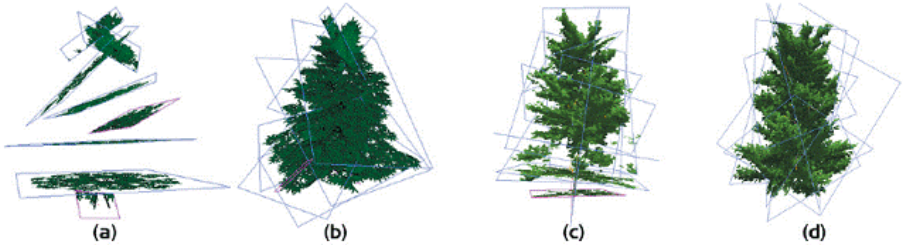
110000多边形树木模型（a）简化到11个公告牌；（b）误差阀值由顶点有效区域显示；（c）各个公告牌纹理在图c中显示

简化的质量由误差阀值ε决定，以模型的边界包围球半径的百分比来度量。一个顶点只能简化到一个平面如果它的法向量距离小于ε。因此，一个顶点周围带有半径ε的球形区域是它的有效区域。

如果一个平面和一个面的所有顶点的有效区域相交，那么表明它对于这个面是合法的（例如，它能简化到这个面）。以此类推，对于每个面有无穷的有效平面。

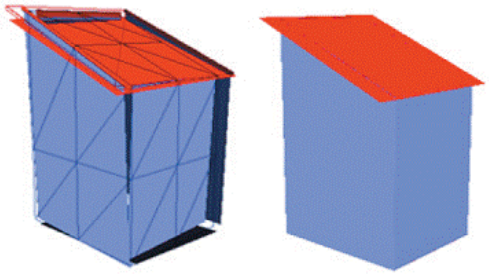
Billboard Clouds（云团，族）

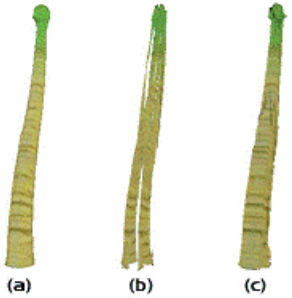




连通性问题

视点相关性问题





运最好的简化可以通过压缩多层植物至一个近似水平的平面来实现，但是其效果在视觉上并不一定正确(如图8a)。**在很多3D的模拟和游戏程序中，俯视角非常小，因此，观察者处在与树木差不多的高度上，所以应该避免纯粹的水平面。**

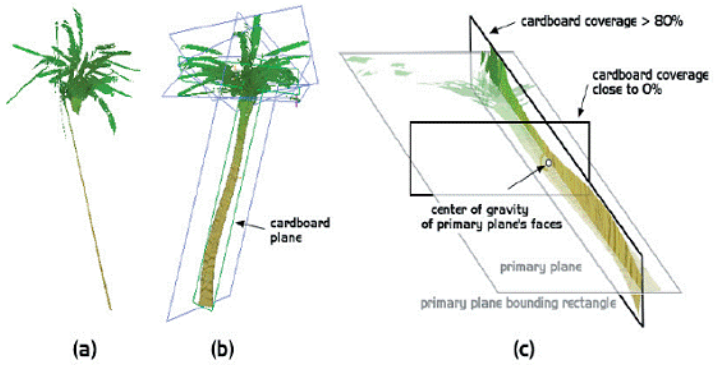
完全垂直的平面也是不可取的因为完全处置地看下去树木会显得很难看，而且植物的很多水平元素在垂直面上无法获得。

因此，需要一个方法来阻碍水平面，但不是完全避免它们。因为在一个平面被选中后将无法通过扭曲避免水平面。

平面扭曲问题

顶点链接问题-裂缝

Billboard Clouds（云团，族）



人们经常将几个公告牌成固定角相交连接，做成一个三维的图象，就有了"纸板"状的效果

纸板状模式 (Cardboard mode)：纸板状外观（Cardboard look）是一种消耗不昂贵的建模技术以致它看起来不真实并且在今天很少使用。但是，基本原理并不是完全没有优点。

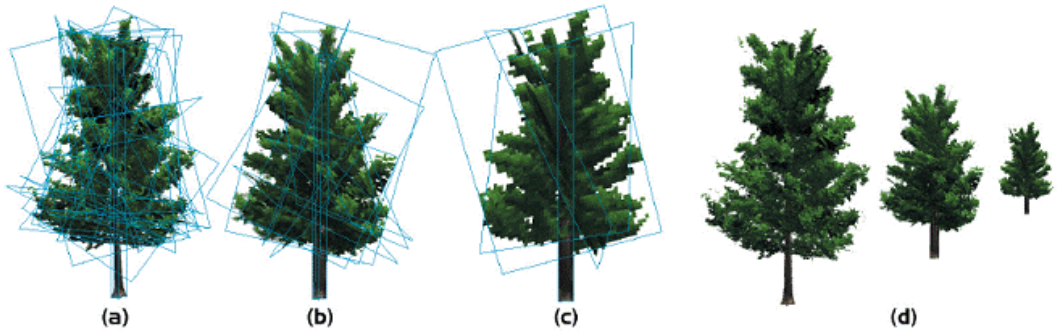
长的，薄的对象如树木的枝干通常简化到一个平面通过Billboard Cloud算法，它看起来显然平坦从一个方向（图7a）。

一个针对这样的平面的额外的正交公告牌（双纸板平面在我们的算法中）带来一个更好的视觉感知效果

Billboard Clouds（云团，族）

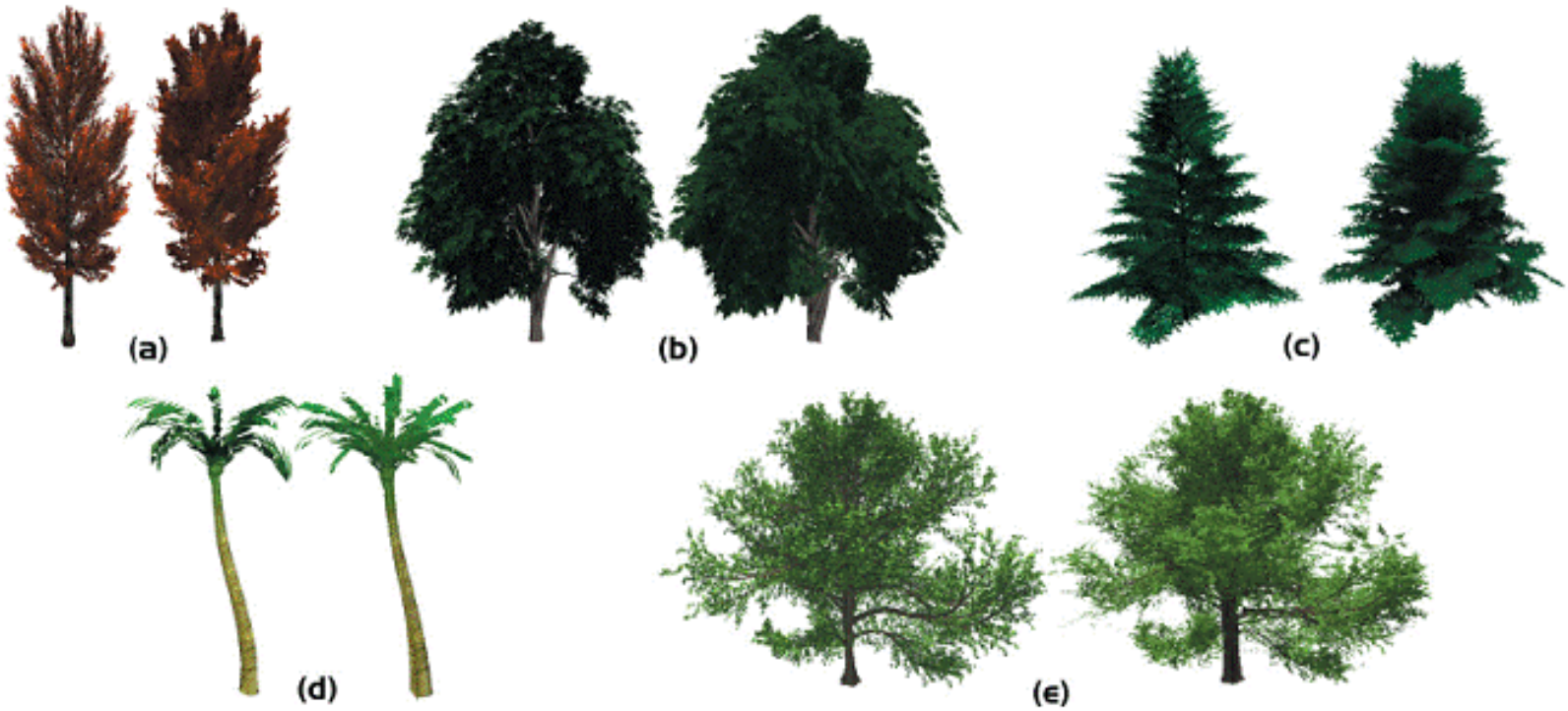
纹理生成 **(Texture generation)**：为Billboard Cloud平面生成纹理是一个非常直接的过程：通过反转平面参数旋转并平移输入模型，通过正交投影为平面渲染各个有效的三角形面片到指定纹理分辨率大小的一个离屏（off-screen）缓冲

细节层次 **(Incremental levels of detail)**：不同的细节层次（LOD）能够容易地生成通过改变误差阀值参数和纹理分辨率。这个方法的主要优点是完全独立地为不同LOD选择简化平面。对于大部分具有细节的LOD，我们采用增l量的方法，把上一个LOD作为下一个更低的LOD的输入。尽管这可能在极度化简的情况下产生差的结果(小于10个公告牌)，对于具有更高细节的各个LOD层次，层次之间的切换是可见的，这减少了切换的闪烁效果



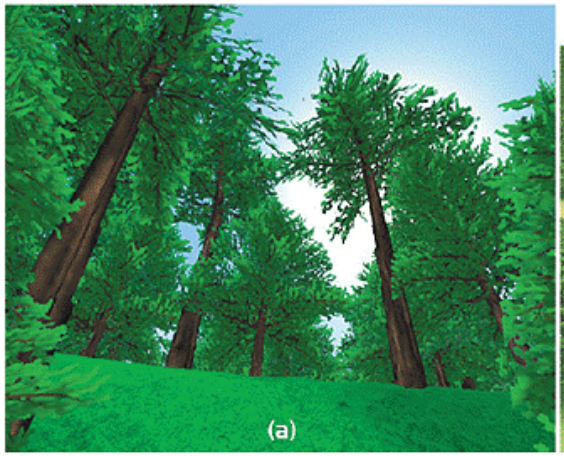
Billboard Cloud树木的不同细节层次（a）24个（b）11个（c）4个

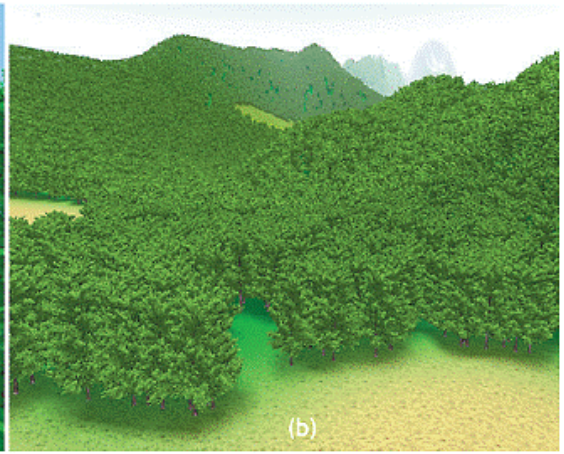
Billboard Clouds（云团，族）

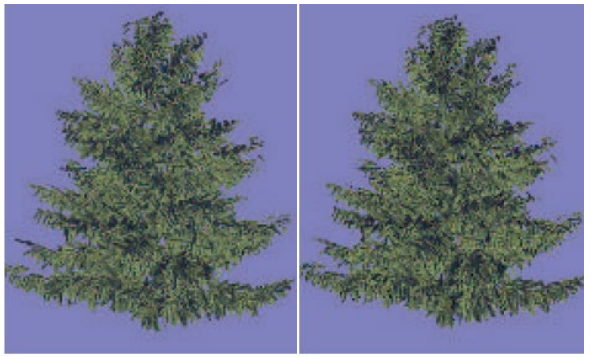


多边形建模后绘制的树木（左面），Billboard Clouds技术绘制的树木（右面）

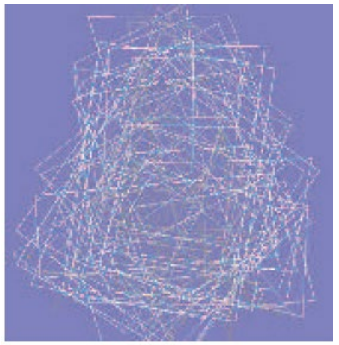
Billboard Clouds（云团，族）







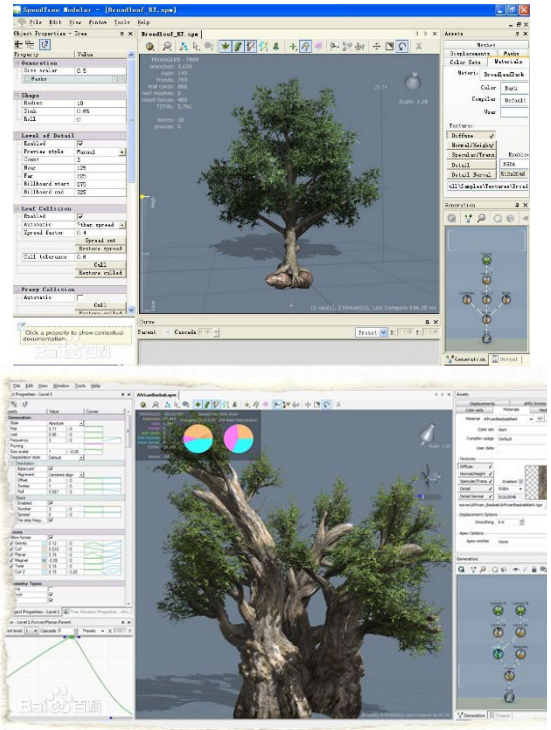
20610个面　　78张图



大规模的森林场景绘制

https://www.cnblogs.com/javazjh/archive/2011/09/08/2171655.html

树木的快速建模：基于插件与引擎



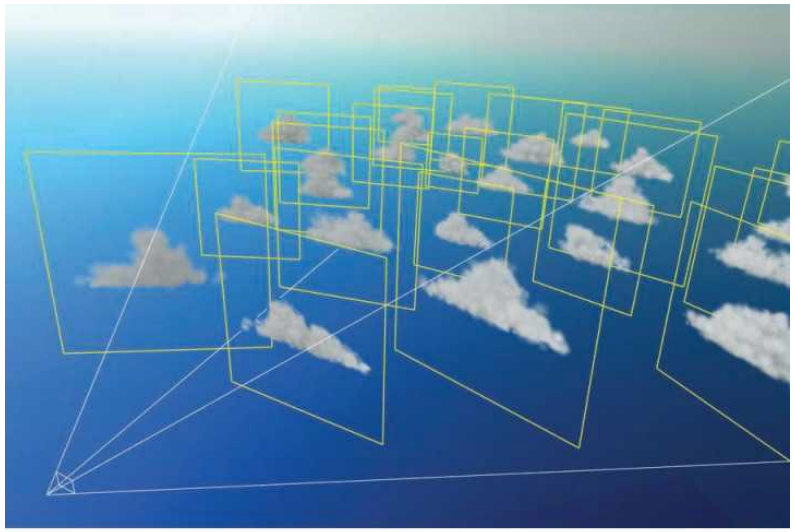
SpeedTree 插件

Axial 公告板技术

Unreal 引擎

Billboard Clouds（云团，族）

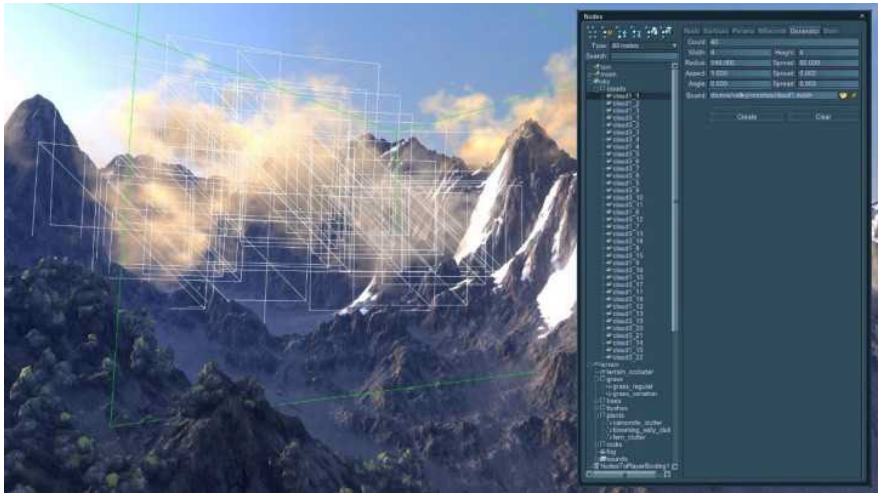
天空云层与云彩效果



飞行模拟类游戏画面

云彩的快速建模：插件和引擎

UNIGINE 引擎



游戏中的应用示例

《地平线：黎明》中远处的树木便利用了Billboard进行渲染

粒子系统（1）

- 粒子动画
  - 逐个地运动一大堆粒子，模拟自然界的“点云”运动的整体效果。

    比如焰火.
- 粒子
  - 是一些小的物质，每个都有自己的运动脚本

粒子系统（2）

- 工作原理
  - 基本思路是某些自然现象可以通过描述一大群单个粒子的运动和绘制来加以模拟.
- 工作方式
  - 单个粒子一般被视做几何上微小或者为零的基本单元，小到多个粒子可能投影到屏幕的一个象素—但是每个粒子拥有自己的属性，如颜色。

粒子系统（3）

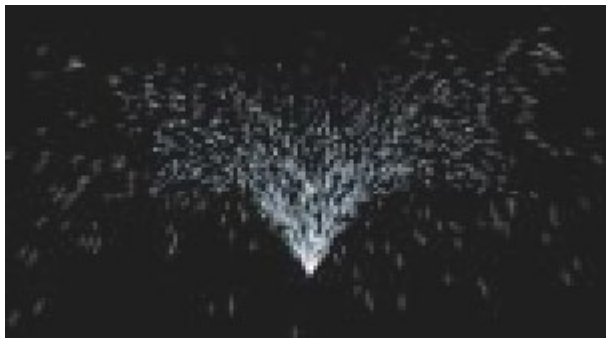
- 粒子的建模
  - 一般可以设计一个脚本
    - 赋予每个粒子一定的随机性，产生彼此之间的差异性。例如，每个粒子的位置在随时演化之中。
    - 不同的自然现象通过群体粒子属性脚本和单个粒子差异脚本描述。

粒子系统（4）

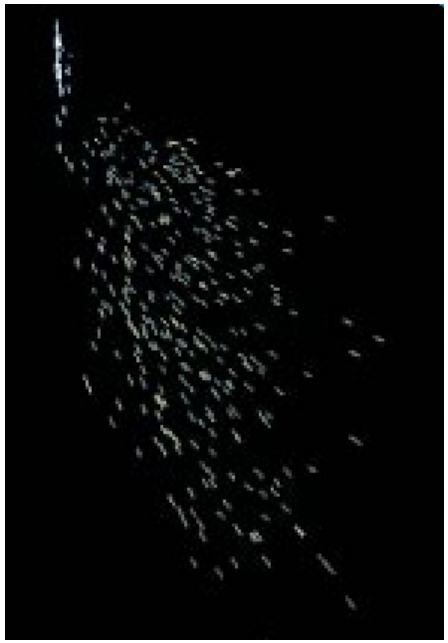
- 控制每个粒子差异性的参数包括下列特征，它们与粒子的位置和粒子本身的生命周期有关：
  - 运动轨迹,
  - 形状,
  - 颜色.
- 粒子的动力学行为和外观是时间函数，可以集成到同一个脚本描述

粒子系统（5）

- Reeves (1983)描述了生成单帧粒子动画的五个步骤:
  - 生成新的粒子并放置到当前系统中；
  - 赋予每个新的粒子属性.
  - 删除超越生命周期的粒子
  - 活性粒子根据各自的脚本运动
  - 绘制活性粒子



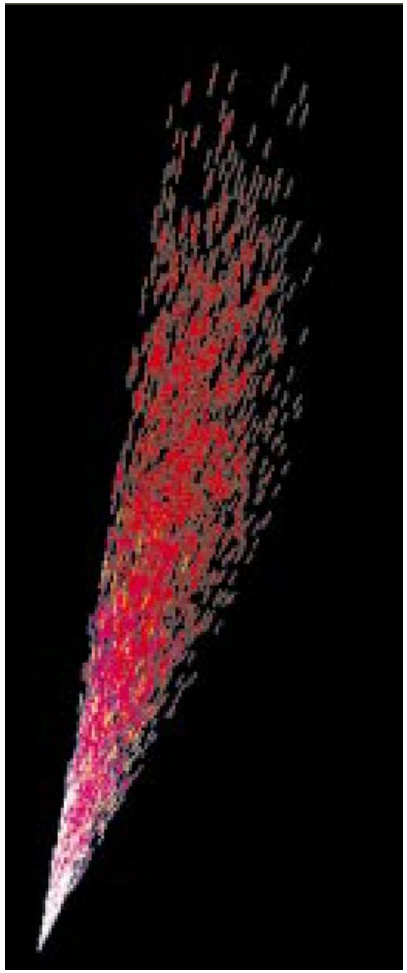
粒子系统（6）



- 单个粒子的脚本描述包含下面一系列属性:
  - 初始位置与速度
  - 发射的方向
  - 尺寸（如半径）和形状
  - 弹性系数和碰撞系数
  - 透明度
  - 摩擦系数、重力和风力
  - 生命周期
  - 纹理
    - 一般地，一个粒子总是被作为一个带纹理映射的长方形在绘制在屏幕上并与背景做透明度融合
      - GL_POINT
      - 线段

粒子系统（7）

- 粒子的发射
- 固定位置
  - 喷水
- 分布式
  - 分布在某一个三维空间的表面，
- 半-随机
  - 指定一个中心点



粒子系统（8）

- 粒子的物理特性
  - F = MA
  - A = 加速度 (初始的推力, 重力, 等)
  - V = 当前速度 = dt * A + V_{prev}
  - X = 当前位置 = dt * V + X_{prev}

粒子系统（9）

- 某时刻 t的粒子数量由下式决定：

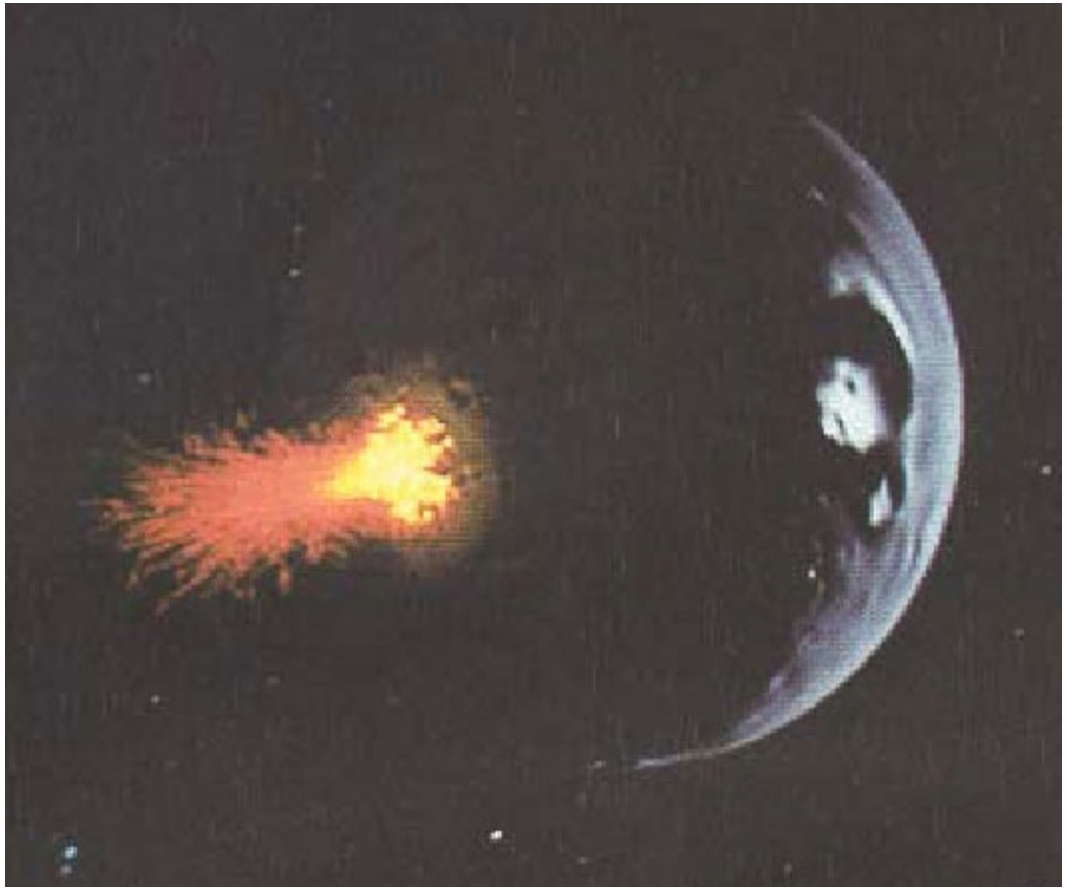
  $N(t)=M(t)+rand(t)\ V(t)$

  其中，M(t)是数学均值期望，V为方差.

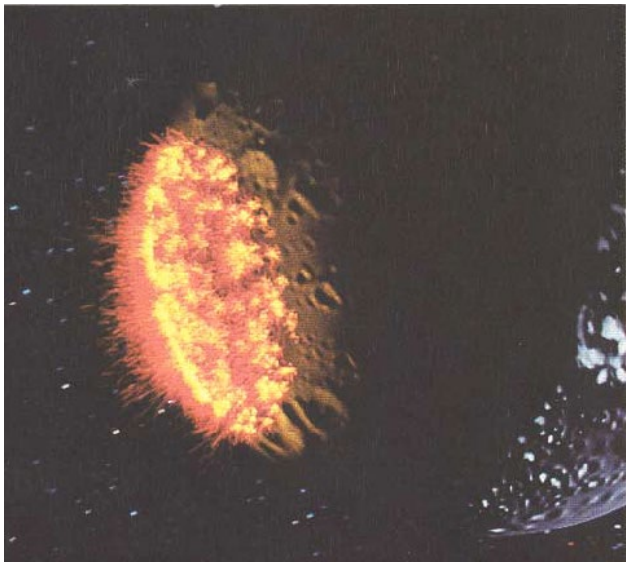
- 方程用于控制总体的点云变化（收缩和增长）尺寸.

粒子系统（10）

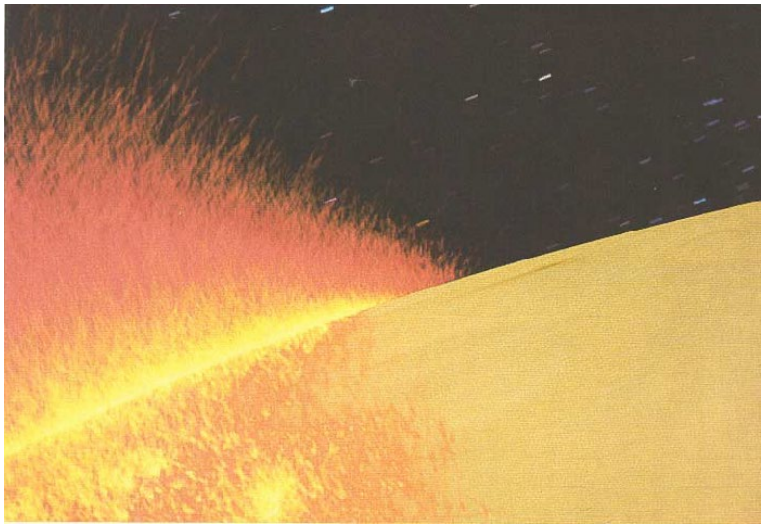
- 环境因素
  - 风力等
  - 全局作用力
  - 局部作用力
  - 碰撞的检测
  - 碰撞的响应：



粒子系统（11）



- 对于一个粒子系统：
  - 初始化所有粒子
  - 启动粒子系统
  - 处理所有粒子
  - 工作完成后清除所有粒子



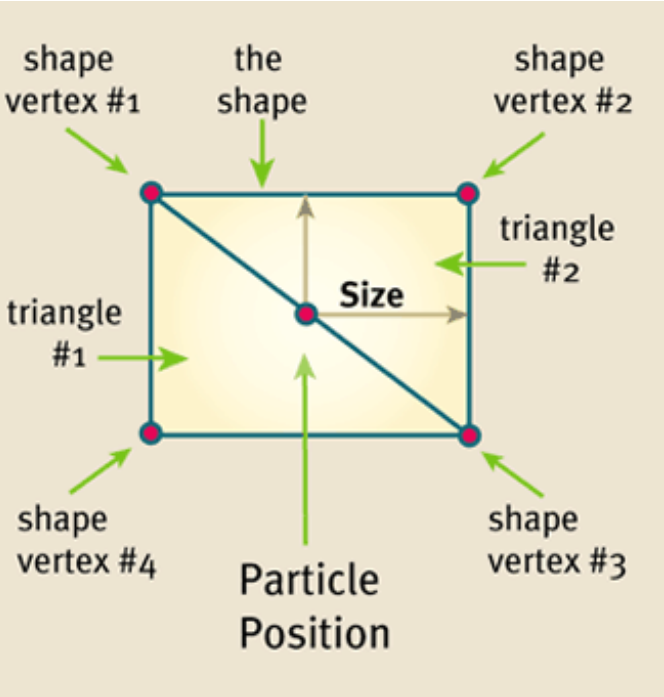
- 演示
  - Particle demo

粒子系统（12）

- 游戏中常见粒子效果
  - 两个三角形（4个顶点）
  - 烟雾是半透明
  - 纹理

粒子系统（13）

- 光晕火花
  - 纹理、碰撞形变模拟



粒子系统（14）

- 光照闪电
- 建立光照路径树
- 在各个分支上衍生粒子
- 把效果纹理一次性加载

粒子系统（16）

- 下雨的效果
- 粒子形状的拉伸变形

谢谢！

本文中有多幅图片来自网络，一并致谢！