目录

[第1章 导论 3](#_Toc5894658)

[1.1 典型游戏团队的结构 4](#_Toc5894659)

[1.2 游戏是什么 5](#_Toc5894660)

[1.3 游戏引擎是什么 5](#_Toc5894661)

[1.4 不同游戏类型中的引擎差异 5](#_Toc5894662)

[1.5 游戏引擎概观 6](#_Toc5894663)

[1.6 运行时引擎架构 6](#_Toc5894664)

[1.7 工具及资产管道 9](#_Toc5894665)

[第2章 专业工具 10](#_Toc5894666)

[2.1 版本控制 10](#_Toc5894667)

[2.2 微软Visual Studio（编译器、链接器、调试器） 11](#_Toc5894668)

[2.3 剖析工具profiler 13](#_Toc5894669)

[2.4 内存泄露和损坏检测memory leak and corruption 13](#_Toc5894670)

[2.5 其他工具 13](#_Toc5894671)

[第3章 游戏软件工程基础 14](#_Toc5894672)

[3.1 重温C++及最佳实践 14](#_Toc5894673)

[3.2 C/C++的数据、代码及内存 14](#_Toc5894674)

[3.3 捕捉及处理错误 17](#_Toc5894675)

[第4章 游戏中所需的三维数学 18](#_Toc5894676)

[4.1 在二维中解决三维问题 18](#_Toc5894677)

[4.2 点和矢量 18](#_Toc5894678)

[4.3 矩阵 18](#_Toc5894679)

[4.4 四元数 20](#_Toc5894680)

[4.5 比较各种旋转表达方式 21](#_Toc5894681)

[4.6 其他数学对象 22](#_Toc5894682)

[4.7 硬件加速的SIMD运算 22](#_Toc5894683)

[4.8 产生随机数 23](#_Toc5894684)

[第5章 游戏支持系统 23](#_Toc5894685)

[5.1 子系统的启动和终止 23](#_Toc5894686)

[5.2 内存管理 24](#_Toc5894687)

[5.3 容器container 26](#_Toc5894688)

[5.4 字符串 28](#_Toc5894689)

[5.5 引擎配置 29](#_Toc5894690)

[第6章 资源及文件系统 30](#_Toc5894691)

[6.1 文件系统 30](#_Toc5894692)

[6.2 资源管理器resource manager 32](#_Toc5894693)

[第7章 游戏循环及实时模拟 34](#_Toc5894694)

[7.1 渲染循环render loop 34](#_Toc5894695)

[7.2 游戏循环game loop 34](#_Toc5894696)

[7.3 游戏循环的架构风格 34](#_Toc5894697)

[7.4 抽象时间线abstract timeline 35](#_Toc5894698)

[7.5 测量及处理时间 35](#_Toc5894699)

[7.6 多处理器的游戏循环 36](#_Toc5894700)

[7.7 网络多人游戏循环 37](#_Toc5894701)

[第8章 人体学接口设备（HID） 37](#_Toc5894702)

[8.1 各种人体学接口设备 37](#_Toc5894703)

[8.2 人体学接口设备的接口技术 38](#_Toc5894704)

[8.3 输入类型 38](#_Toc5894705)

[8.4 输出类型 38](#_Toc5894706)

[8.5 游戏引擎的人体学接口设备系统 39](#_Toc5894707)

[8.6 人体学接口设备使用实践 40](#_Toc5894708)

[第9章 调试及开发工具 40](#_Toc5894709)

[9.1 日志及跟踪 40](#_Toc5894710)

[9.2 调试用的绘图功能 40](#_Toc5894711)

[9.3 游戏内置菜单in-game menu 41](#_Toc5894712)

[9.4 游戏内置主控台in-game console 41](#_Toc5894713)

[9.5 调试用摄像机和游戏暂停 41](#_Toc5894714)

[9.6 作弊cheat 41](#_Toc5894715)

[9.7 屏幕截图及录像 41](#_Toc5894716)

[9.8 游戏内置性能剖析in-game profiling tool 41](#_Toc5894717)

[9.9 游戏内置的内存统计和泄漏检测 42](#_Toc5894718)

[第10章 渲染引擎 42](#_Toc5894719)

[10.1 采用深度缓冲的三角形光栅化基础 42](#_Toc5894720)

[10.2 渲染管道 47](#_Toc5894721)

[10.3 高级光照及全局光照 51](#_Toc5894722)

[10.4 视觉效果和覆盖层 53](#_Toc5894723)

[第11章 动画系统 54](#_Toc5894724)

[11.1 角色动画的类型 54](#_Toc5894725)

[11.2 骨骼 54](#_Toc5894726)

[11.3 姿势pose 55](#_Toc5894727)

[11.4 动画片段 55](#_Toc5894728)

[11.5 蒙皮及生成矩阵调色板 56](#_Toc5894729)

[11.6 动画混合animation blending 57](#_Toc5894730)

[11.7 后期处理post-processing 59](#_Toc5894731)

[11.8 压缩技术 59](#_Toc5894732)

[11.9 动画系统架构 60](#_Toc5894733)

[11.10 动画管道animation pipeline 60](#_Toc5894734)

[11.11 动作状态机action/animation state machine，ASM 61](#_Toc5894735)

[11.12 动画控制器animation controller 63](#_Toc5894736)

[第12章 碰撞及刚体动力学 63](#_Toc5894737)

[12.1 你想在游戏中加入物理吗 63](#_Toc5894738)

[12.2 碰撞/物理中间件 64](#_Toc5894739)

[12.3 碰撞检测系统 64](#_Toc5894740)

[12.4 刚体动力学 67](#_Toc5894741)

[12.5 整合物理引擎至游戏 70](#_Toc5894742)

[12.6 展望：高级物理功能 71](#_Toc5894743)

[第13章 游戏性系统简介 72](#_Toc5894744)

[13.1 剖析游戏世界 72](#_Toc5894745)

[13.2 实现动态元素：游戏对象 72](#_Toc5894746)

[13.3 数据驱动游戏引擎 72](#_Toc5894747)

[13.4 游戏世界编辑器game world editor 72](#_Toc5894748)

[第14章 运行时游戏性基础系统gameplay foundation system 73](#_Toc5894749)

[14.1 游戏性基础系统的组件 73](#_Toc5894750)

[14.2 各种运行时对象模型架构 74](#_Toc5894751)

[14.3 世界组块的数据格式 75](#_Toc5894752)

[14.4 游戏世界的加载和串流 76](#_Toc5894753)

[14.5 对象引用与世界查询 77](#_Toc5894754)

[14.6 实时更新游戏对象 77](#_Toc5894755)

[14.7 事件与消息泵 78](#_Toc5894756)

[14.8 脚本scripting language 80](#_Toc5894757)

[14.9 高层次的游戏流程 82](#_Toc5894758)

使用书籍《游戏引擎架构》（Game Engine Architecture）

书籍网站：http://gameenginebook.com

计算图形学 14,42,1

二元空间分割树（binary space partitioning，BSP tree）

基于入口（portal）的渲染系统

遮挡剔除（occlusion culling）

**第一部分 基础**

# 第1章 导论

reusable software：可复用软件

middleware：中间件

1. 游戏引擎
   1. low-level foundation system
   2. rendering engine
   3. collision system
   4. physics simulation
   5. character animation
   6. gameplay foundation layer
      1. game object model
      2. world editor
      3. event system
      4. scripting system
2. 游戏性编程gameplay programming
   1. player mechanics
   2. camera
   3. artificial intelligence

游戏引擎及制造商举例

a）id Software

i. Quake; ii. Doom;

b）Epic Games

i. Unreal;

c) Valve

i. Source

## 1.1 典型游戏团队的结构

游戏工作室game studio

1.1.1工程师engineer

i. 运行时程序员runtime programmer：制作引擎和游戏本身

ii. 工具程序员tool programmer：制作离线工具，供整个团队使用

iii. 首席工程师lead engineer

iv. 技术总监technical director (TD)

v. 首席技术官chief technical officer (CTO)

1.1.2艺术家artist

i. 概念艺术家concept artist：通过素描与绘画，让团队了解游戏的预设最终面貌，即对于游戏内容的美术概念设计

ii. 三维建模师3D modeler：为游戏世界所有事物制作三维几何模型（按照游戏类 型的不同对于不同游戏内容中的对象有不同分工）

i) 前景建模师foreground modeler：制作物体角色载具武器等

ii) 背景建模师background modeler：制作背景地形建筑等

iii. 纹理艺术家texture artist：制作纹理的二维影像，用于增加模型细节和真实感

iv. 灯光师lighting artist：布置游戏世界的静态和动态光源，并通过颜色、亮度、 光源方向等设定，增加美感

v. 动画师animator

vi. 动画捕捉演员motion capture actor

vii. 音效设计师sound designer

viii. 配音及作曲家等voice actor/composer

ix. 艺术总监art director

1.1.3游戏设计师game designer：即设计游戏体验互动部分，称为游戏性；比如剧情关卡 目标对话等等；可由工程师出身

i. 游戏总监game director：保证设计的一致性等

1.1.4制作人producer：有时承担人力资源经理，有时担任游戏设计师，有时作为和其他 支持部门的联络人，可由游戏设计师出身

1.1.5 其他管理/支持人员（市场策划，法律，信息科技/技术支持，行政等）

1.1.6 发行商与工作室

发行商（publisher）负责游戏市场策划，制造与分销

•发行商举例：EA，THQ，Vivendi，Sony，Nintendo

•第一开发商（first-party developer）：直接隶属于游戏主机生产商的游戏工作室

## 1.2 游戏是什么

1.2.1 电子游戏作为软实时模拟

游戏的定义、组成与性质

软实时（soft real-time）互动（interactive）基于代理（agent-based）计算机模拟

•代理：模拟中独立的实体，比如载具，人物等

•时间性模拟（temporal simulation）：即游戏世界是动态的，游戏必须回应玩家输入

•时限（deadline）：所有实时模拟的核心概念，达不到时限次数的事件，比如刷新率等会造成游戏互动性下降

•软实时：指一些系统错过期限却不会造成灾难性后果，对比于硬实时系统，比如飞机和核能的控制系统

•数值建模（numerial）：对比于解析式（analytic）或闭合式（closed form），自变量（independent variable）可以取任何值；大部分数学问题没有闭合式解，而且对于有解析解的问题，在游戏中也仍然采用数值建模

•游戏循环（game loop）：游戏的运作机制，在循环的每次迭代中，多个游戏子系统计算并更新下一离散时步的状态，这也是采用数值建模的好处

## 1.3 游戏引擎是什么

游戏引擎

游戏引擎和游戏本身的界限往往很模糊，根据游戏的不同，二者有时可以区分有时难以 区分，只能定性的理解

•可复用性：游戏引擎是具有可扩展性的软件，不需要大量修改就可以作为多个游戏的基础，可复用性低的游戏就难以成为（或提取出）游戏引擎

•针对性：游戏引擎一般只适合开发某种类型的游戏，如RPG，射击，针对性越好制作出的游戏质量越佳

## 1.4 不同游戏类型中的引擎差异

游戏类型与引擎特点

1.4.1 FPS第一人称射击

a）高效渲染大型三维虚拟世界，或大型小型之间的切换

b）快速反应的摄像机控制与瞄准

c）游戏内容对象的刻画，AI等

d）玩家角色运动与碰撞模型

e）在线多人游戏能力

HUD（heads-up display）：平视显示器，来源于现代航空器，不需要低头就可以在视野中查看数据的仪器，游戏中指悬浮于视野边界的用户界面

1.4.2平台及第三人称游戏

平台游戏（platformer）：主要游戏机制是平台之间跳跃，显然这是按照2维横版清关模式游 戏定义的，和一般第三人称射击动作历险游戏一并考虑

a）主角的能力（ability）和地点运动模式（locomotion mode）以及逼真的主角动作刻画， 包括一切和主角互动的场景及互动模式

b）摄像机碰撞系统，保证视点不会穿过背景

1.4.3格斗游戏fighting game

a）丰富的格斗动画和角色的细节刻画

b）准确的攻击判定和复杂的控制信号侦测

1.4.4竞速游戏racing game

1.4.5实时策略游戏RTS，real-time strategy

基于栅格（grid-based）或基于单元（cell-based）构建游戏世界，使用正射投影（orthographic projection）

1.4.6大型多人在线游戏

场景建模和玩家数量庞大，图形逼真度一般逊色于其他游戏

massively multiplayer online game (MMOG)：大型多人在线游戏

dedicated server：专用服务器，大型线上游戏需要专用服务器，小型的不需要

peer-to-peer：点对点，以一个玩家的客户端兼任服务器来进行在线游戏，专用服务器的功能 是保证所有玩家同步互动

persistent world：持久世界，状态能持续很长的时间的游戏世界，比特定玩家的游戏时间都 要长很多

voice over internet protocol（VoIP）：IP电话，在线游戏玩家相互之间联络方式

1.4.7其他游戏

a）体育游戏

b）角色扮演游戏

c）上帝模拟游戏（god game）

d）环境与社会模拟游戏（environmental/social simulation）：模拟游戏

e）解密游戏（puzzle）：俄罗斯方块（Tetris）

d）非电子游戏移植：棋牌类游戏

## 1.5 游戏引擎概观

游戏引擎实际发展概观

首个FPS游戏是《德军总部Castle Wolfenstein 3D》1992，id software。

1.5.1雷神之锤（Quake）引擎家族，id Software

1.5.2虚幻引擎（Unreal），Epic Games

UDN，Unreal Developer Network

1.5.3 Source引擎，Valve公司

CryEngine3，CryTek公司，德国

1.5.4 XNA Game Studio，微软

1.5.5其他：C4引擎，Terathon Software

1.5.6专有内部引擎：SAGE，Westwood

1.5.7开源引擎

OGRE，Panda3D，Yake（基于OGRE），Crystal Space，Torque（便宜）及Irrlicht

授权方式：a）GNU公共许可证，General Public License，GPL，免费使用代码， 但作品必须为GPL；b）GNU lesser general publiclicense，LGPL，允许商业盈利

## 1.6 运行时引擎架构

运行时引擎架构概观p26/50

循环依赖（circular dependency）：软件层（software layer）之间的关系，通常上层依赖下层， 反之不然，如果下层依赖上层，则为循环依赖；这种关系导致系统间耦合（coupling）， 妨碍代码重用

由底至上：

1.6.1 目标硬件：即计算机或游戏主机硬件

1.6.2 设备驱动程序（device driver）：管理硬件资源的程序

1.6.3 操作系统（operating system，OS）

时间片（time-slice）方式，使多个程序共享硬件，称为抢占式多任务（preemptive multitasking）

1.6.4 第三方软件开发包和中间件

软件开发包（software development kit，SDK）提供基于函数或类的接口，一般称为应 用程序接口（Application programming interface，API）

1.6.4.1数据结构及算法

i）STL（C++标准模板库，standard template library）

ii）STLport

iii）Boost

iv）Loki

1.6.4.2 图形

i）Glide

ii）OpenGL

iii）DirectX

iv）libgcm

v）Edge

1.6.4.3 碰撞和物理

i）Havok

ii）PhysX

iii）Open Dynamics Engine（ODE）（开源）

iv）Bullet（开源）

1.6.4.4 角色动画

随着游戏角色真实化提高，物理和动画之间的分界线不明显

i）Granny

ii）Havok Animation

iii）Edge

1.6.4.5 人工智能

i）Kynapse

1.6.4.6 生物力学角色模型

物理软件包的一个特例，专门处理角色动作

i）Endorphin和Euphoria

1.6.5 平台独立层

平台独立层（platform independence layer）包装了常用的标准C语言库，操作系统调用 及其他基础API，确保包装了的接口在所有硬件平台上均为一致。这样一款游戏可以运 行于不同的平台上

1.6.6 核心系统

大规模复杂的C++应用软件，都需要有一些有用的实用软件（utility），这类软件在此称 为核心系统（core system）

a）断言（assertion）：一种检查错误的代码，最终版本中会移除断言检查

b）内存管理：自定义内存分配系统，以保证高速的内存分配和释放

c）数学库：解决数学问题的库

d）自定义数据结构及算法：若非完全依靠第三方软件包，如STL，则需自定义工具去 管理数据结构以计算法，有时需要手工编码，以减少或完全消去动态内存分配，最大程 度提高目标平台运行效率

1.6.7 资源管理器

调用各种游戏资源的方式（如字体，材质纹理，模型，地图，参数等）

方式有：专案（ad hoc），虚幻引擎的包（package），OGRE引擎的ResourceManager类， 雷神之锤的PAK压缩文件

1.6.8 渲染引擎

分层架构（layered architecture）

1.6.8.1 低阶渲染器（low-level renderer）

着重与高速渲染丰富的几何图元（geometric primitive）

i）图形设备接口（graphics device interface，GDI）：负责初始化图形设备，建立渲 染表面的组件

ii）其他渲染器组件

1.6.8.2 场景图/剔除优化

背面剔除（back-face culling）

平截头体剔除（frustum cull）

空间细分（spatial subdivision）：潜在可见集（potentially visible set，PVS）

二元空间分割树（binary space partitioning）

四叉树（quadtree）

八叉树（octree）

kd树

包围球树（bounding sphere tree）

1.6.8.3 视觉效果

i）粒子系统（particle system）：烟，火，水花等

ii）贴花系统（decal system）：弹孔和脚印等

iii）光照贴图（light mapping）及环境贴图（environment mapping）

iv）动态阴影（dynamic shadow）

v）全屏后期处理效果（full-screen post effect）

•高动态范围（high dynamic range，HDR）光照及敷霜效果（bloom）

•全屏抗锯齿（full-screen anti-aliasing，FSAA）

•颜色校正（color correction）及颜色偏移（color-shift）效果，包括略过漂白 （bleach bypass）、饱和度（saturation）、去饱和度（desaturation）等

1.6.8.4 前端

大多数游戏为了不同目的，会使用一些二维图形去覆盖三维场景

i）平视显示器，HUD

ii）内置菜单，主控台，其他开发工具（可能不会最终发行）

iii）内置GUI，用于让玩家设定角色，配置游戏等等

iv）全动视频（full-motion video，FMV），负责播放之前录制的全屏电影

v）游戏内置电影（in-game cinematics，IGC）在游戏本身以三维形式渲染电影情节

1.6.9 剖析和调试工具

通用软件剖析工具（profiling tool）和内存分析工具（memory analysis tool）

1.6.10 碰撞和物理

一般都采用第三方SDK，而不是自己编写

碰撞检测（collision detection）

运动学（kinematics）：研究运动的物理学分支

动力学（dynamics）：研究力和力矩

1.6.11 动画

游戏中的五种基本动画：

i）精灵/纹理动画（sprite/texture animation）

ii）刚体层次结构动画（rigid body hierarchy animation）

iii）骨骼动画（skeletal animation）

蒙皮（skinning）

布娃娃（ragdoll）

iv）每顶点动画（per-vertex animation）

v）变形目标动画（morph target animation）

1.6.12 人体学接口设备（human interface device，HID）

键盘和鼠标

游戏手柄

其他

1.6.13 音频

1.6.14 在线多人/网络

1）单屏多人（single-screen multiplayer）：所有玩家均处于同一屏幕，格斗游戏居多

2）切割屏多人（split-screen multiplayer）：每个玩家居于一个窗口内，共处一屏幕

3）网络多人（networked multiplayer）：多台计算机或主机用网络连接，每个机器对应 一个玩家，类似局域网

4）大型多人在线：服务器运行

客户端于服务器之上（client-on-top-of-server）：单人模式运行多人游戏

1.6.15 游戏性基础系统

游戏性（gameplay）：游戏内的活动，支配游戏世界的规则，玩家角色能力（或玩家机 制player mechanics），玩家游戏目标等

游戏性基础层gameplay foundation layer：包括

1.6.15.1游戏世界和游戏对象模型game object model

软件对象模型和游戏对象模型紧密结合：

1.6.15.2事件系统

事件驱动架构event-driven architecture：发送者建立一个称为事件event或消息message的数据结构，包含消息类型和参数。（某个管理对象）调用接收对象的事件处理函数event handler function来处理消息，或者事件也可以存储于队列中。

1.6.15.3 脚本系统

集成脚本语言的引擎，更改游戏逻辑或数据后，无需重新编译、链接所有程序，只需重载脚本代码即可。

1.6.15.4 人工智能基础

Kynogon公司开发的Kynapse商用AI引擎

1.6.16 个别游戏专用子系统

## 1.7 工具及资产管道

游戏资产game asset：即游戏所需的各种模型、动画、纹理等

1.7.1 数字内容创作工具digital content creation, DCC

动画，模型，纹理制作工具

1.7.2 资产调节管道asset conditioning pipeline

定义：从DCC到游戏引擎加工游戏资产的过程

DCC软件的数据格式往往无法直接用于游戏中，因为一方面这些数据格式为了便于编辑修改而加入了很多附加信息（在成品游戏中不再需要），另一方面这些数据格式并不为了游戏快速加载而设计，甚至干脆不公开。

1.7.3 三维模型/网格数据

1.7.3.1 笔刷几何图形brush geometry

原始

1.7.3.2 三维模型（网格）

导出器exporter：用于从DCC工具获取数据的软件

1.7.4 骨骼动画数据

骨骼网格skeletal mesh：又称为skin皮肤，因为它在不可见的骨骼上形成皮肤（这里的皮肤是指骨骼网格，也就是带有动作的角色的模型，而不是一般意义的纹理）。骨骼网格的顶点受到一组关节joint的不同程度weight的影响。为关节动画而绑定到骨骼层次结构上hierarchy

换句话说，骨骼包括关节的定义，网格就是三维模型，二者联系起来称为骨骼网格。用于给有动作的模型定义动作动画。

1.7.5 音频数据

1.7.6 粒子系统数据

1.7.7 游戏世界数据及世界编辑器

一般需要自行编写，没有商用的

1.7.8 一些构建工具的方法

不同种类的游戏引擎和工具软件的构架关系。当然，不同的工具软件有不同的架构。一般世界编辑器和引擎的耦合程度最高。

# 第2章 专业工具

## 2.1 版本控制

版本控制系统version control system：也称源代码控制source control，容许多位开发者协同在同一组文件上工作。

2.1.1 为何使用版本控制

版本控制系统的功能：  
提供中央版本库repository，即代码库  
保留文件更改记录  
为版本增加标签，以便查找  
允许从主生产线建立分支，可供制作示范程序和补丁

2.1.2 常见的版本控制系统p54/78

·SCCS和RCS

·CVS

·Subversion(SVN) 开源小规模最优

·Git

·Perforce 大规模最优

·NxN Alienbrain

·ClearCase

·Visual SourceSafe

2.1.3 Subversion和TortoiseSVN概览

大部分版本控制系统采用客户端/服务器架构client-server architecture

2.1.4 在Google上设置代码版本库

Google Code即开源免费服务器端设置，如需非开源项目，须自行设置服务器

2.1.5 安装TortoiseSVN

这是一个流行的Subversion前端，使用方法类似网盘客户端

2.1.6 文件版本、更新和提交

2.1.7 多人签出、分支及合并

签入check-in：即对被编辑文件进行提交submit，至版本库进行保存

签出check-out：指定某个版本库的文件，下载到本地进行编辑，完整意义上的签出也保证没有被签出的文件不可修改同步至库。

独占签出exclusive check-out：只能有一个人同时编辑文件，编辑时锁定文件

多人签出multiple check-out：多人可以同时编辑同一文件，签入时需要合并修改部分

三路合并three-way merge：多人签出修改相同部分内容后的合并方式

2.1.8 删除

版本库中的文件删除实则为在最新版本里不予显示。

## 2.2 微软Visual Studio（编译器、链接器、调试器）

2.2.1 源文件、头文件及翻译单元

翻译单元translation unit：编译器每次翻译的最小单元，一般为一个源文件

2.2.2 程序库、可执行文件及动态链接库

对象文件object file：也称目标文件，是编译后的机器码，然而仍不能直接执行，需要重定位relocatable和链接link

链接link：将目标文件和程序库library进行链接，解析所有的外部函数和数据引用

程序库library：实际上就是多个对象文件的集成，以供外部链接之备用

重定位relocatable：只有最终链接成的可执行文件才能确定该程序被执行时机器码的内存分布。然而这仍然只是相对分布，最终在内存中的绝对地址仍需在执行时临时确定。

动态链接库dynamic linked library，DLL：相较于静态链接库，DLL不会被链接至可执行文件中成为其固定的一部分，因此可以独立的更新。在程序启动时才会被链接至可执行文件上。

2.2.3 项目及解决方案

项目project是源文件的集合，解决方案文件solution file是项目的集合

2.2.4 生成配置build configuration

含义：解决方案内项目编译所使用的各个组件的设置选项集合，如编译器、链接器和调试器

常见的生成配置为debug和release

2.2.4.1 常用生成选项

①预处理器设置

可以使用形如#pragma等指令使用预处理器对生成过程进行操控

可以通过命令行定义预处理宏（和在文件中用代码定义等效），再在代码中使用条件编译功能conditional compilation来区别对待不同的生成设置。这些预处理宏可以涉及代码语言种类、编译器种类、目标平台种类等信息。即一旦某个相关符号被宏定义（如\_cplusplus），说明该符号代表的性质对本文件适用（本文件使用c++编写）。

②编译器设置

调试信息debugging information：调试生成设置下，编译器会保留或产生大量用于调试器追踪的信息，这些信息是最终版本的软件不需要的。

优化optimization：是编译的一部分可选功能，但是调试过程中往往关闭优化便于在前后一致的代码中寻找问题

③链接器设置

决定了连接那些库，以及输出文件的类型是可执行文件还是库文件等。

按照惯例，调试时链接也会使用调试用的外部库，发布时才链接优化的库。

2.2.4.2 典型生成配置

·调试debug 未优化，含有大量调试信息，运行最慢

·发布release 并非是最终版本的生成配置，可以理解为阶段性测试的版本，仍有调试信息保留但速度更快（已优化）

·制作production 这是最终版本的生成配置，也称最终final或光盘disk配置

·工具tools 专门用于制作工具软件的生成配置，这是因为工具有时和游戏本体共用代码库，但使用不同的部分。tools实际上也分调试和发布两种生成配置

①混合生成版本hybrid build：大部分翻译单元使用发布模式，部分使用调试模式，用于调试大型软件的一部分

②生成配置和可测试性

理论上，所有的生成配置均需要彻底测试，因为可能存在不同的bug。但是实用中，一般只会彻底测试发布和制作两个生成配置。

母片gold master：指用于发售最终版本的光盘或类似载体

2.2.4.3 项目配置教程

配置下拉组合框

常规属性页

输出目录：最终输出内容  
中间目录：目标文件的输出目录

调试属性页

C/C++属性页  
设置编译器的属性

链接器属性页  
附加库目录：存储库文件和对象文件的目录  
依赖项dependencies：指定外部库文件和对象文件

注意：C++可以使用预处理命令#pragma设置连接库，即在链接器属性页不一定看到所有依赖库的信息

2.2.4.4 创建新的.vcproj文件

使用向导

复制现有项目  
可以使用xml编辑器编辑vcproj文件，查找替换需要改动的原内容

2.2.5 调试代码

2.2.5.1 启动项目

2.2.5.2 断点breakpoint

2.2.5.3 单步执行代码

f11 step in和f10 step over

2.2.5.4 调用堆栈call stack

即函数调用堆栈，显示当前被调用函数

2.2.5.5 监视窗口watch window

2.2.5.6 数据断点data breakpoint

也称硬件断点hardware breakpoint。即当某个数据被写入时产生中断，用于查找数据值错误的原因。

2.2.5.7 条件断点

条件断点conditional breakpoint：在设置断点时，额外指定一个条件表达式，当其值为真时才中断程序

命中次数hit count：设置断点时的一种条件，命中次数达到设定值时才中断，用于在循环次数较多的循环体中击中某一次循环

2.2.5.8 调试已优化的生成

bug只出现在发布生成的常见原因（均为优化器问题和两种生成方式之间的区别）：

①未手动初始化变量（调试模式中自动设为0，但发布模式中没有自动初始化）

②发布生成中意外的略去一些代码（重要代码错误地放进断言）

常见窍门：

①学习在调试器中阅读及单步执行反汇编disassembly：汇编语言和平台架构的知识需求

②运用寄存器去推理变量的值或地址：利用近期CPU寄存器的值来查看可能的变量值或地址

③使用地址去检查变量及对象内容

④利用静态和全局变量：成熟的软件体系中，可能有静态和全局变量存有部分当前变量的值，这些全局变量即便在优化的生成中一般也能够被调试器正常检查。

⑤修改代码：通过加入一些便于调试的代码，比如增加打印语句显示情况，引入全局变量用于存储值（可能不会被优化），或增加代码孤立一个类的实例。

## 2.3 剖析工具profiler

Pareto principle：或称80-20rule。指大部分问题只由少部分原因造成。

剖析器：能够度量代码执行时间、每个函数的调用时间、调用次数等信息的工具。可用于锁定需要重点优化的代码部分。

统计式剖析器statistical profiler：也成为不唐突的unobtrusive剖析器。它不会严重拖慢程序，原理是对CPU运行状态采样并估算使用情况。

测控式剖析器instrumental profiler：使用这种剖析器需要对源程序进行修改，加入一些调用剖析器的代码，非常缓慢但是提供的数据非常精确。

也存在两种剖析器的结合版或中间版，如LOP（low over-head profiler）

2.3.1 剖析器列表

见维基百科

## 2.4 内存泄露和损坏检测memory leak and corruption

问题的关键就是正确合理使用指针。养成良好的编程习惯（或使用工具）可以杜绝这个问题。

## 2.5 其他工具

p80/104

区别工具difference/diff tool：用于比较两个文本的区别

三路合并工具three-way merge tool：

十六进制编辑器hex editor：用于查看及修改二进制文件的内容。

# 第3章 游戏软件工程基础

## 3.1 重温C++及最佳实践

3.1.1 扼要重温面向对象编程

略

3.1.1.6 设计模式design pattern

即解决一些类型问题所采用的套路：

常见的通用设计模式：

单例singleton：每个类只有一个实例

迭代器iterator：提供高效存取一个集合的方法

抽象工厂abstract factory：提供一个接口，创造一组相关类，但不需要指明这些类的具体

3.1.2 编码标准：为什么及需要多少

常用的编码标准（编程习惯）：

·保持接口文件和头文件的整洁、简单、易于理解

·使用直白易于理解的好命名

·合理使用命名空间以达到清晰目的

·学习并遵从最好的C++实践  
Scott Meyers《Effective C++》  
John Lakos《Large-Scale C++ Software Design》

·遵守代码已用的约定如一

·让代码自己能够显露错误  
Joel Spolsky《Making Wrong Code Look Wrong》

## 3.2 C/C++的数据、代码及内存

3.2.1 数值表达形式

略

3.2.1.2 有符号及无符号整数

有符号整数一般采用二补数two’s complement记法，最高位为1是负数，但是并非去掉最高位剩下的数就是负数的绝对值，而是从全1（0xFFFFF…代表-1）开始倒数计数。

3.2.1.3 定点记法fixed-point

笨拙的小数记法

3.2.1.4 浮点记法floating-point

浮点数一般由三部分组成：尾数mantissa，指数exponent和符号位。但是如何组织三部分数字的位数分配进而表达实际的小数值存在不同的标准。一般情况下，尾数和指数以2进制科学计数法的形式存在。

IEEE-754 32bit p92/116

范围和精度的取舍：浮点数的总位数是一定的，所以其有效数字significant figure是一定的。数值越大，指数越大，精度较低（数字全部在小数点左边）；数值越小，指数越小，精度较高（数字全部在小数点右边）。这个特点使得如果利用一个固定位数的浮点数从低至高开始以一定间隔计数，那么一定会在到达最大值之前遇到计数间隔小于精度间隔而计数无法增加的失效情况（如游戏时间）。

机器的epsilon：一种度量计数精度的数据，满足1+ε≠1的最小值，即尾数的最小值。注意epsilon不等于该计数精度所能表达的最小值，而是在1附近的最小精度。

浮点数的位操作技巧

注意积累一些快速的浮点数运算算法

3.2.1.5 基本数据类型

①编译器专属特定大小类型：

一般高级语言的基本数据类型为了做到硬件平台可移植性，不做明确的大小规定。为不同的平台编译的程序，编译器会自动改变类型的位数。但是VS的编译器可以使用扩展关键字特别声明确定位数的变量类型，如\_int8、\_int16等

②SIMD类型

算数逻辑单元arithmetic logic unit，ALU：CPU中的计算硬件

矢量处理器vector processor：也称矢量单元vector unit。提供一种并行计算处理方式，称为SIMD，单指令多数据，single instruction multiple data。对于一些矢量和矩阵计算，矢量处理器比单指令单数据SISD单元更加高效。一般SIMD数据类型是由两个或更多数值打包而成的。

③可移植的特定大小类型

因为不同编译器使用不同的自定义类型和特定大小类型，游戏一般会自定义基本数据类型，保证代码可移植性。

④ORGE的基本数据类型

3.2.1.6 多字节值及字节序

多字节量multi-byte quantity：大于8位的值

最高有效字节MSB，most significant byte：数值的前八位

最低有效字节LSB，least significant byte：数值的后八位

字节序endianness或byte order：分为大端big-endian和小端little-endian。小端处理器将LSB放在较低内存位置，大端正好相反。一般PC上的intelCPU是小端，主机上是大端

整数字节序转换p98/122

浮点字节序转换p99/123

3.2.2 声明、定义及链接规范

3.2.2.1 再谈翻译单元

未解决引用unresolved reference：编译后的对象文件内对其他外部文件的引用，在链接之前所处的状态

函数级链接function-level linking：一种以函数作为链接单位而不是目标或对象文件的机制

3.2.2.2 声明与定义

略

①多重声明和定义

同一个文件中的多重定义（不是重载）会被编译器发现，不同文件中的多重定义会被链接器发现

②头文件中的定义，以及内联

头文件中除了内联函数以外，其他的函数只能声明不能定义。内联函数必须定义，否则编译器无法对其工作。而是否真的内联编译器可自行判断，也可以通过人为强制。

3.2.2.3 链接规范linkage

定义：链接规范就是一个变量或函数的定义是否可以被外部引用，能够被外部引用的外部链接是默认的，而内部链接需要用static关键字，两个不同文件内的同名内部链接不冲突。而若要使用其他文件中定义的外部链接实体，在本文件中须使用extern先声明

广义上来说，声明总是内部链接的，即只对当前文件有效。内联函数也是内部链接的，因而可以置于头文件中。

3.2.3 C/C++内存布局memory layout

3.2.3.1 可执行映像executable image

可执行与可链接格式executable and linkable format，ELF：主机和UNIX系统的可执行文件格式。exe是windows上的

可执行文件：包含程序的部分映像，执行时会置入内存。一般包含4个段segment或节section：  
·代码段text/code segment：程序定义的全部函数的机器码  
·数据段data segment：全部在代码中获初始化的全局变量和静态量（包括函数内静态变量）  
·BSS segment：block started by symbol，包含程序中全部未初始化全局变量和静态变量（包括函数内的），C++中这些量全部为零，但是仅以零的字节个数储存这些信息以降低文件大小。  
·只读数据段read only data segment：rodata段，包含程序中的只读全局常量，但是不包括整数常量（也称明示常量manifest constant），后者直接插入代码段

3.2.3.2 程序堆栈program stack

堆栈帧stack frame：调用一个函数时，其相关信息被写入一帧，进而压入堆栈。相关信息包括3类数据：  
·调用函数的返回地址return address：返回后继续在原来代码段中位置  
·CPU寄存器内容：被调用方开始执行之前，CPU寄存器状态的备份，调用结束后调用方可以恢复寄存器状态继续工作（返回值所用寄存器不在此列）  
·函数中的所有局部变量local variable：也称自动变量automatic variable，每个函数调用都有自己一组独特的局部变量集合

堆栈帧结构：p108/132

堆栈指针stack pointer：一个CPU寄存器的值，一般通过调整它来实现堆栈帧的压入和弹出

函数返回后，虽然堆栈帧实际上仍存在，但是已不被保护，不能再使用任何相关的地址。

3.2.3.3 动态分配的堆

堆内存heap memory：也称自由存储free store，即用于动态分配内存的内存空间。

3.2.4 成员变量

3.2.4.1 类的静态成员

3.2.5 对象的内存布局

使用方形内存布局图描述对象p111/135

3.2.5.1 对齐和包裹

对齐alignment：数据在内存之中按照一定的对齐方式存储，便于CPU读写。一般对齐是在内存地址最低有效半字节least significant nibble为2的幂的倍数的位置上。分为1字节对齐（任何地址）、2字节对齐（最低半字节为偶数的地址）、4字节对齐（4的倍数）、16字节（只有0x0）对齐等几种。不同的数据结构需要不同的对齐，一般和其字节大小一致，即32位数据采用4字节对齐。

注：每个内存地址对应一个字节的内存空间。4G内存意味着每个内存地址的地址码长度为4字节。

不对齐的数据也可以读取（依赖于机器，也有的不能），但是需要读取对齐的两块（至少）区域然后将结果移位、拼合。

对于一个含有不同类型数据成员的数据结构，整个结构的对齐需求等同于成员中最大的对齐需求。一个良好的习惯是了解每个数据的大小，将较小的数据声明放在一起，这样等同于手动对齐，节省空间。

3.2.5.2 C++中类的布局

每个新的派生类都会简单地把其数据成员附加到基类布局末端。

虚表指针virtual table pointer或vpointer：每个含有虚函数（定义或继承的）的具体类都有一个指向虚函数表vtable的指针，它位于类的内存布局最前端，也会保留在类的实例中。虚函数表和类一一对应，包含每个类中的虚函数指针。

## 3.3 捕捉及处理错误

3.3.1 错误类型

用户错误user error

程序员错误programmer error

3.3.2 错误处理

3.3.2.1 处理玩家错误

属于游戏性的一部分

3.3.2.2 处理开发者错误

开发者错误即开发人员在彼此合作构建游戏期间产生的错误。

应该避免因为某个小错误导致整个游戏都不能运行，这样会让其他人无法工作。而且游戏引擎应该让错误变得明显：在游戏中显示错误位置好于在错误日志中显示错误。当然问题严重时也别无选择。no silver bullet

3.3.2.3 处理程序员错误

略，见后

3.3.3 实现错误检测及处理

3.3.3.1 错误返回码error return code

当函数执行错误时可根据情况返回：布尔值、“不可能”值（函数合法返回值以外的值）、以及最优的返回错误枚举值。

错误返回码是函数返回值，缺点是必须要让所有函数逐一处理返回值，最后返回到能够解决问题的函数。

3.3.3.2 异常exception

异常处理的功能可以让错误或异常对象被直接抛出到可以解决问题的代码处。程序自动找到可以解决异常的函数称为堆栈辗转开解stack unwinding。实际上就是一种略过式的执行函数栈，不断向函数栈底移动，立刻析构每一个函数直到遇到对应的try-catch块。

异常处理功能的缺点是，为程序增加了额外开销，因为为了能够辗转开解，函数调用帧会变大以承载更多信息。另外，辗转开解过程也并不快速，比返回函数过程慢1-2倍。因此，游戏主机上的引擎一般不会使用异常，PC机上性能会好一些。

3.3.3.3 断言assertion

断言是一种常用的用于避免改动代码引入bug的预防机制。当断言表达式为真，一切正常，一旦假设不成立，就会暂停启动调试。而这个功能开销不大，也可以在发布时去除。

①断言实现

一个可仿例子p122/146

使用断言的建议：  
使用两种以上的断言，用于在不同的生成中使用，比如调试版本使用最多种断言，制作版本去掉断言。  
断言应该只用来捕捉严重错误，且应该总是终止整个程序，以保证引起重视。其他低程度错误使用界面提醒为宜。

# 第4章 游戏中所需的三维数学

## 4.1 在二维中解决三维问题

## 4.2 点和矢量

4.2.1 点和笛卡尔坐标Cartesian coordinate system

圆柱坐标系cylindrical coordinate system

球坐标系spherical coordinate system

4.2.2 左手坐标系left-handed LH与右手坐标系right-handed RH的比较

利手handedness：即左手系或右手系的选取

约定：三维图形程序一般以左手坐标系工作，y向上，x向右，z向内

4.2.3 矢量

位置矢量position vector：也称矢径radius vector。从原点到一个点的矢量为该点的位矢。

4.2.3.1 笛卡尔基矢量basis vector

4.2.4 矢量运算

4.2.4.1 矢量和标量的乘法

非统一缩放nonuniform scaling：即分量积

分量积component-wise product：也称为阿达马积Hadamard product，即两个矢量的分量相应乘积。使用“”表示

4.2.4.2 加法和减法

略

4.2.4.3 模magnitude

4.2.4.4 矢量运算的实际应用

平方根一般是个费时运算，尽量改用模的平方来进行比较大小等操作，尽管这意味着要将其他的值取平方。【优化】

4.2.4.5 归一化normalization和单位矢量unit vector

倒数reciprocal

4.2.4.6 法矢量normal vector

4.2.4.7 点积和投影

矢量投影

模作为点积

点积判定dot product test 是否共线collinear

其他点积的应用

4.2.4.8 叉积

略

4.2.5 点和矢量的线性插值linear interpolation

## 4.3 矩阵

特殊正交矩阵special orthogonal matrix：也称各向同性矩阵isotropic matrix、标准正交矩阵orthonormal matrix。即一个3×3矩阵所有行和列矢量均为单位矢量。

仿射矩阵affine matrix

4.3.1 矩阵乘法

有时也称为串接concatenation

4.3.2 以矩阵表示点和矢量

转置transpose

4.3.3 单位矩阵identity matrix

4.3.4 逆矩阵inverse matrix

高斯消去法Gaussian elimination

LU分解LU decomposition

4.3.5 转置矩阵

4.3.6 齐次坐标homogeneous coordinates

4.3.6.1 变换方向矢量

方向矢量的齐次坐标第四分量为0，刚好可以消去仿射矩阵的平移作用

数学概念：三维空间的纯方向矢量，在四维齐次空间是无穷远的点

4.3.7 基础变换矩阵

4.3.7.1 平移p144/168

4.3.7.2 旋转p145/169

4.3.7.3 缩放p146/170

统一缩放uniform scale：大部分游戏引擎都只支持统缩，为了计算速度

统缩和旋转串接时，次序可换

4.3.8 4×3矩阵

仿射变换矩阵第四列必然为0001，所以为了提高性能，经常略去来存储

4.3.9 坐标空间

参考系frame of reference：也称坐标系coordinate frame，游戏业内称坐标空间coordinate space

4.3.9.1 模型空间model space

定义：也称物体空间object space或局部空间local space。即绑定于模型自身的空间，原点一般置于中心或质心center of mass，对于人或动物，则一般置于脚底和地面间

由于绑定于物体，模型空间的坐标轴一般也绑定于物体自身的方向性。采用向前F，向上U，向左L或向右R（取决于利手）作为基矢的标签或别名。

标签和坐标轴的名称xyz之间的映射理论上是随意的，但是必须从一而终。

3个欧拉角Euler angle的定义：  
俯仰角pitch：绕R或L的角度  
偏航角yaw：绕U旋转  
滚动角roll：绕F

4.3.9.2 世界空间world space

定义：一个固定的坐标空间，所有物体的位置、方向和缩放都用此空间表示。原点一般置于游戏空间的中心。基矢的方向理论上任意，常用z轴或y轴向上

俯览正射视角top-down orthographic view：当z轴向上时，该视角为xy坐标

4.3.9.3观察空间view space

定义：也称摄像机空间camera space。是固定于摄像机的坐标系。原点在摄像机的位置，称为视点view point

4.3.10 基的变更change of basis

4.3.10.1 坐标空间的层次结构

世界坐标空间是坐标空间树的根，其他空间依附或相对于世界空间

4.3.10.2 建构改变基的矩阵

坐标变换矩阵的计算p151/175

4.3.10.3 从矩阵中获取单位基矢量

利用坐标变换矩阵可以直接获得原坐标系的基矢在现坐标系的表达

4.3.10.4 变换坐标系还是矢量

两种正反角度理解变换的操作含义，如果能够自由理解便于记忆，无需费心此处

4.3.11 变换法矢量

使用坐标变换矩阵的逆转置矩阵变换法向量

4.3.12 内存中存储矩阵

C++中的二维数组以最后一个维度为准在内存中连续存放。有两种方式存储矩阵，即每个一维子数组存储一个矢量，或分散对齐stride存储。大多数引擎采用第一种方法，也和行矢量表达式中的矩阵形式一致。

分散对齐的方式利于SIMD进行快速矢量矩阵乘法

## 4.4 四元数

单位长度unit quaternion的四元数，即分量平方和为1的四元数可代表三维旋转。四元数是力学问题领域的被发明的数学工具，由于三维旋转使用矩阵表达存在诸多不便，引入四元数。

四元数可以以前3个分量组成的矢量和后1一个分量组成的标量来看待。

四元数的印刷体一般写成非斜非粗字体（即正常字体），粗字体非斜代表矢量，斜体非粗代表标量。

4.4.1 把单位四元数视为三维旋转

单位四元数表示旋转操作的方法：，**a**为旋转轴方向的单位矢量，θ为旋转角度，方向采用右手法则。p156/180

4.4.2 四元数运算

须注意四元数之和必须单位化才能表示旋转，但这并不是旋转叠加的正确表达

4.4.2.1 四元数乘法

格拉斯曼积Grassmann product：四元数乘法的其中一种，结果为两乘数所表达的旋转的合成结果。注意合成结果为先进行后一个乘数的旋转，再进行前一个乘数的旋转。

格拉斯曼积定义：p157/181

4.4.2.2 共轭conjugate及逆四元数

四元数的逆inverse和矩阵的逆类似，互逆四元数乘积结果为标量1

共轭定义：矢量部分求反，标量部分不变

求逆：共轭除以模平方（三维旋转应用下，直接等于共轭，计算迅速！！！）

①积的共轭及逆四元数

四元数乘积结果的共轭与求逆与矩阵乘积的求逆（或转置）规律相同。

4.4.3 以四元数旋转矢量

公式：

优化形式（无需将结果从四元数转化为三维，且运算更简单，证明见Kavan等人论文）：

4.4.3.1 四元数的串接

四元数的相乘次序和旋转的次序相反，实际上只要按照公式逐层嵌套即可

4.4.4 等价的四元数和矩阵

矩阵和四元数之间的相互转换p160/184

4.4.5 旋转性的线性插值LERP

四元数的线性插值（和点一样直接简便）：p162/186

4.4.5.1 球面线性插值spherical linear interpolation，SLERP

四元数是四维超球hypersphere上的点，准确的插值应该是球面插值。公式上用特殊的权重代替线性插值的权重即可

4.4.5.2 SLERP还是不SLERP

SLERP显然是最优的选择，但是不应该以性能大幅下降为代价（少许代价可以接受）。如果做不到，只使用LERP也可以。【优化】

## 4.5 比较各种旋转表达方式

4.5.1 欧拉角

单轴旋转插值容易，但是任意方向轴插值不行

有万向节死锁gimbal lock

3个分量的旋转先后次序对结果有差别

欧拉角的定义依赖于对于F、U、R/L的标签定义

4.5.2 3×3矩阵

优点：无死锁，唯一性，有硬件加速的支持，反转容易，很容易扩展为仿射变换

缺点：不直观、不容易插值、存储空间占用大

4.5.3 轴角axis-angle

以单位矢量定义旋转轴，再加一个标量定义旋转角，形如[**a**,*θ*]

优点：简单直观

缺点：不能简单插值，不能直接应用于变换

①旋转矢量rotation vector

更简洁的轴角表达，是一个非归一的三维矢量，矢量表示旋转轴，矢量的模为弧度单位的转角。

4.5.4 四元数

优点：可串接，可用于变换，可插值，只需要存储3个浮点数（本身为4个，可以仅存储矢量部分，调用时用模为1反求标量部分。若原本标量部分为负，则将整个四元数取反再存储，之后求标量开平方时永远取非负即可。利用了四元数取反对于旋转无影响的特点，即反方向旋转了360-θ度）【优化】

4.5.5 SQT变换

定义：将4×4仿射变换矩阵转化为平移矢量translation，四元数quaternion，和缩放因子scale的形式

插值时平移矢量和缩放因子用LERP，四元数可用SLERP和LERP

4.5.6 对偶四元数dual quaternion

另一种可以完整表示旋转，平移和缩放的变换的数学对象，比四元数复杂一些。可用状况不详。【优化】

4.5.7 旋转和自由度degree of freedom，DOF

旋转自由度为3 = 参数个数 - 约束个数

欧拉角：3-0

四元数：4-1（模必须为1）

3×3矩阵：9-6（3个行3个列矢量都必须为单位长）

## 4.6 其他数学对象

4.6.1 直线、光线ray及线段line segment

参数方程parametric equation

矢量函数vector function

4.6.2 球体

可表示为4元素矢量，3个表示点，1个表示半径

4.6.3 平面

一般式general form

点法式point-normal form

三点式three point form

截距式intercept form

海赛正规式Hessian normal form：一般式除以法向量的模

可表示为4元素矢量，3个表示法向，1表示到原点距离

使用逆转置矩阵可以正确变换四元素矢量定义的平面，类似法向量的变换

4.6.4 轴对齐包围盒axis-aligned bounding box，AABB

用六元素矢量或两点定义的一个三维长方体，用于碰撞检测的早期淘汰。

4.6.5 定向包围盒oriented bounding box，OBB

一般是局部空间或模型坐标系内的轴对齐包围盒，包围物体

4.6.6 平截头体frustum，frusta（plural）

截断头的四角锥，也称视景体

用6个包围平面的数组表示

齐次裁剪空间homogeneous clip space

4.6.7 凸多面体区域convex polyhedral region

## 4.7 硬件加速的SIMD运算

多媒体扩展指令集multimedia extension，MMX

单指令多数据流扩展streaming SIMD extensions，SSE指令集

32位浮点数打包模式packed 32-bit floating-point mode

4.7.1 SSE寄存器

SSE的SIMD指令（汇编）和对应VC++内部函数  
·addps xmm0, xmm1 将两寄存器中4个float分别相加，结果返回xmm0中  
\_mm\_add\_ps(a, b)  
·mulps xmm0, xmm1 将两寄存器中量分别相乘  
\_mm\_mul\_ps(a, b)  
·shufps  
\_mm\_shuffle\_ps(a, b, c)

其他CPU的额外SIMD指令：  
·madd 乘并加multiply-and-add，将前两个参数相乘，再和第三个相加

FPU浮点运算器floating-point unit

SIMD寄存器、FPU寄存器和内存之间的数据交换代价很大，因此尽量减少普通浮点运算和SIMD运算代码的混用，同时可以使用SIMD寄存器存储标量，使用时将4个寄存器的值均设为该标量即可。

4.7.2 \_\_m128数据类型

visual studio提供内建SIMD数据类型

当编译器要动态分配数据结构，使用malloc()或new时，必须手动对齐16字节的\_\_m128数据类型，如使用\_\_declspec(align(16))强制声明16字节对齐

4.7.3 用SSE内部函数编码

内联汇编inline assembly：顾名思义

内部函数intrinsic：用法和函数类似，但是会被编译器转化为内联汇编代码，多数内部函数都会翻译成单个汇编语言指令，其他的则是宏（一串指令）。因此二者的使用功能相同，但是如果条件允许优选内部函数，因为可移植性强。

具体二者的区别是，汇编指令对应的是CPU，而内部函数对应的是编译器和操作系统平台，显然后者的移植性更强。

4.7.4 用SSE实现矢量对矩阵相乘

不能用矢量\_\_m128变量和矩阵的列\_\_m128变量进行直观相乘，这样结果会分散在4个SSE寄存器中，且需要将每个寄存器中的4个值相加再合并4个寄存器的结果。

应该将矢量的每个分量和矩阵的行相乘，再将4个寄存器结果直接相加。具体操作代码p178/202

矩阵和矩阵相乘可以用类似的方法延伸。

## 4.8 产生随机数

伪随机pseudo-random

完全确定性deterministic

代码中的rand()不一定会使用确定的随机数产生算法，依平台而定，除非自己定义。

4.8.1 线性同余产生器linear congruential generator，LCG

不够好

4.8.2 梅森旋转算法Mersenne Twister，MT

较LCG有很大改进

优点：长周期，均匀分布维度dimensional equidistribution很高，测试证明有效（著名的Diehard随机性测试），运行很快

4.8.3 所有之母mother of all pseudo-random number generator及Xorshift

所有之母比MT更加易于实现，且更快，由Diehard测试组发明人George Marsaglia发明

Xorshift随机性介于MT和所有之母之间，比所有之母更快，发明人相同。

**第二部分 低阶引擎系统**

# 第5章 游戏支持系统

游戏引擎中的子系统的常见设计模式是为每个子系统定义单例类singleton class，即管理器manager。设计者需要能够决定多个子系统的初始化及终止次序。

## 5.1 子系统的启动和终止

5.1.1 C++的静态初始化次序（不可用）

对于全局变量，直接使用C++类的构造和析构函数无法控制其构造次序，因为在主函数调用前，所有全局变量被自动创建。

5.1.1.1 按需建构

利用静态变量（函数内）不会早于main()之前创建，而是在第一次调用函数时创建，来满足按需建构的需求。是为Meyers单例。

代码示例p186/210 或（使用动态分配内存的变种）p187/211

优点：可以按需构造

缺点：不能控制析构次序，可能造成很高开销。

5.1.2 行之有效的简单方法

通过定义空的构造和析构函数覆盖默认构造/析构函数（等于关闭自动创建/析构），然后手动添加额外的启动/关闭函数来控制。也可以通过一个优先队列管理所有管理器。

优点：简单、明确直观、易于调试维护

缺点：完全手动

5.1.3 一些实际引擎的例子

5.1.3.1 OGRE

OGRE的一切子系统都由Ogre::Root管理，也由其明确地进行动态分配内存。影响了闭源引擎可扩展性

5.1.3.2 顽皮狗的《神秘海域：德雷克船长的宝藏》

依靠静态分配的对象而非动态内存

## 5.2 内存管理

原则：【优化】

1）动态分配内存dynamic memory allocation：减少使用系统自带的动态内存分配机制或称堆分配heap allocation（malloc()和new等），最好避免在紧凑循环中使用堆分配，其次自行定义动态内存分配器custom allocator提高效能。

2）内存访问模式memory access pattern：将数据紧凑排列更加高效

5.2.1 优化动态内存分配

C++自带的动态内存分配机制很慢，因为必须考虑到各种通用请求，且执行时必须切换至内核模式的context-switch。

定制分配器可以从预分配的内存中完成分配从而保证大部分动态内存分配过程在用户模式下，而且定制分配器的使用模式肯定被设计者高度模式化（而非通用化自由化）因此代码执行更快。

5.2.1.1 基于堆栈的分配器stack allocator

适用条件：一次性分配大量内存，分配后一段时间不会动态分配内存，之后再将该大量内存释放。

堆栈分配器接口类定义p195/219

①双端堆栈分配器double-ended stack allocator

一段内存从两端共同分配使用，一般用底堆栈载入及卸下游戏资源，顶堆栈分配临时内存块。

详细见p196/220

5.2.1.2 池分配器pool allocator

作用：用于分配大量同等尺寸的小块内存，如矩阵、迭代器、网格实例等。

机理：首先预分配一大块内存，刚好为分配元素大小的倍数。在每个空闲的小块内存中存储下一块空闲内存的地址——以链表的形式存储空闲的小块内存，是为自由列表free list。若元素大小不够存放地址，则可以用元素的索引替代（但是池子的大小也受到限制）。

5.2.1.3 含对齐功能的分配器

直接内存访问DMA，direct memory access：PS3的控制器，128字节的最大吞吐量，需要128字节对齐。

分配对齐内存块的过程：分配时，额外分配1倍所需对齐字节长度的内存，然后按照需要向上调整地址以对齐，因为额外分配一轮对齐地址，所以一定够用。

计算地址对齐偏移量的方法：p197/221 和对齐内存分配器的完整实现 p198/222

5.2.1.4 单帧和双缓冲内存分配器

①单帧分配器single-frame allocator

实现：先预留一块内存，以简单堆栈分配器管理，每一帧开始时清除所有内存内容。

②双缓冲分配器double-buffered allocator

实现：建立两个相同的单帧堆栈分配器，每一帧交替使用。这样，每一帧可以访问前一帧的内容。

5.2.2 内存碎片memory fragmentation

描述：动态堆分配的过程进行到一定程度，产生的自由内存块和占用块彼此交替连接的情况。由于分配内存一定需要足够大小且连续的内存块，因此内存碎片问题将影响后续分配。

虚拟内存virtual memory：操作系统的一种功能，详见《计算机操作系统笔记》。pc系统上已经广泛应用，因此内存碎片问题基本不存在。但虚拟内存技术会导致额外性能开销，嵌入式系统和主机游戏引擎仍然一般不会采用。

5.2.2.1 以堆栈和池分配器避免内存碎片

两种方法都可以避免内存碎片

5.2.2.2 碎片整理defragmentation及重定位relocation

碎片整理：通过将碎片区域地址上浮，占用区域地址下沉来拼接碎片

重定位：由于碎片整理导致地址更改，所以需要对指针等工具更新。C++本身没有方法搜寻指向某地址范围的指针，因此不支持直接实现的重定位，要么手动记录维护指针或采用智能指针或句柄。

智能指针smart pointer：用类封装指针产生的对象，包含原始指针raw pointer成员。可以将所有智能指针加入一个全局链表进行跟踪，移动内存时，扫描更新对应的智能指针。

句柄handle：也是一种原始指针的封装，将索引和原始指针通过句柄表联系起来，使用句柄的对象便使用索引来代替指针。更新地址时，只需更新句柄表内的指针，无需触动对象本身。实用中句柄相对智能指针更优，因为后者是一个类。

·仍然无法应用重定位：  
对于仍然无法使用重定位的情况（如使用不支持重定位的库、各种原理性原因），可以设立一个特别缓冲区为这些对象分配内存，将其和可重定位内存分离；或在影响很小的情况下，干脆容忍这些不可重定位内存。

·分摊amortize碎片整理成本：  
每一帧只进行特定次数的内存块移动操作，而不是将所有的碎片整理。另外，对于特别大的内存块，移动一块便需要大量时间，一方面可以将其拆分，让每个小块可以独立重定位（实际上，设计之初就可以避免这个问题，使用轻量级的类）；另一方面，只对细小的内存块进行整理，比如游戏对象（大的内存块释放和分配带来的碎片问题本就不大）。

5.2.3 缓存一致性

内存的结构：内存包括主系统内存和内存缓存cache

内存缓存：可以高速读写的一种特殊内存部分。但是一般硬件角度讲，缓存置于CPU芯片上，而主内存置于主板。缓存显然也使用更优更高级的硬件才能保证性能高于主内存。

读取：每次读取主内存，该内存块就会载入缓存，该块内存单位也称缓存线cache line（8-512B，依赖CPU）。若要读取的内容不在缓存中，就会发生上述情况，也称缓存命中失败cache miss。

写入：一种方式叫透写式缓存write-through cache，写入缓存时，同时写入主内存。另一种叫回写式write-back或copy-back，写入缓存后，特定情况下再写入主内存，如因各种原因不得不清除该缓存时。

5.2.3.1 一级及二级缓存

随着发展演进，缓存或内存出现分级，最低级的内存是主内存，最高级的缓存称为一级缓存L1。越高级的缓存速度越快（一般由于技术原因总空间也越小），离CPU越近（系统架构意义上），亦即某一级缓存只能和其相邻的缓存或内存直接传输数据。

load-hit-store：一种PowerPC架构上的缓存命中失败，实则为设计缺陷，导致写入数据后无法立刻读取相同地址的该数据，因为必须等待写入主内存的过程完成。

5.2.3.2 指令缓存和数据缓存

缓存又在功能上分为两种，一般硬件层面也在物理上独立分开两种缓存。

指令缓存instruction cache，I-cache：预载即将执行的机器码

数据缓存data cache，D-cache：加速主内存读写的缓存

5.2.3.3 避免缓存命中失败

数据缓存方面：将数据编排进连续的内存块中，尺寸越小越好，并且要尽量顺序访问这些数据。即紧凑、有序原则，最大程度利用一次缓存线加载。

指令缓存方面：遵照相似的原则，尽量设计程序使代码执行的顺序和代码存储的顺序一致，避免有太多的跳跃式读取。

指令缓存优化【优化】的经验准则：p207/231。注意，编译器也会提供一些链接和编译选项从而决定代码布局。

## 5.3 容器container

也称集合collection

举例：（不常见的）  
动态数组dynamic array：可变长数组，如向量vector  
双端队列double-ended queue，deque：两端可用的队列

5.3.1 容器操作

略

5.3.2 迭代器

优点：  
（1）不用迭代器可能会破坏容器类的封装  
（2）封装了迭代的复杂细节，使用方便，无需每次实现遍历算法

5.3.2.1 前置递增preincrement与后置递增postincrement【优化】

一般情况下，后置递增比前置递增多一步备份原值的过程。因此，除非绝对需要，在两者均可用的环境下，优先使用前置递增。

5.3.3 算法复杂度

略

【优化】讨论：如果插入及移除元素的速度重要，那么使用链表，如果不是，应使用数组，因为更加缓存友好。

5.3.4 建立自定义的容器类

5.3.4.1 建还是不要建

库技术报告Library Technical Report，TR1：C++标准委员会的一个草案，有可能成为未来的C++标准。

第三方库举例：  
①STL标准模板库，standard template library：优劣势分析p213/p237  
STLport扩展版的STL，多平台性较强，优于STL  
②Boost：开源的STL扩展p214/238  
③Loki：模板元编程template metaprogramming，TMP，利用C++模板功能诱使编译器做一些原本设计之外的事  
基于原则的设计policy-based design

5.3.4.2 动态数组和大块分配

动态数组具有数组（数据紧凑易于访问）和链表（易于扩充）的双重优点。但是并非绝对更优，因为存在性能代价（扩充和复制到新内存的过程）。

STL的vector变量可以通过调用自身的swap函数和自身交换，来将其缩小。

【优化】开发期间使用的动态数组在后期能够确定内存预算时，改用固定数组实现的相同接口类来替换。因此，使用上没有差别，但是完全避免了动态分配和内存碎片问题。

5.3.4.3 链表

①链表基础

略

②节点数据结构

略

③外露式表extrusive list

含义：节点数据结构和元素数据结构分离，用指针联系。STL所有容器均为外露式

优点：不用更改源类的代码即可灵活使用

④侵入式表intrusive list

含义：节点数据结构被嵌进目标元素本身（通过让元素继承节点类或将节点对象作为元素类成员）

优点：每次分配元素时就已经分配节点，避免了动态分配内存

缺点：但是不能灵活地将元素加入多个链表，因此弹性不足

⑤头尾指针：循环链表circular linked list

含义：用一个称为根节点的特殊节点包含链表的首尾指针，并加入链表内。

优点：（相较于一般带有独立首尾指针链表）所有的节点的内部指针都不会为空，若为空则必为未使用节点。

⑥单向链表singly-linked list

含义：每个节点只有后继的指针，没有前驱的指针。

缺点：不能高效地逆向遍历，删除操作需要线性复杂度。

优点：用以实现堆栈或队列时可以节省内存，同时避开固有缺陷

5.3.4.4 字典和散列表

两种实现：

二叉搜索树——按照键的大小排序和查找，键值对存于节点

散列表——略

①冲突collision：开放和闭合散列表

开放式散列open hashing或分离链式separate chaining

闭合式散列closed hashing或开放定址open addressing

②散列法

散列函数举例：

LOOKUP3，由Bob Jenkins开发

CRC32

MD5

③实现闭合散列表

## 5.4 字符串

5.4.1 字符串的使用问题

字符串处理函数是开销极大的函数，比如strlen()和strcpy()，如果能够避免必须注意。应该尽量用引用传递字符串而不是值传递。

5.4.2 字符串类

空结尾字符串null-terminated string：以’\0’为结尾的字符串

游戏引擎中一般避免使用字符串类（因为可能含有隐性性能成本），如果必须使用字符串类，须查看其性能特点：

所有字符缓冲区是否为只读？

是否使用写入时复制copy-on-write优化

是否尽量使用引用传递参数

有必要使用字符串类的情形——Path类，负责提供和文件路径有关的服务，以及跨平台转换路径格式。

路径分隔符path separator：Windows反斜线，UNIX正斜线

5.4.3 唯一标识符unique identifier

全局唯一标识符globally unique identifier，GUID

5.4.3.1字符串散列标识符string id

使用散列函数将字符串转化为一个唯一的数字标识符，其比较操作较字符串本身更加便捷。也可以使用其他的方式指定数字标识符，将字符串和字符串标识符封装或联系在一起即可。

虚幻引擎的FName类用于包装字符串和字符串标识符，但是标识符只是一个FName对象所在数组的索引，不是散列值。而字符串和FName对象的索引关系也还是靠散列建立的。

5.4.3.2 一些关于实现的主意

SID任何字符串

字符串扣留string interning：从字符串产生字符串标识符的过程

字符串扣留速度缓慢代价高昂，如果在运行时执行，最好只执行一次并记录结果。也可以在编译时进行或在预处理阶段之前完成并直接利用宏来替换字符串。后一种方法的优点是可以将字符串标识符等价视为一个常量，如果是运行时执行的函数结果，须额外处理才能视为常量。

字符串扣留代码internString()实现：p229/253

利用调试内存debug memory储存字符串——字符串本身只供人类用户或开发者使用，转换为字符串标识符之后就完全失去了在机器中使用的必要性。因此可以将字符串表置于零售版机器内存（retail memory）以外的内存中（开发机型才有），但是要注意不要让游戏代码依赖于字符串表（仅用作开发调试）。

5.4.4 本地化

字符集编码的历史沿革：

EBCDIC——最古老的字符集

ASCII——7bits 128个字符和操作信号

OEM——8bits 256个字符，ANSI的前身，但未标准化，有很多不同版本

ANSI——标准化的OEM，低128个字符相同，高128个字符被不同的code pages定义

Unicode——无限的字符集，长度和大小视具体编码方式而定

其他传统字符集编码方式：

Windows-1252——windows9x标准，适合西欧语言

ISO-8859-1或Latin-1——类似前者

Unicode的基本原理：

以前的字符集是定义字符与编码（数字）的直接对应关系。Unicode定义的是字符与code point之间的关系，后者只是一种用于记录和区分所有字符用的唯一标识码（即Unicode本义），具体机器中存储使用的编码依不同的编码方式而定。

Unicode的常见编码方式：

UTF-7，UTF-8（1-6字节），UTF-16（2字节或4字节），UCS-2（被UTF-16替代，固定占16位），UTF-32

UTF-8不代表所有字符只占8bits，只有前128个字符占1字节，后续的字符每个可能会占更多字节。

5.4.4.1 Unicode

中文名有统一码，万国码，单一码，标准万国码等。

Unicode Character Set，UCS：Unicode字符集的缩写。

①UTF-8

每个字符占1-3字节，可以向后兼容ASCII，因为所有多字节字符的首字节最高有效位必然为1，因此不会和ASCII混淆。

多字节字符集multibyte character set，MBCS：每个字符占多于1个字节的存储空间。注意该定义有时和可变长度编码混用，表示每个字符可能占1至多个字符的空间。

②UTF-16

针对基本多文种平面Basic Multilingual Plane，BMP每个字符使用16位，如果要覆盖全部，每个可能需要2或4字节。

宽字符集wide character set，WCS：相对于ASCII而言的定义，每个字符占多于1字节空间。Windows，Java，.Net里为16位，UNIX定义为32位。

③Windows下的Unicode

wchar\_t表示UTF-16字符

char表示ANSI字符

TCHAR字符集无关类型character set independent，编译时会typedef依生成模式而定类型

WCHAR和wchar\_t相同

C标准库中针对ANSI、宽字符集、多字节字符集提供了不同版本的操作函数库p232/256

④游戏机上的Unicode

Xbox 360 software development kit，XDK：完全贯彻采用WCS字符集

引擎使用何种字符集不重要，贯彻即可。

5.4.4.2 其他本地化要考虑之事

文本字符串不是唯一需要翻译的东西，还有语音、图片（不同符号的不同文化含义）、纹理文字等。

永远不要直接使用原始字符串，必须采用查找标识符的方式，便于统一管理和本地化！

逗号分隔型取值comm-separated values，CSV：一种纯文本文件，用于程序读取数据信息

## 5.5 引擎配置

5.5.1 读/写选项

①文本配置文件

如windows的INI文件，以逻辑段[xxx]分组的键值对构成

如xml文件

②压缩的二进制文件

旧时游戏主机的策略，如SNES/SFC（超级任天堂的欧美或日本名）

③windows注册表registry

一个全局选项数据库，以树形式存储，内部节点为注册表项key，类似文件夹，叶节点为键值对。实际上就是以文件管理形式存在的用于替代INI文件功能的机制，注册表实际上就是键值对数据库。每个应用程序都可以预留一个子树供自己专用。

④命令行选项

通过命令行来配置

⑤环境变量environment variable

⑥线上用户设定档online user profile

云端存储的配置文件

5.5.2 个别用户选项per-user option

即用户的个人游戏配置选项

·可存储于用户游戏进度文件中

·存储于windows中的Documents and Settings\Application Data文件夹，该文件夹专用存储个别应用程序的个别用户数据，而Documents and Settings也是区分用户的，随着windows版本区别，该文件夹可能位置不同，但是一定存在

·存储于注册表中，注册表中有区分用户的HKEY\_CURRENT\_USER表项，即注册表不仅可以用来存储全局选项也可以存储个别用户选项。

5.5.3 真实引擎中的配置管理

5.5.3.1 雷神之锤的CVAR

主控台变量console variables，CVAR

p237/261

5.5.3.2 OGRE

p237/261

5.5.3.3 《神秘海域：德雷克船长的宝藏》

①游戏内置菜单设置

p237/261

②命令行参数

③Scheme数据定义

Scheme是Lisp的方言之一，一种特殊的编程语言，被自定义的编译器编译，主要用于定义和编译二进制数据供引擎读取。

# 第6章 资源及文件系统

资源管理器resource manager：也称资产管理器asset manager或媒体管理器media manager

## 6.1 文件系统

文件系统API的典型功能：

操作文件名和路径

开、关、读、写个别文件

扫描目录下的内容

处理异步文件IO请求

6.1.1 文件名和路径

路径：卷指示符volume specifier（可选，也称根目录） + 路径成分（以路径分隔符separator分割）+ 文件名（可选）

6.1.1.1 操作系统间之区别

①分隔符与卷符

UNIX或基于其的系统使用/正斜线作为分隔符，但不出现卷指示符，因为其文件系统不像windows一样被卷分隔，而是单一主层次

Mac OS的传统是冒号：分隔符，后续支持正斜线

DOS和早期Windwos使用反斜线\分隔符，后续windwos支持正反两种斜线分隔。卷符可以使用如C:\形式，远端网络可以使用双反斜线加计算机名，如\\some-computer形式

游戏主机上的卷符使用单斜线加名称的方式，如/dev\_hddx/代表第x号硬盘。

②当前工作目录current working directory，CWD

或称present working directory，PWD

③当前工作卷current working volume

只有支持多卷操作的系统，如windows，有这个意义

6.1.1.2 绝对和相对路径

绝对路径absolute path：路径相对于根目录，但不一定必须出现卷符或文件名（可以是文件夹），若不出现卷符表示使用当前工作卷（windows）或干脆不需要（unix），但路径必须可以追溯到根目录层次，对windows而言是卷，而unix则是主层次

相对路径relative path：路径相对于文件系统层次架构中的其他目录，也不一定不可以出现卷符，但是一定省略了一部分路径表示，由当前工作目录来指定

6.1.1.3 搜寻路径search path

定义：含一串路径的字符串，路径之间以特殊字符相隔

6.1.1.4 路径API

windows中提供shlwapi.dll可用于win32平台的路径字符串处理。

若要实现跨平台游戏引擎，路径API显然需要自行定制，但是可以分别包装现有的实现。

6.1.2 基本文件I/O

C标准程序库standard C library有两组API用于读写文件：

有缓冲功能buffered：API负责管理输入输出数据缓冲，也称流输入/输出stream I/O

无缓冲功能unbuffered：由程序员自行管理缓冲，也称低阶I/O

6.1.2.1 包装还是不包装

包装操作系统原生I/O是最好的方式，也最最自由易于维护。

6.1.2.2 同步文件I/O synchronous

C标准库的两种文件I/O API都是同步的，即程序必须等待数据读写完毕才能继续运行。

6.1.3 异步文件I/O

或称为串流streaming，在后台或背景载入数据，让主程序同时运行。

异步I/O操作的代码举例（不是实现机制，需要多线程）p249/273

6.1.3.1 优先权

异步I/O操作在引擎开发领域会有优先级、时限（实时性）等要求。超出时限的请求如何处理会有不同的需求。

6.1.3.2 异步文件I/O如何工作

利用线程或独立硬件

信号量semaphore：每个请求对应一个信号量，I/O线程完成工作后通知信号量，进而唤醒主线程

协同处理器synergistic processing unit，SPU：PS3上的硬件

## 6.2 资源管理器resource manager

资源管理器2个元件：

管理离线工具链，创建并转换资产格式

管理运行时资源，执行期管理资源和内存的载入

6.2.1 离线资源管理及工具链

6.2.1.1 资产的版本控制

①解决数据量的问题

Alienbrain版本控制系统，针对极大量数据而设

顽皮狗工作室和UNIX的符号链接symbolic link

6.2.1.2 资源数据库

资产调节管道asset conditioning pipeline，ACP：用于控制资产的格式转换，以便被引擎使用，也称资源调节管道resource conditioning pipeline，RCP或工具链tool chain。

资源数据库resource database：每个资源都需要一些元数据metadata来定义其处理和转换方式，而所有的数据存于数据库

资源数据库的的功能概述：p253/277

6.2.1.3 一些成功的资源数据库设计

①虚幻3

由其万用工具UnrealEd所管理，其为游戏引擎的一部分，因此可以立刻看到资源对游戏影响

一站式购物one-stop shopping：UnrealEd的优点，其通用浏览器generic browser可以阅览和存取引擎支持的一切资源，而非将资源数据分散在不同的工具中

资产必须明确导入虚幻的资源数据库，避免了出现数据失效的情况。

②顽皮狗的《神秘海域：德雷克船长的宝藏》引擎

略p255/279

③OGRE的资源管理系统

略p258/282

④微软的XNA

略p258/282

6.2.1.4 资产调节管道

资产经历的三个处理阶段

导出器exporter：将源生格式转化为可处理格式，但不一定是最终引擎可用的

资源编译器resource compiler：可选的对数据进行适当处理的步骤

资源链接器resource linker：可选的对部分种类资源进行整合，如模型动画

①资源依赖关系及生成规则

资源之间彼此存在依赖关系，改动后也有对应的生成顺序。可自行开发，也可以使用工具make。

6.2.2 运行时资源管理

6.2.2.1 运行时资源管理器的责任

p260/284

6.2.2.2 资源文件及目录组织

资源文件一般按照树状目录形式管理，但是这只是为了方便创作者的设计。实际上，经常会将已经创作好的资源打包为单一文件，目的是为了减少载入时间。

从硬盘文件载入数据的3大开销：

寻道时间seek time：磁头找到物理媒体正确位置的时间

开启文件时间

读取数据时间

已有的资源打包格式zip的特点描述：p262/286

6.2.2.3 资源文件格式

纹理：

（开放标准格式）

便携式网络图像portable network graphics，PNG

标记图像文件格式tagged image file format，TIFF

联合图像专家小组joint photographic experts group，JPEG

视窗位图windows bitmap，BMP

Truevision Graphics Adapter，TGA

（标准纹理压缩格式）

S3TC，或DXTn或DXTC，DirectX的S3纹理压缩家族

也可以定义自设的文件格式，目的在于更好的存储引擎所需信息，而且也可以便于引擎更快速的读取和处理资源数据，标准的格式不一定为了这个目的。

6.2.2.4 资源全局统一标识符

全局唯一标识符GUID

可以是资源的系统文件路径，或散列码，或其他类似的不重复编码、虚拟路径等

6.2.2.5 资源注册表resource registry

用以记录已经加载的资源，保证不会加载超过一份副本，一般实现为字典，以GUID为键

加载资源是非常耗时的，要么使用完全加载的方式，会导致加载完成前无法游戏，或者异步加载，但更难实现。

6.2.2.6 资源生命期lifetime

载入并驻留load-and-stay-resident，LSR：无限生命期的资源

资源引用计数：对资源的使用和卸载进行计数，避免失误卸掉共享资源，造成重复加载的浪费p265/289

6.2.2.7 资源所需的内存管理

显存video RAM资源

纹理、顶点缓冲vertex buffer、索引缓冲index buffer、着色器

主内存main RAM

其他资源

①基于堆的资源分配

p266/290

②基于堆栈的资源分配

p266/290

③基于池的资源分配

数据结构的大小和特点有限制，空间利用率不能达到最高

p268/292

③资源组块分配器resource chunk allocator

用以管理和重复利用池分配器分配的未完全利用的块，由于每个块必须一起分配或卸下，因此未完全利用的块也只能分配给和原本占用该块的资源生命期相同的资源。

④分段的资源文件

文件段file section：p270/294

6.2.2.8 复合资源及引用完整性

复合资源composite resource：多个相互依赖的资源合成的资源，如3d模型，包括网格、骨骼、动画、材质、纹理。

6.2.2.9 处理资源间的交叉引用

①使用全局统一标识符做交叉引用

②指针修正表pointer fix-up table

二进制文件中对象存储位置的偏移值和存储在内存中时的指针转换（亦即序列化和反序列化）

指针修正表：序列化和反序列化中唯一需要改变数值的是指针，而指针修正表就是记录对象中哪里有指针需要更改的表。它和二进制对象一起存于文件中，实际上只是一串偏移值，每个对应一个对象中的指针成员。

③存储C++对象为二进制映像：构造函数

如果要使用C++的类OOP功能必须对每个载入的C++对象再次使用构造函数，否则只能当做POD结构处理

对已分配的内存块调用构造函数须使用placement-new语法：p274/298

④处理外部引用

交叉引用表

6.2.2.10 载入后初始化post-load initialization

也称登入资源，指在资源载入后还必须进行的处理，最后才能供引擎使用。虽然应该在载入前尽量离线处理，但是有些需求只能此时完成。另外读取数据也需要花时间，如果计算步骤简单，但是结果庞大，那么在线处理肯定更有效率，否则就需要读取大量数据进内存而不是计算出来。

最主要的载入后初始化需求（不可避免）是定义三维网格的顶点和索引值，包括一系列步骤但是必须在运行时完成。

载入后初始化也应用在兼容旧版本数据的过程，此时有一个策略——允许载入至一个临时的内存位置，更新转换至新版本后必须将其删除或移走。这样不会对主内存的分配产生影响。

# 第7章 游戏循环及实时模拟

## 7.1 渲染循环render loop

矩形失效rectangle invalidation：2d游戏采用的渲染技术，仅仅重绘屏幕中有改动的内容，降低重绘像素数目。

代码结构p277/301

## 7.2 游戏循环game loop

7.2.1 简单例子：《乒》pong

单一循环更新所有子系统的架构p278/302

## 7.3 游戏循环的架构风格

7.3.1 视窗消息泵

消息泵message pump：将来自操作系统的消息和游戏的消息进行分离处理，一般优先处理系统消息。

7.3.2 回调驱动框架

软件开发的第三方工具和套件以两种方式存在：

·程序库library：一组外部函数或类，可以被自由使用，但是一般不能篡改

·框架framework：半完成的应用软件，只能编写一部分空缺的自定义实现，一般其控制流程等已经确定

常见的游戏引擎框架就是由游戏引擎自行运行主循环，但是可由程序员实现一系列回调函数来处理对应的事件。

7.3.3 基于事件的更新

有些游戏引擎使用事件/消息系统来处理游戏循环或者更新子系统，即将所有的更新事件反复周期性的加入消息队列。

## 7.4 抽象时间线abstract timeline

7.4.1 真实时间real timeline

使用CPU的计时寄存器

7.4.2 游戏时间game timeline

良好的游戏引擎应该为很多游戏实体设定自己的时钟，比如游戏时间、摄像机时间等，目的是既可以实现暂停、慢放等时间效果，也可以暂停游戏同时继续摄像机和渲染过程，用以调试问题。

需要注意的是单步调试游戏时间和单步调试游戏程序概念不一样，前者游戏程序始终在运行。

7.4.3 局部及全局时间线p284/308

利用多个相对的时间线可以实现不同游戏元素之间的各种不同相对时间关系。

## 7.5 测量及处理时间

7.5.1 帧率及时间增量

帧时间frame time：也称时间增量或增量时间delta time，即两帧之间所经过的时间

7.5.2 从帧率到速率

感知速度perceived speed

7.5.2.1 受CPU速度影响的早期游戏

早期游戏中的速率不使用真实时间度量，而是依赖帧速率

7.5.2.2 基于经过时间的更新

帧率尖峰frame-rate spike：绝大多数游戏引擎使用过往一帧的实际时间来估计下一帧的持续时间。但这样可能会导致低帧率恶性循环，即由一次低帧率导致的后续帧所需计算量增大（因为时间增量变大，而计算跟不上），或一次低帧率导致后续帧短暂快进（计算可以跟上）。

7.5.2.3 使用移动平均

计算连续几帧的平均时间来估计下一帧的时间，可以缓和帧率尖峰问题。

7.5.2.4 调控帧率frame-rate governing

含义：限制帧速率不会超过一个值，过快则让游戏等待休眠，过慢则跳过本帧直到下一目标时间。这个机制只有在帧速率完全可以超过阈值时才可以发挥优势，否则会导致将帧率降得比原本更低（也许帧率稍降还可以运行，但调控下会导致完全跳过整帧）。

帧率稳定的好处：

·物理模拟的数值积分，以固定时间运行更加

·可以起到垂直同步作用

·游戏录播功能依靠稳定帧率才能重放游戏（帧率不定显然游戏行为不可测，更无法准确重现）

游戏录播功能：不是录像，而是重放游戏过程，理论上来说仍然是游戏过程，只是玩家的操作全部被记忆重放，而其他机制和普通游戏过程无异。

7.5.2.5 垂直消隐区间vertical blanking interval

定义：电子枪扫描一轮之后重归到屏幕上角的时间区间，根据显示技术不同而可能指代有类似特征的不同对象。

垂直消隐同步：简称垂直同步，是指将渲染帧率设定为显示器屏幕刷新率的倍数。也算是一种特殊的帧率调控。

画面撕裂tearing：在显示器刷新屏幕的过程中途交换帧缓冲，导致屏幕显示不一致的帧画面。这个现象也可能是由于没有使用双缓冲法，而渲染和扫描不同步导致扫描了已更新的和过时的图像。

7.5.3 使用高分辨率计时器测量实时

高分辨率计时器high-resolution timer：CPU所带的硬件寄存器的一种，以CPU周期为单位的时间测量装置。

查询高分辨率计时器的各个操作系统方法：p289/313

7.5.3.1 高分辨率计时器的漂移

多核处理器的多个计时器之间的漂移drift误差是存在的

7.5.4 时间单位和时钟变量

7.5.4.1 64位整数时钟

即高分辨率计时器，整数指的是以CPU周期为单位，195年才会溢出归零

7.5.4.2 32位整数时钟

可以用CPU周期量度的32位整数存储高精度但是较短的时间间隔，即少于1.4s（超出1.4s会溢出归零）

7.5.4.3 32位浮点时钟

另一种用于存储较小的持续时间的方法，以秒为单位。注意必须先用64位变量计算差值，最后再转化为32位，以免数据溢出丢失

7.5.4.4 浮点时钟的极限

浮点数的性质，参考前文基础部分，应该有准备有针对性使用这个类型，比如经常重置为零或强制用模计算周期性循环该变量的值

7.5.4.5 其他时间单位

常用的一个时间间隔单位是1/300s，其特点有：p292/316

7.5.5 应付断点

在断点调试时，CPU时钟仍然在运行，下一次继续运行游戏可能会使得时间增量非常巨大。可以通过检查时间增量使其不超过一个设定的阈值。

7.5.6 一个简单的时钟类

p293/317

## 7.6 多处理器的游戏循环

7.6.1 多处理器游戏机的架构

7.6.1.1 Xbox 360

Xbox 360硬件架构简介p297/321

统一内存架构unified memory architecture，UMA：所有处理器共用一块主内存

7.6.1.2 PlayStation 3

Cell Broadband Engine, CBE架构：p297/321

Power处理部件Power Processing Unit，PPU：PS3中的主CPU名称，实为PowerPC

协同处理部件Synergistic Processing Unit，SPU：PS3中的6个副处理器

局部存储local store，LS：SPU的专属一级缓存

直接内存访问控制器direct memory access：负责管理SPU读取内存的部件

7.6.2 SIMD

略

7.6.3 分叉及汇合fork/join

一种多处理器程序架构（不是多线程），将数个选定的CPU使用密集阶段并行化至多个核，是为分叉；然后将结果汇总到主线程，是为汇合。

7.6.4 每个子系统运行于独立线程

另一种多处理器多线程程序架构，如题目所示。一般分离的线程有动画、物理和渲染，主线程仍然运行主游戏循环。

SPURS：PS3的多线程库，提供两种运行多线程程序的方法，任务模型task model和作业模型job model。任务模型类似于子系统独立线程模式。

7.6.5 作业模型

另一种多处理器程序架构，将子系统计算量布置为一系列作业（数据和操作该数据的代码），并且置于队列中等待，一旦有任何处理器闲置，便会处理一份作业。主线程和主循环仍然独立存在。

7.6.6 异步程序设计

实则为多线程来实现的

举例：p302/326

## 7.7 网络多人游戏循环

7.7.1 主从式模型client-server model

服务器：负责运行大部分游戏逻辑

客户端：只充当渲染引擎和音频引擎，以及本地操作数据读取和网络代码

专属服务模式dedicated server mode：服务器独立

客户端于服务器之上模式client-on-top-of-server mode：服务器和客户端共体

专属服务模式下，客户端和服务器运行于不同的进程或线程。客户端于服务器之上模式下，客户端和服务器可置于同一个线程和循环中。

服务器和客户端的更新频率不一样，若两者在同一个循环中更新，那么应该以较快速的一方进行循环，而为慢速的一方设立计时器，当计时器到达时在当次循环中执行慢速一方的逻辑。

7.7.2 点对点模型peer-to-peer

略

7.7.3 案例分析：《雷神之锤II》

p306

# 第8章 人体学接口设备（HID）

human interface device，HID：

摇杆joystick

手柄joypad

轨迹球trackball

## 8.1 各种人体学接口设备

略

## 8.2 人体学接口设备的接口技术

8.2.1 轮询poll

含义：定期询问硬件来读取输入，通常每个主游戏循环一次

举例：Xbox 360，XInput API

8.2.2 中断hardware interrupt

含义：硬件产生信号，让CPU暂停，并执行中断服务程序interrupt service routine，ISR

举例：鼠标

8.2.3 无线设备

虽然通过蓝牙协议进行包装，但是用法和轮询类似。

## 8.3 输入类型

8.3.1 数字式按钮digital button

举例：键盘、鼠标、手柄的按钮等

有时可以将所有按键的状态整合为一个整数值，每个位对应一个按钮的两种状态

8.3.2 模拟式轴及按钮analog input

含义：可获取的数据不是一个固定状态，而是一个范围内的值。通常表示某些轴的转角（对应于操作介质的状态），因此也成为模拟式轴analog aixs

举例：手柄的摇杆位置由XY坐标两个数值模拟，手柄的左右扳机Left/RightTrigger也有按压程度的度量

8.3.3 相对性轴

有一些模拟式轴的输入值不是绝对性的，一般代表一种相对性的移动。

举例：鼠标和其滚轮、轨迹球

8.3.4 加速计accelerometer

即能够感知受力或加速度的输入设备，可以感知3维空间中的3个方向上的加速度，并产生3个输入值。一般一个读数由一个加速计测量产生。

举例：Wii遥控器和其他体感设备

8.3.5 以Wii遥控器或Sixaxis做三维定向

如何使用加速计传感估算设备的定向（只是估算，若存在抖动则无效）？p315/339

8.3.6 摄像机

红外线infrared，IR传感器

Wii遥控器红外定向功能简介p316/340

也许也是VR定向的技术雏形

## 8.4 输出类型

8.4.1 震动反馈vibration feedback

rumble或vibrate：手柄震动，一种触觉技术haptic technology

8.4.2 力反馈force-feedback

致动器actuator：马达驱动的一种阻力装置，以给玩家产生阻力效果，常用于赛车游戏的方向盘

8.4.3 音频

扬声器、耳机、网络语音

8.4.4 其他输入/输出

最新：

姿势界面gesture interface

思想控制设备thought-controlled device

## 8.5 游戏引擎的人体学接口设备系统

按钮映射表button-mapping table：用来将原始按钮输入转化为游戏逻辑的动作，不同的映射表就代表着不同的按钮与游戏操作之间的映射。这种封装隔离使得玩家可以一定程度选择不同的按钮定义的操作方式。

8.5.1 典型需求

p318/342

8.5.2 死区dead zone

定义：由于模拟设备输入存在噪声，将静置情况下的0输入值周围的小范围区域设定为不予处理的死区来忽视噪声，同时死区不能过大影响操作。

8.5.3 模拟信号过滤

①离散低通滤波器discretized low pass filter

p319/343

②窗口化移动平均 moving average

p320/344

8.5.4 输入事件检测

8.5.4.1 按下和释放按钮

使用按位异或与按位和来比较当前状态来确定按钮事件以及何事件

代码：p322/346

8.5.4.2 弦chord

定义：一组按钮或玩家操作的集合，被赋予一个独立的指令或行为。如同时按下两个以上按键等。

问题：

·要区分弦的动作和组成弦的每个单独按键的动作

·人的反应往往无法在同时刻触发弦

解决该问题的几种方案：p323/347

8.5.4.3 序列和手势检测sequence and gesture detection

定义：类似于弦操作的另一种组合式操作，玩家按照一定顺序、节奏和快慢操作输入，该输入被赋予一个不同于每个组成操作的独立的指令或行为。

几个例子：

①迅速连打按钮

代码示例：p324/348

②多按钮序列

代码示例：p325/349

③旋转摇杆

对旋转摇杆的侦测和多按钮序列类似，只要将摇杆的位置看成是按钮值即可。

8.5.5 多HID和多玩家的管理

未详述技术细节，只是表面描述

8.5.6 跨平台HID系统p328/352

将游戏代码和硬件细节或平台细节之间加入隔离机制。可以是将按键标识符泛化，不同平台的操作经过处理后都会对应单一功能的泛化标识符，进而在游戏代码逻辑中不必区分。只是处理输入的部分需要针对性处理。

8.5.7 输入的重新映射p330/354

可以将各种类型的输入分别归一化，这样各种输入都可以产生同样的映射结果。

8.5.8 上下文相关控制context-sensitive

由游戏状态决定的不同的环境下，相同的输入有不同的操作含义。

控制拥有权control ownership

8.5.9 禁用输入

最好是在玩家或摄像机对象中设置忽略输入，当然也可以用掩码清零HID读取的数值，但是其他对象就无法使用输入值了。

## 8.6 人体学接口设备使用实践

技术要求清单technical requirements checklist，TRC

# 第9章 调试及开发工具

## 9.1 日志及跟踪

对于很多复杂的bug，使用打印语句在关键位置节点用以检测bug是非常重要的手段。完全手动跟踪bug只限于逻辑相对简单和bug比较稳定出现的情况。

9.1.1 使用OutputDebugString()做格式化输出

该函数本身不具有格式化输出的能力，但是可以通过包装来自行实现

例子：p334/358

9.1.2 冗长级别verbosity level

含义：通过设定冗长级别来全局控制打印信息的函数是否被调用，而无需手动注释掉

例子：p335/359

9.1.3 频道channel

含义：将游戏中的不同子系统给予不同的频道，这样就可以分类管理、输出或过滤掉不同子系统的信息，便于阅读

实现思路：p336/360

9.1.4 把输出同时抄写至日志文件

顾名思义，日志文件中可以永久记录调试输出产生的内容，最好将所有输出都写入文件（可以按照频道分配不同的文件）。另外应该设置选项——是否每次调试输出后都立刻对日志文件进行清空缓冲刷新，这个选项优点在于若游戏崩溃可以在文件中看到最后一次输出，但总体上消耗了性能，应视情况而选用。

9.1.5 崩溃报告

崩溃时放出特别的文本输出或日志文件，很多操作系统本身就有这样的顶层异常处理函数top-level exception handler，可以直接利用。

## 9.2 调试用的绘图功能

调试绘图debug drawing：用以起到调试作用的绘图功能（包括绘制文字），绘制的图形并不会存在于发布版

9.2.1 调试绘图API

功能简述：p340/364

绘制图元primitive，包括控制其颜色，尺寸等

可将图元或文字绘制于3d空间作为注释或2d屏幕上作为类似HUD

设置是否使用深度测试depth testing

为图元设置生命期，和一般游戏物体一样

可任何时刻调用API，应将请求加至队列，因为最终只有在渲染阶段才会真正绘制图形

## 9.3 游戏内置菜单in-game menu

含义：也称开发菜单，用以设定游戏内部机制、参数等作用的菜单，但是是开发和调试工具，不是面向玩家的游戏菜单。仍然可以采用用户输入操作（不常用的输入组合）来唤出菜单。至于是否在发布版本取消，另行决定。

## 9.4 游戏内置主控台in-game console

含义：另一种开发调试工具系统，可以替代开发菜单或并存

## 9.5 调试用摄像机和游戏暂停

唤出内置菜单或主控台进行调试时，最好保证两项功能：

可以自由使用摄像机

可以暂停、恢复或单步游戏

慢/快动作模式slow/fast motion mode

## 9.6 作弊cheat

常见作弊功能——作弊是用于开发和调试的功能：

自由飞行，无视碰撞

无敌invincible

无限弹药

获得任何物品

## 9.7 屏幕截图及录像

截图的一般原理：利用图形API将帧缓冲从显存移至主内存，在主内存中可以扫描转换为预期格式。

功能简述：p349/373

## 9.8 游戏内置性能剖析in-game profiling tool

用于弥补第三方剖析软件的不足，第三方剖析软件仍然是性能剖析的主力工具。程序员标记一些代码段落，剖析器会为段落计时。

9.8.1 层阶式剖析

命令式语言imperative language

9.8.1.1 以层阶式量度执行时间

包含执行时间inclusive：一个函数包含其调用函数的总执行时间

排他执行时间exclusive：一个函数本身的执行时间，不包括其调用的函数

·实现原理1（简单粗粒度，不考虑调用层阶，即标记段落彼此不覆盖）：p354/378

将程序代码段落以大括号包围，在代码段首创建一个特殊的对象并在其构造函数中开启一个计时器，当代码段结束之时，特殊对象会自动被析构，只需在析构函数中加入返回时间结果即可。整个创建对象的过程，或称为标记过程，可以用宏来包装实现。

·实现原理2（考虑层阶，但是限于每个函数在层阶中只有一个父函数）：p355/379

将函数的标记提前声明父子关系。之所以有这么多限制，是因为在代码中添加标记和函数本身执行的栈结构无法形成完美的映射关系。

·调用次数计数原理：p356/379

9.8.2 导出至Excel

将性能剖析数据导出至文本文件，最好是CSV样式，可被Excel读取

## 9.9 游戏内置的内存统计和泄漏检测

内存追踪工具memory tracking tool

内存泄漏memory leak

内存分配钩子memory allocation hook：第三方库可能会提供的给予用户自定义内存分配的接口

·基本原理：包装内存分配指令或用自定义内存分配器来分配内存，从而在分配器中跟踪内存的分配使用情况。

·常见问题：p357/381

**第三部分 图形及动画**

# 第10章 渲染引擎

## 10.1 采用深度缓冲的三角形光栅化基础

·渲染方程rendering equation，也称着色方程shading equation

即计算每个像素颜色和强度的过程。渲染和着色两个词意思有时相似但是仍有本质不同，渲染即生成图像像素的全部过程，无论图像到底是什么类型，是否包含光照计算等；着色一般指光照效果和其他高级视觉效果的计算实现。

·渲染技术领域分为为了平衡视觉流畅和视觉效果的实时渲染技术（游戏）和单纯为了视觉效果的非实时渲染技术（做图片或电影视频）。

10.1.1 场景描述

10.1.1.1 高端渲染软件所用的表示法

一言以蔽之，高端渲染或非实时渲染采用的是光滑曲面或曲线，而实时渲染只能用多边形网格近似。严格来说，光栅化显示后所有的像素都是一种近似，但实时渲染的构图本身就已经在近似了。

10.1.1.2 三角形网格

分段线性逼近piecewise linear approximation

游戏行业采用三角形网格的原因：p364/388

镶嵌tessellation：把表面分割为一组离散多边形的过程，一般是三角形或四边形quadrilateral，而三角化triangulation特指用三角形

·根据摄像机和物体的距离不同进行不同程度的镶嵌，始终保证一定的三角形对像素比例。实现方法可以是通过细分曲面技术预先定义好不同层次细节level-of-detail，LOD的网格，在摄像机移动过程中切换，这仍是静态镶嵌的一种。

·动态镶嵌dynamic tessellation：一般只用于可扩展的网格上，如水面和地形。动态地在距离相机近处细分网格，远处粗化网格。

·渐进网格progressive mesh：一种动态生成全局镶嵌的技术，可以视为是第一种静态镶嵌技术的自动化。

10.1.1.3 构造三角形网格

①缠绕顺序winding order

②三角形表triangle list

使用顶点的三角形顺序描述定义网格

③索引化三角形表indexed triangle list

顶点缓冲vertex buffer：用于存放顶点数据，每个顶点存一次，也叫顶点数组vertex array

索引缓冲index buffer：用于以顶点索引形式存放三角形表，也叫索引数组index array

④三角形带及三角形扇triangle strip & triangle fan

p368/392

⑤顶点缓存优化vertex cache optimization

缓存一致性cache coherency：三角形扇和带的存储顺序和读取顺序相当，有利于加速数据的读写（见前文内容）

三角形带和扇可以不需要索引缓冲，但是也可以使用索引缓冲，也仍可以降低内存用量

顶点缓存优化器：以独特算法对顶点顺序进行处理以达到提升缓存一致性的作用

10.1.1.4 模型空间model space

也称局部local或物体object空间

质心centroid

10.1.1.5 世界空间及网格实例化

网格实例mesh instance

模型至世界矩阵model-to-world matrix或世界矩阵world matrix

10.1.2 描述表面的视觉性质

10.1.2.1 光和颜色的理论

光谱颜色spectral color与光谱图spectral plot

①光和物体的交互作用

·传播transmit折射refract反射reflect吸收absorb（常见）

·衍射diffract极化polarization（被渲染器经常忽略）

·干涉interference（需借助特殊模型）

反照率albedo：光反射和被物体吸收的比例

次表面散射subsurface scattering，SSS：用于动物皮肤、蜡、大理石等物质显示柔和特性

②颜色空间和颜色模型

颜色通道color channel：RGB分量

颜色格式color format：颜色存储的占用空间和编码方式

RGB888, RGBA8888, RGB565, RGBA5551等

每像素位数bits per pixel，BPP：色深

调色板格式paletted format

③不透明度和alpha通道

10.1.2.2 顶点属性vertex attribute

顶点可以或必须具备的属性举例：p374/398

·位置position

·法矢量vertex normal：顶点处的法向量，用于计算该顶点处的dynamic lighting动态光照

·切线矢量vertex tangent

·副切线矢量vertex bitangent或副法矢量binormal（应优先使用前者作正式名称）：tangent space切线空间，是基于网格单元的local空间，由顶点法向量、切线矢量和副切线矢量组成。和顶点法向量类似，切线空间的基向量以网格单元为单位计算，但相邻网格单元的计算结果往往取平均作为共享顶点的属性。

·漫反射颜色diffuse color

·镜面颜色specular color

·纹理坐标texture coordinates：纹理贴图texture mapping

·蒙皮权重skinning weight：顶点受各个骨骼关节影响的情况

10.1.2.3 顶点格式vertex format

即顶点属性在代码中的数据结构格式，举例：p375/399

不同顶点格式的管理

10.1.2.4 属性插值

透视校正插值perspective-correct interpolation

①顶点法线及圆滑化

略

10.1.2.5 纹理

大部分的图形硬件要求纹理位图尺寸必须为2的幂且为正方形。其他的图形硬件可能会有不同程度的宽容。

①纹理种类

·漫反射贴图diffuse map，也称反照率贴图albedo map

存储表面的漫反射颜色

·法线贴图normal map

存储以RGB值编码后的法矢量

·光泽贴图gloss map

描述表面的光泽程度

·环境贴图environment map

含周围环境的图像以渲染反射效果

·光照贴图light map

见光源模型-静态光照

纹理贴图可以存储任何计算着色时所需信息，甚至非图形表格信息。

②纹理坐标

一般左下0,0，右上1,1。Direct3D中左上0,0，右下1,1

③纹理寻址模式texture addressing mode

OpenGL中称为纹理缠绕模式texture wrap mode

·缠绕模式wrap mode：对于任意正整数j，k，(ju,kv)=(u,v)。OpenGL中称为重复模式repeat mode。Direct3D中的纹理缠绕texture wrapping又有其他含义。

·镜像模式mirror mode：

·截取模式clamp mode：超过的坐标自动截取至最近边界

·边缘颜色模式border color mode：超过的坐标全设为一个颜色

对于u，v两个坐标，一般可以分别设定不同的寻址模式。

④纹理格式

压缩纹理compressed texture：不同于用二维像素矩阵存储纹理，压缩一般会采用调色板color palette+索引模式来存储纹理——将一组像素块视为一个整体存于调色板，使用索引来表达图片。显然压缩效果、图片复杂程度、为了压缩而舍弃的真实度彼此间有平衡关系。

压缩格式：

DXT或BC系列：BC1/2/3/…/7 block compression。也称S3 Texture Compression，S3TC

使用压缩格式不仅仅占用较少内存，而且因为其存储方式使得其内存存取模式更加缓存友好——一次性存储周围数个像素，便于一些计算的就地执行，相比于线性扫描的纹理存取不必反复跳跃式查询内存。

⑤纹素密度及多级渐远纹理

纹素密度texel density：纹素和像素对应关系中二者数目的比值。该值理想下应为1，越低纹理会出现锯齿，越高会出现像素颜色的浮动和闪烁，以及莫列波纹moiré banding pattern

多级渐远纹理mipmapping：一般相邻两级之间的像素尺寸相差一倍，即宽高各为上一级的一半

渐远纹理级数mip level：图形硬件一般会逐像素或逐采样计算渐远纹理级数，而不是每个三角形甚至每个网格

⑥世界空间纹素密度world space texel density

定义：即纹素数量在世界空间单位面积内的数值，该数值的绝对值没有太大意义，有意义的是其数值无论如何计算在全空间中应该保持恒定。这意味着每个模型纹理的细节程度差不多都相同

⑦纹理过滤texture filtering

定义：对像素的纹理采样通常采样会多于一个纹素，并将结果按一定规则混合得出最终颜色

技术种类：p383/407

最邻近nearest neighbor

双线性bilinear

三线性trilinear

各向异性anisotropic

10.1.2.6 材质material

定义：是网格视觉特性的完整描述，即包含纹理设置、着色器设置、图形加速硬件参数等，但是不包括顶点属性即网格本身。网格-材质对，也称渲染包render packet，包含所有需要渲染物体的信息。

子网格submesh：网格一般会切割成子部分，每个子网格对应一个材质，来体现实际中一个完整模型不同的部分——如人的各个表面器官和衣物等特征。

10.1.3 光照基础

康乃尔盒子Cornell box

着色shading：也作浓淡处理，即光照加上其他视觉效果的泛称，严格来说着色是渲染的一部分。

10.1.3.1 局部及全局光照模型

光传输模型light transport model

·局部光照模型local illumination model：只考虑直接光照，一般间接光照只用一个环境光简单估计

·全局光照模型global：考虑到间接光照

针对某种视觉现象的：阴影，镜像

模拟多种光学现象的：光线追踪ray tracing，辐射度算法radiosity

10.1.3.2 Phong氏光照模型

也译作冯氏光照模型，是一种局部光照模型。注意还有Phong shading着色，是另一回事。

哑光表面matte surface：几乎不会产生反射行为的表面，如木块或布料。只用漫反射模拟即可。

光滑度glossiness

①Blinn-Phong

略

②BRDF图表

双向反射分布函数bidirectional reflection distribution function，BRDF：沿视线方向的向外辐射与入射光线的进入辐射之比

10.1.3.3 光源模型

①静态光照

光照贴图light map：也称为顶点光照烘焙baking。存有静态光照条件下（静态光源和物体）物体表面光照预计算结果的纹理。首先能够使用静态光照贴图说明至少漫反射与视线方向无关（如Phong模型），另外光照贴图一般比漫反射纹理分辨率较低，且更易压缩，因为它可以不含有物体漫反射颜色的绝对信息，只有漫反射强度的相对信息。

②环境光ambient light

③平行光directional light

④点光point light/全向光omni-directional light

⑤聚光spot light

⑥面积光area light

真实光源都为面积光

本影umbra和半影penumbra

实现方法：投射多个阴影再混合结果，或将阴影边缘模糊化

⑦发光物体

放射光贴图emissive texture map：用来替代发光物体原本的漫反射纹理，此纹理颜色永远以完全强度发射。

发光物体使用多种技术渲染（根据物体本身有可选项）：

·放射光贴图

·光源

·半透明网格模拟光形成的几何体（如手电光柱）——实际上是一种空气和灰尘散射

·镜头光晕lens flare：由于强光在镜头内反射所形成，一般表现为排列在一条直线上的一串大小不一、颜色不一的圆形光圈，以及光源周围向特定几个方向辐射至远处的光带

·敷霜效果bloom：成因有可能是不完美对焦或强光在相邻感光元件溢出，一般表现为光源或强光处周围附近区域形成的模糊光晕。

·投射纹理（如聚光灯照射在物体上形成的环状光图样，这和聚光灯的结构有关）

10.1.4 虚拟摄像机

成像矩形imaging rectangle

10.1.4.1 观察空间view space

也称摄像机空间camera space，原点为摄像机焦点，准确的说就是摄像机的位置。

利手p393/417

观察至世界矩阵view-to-world matrix：用于为摄像机在世界中定位的矩阵，充当了摄像机自身的模型矩阵。但是摄像机就是观察点，这个矩阵（实为其逆矩阵，也称观察矩阵）的真正作用是将世界中的物体变换至观察空间。

OpenGL中会预先将模型空间和观察空间的变换矩阵串接，称为模型观察矩阵model-view matrix

10.1.4.2 投影projection

透视收缩perspective foreshortening：透视投影的近大远小效果术语

10.1.4.3 观察体积及平截头体

观察体积view volume：摄像机能看到的空间范围

平截头体frustum：透视投影的观察体

10.1.4.4 投影及齐次裁剪空间homogeneous clip space

OpenGL中z轴由-1到1，DirectX中为0到1。其他两个坐标轴均为-1到1。齐次裁剪空间通常都是左手系。

①透视投影（准确的应为透视变换）

OpenGL和DirectX中的透视变换矩阵p397/421

准确的说，透视变换的过程是将物体进行透视投影（2维）并为其赋予一个伪深度（第3维），之后对深度之外的两个坐标进行伸缩平移，最后变成齐次裁剪空间的标准观察体积或正则视景体CVV。

②除以Z（透视除法或透视收缩）

③透视正确的顶点属性插值

透视校正插值perspective-correct interpolation

由于透视投影造成的近大远小扭曲效果，直接对属性进行线性插值已不再适合

④正射投影p398/422

10.1.4.5 屏幕空间及长宽比

长宽比aspect ratio：宽度和高度之比

屏幕映射screen mapping：将齐次裁剪空间缩放平移至屏幕空间

10.1.4.6 帧缓冲frame buffer

双缓冲法double buffering：保持两个缓冲，扫描一个时，渲染另一个，在垂直消隐区间互换缓冲。

三缓冲法triple buffering：相当于给双缓冲法多提供了一个帧缓冲区，如果图形硬件性能远超过显示硬件，这么做便有一定的好处，不过也并不必要。

①渲染目标render target

定义：供渲染引擎绘画图形和储存中间渲染结果的缓冲，包括深度depth、模板stencil等

10.1.4.7 三角形光栅化及片段

片段fragment：见计算图形学

①抗锯齿antialiasing

也作边缘柔化、抗混叠（信号学中的术语）

见计算图形学

由于延迟渲染不能使用MSAA，因此诞生了新的AA技术。如MLAA，DLAA，FXAA，SRAA

10.1.4.8 遮挡occlusion及深度缓冲depth buffer

或称z缓冲z-buffer

①深度冲突z-fighting及W缓冲

w缓冲使用的是顶点原坐标z进行深度测试，而z缓冲使用的是伪深度（正比于1/z）进行深度测试。z缓冲有近处精度大，远处精度小的特点，远处可能会形成深度冲突，w缓冲处处精度相同。然而z缓冲的伪深度在透视变换后是可以线性插值的，计算较快，但是w缓冲的z坐标是原始坐标，不能直接插值（先倒数，再插值，再倒数），计算较慢。

## 10.2 渲染管道

并行化parallelization：渲染管道应该是并行化的（虽然表观上是串联的），指的是每个阶段都在同时运行处理上一个阶段的结果。

吞吐量throughput：总体每秒可产生的数据量

潜伏期latency：单个数据需要花多少时间才能走完整个管道

10.2.1 渲染管道概观

（1）工具阶段（脱机）：制作模型与材质

（2）资产调节阶段（脱机）：

（3）应用程序阶段（CPU）：调度管理GPU行为

（4）几何阶段（GPU）：几何计算过程，以及光照计算等

（5）光栅化阶段（GPU）：着色计算和深度测试等测试

10.2.1.1渲染管道如何变换数据

p405/429

10.2.1.2 管道的实现

略

10.2.2 工具阶段

图形化着色语言graphical shading language：如node-based的一些渲染器所提供的设置材质与着色器的方式。

10.2.3 资产调节阶段

预计算辐射传输precomputed radiance transfer，PRT

10.2.4 GPU简史

硬接线hard-wired：对立的是可编程programmable，嵌入式系统的一种类型，可能只提供少量可配置的可选项，整体逻辑无法修改。产生的管道叫固定功能管道fixed-function pipeline，也称硬件变换及光照hardware transform and lighting，hardware T& L

图形处理器graphics processing unit，GPU

10.2.5 GPU管道

渲染管道的GPU部分：p410/434

10.2.5.1 顶点着色器

完成变换、光照计算的部分

程序式动画procedural animation：通过顶点着色器修改顶点位置来产生的。

顶点纹理拾取vertex texture fetch，VTF：顶点着色器可以完全存取纹理数据，尤其是以纹理形式存储的非颜色信息。这个功能在早期的GPU中不存在。

10.2.5.2 几何着色器

可选，处理以齐次裁剪空间表示的所有的整个图元，可以创建、修改和删除图元，以实现各种高级效果。

应用：

·阴影提及拉伸shadow volume extrusion

·渲染立方体贴图cube map

·在网格的轮廓边拉伸毛发的鳍fur fin

·从点数据生成粒子四边形

·动态镶嵌

·把线段以分形细分fractal subdivision模拟闪电效果

·布料模拟

10.2.5.3 流输出stream out

允许将几何着色器的输出写回内存，重新从管道之始处理。尤其在几何着色器增添了图元的情况下，可以送回顶点着色器重新渲染。没有几何着色器和流输出功能，一般编辑图元的功能只能由CPU负责

举例：头发物理效果与渲染p410/434

10.2.5.4 裁剪clipping

略

10.2.5.5 屏幕映射screen mapping

略

10.2.5.6 三角形建立triangle setup

略

10.2.5.7 三角形遍历triangle traversal（光栅化）

略

10.2.5.8 提前深度测试early depth test

也称提前z测试early z-test。这是一个可选的优化功能，可以提前舍弃因被遮挡而无存在意义的片段，进而节省像素着色器或片段着色器的性能。但是若在像素着色器中更改深度值，或增加alpha测试，都无法再使用这个功能。

10.2.5.9 像素着色器（或片段着色器）

为像素计算颜色和光照，对纹理采样以及其他计算。

输入为每片段属性，输出为颜色矢量

像素着色器是主要的光照计算场所（一般采用世界坐标系计算光照，因此一般不需要顶点着色器的输出——齐次裁剪空间坐标），因为它可以获取纹理。然而随着技术进步和程序变得复杂多样化，顶点着色器也可以获取纹理并且进行顶点的光照着色计算，并将顶点的颜色输出。但是可以肯定，像素的着色仍然要在像素着色器中进行，而且顶点的颜色在像素着色器中是一个对片段插值的结果，并不是该片段的颜色。如何使用和计算都是视算法而定的。

10.2.5.10 合并merge或混合blending/光栅运算阶段raster operations stage，ROP

负责片段测试（深度、alpha、模板stencil）

合并和混合是指通过测试的片段与帧缓冲中的颜色进行alpha混合。

alpha混合计算：p412/436，注意权重系数的位置

通用混合函数的形式：p413/437

·如何渲染半透明物体？

①开启深度写入并先渲染不透明物体，再以物体为单位对半透明物体从后往前排序，然后关掉深度写入去渲染半透明物体。（**理论上来说，不透明物体由近及远渲染性能更好，半透明物体由远及近渲染结果才正确**）

②次序无关透明order-independent transparency，OIT技术：包括深度剥离depth peeling、alpha至覆盖掩码转换alpha to coverage，片段链表fragment linked list等。

10.2.6 可编程着色器

着色器编程语言：

Cg——C for graphics，Nvidia

HLSL——高级着色语言，high-level shading language，微软的Cg

GLSL——OpenGL shading language

统一着色器架构：DirectX中称为着色器模型4.0，将三种主要的着色器使用差不多的指令集和能力。

10.2.6.1 内存访问

GPU的数据处理管道是高度已配置的，所以无法自由地访问其内存。一般有两个间接办法

①着色器寄存器

所有GPU寄存器是128位SIMD格式

寄存器类型：

·输入寄存器input register

·常数寄存器constant register：包含计算所需但是输入不包含的数据

·临时寄存器temporary register：中间结果

·输出寄存器output register：

变换后顶点缓存post-transform vertex cache：用于暂时存储顶点着色器结果的缓存

②纹理

着色器可以直接读取纹理贴图，但是不能直接写数据进纹理，只能先渲染至帧缓冲然后保存为纹理贴图，是为渲染到纹理render to texture，RTT

10.2.6.2 高级着色器语言的语法入门

整体简介p415/439

纹理采样器texture sampler

10.2.6.3 效果文件effect file

p417/441

用来定义GPU渲染流程的说明性文件，并非是着色器编程代码，而是以着色器为模块的角度定义渲染管道的文件。其作用是实现某些需要多次使用各种着色器的技术，和定义次级配置效果供低端硬件实现。

10.2.6.4 延伸阅读

略

10.2.7 应用程序阶段application stage

包含3个部分的子内容，其实也是渲染阶段游戏引擎作为一个程序的主体工作

10.2.7.1 可见性判断visibility determination

定义：建立可见网格实例表的过程，剔除cull不会对最终影像有贡献的物体

①平截头体剔除frustum culling

包围体积bounding volume：用来代表物体进行剔除测试的体积，一般为包围网格全部顶点的球体

常用的平截头体剔除方法：p418/442.这是一种对结果保守的方法，只能保证一定不会剔除平截头体内或相交的物体

平截头体剔除法在大型场景中性能较差（要剔除的太多），可以用场景图数据结构优化。

②遮挡剔除occlusion culling及潜在可见集potentially visible set，PVS

潜在可见集：顾名思义，在渲染之前预计算场景中对摄像机可能可见的物体，有一些物体可能最终仍不可见，但是不会排除真正可见的物体。是为保守的conservative方法

实现方法1：将场景切割成区域，每个区域维护一个摄像机在该区域可以看到的其他区域的表。

③入口portal

另一种剔除物体的方法，适用于渲染封闭的室内环境，不同房间以少量门窗联通。入口就是以门窗为视平面和视点形成的视角锥体，然后用这个锥体做类似平截头体剔除的检测。

④遮挡体积occlusion volume（反入口anti-portal）

概念上与实现上和入口相对，但是功能类似——完全排除不影响渲染结果的物体。

10.2.7.2 提交图元

①渲染状态render state

可配置参数例子：p420/444

也称硬件状态hardware state。是指GPU管道内所有可配置参数。渲染状态是和网格-材质对对应的，即不同的子网格和材质就有一种不同的渲染状态。

②状态泄露state leak

忘记设置某个参数状态，而沿用上一个参数

③GPU命令表command list

应用程序使用命令表的方式和GPU进行沟通。

【优化】有时手工建立GPU命令表，或调用底层的渲染API，可能比使用包装好的API函数性能更好，而且便于在多核机器上利用多线程。

10.2.7.3 几何排序

按照渲染状态（即材质）来为物体排序，可以降低更改渲染状态的次数，进而在此方面提高性能。但是从另一个角度来说，按照物体距离摄像机的位置由近及远地排序渲染，可以利用深度测试提前舍弃不必要的片段，反之需要在同一像素上绘制多次（覆绘overdraw）

①深度预渲染步骤z prepass是救星

对于不透明物体，可以采用渲染两次的方式——第一次按照由近及远顺序只产生深度缓冲，第二次按照材质排序来正式渲染场景（应该需要开启提前深度检测）

对于半透明物体，只能按照由远及近的顺序渲染，但是可以采用纹理图谱texture atlas技术把多个材质合成为一个，在一定程度上降低更改状态的次数（尤其是只修改纹理的情况）

10.2.7.4 场景图scene graph

作用是管理场景中的物体，能够快速的分离出少部分需要和平截头体进行剔除测试的物体集合，同时能够帮助对场景中的物体按照距离或材质排序

①四叉树和八叉树p423/447

四叉树用于从平面角度划分区域，八叉树用于3维角度划分。并非是游戏物体的维度，而是划分空间的方式。对于有些3d游戏，从平面（俯视）划分空间足够了。

②包围球树bounding sphere tree

p424/448

③BSP树binary space partitioning

即二元空间分割树，常见的BSP树有轴对齐划分空间或以三角形为平面划分。

如何使用BSP树顺序遍历三角形p425/449

10.2.7.5 选择场景图

根据游戏类型和场景类型选择场景图p426/450

## 10.3 高级光照及全局光照

10.3.1 基于图像的光照image-based lighting

定义：高级的光照和着色技术都会使用大量含有不同数据类型的纹理贴图

10.3.1.1 法线贴图normal map

如何编码法向量至非负的RGB通道也是个技术：见注释p427/451

一般只是将单位法向量压缩平移至[0,1]的颜色范围

10.3.1.2 高度贴图height map：视差贴图parallax mapping和浮雕贴图relief mapping

定义：编码高于或低于三角形表面的高度值，一般为灰度图

高度贴图和法线贴图的区别在于，前者可以用于计算出后者进而计算表面光照，当然为了性能后者的意义不容小视；前者可以实现出自遮挡self-occlusion和自阴影self-shadow效果，而后者很难直接实现。最后就是两者存储的数据类型不一样，前者是单一高度值，后者常用RGB通道存储法向量。

视差贴图法和浮雕贴图法是利用高度贴图的两个例子。

视差遮挡贴图法parallax occlusion mapping，POM：详见Learn OpenGL

10.3.1.3 镜面贴图specular map/光泽贴图gloss map

光滑度glossiness=镜面幂specular power

定义：将不同区域的镜面反射属性（反射率、光滑度等）编码至贴图中，则称为镜面贴图。一般用于同一材质中反射光程度不同的情况，如汗珠在皮肤上等（其实也是一种不将材质继续细分的近似方法）

存储反射率ks的镜面贴图称为光泽贴图，也称镜面遮罩specular mask。如果存储的是光滑度，称为镜面幂贴图specular power map

10.3.1.4 环境贴图environment map

定义：某物体周围场景的全景照片panoramic photograph。用于低成本反射渲染。

环境贴图不再是纯2维纹理，有球面环境贴图spherical environment map，立方环境贴图cubic environment map，以及双抛物面环境贴图dual paraboloid environment map。

球面主要用于不支持立方贴图的硬件，以及寻址较简单，其他方面远不及其他贴图优点多，而且不利于动态渲染环境贴图（立方和双抛物面均可）。

环境贴图也可以模拟折射，但是一般只适用于曲面的物体反射，不适用于大面积平面物体，如水面等。

10.3.1.5 三维纹理

也称体积纹理volume texture。即由3个坐标确定的纹理。现时3维纹理占用空间太大，有时会用程序纹理procedural texture代替，即由程序生成的纹理。

10.3.2 高动态范围光照high dynamic range，HDR

原理：将高亮的光照结果原样（当然要用不同于传统RGB的格式）保存，然后调用色调映射tone mapping将光强范围映射至显示设备可以显示的范围。

HDR技术可以用于模拟人眼对于强光的暂盲，和bloom效果（这其实是一种虚假的视觉特效，用来给人以光线特别明亮的感觉。HDR是bloom实现的有效基础，但是二者本质上是两种技术）

HDR的格式：

·每个通道32位浮点数代替8位整数

·使用log-LUV色彩模型，L代表强度intensity，16位，使用log2为底；UV代表两个色度chromaticity，各8位。因为人眼对于光强比色度更敏感。

10.3.3 全局光照global illumination，GI

10.3.3.1 阴影渲染

自阴影self-shadow：发光且能投射阴影的物体

①阴影体积技术shadow volume

原理：将投射阴影的物体轮廓与阴影形成的直纹面体积称作阴影体积，并作为一个几何体渲染。但是渲染的只是模板缓冲stencil，再用模板缓冲分步渲染场景。

具体细节：要参考注释p432/456

②阴影贴图shadow map

原理：从光源处渲染一张深度缓冲作为阴影贴图。聚光灯使用透视投影来渲染阴影贴图，平行光使用正射投影，而点光源严格来讲要使用立方贴图或双抛物面贴图。

细节：p432/456（有不准确的地方，比如摄像机空间中的片段需要使用投射变换来获得其在光源贴图纹理中的位置，具体涉及到摄像机的投影方式和光源的投影方式）

相较而言，阴影贴图的性能比阴影体积的性能好，因为后者填充率一般更大。还有一种类似阴影贴图计算的另一种技术——投影贴图

10.3.3.2 环境遮挡ambient occlusion，AO

定义：用于渲染接触阴影的技术。

接触阴影contact shadow：指场景仅以环境光照明时所产生的软阴影——区别于具体物体阻挡光线产生的轮廓或硬阴影。所谓软阴影其实也就是用环境光近似模拟场景中的散射光所催生的又一近似产物。

使用几何方法计算AO值（可接触环境光线的程度）的细节：p433/457

屏幕空间计算AO，即屏幕空间环境遮挡screen-space ambient occlusion，SSAO：见Learn OpenGL

研究新进展：注释85，p433/457

10.3.3.3 镜像

环境贴图用于光滑物体表面产生的镜像，实际上一定会有扭曲，但是一般无法识别。而镜面和水面的反射效果必须做到足够准确。所以一般采用镜面对称的摄像机来渲染至纹理再贴图至镜面的方法。

10.3.3.4 焦散caustics

即水光摇曳的效果，通过投影动画纹理（一些半随机亮点）至物体表面。

10.3.3.5 次表面散射subsurface scattering，SSS

原理：光线在表面下散射，然后在表面的其他位置离开。适用于皮肤、大理石、蜡等材料的柔和淡光效果。

双向表面散射反射分布函数bidirectional surface scattering reflectance distribution function, BSSRDF：不同于BRDF的一种描述光照效果分布的函数

次表面散射的模拟方法有很多，基于深度贴图或光源贴图的方法见p435/459

10.3.3.6 预计算辐射传输precomputed radiance transfer，PRT

简介p436/460

10.3.4 延迟渲染deferred rendering

几何缓冲geometry buffer，G-buffer

详见Learn OpenGL

## 10.4 视觉效果和覆盖层

大部分视觉效果和后期处理都是二维层面意义的操作，而不再是三维渲染。

晕影vignette：也称暗角，画面边缘稍暗和模糊的效果

10.4.1 粒子效果particle effect

如烟雾、火花、火焰

进一步阅读p439/463

10.4.2 贴花decal

如弹孔、脚印、抓痕、裂缝

进一步阅读p440/464

10.4.3 环境效果environmental rendering effect

10.4.3.1 天空

一般会将天空渲染成为一个大纹理，根据其着色复杂程度可以选择最先渲染或最后渲染天空，同时关闭深度写入。而且也会将天空纹理设置为sky box或sky dome，中心置于摄像机位置且随之移动，这样保证无论如何移动天空不变。

10.4.3.2 地形terrain

高度场地形height field terrain：地形建模的方法，使用高度场定义地形走势

层次细节level-of-detail，LOD：用来将细节细致程度分层的系统

进一步阅读p442/466

10.4.3.3 水体

10.4.4 覆盖层overlay

10.4.4.1 归一化屏幕坐标

一般将y轴或纵向高度方向设置为0-1，然后根据宽高比设定x轴的范围，如0-1.33（4:3），0-1.77（16:9）等。这样做的原因是保证坐标不会被拉伸。

10.4.4.2 屏幕相对坐标

略

10.4.4.3 文本及字体

略

10.4.5 伽马校正

略

10.4.6 全屏后期处理效果full-screen post effect

简介，略

10.5 延伸阅读

p446/470

# 第11章 动画系统

角色动画系统character animation system

## 11.1 角色动画的类型

11.1.1 赛璐璐动画cel animation

精灵动画sprite animation

循环动画looping animation：可以循环播放而不产生断档的动画，一般如跑动等。并且有闲置周期idle cycle，跑动周期run cycle等代表循环动画的一个周期。

11.1.2 刚性层阶式动画rigid hierarchical animation

动画纹理animated texture：一连串纹理位图，用以连续渲染产生动感，原本属于早期动画技术，现在用于低分辨率或远景物体渲染，如远处的人群等。实际上是2维面片的纹理，并非三维网格。

刚性层阶式动画：将角色以一堆刚性部分建模，其层次结构以骨盆为基，符合人体学，但是由于刚体的使用导致动画僵硬，而且关节处会出现裂痕。不过适合于模拟真正的机器人。

11.1.3 每顶点动画及变形目标

每顶点动画per-vertex animation：采用蛮力法记录每个顶点的位置和属性信息，信息代价太高昂，不适于实时

变形目标动画morph target anim：采用插值极限姿势extreme pose的方法存储少量顶点属性，通过计算得出动作过程。一般用于面部动画facial anim，因为面部动画细微而且有复杂的解剖结构，但是人物常见表情的种类（喜怒哀乐等）往往有限。

11.1.4 蒙皮动画skinned anim

简述，略

骨骼→骨头→关节→顶点→皮肤（网格）

11.1.5 把动画方法视为数据压缩技术

略

## 11.2 骨骼

术语中关节joint是一个正式称谓，但是也被俗称为骨头bone，但是严格来说二者不是一回事，而且动画模型中并不在乎骨头也不是以骨头为设计对象。不过关节与关节之间也确实可以存在某种类似骨头一般的联系，这种联系可以由树结构中二者的直接从属关系（即连接二者的边）看出。

11.2.1 骨骼层阶结构

一般是和刚性层阶结构相似的。通常会为关节赋予索引，通过存储每个关节的父关节来存储整个数据结构

11.2.2 在内存中表示骨骼

关节数据结构范例p453/477

## 11.3 姿势pose

姿势（定义）：关节相对某参考系的位置、定向和缩放。所有关节的姿势构成骨骼的姿势。姿势一般用平移矢量、四元数或变换矩阵等方式组合表示，视具体需求而定。

11.3.1 绑定姿势

绑定姿势bind pose：也做参考姿势reference pose或放松姿势rest pose或T-pose。是三维网格绑定至骨骼之前的默认形态。

11.3.2 局部姿势local pose

定义：子关节相对父关节的姿势，一般调整父关节的属性时，所有子关节相对姿势不变。

顺带一提，动画软件中严格来说不应该将关节表达为一个球或点，而应该是一个局部坐标系，称为关节空间。

11.3.2.1 关节缩放

一般所有的引擎都会禁止切变shear，一部分允许非统一缩放。对于最简单的情况，除了平移旋转只有统一缩放，可以大幅度降低各种计算，如光照中的矩阵计算、无椭圆形包围球有利于进行平截头体剔除和碰撞检测

11.3.2.2 在内存中表示关节姿势

数据结构范例p456/480

11.3.2.3 把关节姿势当作基的变更

略

11.3.3 全局姿势global pose

定义：在模型空间或世界空间表示的关节姿势。可由局部姿势按照层阶结构串接矩阵计算得出

11.3.3.1 在内存中表示全局姿势

范例p458/482

## 11.4 动画片段

动画片段animation clip：实时游戏中无法像电影一样将整个动画作为整帧渲染，只能将动画拆分为对应角色的小粒度动作。

游戏内置电影in-game cinematics，IGC：也称非交互连续镜头nointeractive sequence，NIS，即游戏中的实时渲染动画，和一般游戏不同的只是不能交互（可能包括摄像机）

全动视频full-motion video，FMV：提前录制好的视频，用于在游戏内播放

快速反应事件quick time event，QTE：半互动的动画序列

11.4.1 局部时间线local timeline

定义：每个动画片段播放的时间线

时间索引time index：局部时间变量的值

11.4.1.1 姿势插值及连续时间

关键姿势key pose：或关键帧key frame，动画制作中并不会按照渲染频率制作帧，而是制作几个特殊的姿势，然后使用插值（线性或曲线）来计算其余的姿势

实时渲染动画和动画电影的另一个不同就是动画电影的帧采样严格按照动画的帧率，而实时渲染动画因为各种原因必须依靠连续的插值计算“获得”所需动画帧。

11.4.1.2 时间单位

选取精确的值而不是整数

11.4.1.3 比较帧与采样

为了避免术语混淆，应该区分代表一帧的一小段时间（后面直接简称为帧）和该帧的时刻（也称采样）

11.4.1.4 帧、采样及循环片段looping

若片段非循环，则采样数比帧数多1。若循环片段，则二者相等，因为最后一个采样和第一个一定相同

11.4.1.5 归一化时间（相位）

为所有片段使用0-1的相同时间索引，不是用于度量时间而是用于同步动画

11.4.2 全局时间线global timeline

即每个动画角色的生命期时间线

播放速率playback rate

·全局时间和局部时间的映射关系公式p464/488

全局时间的使用有利于同步动画，一般可选择全局或局部时间之一存储为动画进度，需要时通过计算得出另一个数据。

11.4.3 比较局部和全局时钟

略

11.4.3.1 用局部时钟同步动画

会造成动画的同步相差至少一帧的情况，因为动画由不同的子系统负责，而游戏循环中会按照顺序轮转子系统。若发起方代码在接收方代码执行之后，那么接收方要在下一次循环反应。

11.4.3.2 用全局时钟同步动画

略

11.4.4 简单的动画数据格式

未压缩的动画片段由所有离散采样的所有关节的完整姿势（SQT向量）所组成。当然实际中需要压缩存储。

11.4.4.1 动画重定目标

即有关于将动画片段应用于不同的骨骼之上，一般严格来说一个动画片段只能用于一个骨骼或类似的骨骼。

11.4.5 连续的通道函数

通道函数channel function：动画片段的数据实际上可以看成SQT向量每个分量的连续函数，理论上是连续光滑的，但是实际上会使用分段线性逼近piecewise linear approximation

11.4.6 元通道metachannel

即非关节姿势信息所存储使用的通道，用于和动画同步过程中的其他功能和计算。

举例：p471/495

## 11.5 蒙皮及生成矩阵调色板

11.5.1 每顶点的蒙皮信息

蒙皮skinning：计算过程为将顶点依次按照只绑定bind一个关节的方式移动，之后将所有关节的结果加权平均。若只考虑绑定一个关节时，顶点完全随着关节移动，亦即在关节空间的位置完全不变。

一般顶点绑定的关节最多4个，便于计算和存储数据，同时更多绑定带来的动画效果也不明显

蒙皮顶点数据结构p472/496

11.5.2 蒙皮的数学

蒙皮矩阵skinning matrix：计算得出的用于将顶点从绑定姿势位置移至当前姿势位置的矩阵，计算前后的参考都是模型空间

11.5.2.1 单个关节骨骼的例子

蒙皮矩阵的推导：p474/498 实际上是关节绑定姿势（逆矩阵）和当前姿势（矩阵）的乘积，参考空间均为模型空间，即全局姿势。

11.5.2.2 扩展至多个关节的骨骼

矩阵调色板matrix palette：一组蒙皮矩阵，代表一个顶点的受其绑定的每个关节影响的情况。或将所有关节的蒙皮矩阵存储，不同顶点按照绑定情况以索引形式调取

11.5.2.3 引入模型至世界变换

【优化】有时可以直接将模型至世界变换矩阵串接至矩阵调色板中，这样可以节省渲染步骤中的一次矩阵计算。但是如果一个动画应用于多个不同的角色，就不能这样用，因为不同角色的模型至世界矩阵不同。

11.5.2.4 把顶点蒙皮至多个关节

绑定binding有时也称rigging，后者一般修饰character，前者修饰joint，实质工作内容一样——确定顶点随关节权重的分布。

将单个关节的蒙皮结果简单加权平均即可

## 11.6 动画混合animation blending

定义：将两个或更多的姿势混合，产生一个输出姿势。简而言之，就是动画姿势插值。可以用于插值动画采样帧，也可以用于插值不同极端姿势，还可以用于圆滑过渡动画

11.6.1 线性插值混合

直接对矩阵使用线性插值不行，所以使用SQT向量表示姿势，即可以分别对每个向量进行直接线性插值。

姿势混合通常都在局部姿势中进行，即每个关节的插值都是独立于自身所在的父关节空间的。这样比使用全局姿势插值更加真实自然。

11.6.2 线性插值混合的应用

11.6.2.1 时间性混合

在时间点尺度上线性插值

11.6.2.2 动作连续性：淡入/淡出cross-fading

混合时间blend time：处于淡入或淡出效果的时间段

·圆滑过渡smooth transition：两个片段在混合时间内一起播放，两个片段一般都为循环动画，且同步至手脚位置大致匹配。

·冻结过渡frozen transition：前一个片段在混合时间开始时冻结，后一个片段逐渐加入动作，适于混合两个不相关且不能在时间上同步的片段

缓入ease-in缓出ease-out：混合因子blend factor按照曲线方式递变的情况

·核心姿势core pose：设定一组核心姿势，然后让每个动画以核心姿势起始，就可以完全避免混合。显然这种方法只适用于部分种类的动画片段。

11.6.2.3 方向性运动

角色运动character locomotion包括两种

·轴转运动pivotal movement：始终面向移动方向，通过轴转旋转

·靶向移动targeted movement：也称strafing，始终面向同一方向，和运动方向独立

靶向移动实现方式：设计向前、向左和向右三种极端移动动画，也称方向性运动片段directional locomotion clip。然后让角色前方对齐向前方向，在3个角度动画中插值。注意不能直接设计包含向后移动的4种姿势插值，因为从人体工程学角度来说，人偏向前和偏向后的靶向移动会使用不同的步伐方式，直接插值会导致腿部骨骼交叉。可以为了向前和向后移动分别设计一组向左和向右移动动画匹配，以及一个用于在两种状态间过渡的机制

轴转移动实现方式：最佳的方法也是设计向前移动、完全向左转和向右转等3个动画（步行和跑步），然后插值混合。注意转向时要有倾斜角度，可以通过角色的移动速度计算。

11.6.3 复杂的线性插值混合

11.6.3.1 泛化的一维线性插值混合one-dimensional LERP blending

方向性运动的半圆是它的一个特例

见p483/507

11.6.3.2 简单的二维线性插值混合

简单即所谓的4个动画，说是二维线性插值，不过是对两组动画的插值结果再进行插值混合。这种情况下，所谓的二维混合因子有两个分量，且独立。

11.6.3.3 三角形的二维线性插值混合

二维混合因子有三个分量，和为1。动画姿势以平面内求三角形重心barycentric的方式进行。

11.6.3.4 泛化的二维线性插值混合

Delaunay三角剖分triangulation

p486/510

11.6.4 骨骼分部混合partial-skeleton blending

混合遮罩blend mask：即对每个骨骼设置不同的混合因子的集合。可以使得角色动画按照预期的方式局部混合或干脆是拼接（一部分设为1一部分设为0的极端情况）

缺点：对骨骼不同部分使用不同的动画会很容易造成角色动作不自然，这可以通过仔细设定混合遮罩参数来改进，但终究有些麻烦。另外本质上人的身体各部分运动因骨骼、肌肉和物理限制并非绝对独立，而这些限制很难直接定量定义。

11.6.5 加法混合additive blending

区别片段difference clip：也称加法动画片段，概念上为D=S-R，即两个输入片段之间的差别

来源片段source clip，S：

参考片段reference clip，R：

目标动画target clip，T：可以将区别片段加入其它动画中增加效果，其它的动画（非用于计算得出D的动画）即为此

11.6.5.1 数学公式

关节姿势是一个变换，因此S-R实际上是SR-1（变换形式）。

对于区别片段的插值和一般动画的插值是一样的。

加法混合百分比：可以使用按百分比混合区别动画来调整程度，只需要对无修改的T和完全修改的DT中间进行比例插值即可。

11.6.5.2 比较加法混合和分部混合

略，加法混合较优，将难以定量描述的运动限制留给计算机计算，留给人的工作更加主观和直观。

11.6.5.3 加法混合之局限

此方法也不是万用方法，虽然有改进但是仍然只是复杂的人体运动的一种近似。

经验：p490/514

11.6.6 加法混合的应用

11.6.6.1 站姿变化stance variation

站姿变化的区别动画可以只有一帧内容，即一个区别姿势，但是可以作为一个恒定姿势片段加入到一整段目标动画中。这个应用非常稳定可靠，因为站姿一般都是固定的。

11.6.6.2 移动噪声

即指角色在进行循环重复的动作过程中，会存在的动作差别。如走路时偶尔顾盼、低头等自然和下意识动作。

11.6.6.3 瞄准及注视

通过制作注视方向或瞄准方向极端情况下的区别动画，再使用加法比例混合即可做到各种角度

11.6.6.4 时间轴的另类用途

举例：利用时间轴和时间插值的功能来插值姿势。此时动画本身不一定是具备连续和独立意义的动作元素，而是各种极端姿势帧的集合。当然这种插值方式只能插值相邻两帧。

## 11.7 后期处理post-processing

11.7.1 程序式动画procedural animation

定义：由程序计算和修改动画参数而产生的动画，不是传统意义上手工定义以及插值形成的动画。用于一些动作简单、易于定义但是可能比较繁复的动画。

11.7.2 逆运动学inverse kinematics，IK

定义：输入是某关节预期的全局姿势，也称末端受动器end effector，求出其他关节的局部姿势，以使受动器达到指定位置

正向运动学forward kinematics，FK：普通的动画片段流程，输入局部姿势，输出全局姿势和蒙皮。

详细阅读：p495/519

11.7.3 布娃娃ragdoll

略，物理系统部分详见12章

## 11.8 压缩技术

用于压缩动画数据的技术

11.8.1 通道省略

按照动画角色的特点，省略一些注定无关的通道数据。比如很多角色不需要非统一缩放，人性角色的骨骼多数不能伸缩和缩放。

也可以按照数据变化情况压缩那些不变的数据。

11.8.2 量化quantization

缩减每通道的尺寸，实际上是降低不必要的精度位数，如一些归一化数据的值无需32位浮点数存储。

量化过程就是将浮点数编码为整数表示法，然后需要时解码为浮点数。这是一个有损压缩lossy compression。

量化算法介绍p497/521

·T编码/R编码

·L重建/C重建

一般最常用的压缩是针对四元数或旋转，对于平移和缩放一般是无界量，如果要量化，必须设定边界，这会导致额外开销，须加以权衡。

11.8.3 采样频率及键省略

压缩动画除了可以从数据入手，还可以从采样数入手，关节数不便于入手

·降低整体采样率

对于某些动画效果仍然可以接受

·省略一些样本

简单来说仍然是利用“可预测性数据”的简化来实现压缩。如一些通道线性变化或不变化即可使用规范化的数据描述而非枚举

11.8.4 基于曲线的压缩

即使用曲线（B样条）拟合动画通道的数据，来代替均匀采样的离散数据，以此达到整体降低数据大小的目的。

Rad Game Tools的Granny 3D API采用此法。

11.8.5 选择性载入及串流

实用中对于动画资源的使用方式，长期留存核心动画资源，选择性载入其他的

## 11.9 动画系统架构

包含后面三个部分的内容，管道（负责处理关节姿势与渲染动画）、状态机（控制游戏角色的动作状态）、控制器（更高层的管理角色的行为模式）

## 11.10 动画管道animation pipeline

管道各个阶段的组成步骤：p502/526

11.10.1 数据结构

11.10.1.1 共享资源数据shared resource data

即相同类型的角色或物体所共享的一组资源数据，包括骨骼、蒙皮网格和动画片段。

11.10.1.2 每实例数据per-instance data structure

即每个角色实例自身的信息，一般也是游戏中的实时本地信息，包括

片段状态——局部时钟和播放速率

混合规格——加权平均法或混合树

分部骨骼关节权重

局部姿势

全局姿势

矩阵调色盘

11.10.2 扁平plain的加权平均混合表示法

略

11.10.2.1 例子：OGRE

p506/530

11.10.2.2 例子：Granny

p508/532

动画时间处理上优于OGRE

11.10.3 混合树animation blend tree

混合树是编译理论中的语法树之例子

11.10.3.1 二元LERP混合

略

11.10.3.2 泛化一维LERP混合

略

11.10.3.3 简单二维LERP混合

略

11.10.3.4 三角LERP混合

三元ternary表达式

11.10.3.5 泛化三角LERP混合

略

11.10.3.6 加法混合

略，区别姿势并不是一个独立的动画，严格意义来说区别姿势只能和正常动画混合而不能自混合

11.10.4 淡入/淡出架构

两种混合规格的不同实现方式

11.10.4.1 扁平加权平均的淡入/淡出

因为数据结构简单，所以对于复杂的动画混合之间的过渡需要额外的数据结构来支持。

11.10.4.2 表达式树的淡入/淡出

因数据结构足够精巧，对于任何种类的混合都可以轻易的表达和执行

11.10.5 动画管道的优化

包括内存存取模式相关优化，SIMD运算优化等，依不同平台特点而有不同工作内容

11.10.5.1 PlayStation 2上的优化

p514/538

便笺内存scratch pad

11.10.5.2 PlayStation 3上的优化

动画作业animation job

11.10.5.3 Xbox及Xbox 360上的优化

略，有时可以借鉴不同平台之间的优化策略

## 11.11 动作状态机action/animation state machine，ASM

底层管道如OpenGL之上有一个中间软件层，用于控制角色的动作状态。

11.11.1 动画状态

ASM中每个状态对应一个任意复杂的动画片段混合。对于混合树架构而言，即对应一个混合树。

11.11.1.1 状态及混合树规格

数据驱动data-driven：一种动画引擎的工作方式，通过参考立刻显示的结果来修改数据，是一种以快速迭代rapid iteration为思想的方式。

数据驱动的具体实现可以依靠文本或脚本，也可以依靠图形界面。但这只是引擎的UI部分，并不一定影响工作结果。

11.11.1.2 自定义混合树节点类型

已定义的4种原子混合节点类型：片段本身、二元LERP，二元加法混合，三元LERP

游戏引擎可以支持一些自定义的复合节点类型来便于各种游戏重复使用某种混合方式。

11.11.1.3 例子：顽皮狗的神秘海域引擎

一个依靠强大功能脚本语言Scheme（基于Lisp）的引擎

11.11.1.4 例子：虚幻引擎3

使用类似node-based的GI界面设计动画，但是是基于巨型混合树而不是状态来管理角色动画的，虽然本质上是一样的。巨型树的子树便是某一种状态的混合树。

11.11.2 过渡

由一个状态过渡至另一个状态时所需经历的中间过程，可以是一个独立状态，但不是一个稳定的状态。如果两个状态的起始姿势完全匹配则不需要过渡，而过渡也可以由淡入/淡出完成，但并不总是可行。

11.11.2.1 过渡的种类

过渡状态transitional state

11.11.2.2 过渡的参数

描述过渡的参数p522/546

11.11.2.3 过渡矩阵transition matrix

即用于描述n种状态之间的n^2种过渡情况的数据结构，但是实际上并非每个元素都有意义，即并非每个状态都可以过渡至另一个任意状态，同时完全独有的过渡也很少。

11.11.2.4 实现过渡矩阵

略，p522/546

11.11.3 状态层state layer

预备动作anticipation：某些身体部位引领其他部位按次序进行跟随动作

状态层：将状态管理分层进行，每个层每时刻只有一个状态，所有层按照设定规则混合。数据结构上即将多个状态混合树再进行混合。

举例：p525/549

11.11.4 控制参数

即控制角色动作的参数，主要是混合权重。如何方便快捷地查找这些参数和修改所需的参数，是一个架构性的问题。

举例：p526/550

11.11.5 约束constraint

动画系统中的角色或物体运动约束

11.11.5.1 依附attachment

定义：子骨骼移动时父骨骼不受影响，父骨骼移动时受依附约束的子骨骼关节相对姿势不变。

依附点attach point：一种特殊的关节，并不用于计算角色的姿势，而是以关节姿势的形式记录一对不同骨骼关节之间的依附约束，或称为偏移offset（依附不仅仅是两个关节重叠）

Maya中有一种定位器locator，即为特殊关节的例子，可以做各种非动画姿势性的定位任务。

11.11.5.2 跨物体对准

即不同物体之间的位置按需对齐。动画片段一般只是针对单个独立的骨骼定义的，要实现多个动画片段的最终姿势的对齐就需要在每个动画片段中增加一个特殊关节（绝对位向并不重要），该特殊关节在所有动画片段按需对齐时全部重合。该特殊关节在场景动画制作时为所有动画片段统一选定，也称参考定位器。最后可以将参考定位器和世界空间中预定义的对应定位器对齐，从而将场景对齐至世界空间中所需位置。

世界空间中的参考定位器可以用两种方式定义，一是将其设为独立的一个特殊角色，也是唯一拥有已知世界空间变换的角色，另一种是选取已经在世界空间中存在定义的并且一般不会移动的物体来计算世界空间参考定位器的变换。

11.11.5.3 抓取及手部IK

有些约束并不能保证随着LERP混合保持原有的约束关系，这种情况下需要最后对关节姿势进行修改和调整。另外IK也属于物理系统的一部分，用于解决和计算物理约束。

末端受动器end effector：IK指定关节的目标位置，进而反求该关节的连续几个祖先以让该关节接近目标位置。需要修正的关节也叫末端受动器。

IK的计算一般仅仅涉及位置（而不是位向）计算，而且要求所求关节和目标本身足够接近才能尽快得出理想结果。

11.11.5.4 动作提取extracted motion及脚部IK

滑步foot sliding：一种脚步动画的失真

步行或其他类似移动动画在动画片段制作时采用真实的移动方式设计，但动画片段是循环播放于游戏中的，直接套用片段会让角色每个周期闪回至原点（或每个周期必须让角色位置瞬移），这不利于游戏性的实现。因此首先将角色移动位置的数据通道单独提取，并将动画片段中的角色移动清零（始终在原点），最后利用位置数据通道的信息更新原点位置即可完美地表现动画同时维持正确的游戏性（角色坐标变化等）。

之所以不使用平均移动速率来更新原点位置，是因为有些移动动画并不是匀速运动的，使用匀速更新位置势必会导致滑步。

脚部IK：略

角色的操控感应该高于角色动画的美观，应该足够灵敏，

11.11.5.5 其他类型的约束

·注视look-at——IK或程序式关节偏移，加法混合

·掩护对准cover registration——参考定位器

·进入及离开掩护

·通行协助——参考定位器

## 11.12 动画控制器animation controller

用于在动画状态机之上更高层地封装和控制动画，是一个可选的软件层。并无固定框架规则。

# 第12章 碰撞及刚体动力学

·碰撞检测系统collision detection system：只是一个检测几何体相交的组件，通常和物理引擎系统整合。当然也可以只使用碰撞检测而不加入物理模拟至游戏

·物理引擎physics engine：进行刚体动力学rigid body dynamics计算的组件，很多时候必须依赖碰撞系统的测试结果

## 12.1 你想在游戏中加入物理吗

12.1.1 物理系统可以做的事情

举例见p538/562

物理模拟有实时的也有离线脱机预处理形式的（动画片段）

12.1.2 物理好玩吗

12.1.2.1 模拟类游戏simulation game

必须物理引擎

12.1.2.2 物理解谜游戏physics puzzle

必须物理引擎

12.1.2.3 沙箱游戏sandbox game

或开放世界类型游戏，可选用物理引擎增添开放探索的乐趣

12.1.2.4 基于目标及基于故事的游戏

物理系统或游戏真实性不能阻碍游戏单一的进度方式，或者可以弃用实时模拟而选用设计好的行为。

12.1.3 物理对游戏的影响

12.1.3.1 对设计的影响

·可预测性predictability：物理模拟没有精确的可预测性，即便可以做到也比灌装的代价高

·调校及控制：调节物理系统比调整动画更加困难

·意外行为：物理模拟的漏洞被利用成为bug

12.1.3.2 对工程的影响

·碰撞检测和物理模拟需要耗费工时也会对其他子系统产生影响，如人工智能，也会影响其他一些子系统的脱机工作，如脱机光照计算等

·物理系统会带来更多潜在的bug和麻烦，尤其是关于物体穿越和碰撞

·实时物理系统会使得网络游戏和重放游戏等功能有更多问题

12.1.3.3 对美术的影响

·更多的模型、物体属性参数和失控性

12.1.3.4 其他影响

略

## 12.2 碰撞/物理中间件

12.2.1 I-Collide、SWIFT、V-Collide及RAPID

略

12.2.2 ODE（Open Dynamics Engine）

开源碰撞及刚体动力学SDK

12.2.3 Bullet

用于游戏和电影的开源碰撞检测及物理程序库

支持连续碰撞检测continuous collision detection，CCD，也称冲击时间time of impact，TOI

12.2.4 TrueAxis

略

12.2.5 PhysX

原本是物理协处理器的程序，但由于GPU可以支持通用计算，协处理器逐渐没有必要

扩展阅读：p543/567

12.2.6 Havok

略，商用，代价高

12.2.7 PAL，Physics Abstraction Layer，物理抽象层

开源程序库，可以让开发者使用多于一个物理SDK

12.2.8 DMM，Digital Molecular Matter，数字分子物质

使用有限元方法模拟变形

## 12.3 碰撞检测系统

互相穿插interpenetrate

深入阅读：p545/569

12.3.1 可碰撞的实体

碰撞表达形式collision representation：游戏程序中物体在碰撞系统中的表达形式，除此之外在其他系统中有gameplay representation和visual representation

可碰撞体collidable：Havok中的对应术语

演员actor：PhysX中的术语

12.3.2 碰撞/物理世界

12.3.2.1 物理世界physics world

也称collision world。是一个概念也是一个数据结构，指游戏世界中具备物理或碰撞性实体的另一种完整表述。这种单独用一个“世界”来表达物理引擎所处理的信息为的是可以避免遍历游戏世界中没有赋予物理的实体，且可以高效组织碰撞数据。

12.3.3 关于形状的概念

多边形polygon和多面体polyhedron

挤压extrusion：一个表面形状的参数，表面surface是一个理论上没有厚度的面，挤压参数赋予表面假想厚度，用来在离散的计算中不会错失和高速细小物体的碰撞

12.3.3.1 相交intersection

12.3.3.2 接触

分离矢量separating vector：用来最快分离接触物体的矢量，由碰撞检测系统计算得出

12.3.3.3 凸性convexity

定义：由形状内发射的光线不会穿越形状表面两次以上。

12.3.4 碰撞原型collision primitive

包围体积bounding volume可以比物体网格小，这可以避免误命中，而且擦碰命中本来就比较难以察觉。

12.3.4.1 球体sphere

四元素浮点矢量即可表达一个球

12.3.4.2 胶囊体capsule

一个圆柱体加上两个半球，由两点和一个半径表示。也可以看做是一个扫掠球体swept sphere。

12.3.4.3 轴对齐包围盒axis-aligned bounding box，AABB

长方体cuboid

表达方式（2维）：两点法，一点+两个尺寸，一个中心点+两个半长尺寸

12.3.4.4 定向包围盒oriented bounding box，OBB

可以由一个仿射矩阵表示，将一个单位立方体变换至所需位置及定向

12.3.4.5 离散定向多胞形discrete oriented polytope，DOP

是AABB和OBB泛化的形状，两者均为6-DOP，详见p550/574

12.3.4.6 任意凸体积arbitrary convex volume

略

12.3.4.7 多边形汤polygon soup

即由多边形连续组成的复杂表面，可能为凹体，可能为表面。

12.3.4.8 复合形状compound shape

定义：将多个基本形状组合成为一个形状，原来的形状可能是一个凹性形状。

中间阶段midphase碰撞检测：将复合形状作为一个整体再外加一个包围形状（是基本形状），可以用该包围形状预先检测碰撞，再进一步测试

12.3.5 碰撞测试及解析几何analytical geometry

12.3.5.1 点与球体的相交

【优化】计算矢量长度（模）有时是不必要的，保留平方省略平方根运算也可以

12.3.5.2 球体与球体的相交

12.3.5.3 分离轴定理separating axis theorem

内容：若找到一个轴，两个凸形状在该轴上的投影不重叠，即可确定二者不相交

分离线（面）：一个可以将两凸物体分开的线或面，若存在该线或面，则分离轴与其垂直（可能有无数个这样的组合）

两个圆的可能分离轴必包含两个圆的圆心连线

12.3.5.4 AABB与AABB的相交

使用分离轴定理，可能的分离轴必为三个坐标轴中，反之，若三个轴都重叠，则必相交。

12.3.5.5 检测凸碰撞：GJK算法

详细阅读算法：p556/580

闵可夫斯基差Minkowski difference：A图形的所有点和B图形的所有点成对相减，所得矢量的集合。若AB相交，则该集合包含原点，反之不包含。

12.3.5.6 其他形状对形状组合

每对不同形状之间的碰撞检测都可能不一样，详细阅读：p558/582

双分派double dispatch：p559/583

12.3.5.7 检测运动物体之间的碰撞

bullet through paper或tunneling效应：高速运动细小物体在每一帧的位置采样之间存在空隙，从而可能越过应该碰撞的某些相对较薄物体

①扫掠形状swept shape

利用相邻帧之间形状的采样形成一个扫掠图形用以计算碰撞，但非常不精确，不适合运动轨迹复杂的情况（非直线且自带旋转等）

②连续碰撞检测continuous collision detection，CCD

线性插值形状的每一帧位置与定向，用以估算帧之间的位向，然后通过迭代计算求出路径上的碰撞

冲击时间time of impact

12.3.6 性能优化

空间散列spatial hashing

空间细分spatial subdivision

层次式包围体积hierarchical bounding volume

12.3.6.1 时间一致性temporal coherency

也称帧间一致性frame-to-frame coherency

缓存一些物体的运动信息，很多情况下这些信息不会改变或大变

12.3.6.2 空间划分space partitioning

使用树形数据结构，略

12.3.6.3 粗略阶段、中间阶段、精确阶段

broad phase：所属AABB包围体的测试

midphase：逼近包围体间的测试

narrow phase：碰撞原型间的测试

①扫掠裁减算法sweep and prune

用于粗略阶段检测，将所有AABB的最大最小坐标分不同的轴排序以检测是否重叠

12.3.7 碰撞查询collision query

碰撞投射collision cast：也称cast，追踪trace，探查probe。是一种“特殊”的碰撞检测，或是预测什么物体会发生碰撞。

12.3.7.1 光线投射ray cast

p563/587

12.3.7.2 形状投射shape cast

p564/588，几何体的光线投射

12.3.7.3 Phantom

Havok中无实体碰撞体的术语名称

12.3.7.4 其他查询类型

邻近查询proximity query：找出接近碰撞体上的最近点集合

12.3.8 碰撞过滤collision filtering

12.3.8.1 碰撞掩码及碰撞层

将碰撞体分层管理，并且可以定义每层碰撞体之间可碰撞性

12.3.8.2 碰撞回调callback

决定是否接受或拒绝碰撞结果

12.3.8.3 游戏专门的碰撞材质material

用来定义物体物理效果的属性，应该将碰撞原型和属性分别集中存储，以索引方式访问达到紧凑数据的效果。

## 12.4 刚体动力学

经典刚体动力学classical rigid body dynamics

刚性

12.4.1 基础

深入阅读：p570/594

12.4.1.1 单位

MKS系统：使用米，千克，和秒的单位组合

12.4.1.2 分离线性及旋转动力学

无约束刚体unconstrainted rigid body

线性动力学linear dynamics：质心运动

旋转动力学angular dynamics

12.4.1.3 质心center of mass

也称CM/COM

12.4.2 线性动力学

12.4.2.1 线性速度和加速度

略

牛顿标记法Newton’s notation：用正上方的点号来表示对时间导数

莱布尼茨标记法Leibniz’s notation：dx/dt形式

12.4.2.2 力及动量

线性动量linear momentum：即最普通的质量和速度乘积

12.4.3 运动方程求解

12.4.3.1 把力作为函数

弹簧常数spring constant或刚度stiffness

粘滞阻尼系数viscous damping coefficient

12.4.3.2 常微分方程ordinary differential equation，ODE

略

12.4.3.3 解析解analytical solution

略

12.4.4 数值积分numerical integration

12.4.4.1 显式欧拉explicit Euler method

即用delta每帧变化量来直接近似微分，或线性外插

12.4.4.2 数值方法的特性

·收敛性covergence

·阶数order：p577/601，显式欧拉法是一阶方法

·稳定性stability

12.4.4.3 显式欧拉以外的选择

深入阅读：p578/602

向后欧拉backward

中点欧拉mid-point

Runge-Kutta方法族（其中RK4）

12.4.4.4 韦尔莱积分Verlet integration

正常韦尔莱积分法p578/602

优点：快速稳定

缺点：△t在前后两帧中必须接近，不然会导致误差

12.4.4.5速度韦尔莱积分法velocity Verlet

更常用，p579/603

12.4.5 二维旋转动力学

12.4.5.1 定向及角速率

平面薄片plane lamina

12.4.5.2 角速率与加速度

略

12.4.5.3 转动惯量moment of inertia

含义：相当于线性动力学中的质量，也称inertial mass。即改变转动状态的难易程度

12.4.5.4 力矩torque

也称转矩、扭矩、转动力矩。概念都是一样的

力矩，角加速度，转动惯量 = 力，线性加速度，质量

12.4.5.5 二维旋转方程求解

类似线性

12.4.6 三维旋转动力学

12.4.6.1 惯性张量inertia tensor

惯量积profuct of inertia：惯性张量（转动惯量的3维表达）的非主对角线元素。若刚体对3个主轴都是对称的，则惯量积均为0

虽然真实情况并非如此，但是游戏中通常都会将所有物体的惯量积设为0，反倒可以呈现观众可以接受的效果

12.4.6.2 三维中的定向

仍然使用四元数表达定向

必须统一一个零旋转方向，在这个世界中

12.4.6.3 三维中的角速度及角动量angular momentum

三维中物体的角速度并不守恒，因为旋转轴并不稳定（原因不详），而且转动惯量可变。必须在求得角动量后才能算出。

12.4.6.4 三维力矩

略

12.4.6.5 三维旋转运动方程求解

角速度矢量和四元数定向之间的计算关系：p586/610

和二维或线性运动方程有区别

12.4.7 碰撞响应collision response

12.4.7.1 能量

略

12.4.7.2 冲量碰撞响应

也就是无摩擦力下瞬时碰撞的牛顿恢复定律Newton’s law of restitution for instantaneous collisions with no friction

冲量impulse

恢复系数coefficient of restitution

·这是基于经典物理理论给出的碰撞响应模型，虽然是科学的但不一定好用。

·包括旋转的碰撞响应更加复杂（超出高中物理范围）：p590/614

12.4.7.3 惩罚性力penalty force

p590/614

一种便于计算的简单建模，但是有些情况下并不准确

12.4.7.4 使用约束解决碰撞

即将碰撞作为一种不许穿透的约束条件来求解

12.4.7.5 摩擦力friction

略

碰撞摩擦力collision friction

12.4.7.6 焊接welding

一种Havok 4.5版本引入的抛弃伪接触点的技术，伪接触点是相邻共面三角形网格边界所产生的。

启发法heuristic

12.4.7.7 休止、岛屿及休眠

物理模拟有时无法准确的模拟出归于静止的物体，需要进行特殊的处理和定义，令其真正静止

①休眠条件

常用休眠条件p593/617

②模拟岛simulation island

一种Havok和PhysX的物体分组技术，用于优化休止性能

12.4.8 约束

12.4.8.1 点对点约束point-to-point constraint

球窝关节ball and socket joint

12.4.8.2 刚性弹簧stiff spring constraint

略

12.4.8.3 铰链约束hinge constraint

略

12.4.8.4 滑移铰prismatic constraint

活塞运动

12.4.8.5 其他常见约束

p596/620

12.4.8.6 约束链constraint chain

略

12.4.8.7 布娃娃ragdoll

略

12.4.8.8 富动力约束powered constraint

带有力或力场影响的约束，而不局限于对自由度的绝对限制

12.4.9 控制刚体的运动

12.4.9.1 引力gravity

略

12.4.9.2 施力

略

12.4.9.3 施力矩

纯力矩pure torque

12.4.9.4 施以冲量

略

12.4.10 碰撞/物理步

计算步骤大致流程：p599/623

12.4.10.1 约束求解程序constraint solver

约束求解的大致方法描述：p600/624

12.4.10.2 各引擎的差异

详细深入阅读：p601/625

## 12.5 整合物理引擎至游戏

12.5.1 连接游戏对象和刚体

游戏对象或逻辑对象是碰撞体/刚体和网格实例之间的桥梁，这也是三种游戏中物体的不同表达方式，当然并非每个物体都有全部三种

12.5.1.1 物理驱动的刚体physics-driven

在Havok中称为dynamic

即一般意义上的带有物理的游戏物体

p602/626

12.5.1.2 游戏驱动刚体game-driven body

在Havok中称为keyframed

带有物理和碰撞性，可影响其他物理驱动刚体，但是自身依靠游戏驱动，如由动画驱动，或直接由玩家控制的物体。虽然是游戏驱动但是为了便于物理计算，可以通过冲量（额外进行计算来获得）移动物体而不是直接定义其位向（当然这样做也可以，需要解决碰撞时的参数定义问题），然后根据需要调整速度。显然一个物体由什么驱动不是固定的、可以根据情况改变。

12.5.1.3 固定刚体fixed body

只有碰撞的刚体，无动力学

12.5.1.4 Havok的运动类型motion type

p604/628

12.5.2 更新模拟

一般每帧一次物理模拟更新，包括物理计算、维护物理对象和游戏对象联系。

完整更新物理模拟所需步骤：p605/629

一般而言，游戏驱动的刚体先更新，物理驱动的游戏对象后同步，最后才渲染。

12.5.2.1 安排碰撞查询的时间

物理更新一般在整个帧的计算之末期，在其之后的几乎只有渲染步。所以使用本帧的物理计算信息更新本帧的游戏逻辑行不通（本帧逻辑已经计算完成），对此情况有多种选择和妥协方式。

p606/630

12.5.2.2 单线程更新

举例：游戏对象和子系统/子引擎之间的互动

p606/630

12.5.2.3 多线程更新

①在另一线程运行物理

竞态条件race condition：两个线程的执行速度和起止时间不同导致问题

线程同步thread synchronization：包括互斥锁mutex，信号标semaphore，临界区域critical section。一种让线程之间彼此等待的同步技术。

命令队列command queue：另一种线程同步技术，利用队列管理线程通信

读取模式或写入模式锁定：赋予锁定两种不同的方式，写入模式锁定排除读取和其他写入操作，读取模式锁定只排除写入操作。

②分叉及汇合fork/join

不存在同步问题，因为主线程继续执行下去必定所有分叉均完成并汇合。能够使用该架构的前提是有分组（例如“岛屿”）概念，将互不影响的刚体分开

③作业模型job model

使用作业模型对于碰撞查询（可以在任何时候进行）有优势

多线程模式和作业模式的对比：p609/633

12.5.3 游戏中碰撞及物理的应用例子

实际引擎中如何构架刚体和游戏对象：

12.5.3.1 简单刚体游戏对象

p609/633

12.5.3.2 弹道

p610/634

12.5.3.3 手榴弹

p610/634

12.5.3.4 爆炸

p611/635

12.5.3.5 可破坏物体destructible object

破裂模拟fracture simulation

·可破坏物体也可以进行更多详细的分类，但是这种分类实际上是对组成材料具有不同特征的简单近似。

p611/635

12.5.3.6 角色机制

游戏中的角色是以游戏驱动刚体方式移动的，但是为了解决碰撞，需要使用球体或胶囊体投射为角色的移动和与物体互动做出预判。

Havok使用胶囊体phantom来模拟角色的刚体，p613/637

12.5.3.7 摄像机碰撞

摄像机碰撞方面的常见问题：p613/637

12.5.3.8 整合布娃娃

存活和死亡的角色应该使用不同的物理表示（刚体），因为存活时一般为游戏驱动，刚体尽量靠近且不禁止穿插，但死亡时由物理驱动，刚体必然不能穿插。

p614/638

## 12.6 展望：高级物理功能

形变体deformable body：DMM

布料cloth：

头发hair：建模为大量细丝或可变形体

水面模拟：water surface simulation及浮力buoyancy

通用流体动力学模拟general fluid dynamics simulation

深入阅读p616/640

**第四部分 游戏性**

# 第13章 游戏性系统简介

游戏性gameplay也称游戏机制game mechanics或G-factor

## 13.1 剖析游戏世界

13.1.1 世界元素

13.1.1.1 静态几何体

实例化几何体instanced geometry：利用少量种类的几何体随机组合成大量独特的整体

笔刷几何图形brush geometry：一个晦涩的Quake引擎概念，实际上就是一个简单的凸形状，用来简化逼近真正的游戏资源，用来初始设计场景布局。优点是渲染快速。

13.1.2 世界组块world chunk

也称关卡level，地图，舞台stage，地区area

·这是一个很古老的技术概念，用至今日仍然有内存限制、分工限制、游戏性控制的作用

13.1.3 高级游戏流程high-level flow

定义：由玩家目标objective（或task，也有时称wave或level或stage）组成的序列、树或图。目标和组块的关系可以一一对应（早期游戏），也可以更加复杂，视游戏设计而定

## 13.2 实现动态元素：游戏对象

游戏对象以OOP形式建模

13.2.1 游戏对象模型object model

对象模型的概念：p625/649

OMT+Booch+OOSE=UML

13.2.2 工具方tool-side的设计和运行时runtime的设计

工具方：以世界编辑器和设计师接触的对象模型

运行时：略

·两者不必采用相同的对象模型

## 13.3 数据驱动游戏引擎

数据驱动data-driven：顾名思义，由数据的更改即可完成修改目标的设计，无需大量重复的软件编程工作。这样的设计可以降低游戏开发成本，最大化美术和设计人员对游戏开发的参与性，加快迭代和测试的效率。

数据驱动模式固然有优势，但是数据驱动模式需要更大的软件开发成本，这是一个需要衡量的事情。

## 13.4 游戏世界编辑器game world editor

13.4.1 游戏世界编辑器的典型功能

13.4.1.1 世界组块创建及管理

p629/653

13.4.1.2 可视化游戏世界

多种可视化方式：p629/653

13.4.1.3 导航

导航功能描述

13.4.1.4 选取

功能描述

13.4.1.5 图层

略

13.4.1.6 属性网格

略

①选取多个对象后的编辑方式

略

②自由格式属性

用户自定义属性，用来试验新游戏性或一次性场合

13.4.1.7 安放对象及对齐辅助工具

略

13.4.1.8 特殊对象类型

其他功能的描述

·光源：

·粒子发射器particle emitter：

·区域region：

·样条spline：

13.4.1.9 读/写世界组块

读写组块内容的粒度有说道

13.4.1.10 快速迭代

说白了就是调整参数和看到结果改变之间的时间要合理地短

13.4.2 集成的资产管理工具

举例：UnrealEd，包含游戏世界编辑器和完整的内容创作软件包

13.4.2.1 数据处理成本

何时将游戏资产进行针对特定平台优化？p635/659

# 第14章 运行时游戏性基础系统gameplay foundation system

## 14.1 游戏性基础系统的组件

包括：

·运行时游戏对象模型runtime game object model

·关卡管理及串流level management and streaming

·更新实时对象模型real-time object model updating

·消息及事件处理messaging and event handling

·脚本scripting

·目标及游戏流程管理objectives and game flow management

运行时游戏对象模型包括：

·动态产生和消灭游戏对象

·联系底层引擎系统

·实时模拟对象行为

·定义新游戏对象类型

·唯一的对象标识符

·游戏对象查询

·游戏对象引用

·有限状态机

·网络复制

·存档及载入游戏、对象持久性

运行时类型识别runtime type identification，RTTI，反射reflection，抽象构造abstract construction

## 14.2 各种运行时对象模型架构

两种对象模型架构：

·对象为中心：一般的OOP理解方式，每个游戏对象对应一个或一组实例，游戏对象属性即实例的属性

·属性为中心：不存在游戏对象对应的实例，而是将属性实现为实例，再通过表将属性们和游戏对象（只是一个标识符）联系

14.2.1 以对象为中心的各种架构object-centric

14.2.1.1 一个简单以C实现的基于对象的模型：《迅雷赛艇》

实际上是用C语言变相实现了OOP，比如struct中带有函数指针成员。

14.2.1.2 单一庞大的类层次结构monolithic class hierarchy

·这是最常用的架构，直接使用OOP支持语言最佳

14.2.1.3 深宽层次结构的问题

①类的理解、维护及修改

深度越大，修改和理解该派生类的困难越大，越容易出现bug，且难以发现

②不能表达多维的分类

分类学taxonomy

每个层次使用单一确定标准分类对象，这种标准一旦确立无法改变，更无法随着新加入的类做出适当改变，除非新加入的类本身就符合原有分类标准

③多重继承：致命钻石multiple inheritance,MI

一般不用这种继承方式

④mix-in类

定义：一种无基类的独立类，区别于主要继承层次结构。一个类最多只能继承主要层次结构中的一个类，但是可以额外联系（使用继承、合成或聚合均可）任意数量的mix-in类，从而避免多重继承问题。

·采用继承mix-in的方式加入额外的分类信息并不是最佳方法，最好使用合成composition和聚合aggregation，因为可以降低耦合程度。

⑤冒泡效应bubble up effect

程序设计过程中使用深宽层次结构的一个现象，即陆续将多个类中共有的功能向上移至基类中统合。这是一个必然和必须的过程，但是也会导致子系统功能封装性的问题。

14.2.1.4 使用合成简化层次结构

合成：两个类是组成关系，或是拥有关系，并且生命周期一致

聚合：两个类有联系（通过指针或引用），但是生命周期无关

①把“是一个”改为“有一个”

组件component：也称服务对象service object，区别于唯一一个根基类，每个组件是一个具备独立功能的独特类，被根基类引用

②组件的创建及拥有权

略

14.2.1.5 通用组件

即将所有的组件继承自一个基类，该基类即为通用组件，根游戏对象类包含通用组件的链表。这样可以让组件选取更有弹性限制更少，但是通用和弹性的代价是更难理解和组件之间更高要求的独立性（实际上部分组件之间的“合作”是常见的）。

14.2.1.6 纯组件模型pure component model

定义：没有游戏对象根类和游戏对象实例，仅仅保留一个标识符用以标识所有组件对象。创建这样的一个游戏对象时，可以使用工厂模式，即使用一个名为工厂的类来帮助创建对象和组件组。

纯组件模型还有通信方面的缺点和不便，虽然有优点，但是并不是当前最通用且最优的选择（当然也不一定最差）。

14.2.2 以属性为中心的各种架构

成功案例：p653/677

14.2.2.1 通过属性类实现行为

利用属性类的函数实现

14.2.2.2 通过脚本实现行为

将脚本对象或函数视为一个特殊的属性，利用该属性来控制行为。

·一般属性类和脚本可以共用，核心属性使用属性类硬编码即可，自定义或新增属性可利用脚本支持

14.2.2.3 对比属性与组件

有时候属性为中心的架构和组件模型（省略对象类的对象中心模型）难以区分，也不必区分。只是注意一下术语混淆即可。

14.2.2.4 以属性为中心的设计优缺点

优点：

·有效使用内存，不存在空闲属性成员

·更容易使用数据驱动模型

·更加缓存友好

缺点：

·直观视觉结构不明晰，难以手工调试

数组之结构struct of array，SoA：数据布局方式，将相同类型的类似数据成员连续集中存储

结构之数组array of struct，AoS：传统布局方式，将相同类型的对象连续存储，但实际内存中数据并不类型连续

内存墙memory wall：内存硬件发展比CPU发展慢很多的现象

14.2.2.5 延伸阅读

属性中心的架构简报

p656/680

## 14.3 世界组块的数据格式

世界组块数据文件的内容：

对象属性的初始值

对象类型的某种规格

14.3.1 二进制对象映像

这是性能上最佳的方式，然而并不利于修改，可以用于更加稳定的数据存储，如网格或碰撞几何。但是并不是最通用最佳的数据格式

14.3.2 游戏对象描述的序列化serialization

C#和java具备原生序列化系统，C++不具备

XML是流行的对象序列化格式，但是缺点是解析慢。JSON是另一个，更加快捷简单。

·序列化和反序列化的实现机制选择

在基类加入一对虚函数，负责序列和反序列化

实现一个反射系统，继而实现一个通用的自动序列化系统

C++中实现反射系统的技巧简述：p659/683

C++中实现反序列化对象和类时，不能直接使用字符串或标识符，即只能硬编码要实例化的类名称，因此可以使用工厂模式：p659/683

14.3.3 生成器spawner及类型架构p660/684

生成器是一种元数据对象，一般是游戏对象类的成员和属性信息，不包括函数。这种数据的存在为了便于从世界编辑器的工具方对象模型建立起运行时游戏对象模型的桥梁。

14.3.3.1 对象类型架构

数据驱动的schema：p660/684

·schema的作用就是对象类型的原型archetype，生成器则是含有初始化信息用来具体实例化对象

14.3.3.2 属性默认值

略

14.3.3.3 生成器及类型架构的好处

略，优点很多，就是很好，p662/686

## 14.4 游戏世界的加载和串流

游戏世界加载系统的功能：

管理所需文件I/O

管理资源的内存分配和释放

14.4.1 简单的关卡加载

载入并驻留load-and-stay-resident，LSR

【优化】注释《猎天使魔女》p663/687

14.4.2 往无缝加载进发：阻隔室

本质就是异步asynchronous文件I/O

14.4.3 游戏世界的串流streaming

串流的目标：p665/689

无缝世界串流的实现案例：p665/689

14.4.3.1 判断要加载哪些资源

使用技术：关卡加载区域level load region系统

描述：p666/690

14.4.4 对象生成的内存管理

载入世界组块后，对动态对象进行运行时模型生成spawning，即实例化

14.4.4.1 对象生成的离线内存分配

即一次性提前分配好内存，运行时完全禁止动态分配

14.4.4.2 对象生成的动态内存管理

①为每个对象类型设内存池

相同的对象类型往往大小相同，有时不同的对象类型也可能大小相同。

缺点在于需要精确估算对象类型有多少实例，不能浪费也不能不够，但是可以避免碎片

②小块内存分配器small memory allocator

原理：使用预设的一组大小不同的池分配器，但是允许对象申请比自己大的池。

浪费了一定的内存，但是可以缓解碎片问题。

③内存重定位memory relocation

无需多言，最难实现的办法也是最好的办法

14.4.5 游戏存档saved game system

和世界组块文件有一定功能上的相似之处，但是内容和总量上几乎完全不同。

只要从游戏存档恢复出来的游戏世界玩家察觉不出区别就可以，另外游戏存档无需存储世界组块文件中已知内容。

14.4.5.1 储存点check point

更加便捷的存档机制，更加细小的存档文件

14.4.5.2 任何地方皆可存档save anywhere

这是需求空间最大的存档模式，但是也可以一定程度进行精简（比如不必完全存储动画索引，只要重生游戏对象即可），利用人类的非完美记忆。

## 14.5 对象引用与世界查询

邻近性查询proximity-based query

14.5.1 指针

指针的问题和缺点：

·孤立对象orphaned object：删除对象后忘记处理它所拥有的对象，残留对象没有其他对象来管理其生命周期，白白占用内存

·过时指针stale pointer：删除对象后忘记将所有指向该对象的指针置空，该错误很难追踪因为在一段时间内该对象仍然在内存中残留。

·无效指针invalid pointer：即一般的对无效指针的解引用，可以用断言来预防

14.5.2 智能指针smart pointer

即用自定义类封装的指针对象，为了和一般指针共用，须重载一切一般指针可用的运算符。另外为了达到智能化规避错误，需要定义一系列操作来代替程序员的记忆。

※智能指针非常难以实现，最好使用成熟的库（如Boost C++模板库）或干脆避免使用。

14.5.3 句柄handle

句柄的介绍、优点和实现示例：p672/696

14.5.4 游戏对象查询game object query

常见游戏查询类型和对应的数据结构：p675/699

## 14.6 实时更新游戏对象

游戏对象的状态state不同于有限状态机定义的状态，实际上一个游戏对象的状态包括了数个有限状态机，每个FSM一般只定义管理游戏对象的某个子系统的状态。

14.6.1 一个简单（但不可行）的方式

14.6.1.1 管理所有对象的集合

游戏引擎通常会利用一个singleton管理所有活跃的游戏对象，比如叫做GameWorld或GameObjectManager。

14.6.1.2 Update()函数的责任

略，只是个反例

14.6.2 性能限制及批次式更新batched update

批次式更新：相较于在每个游戏对象中分别更新所有相关的子系统内容，在主循环中统一进行子系统任务的循环更加高效。前者是游戏对象的遍历在外而子系统遍历在内，后者是游戏对象的遍历在内而子系统遍历在外。

·注意：批次式更新的情况下，并非不会再对游戏对象进行每帧更新，部分游戏对象的属性和配置仍然可以进行每帧赋值和修改，只是不会运行子系统计算结果而已。

优点：p681/705

14.6.3 对象及子系统的相互依赖

关联游戏对象之间、子系统（如动画和物理）之间存在依赖性

14.6.3.1 分阶段更新

有时有的游戏对象每个主循环需要更新1次以上，其更新时间点往往是分阶段的或参照其他子系统更新时间。对于这种对象只能按对象遍历来进行更新，最好可以维护一组数据结构，在特定阶段只遍历有“需求”的那些对象。

14.6.3.2 桶式更新p683/707

依赖树dependecy tree

含义：按照依赖层级关系，从底层到顶层逐层进行更新，同一层中的对象采用批次式更新，不同层的对象只能按照层级顺序更新。

14.6.3.3 对象状态及“差一帧”延迟

介绍一种延迟问题，略

14.6.3.4 对象状态缓冲state buffering

采用两组状态数据交替更新，以确保任意时间内至少有上一帧数据全局一致。

14.6.3.5 加上时戳time stamp

利用时间戳标记来解决状态数据一致性问题

14.6.4 为并行设计parallelism

14.6.4.1 使游戏对象模型本身并行

游戏对象模型本身全部并行和完全相互访问难以实现，最好使用消息传递和部分并行的设计

14.6.4.2 与并发的引擎子系统接口

异步批次式处理所需要解决的问题：p689/713

## 14.7 事件与消息泵

14.7.1 静态函数类型绑定带来的问题p691/715

函数绑定：指调用函数时会运行哪个函数实现，这个含义显然暗示函数的调用语句和实际实现并不一定是确定的

后期绑定late binding：在运行时才能确定绑定关系的情况，一般为虚函数和多态

静态函数类型：给定一个对象类型，就能知道应该调用函数的哪个实现

14.7.2 把事件封装成对象

事件event有时在相同的语境下也称为消息或命令。

把事件封装的好处：

单个事件处理函数：无需每个事件一个

持久性：事件参数可以永久保存

盲目转发：转发事件的对象不必了解事件

14.7.3 事件类型

两种事件类型编码方法的比较：p692/716

·使用枚举或常量

·使用字符串或散列值

14.7.4 事件参数

两种一般方法：p693/717

派生类和硬编码

定义variant数据对象来存储参数

14.7.4.1 以键值对作为事件参数

略

如何解决值的数据类型不统一的问题？

14.7.5 事件处理器

两种实现方式：

·使用单一虚函数和事件类型参数以及分支语句

·使用多个虚函数对应每个事件

·消息映射系统message map：MFC中有应用

14.7.6 取出事件参数

两种方法：

·在事件类的派生类中来实现具体的取参数API

·在处理器中手工取出数据，并且转化为合适的类型。

14.7.7 职责链chain of responsibility

按照对象关系来转发事件，转发事件的顺序一般是提前确定的

14.7.8 登记对事件的关注

加入事件和对象之间的过滤机制

兴趣管理interest management：在分布式互动仿真distributed interactive simulation中的同义术语

14.7.9 要排队还是不要排队

14.7.9.1 事件排队的好处

①控制事件处理的时机

②往未来投递事件的能力

事件的时间优先级管理伪代码p700/724

③事件的优先次序

事件的时间需求和优先级同时决定事件的处理次序

14.7.9.2 事件排队带来的问题

①增加了事件系统的复杂度

②深度复制事件及其参数

事件必须被深度复制（特指其包含和引用的其他参数对象）才能存于队列

③为队列中的事件做动态内存分配

循环队列circular queue和重定位relocation

④调试困难

竞态条件race condition

14.7.10 即时传递事件带来的问题

·非常深的调用堆栈，可能耗尽资源

·多个即时事件之间的并行矛盾

14.7.11 数据驱动事件/消息传递系统

基于简单地从硬编码选项中选取、脚本或可视化编程等几种设计

14.7.11.1 数据路径通信系统

用数据流来模拟对象之间的通信，而取消事件类型等一般事件机制

p705/729

14.7.11.2 视觉化编程的优缺点

简单易用的背后实际上隐藏了复杂和麻烦，区别只是不同人面对的麻烦程度此消彼长。

## 14.8 脚本scripting language

脚本语言可用于开发，或者定制mod一个现存的游戏

14.8.1 对比运行时与数据定义

脚本语言分为两种：

数据定义语言data-definition language：用来定义数据

运行时脚本语言runtime scripting language：类似程序语言用来定义方法

14.8.2 程序语言特性

直译式interpreted：运行时直接解析，或——通过编译为字节码byte code在虚拟机上执行。虚拟机相当于cpu，字节码相当于目标机器语言。

编译式compiled：原生cpu运行编译后的机器码

声明式declarative：只描述目标，不规定具体实现，如markup language

命令式imperative：每个指令执行一个操作或是读写内存，如C/C++

函数式functional：程序由一组或一串函数所定义

过程式procedural：类似函数式

面向对象：略

反射式reflective：数据类型等元信息可以在运行时获得，如C#

14.8.2.1 游戏脚本语言的典型特性

·直译：脚本基本都是虚拟机直译，具有可移植、快速迭代的特点

·轻量：多数游戏脚本语言都是为嵌入式系统设计的，虚拟机结构简单轻便

·支持快速迭代：无需重新编译，更快看到变化

·方便易用：可定制化

14.8.3 一些常见的游戏脚本语言

游戏脚本语言显然可以自行开发定制，但是成本较大，多数情况下不如借用已有的脚本语言改进。

14.8.3.1 QuakeC

基于C语言的简化

不支持指针或任意定义的struct

直译式、命令式、过程式

14.8.3.2 UnrealScript

基于C++

使用对象参考而不是自由指针

直译式、命令式、OOP

①扩展类层次结构的能力

最有名的功能，可以纯粹用脚本实现新的派生类

②Latent函数

一种特殊的与事件或时间互动的函数

③方便与UnrealEd联系

④多人游戏的网络复制

远程代理remote proxy：一个轻量版本的对象，用于存储在其他机器中，只在本机中存储完整内容

14.8.3.3 Lua

优点：

·健壮及成熟：

·优良文档：

·卓越的运行时性能：字节码执行速度快，高效

·可移植性：

·内存占用少：直译器和程序库大约350kb

·强大可扩展：meta-mechanism元机制无限扩展功能

·免费：开源

14.8.3.4 Python

优点：

·代码容易阅读

代码缩进indentation

·反射性语言

·面向对象

鸭子类型duck typing——支持多态而无需使用继承，即动态运行时绑定

·模块化

·优化的异常错误处理

·全面的标准库和第三方模块

·可嵌入

·详尽文档

14.8.3.5 Pawn/Small/Small-C

Pawn是这系列语言的最新版本名称，轻量、动态类型，类C语言。支持有限状态机

14.8.4 脚本所需的架构

·回调脚本：

脚本仅作为一种可定制的函数和方法而存在

钩子函数hook function：也称回调callback

·事件处理器脚本：

类似于回调

·以脚本扩展游戏对象类型或定义新类型

更广泛意义上的定制

·组件或属性脚本

基于组件或属性的游戏对象模型中，有必要使用脚本创建新组件或属性对象

·脚本驱动的引擎系统

即游戏对象模型等核心内容完全由脚本编写并在运行时翻译，除了部分底层组件外

·脚本驱动的游戏

原生代码仅作为库调用而存在

14.8.5 运行时游戏脚本语言的功能

大部分游戏脚本语言的主要功能是为了定制游戏性和游戏对象

14.8.5.1 对原生编程语言的接口

脚本语言和原生语言之间最好实现双向通信和调用

举例（伪代码）：DC（Scheme）p716/740

14.8.5.2 游戏对象句柄

脚本调用游戏对象的方法，绝大多数脚本不支持指针

14.8.5.3 接收及处理事件

脚本处理事件的机制

14.8.5.4 发送事件

略

14.8.5.5 面向对象脚本语言

①脚本中定义类

只要脚本支持新的数据结构，并且可以储存数据和函数，就等价于支持oop

②脚本中的继承

脚本类派生脚本类是支持oop的脚本常见功能，但并非必须。然而原始类和脚本类之间的继承较少有引擎实现，如UnrealScript

③脚本中的合成/聚合

另一种替代继承或类似功能的机制

14.8.5.6 有限状态机脚本

略

14.8.5.7 多线程脚本

略

## 14.9 高层次的游戏流程

游戏流程或任务系统的简单介绍，利用有限状态机管理流程的进度

**第五部分 总结**

# 第15章 还有更多内容吗？

## 15.1 一些未谈及的引擎系统

15.1.1 音频

微软XACT音效制作工具

15.1.2 影片播放器

略

15.1.3 多人网络

深入阅读P730/754

## 15.2 游戏性系统

15.2.1 玩家机制

略

15.2.2 摄像机

注视摄像机look-at camera：聚焦在一个点，可聚焦或拉远

跟随摄像机follow camera：第三人称、载具视角、平台游戏，摄像机绑定在某个对象上

第一人称摄像机first-person camera

即时战略摄像机RTS camera：只能水平移动，或围绕一点旋转方向，游戏性并不需要完全的自由的相机

电影摄像机cinematic camera

15.2.3 人工智能

·游戏中的人工智能和真实的人工智能有一定区别，和真实物理效果一样，游戏的目的是在乎玩家体验。即便不够真实的智能，玩家可以接受即可。

深入阅读p732/756

15.2.4 其他游戏性系统

略

皮毛外壳fur shell p384

光线追踪式渲染和光栅化管道

MLAA DLAA FXAA SRAA p401/425

几何着色器功能

directX11

外壳着色器hull shader

域着色器domain shader

计算着色器compute shader

次序无关透明order-independent transparency，OIT技术：包括深度剥离depth peeling、alpha至覆盖掩码转换alpha to coverage，片段链表fragment linked list等。

BSP树，碰撞检测，构造实体几何constructive solid geometry，CSG。剔除与几何排序

视差贴图和浮雕贴图，POM和四叉树位移贴图法QDM

双抛物面环境贴图dual paraboloid

动画纹理

UML图

[1]注释84 p433/457

动画键

IK计算原理

JSON 序列化 DRY规则

C++ 使用branch table 优化 switch-case

《游戏开发者杂志》

Steve Maguire 《Writing Solid Code》老书

《代码大全code complete》

Scott Meyers《More Effective C++》

Eric Lengyel著作 游戏中三维数学

《Modern C++ Design》 Andrei Alexandrescu ——Loki实现

Hoppe 2002 《Geometry Images》用位图存储网格数据

The Art of Multiprocessor Programming, Maurice Herlihy & Nir Shavit《多处理器编程的艺术》金海，胡侃，机械工业出版社

网络游戏的知识[3]

高级光照和全局光照[8]