Cha1-初识神兵：为什么是 Unity

在上一课的内容中，我们一起简要了解了游戏引擎的诞生和发展。

在本课的内容中，我们将一起来了解一下游戏引擎的基本构成。

虽然如今绝大多数的开发者已经不需要学习如何从零开始创建一个属于自己的游戏引擎了，但是了解一下游戏引擎的基本架构是非常有必要的。

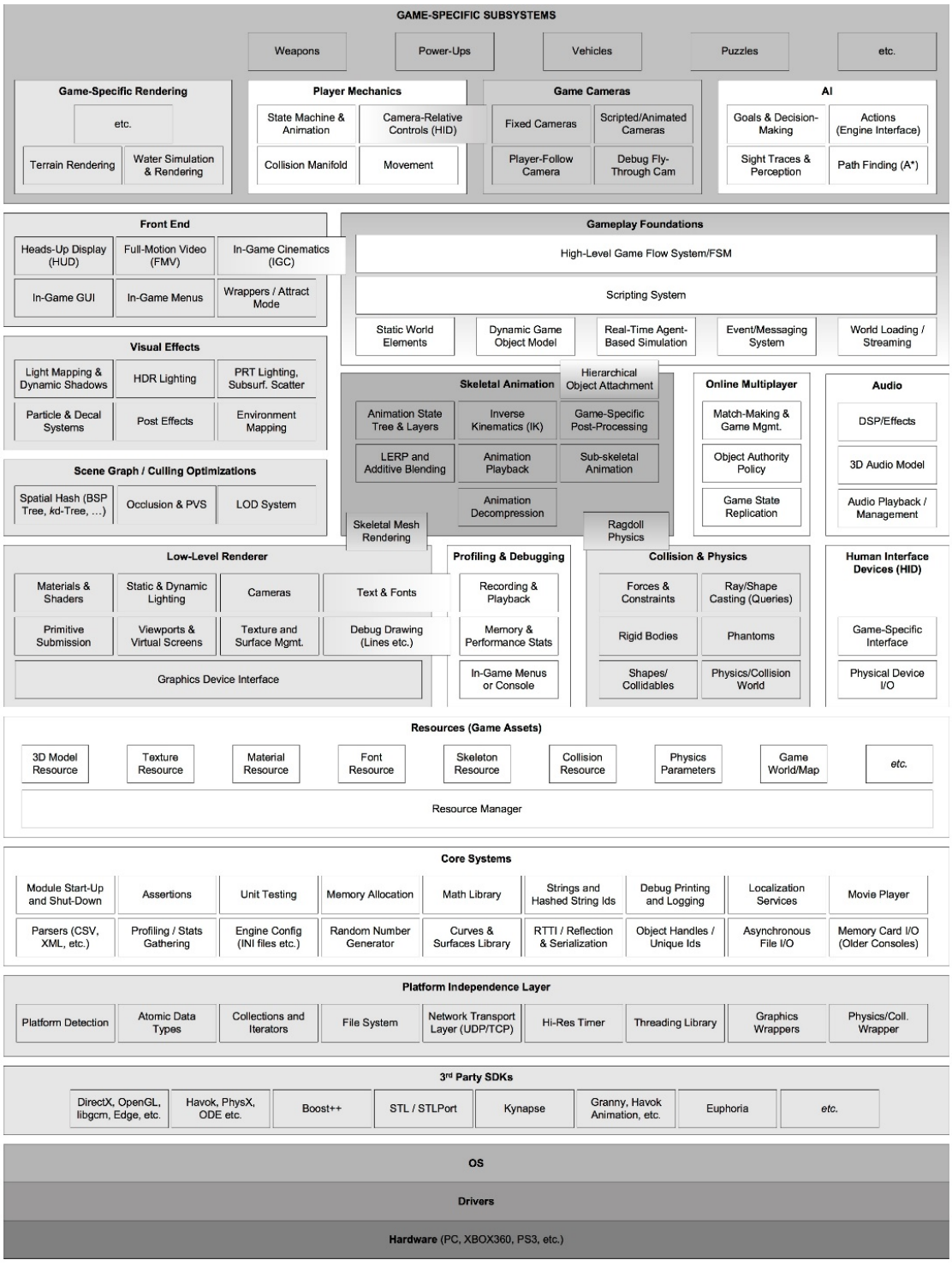
个人微信号：iseedo

微信公众号：vrlife

005- 游戏引擎的基本构成

在由 Jason Gregory 撰写的非常经典的《游戏引擎架构》(Game Engine Architecture)一书中，将游戏引擎分为工具套件和运行时组件两大部分。考虑到本系列课程主要面向刚入门的开发者，所以在这里仅作简单的介绍。

下图显示了一个典型 3D 游戏引擎的基本构成：



游戏引擎架构图

需要特别说明的是，上图显示的是游戏引擎在整个游戏体验环节中所处的位置，但是并不意味着游戏引擎本身就包含了以上的所有元素。

有童鞋要说了，看不懂上面的 e 文怎么破？我的建议是，尽量还是提升下自己的英语水平，特别是阅读和听力，对于如今的开发者仍然是非常重要的。当然，下面也会对以上的架构图做一个简单的介绍~

（1）硬件层（Hardware）

该层代表用于执行游戏的游戏主机系统，包括基于 Windows 或 Linux 操作系统的 PC 设备、基于苹果 Mac 操作系统的 Mac 电脑，微软的 Xbox 系列游戏主机、索尼的 PS 系列主机和掌机、任天堂的游戏主机和掌机、现在主流的 iPhone 或 Android 智能手机及平板设备，以及未来可能成为主流的 AR/VR 独立一体机设备。由于 PC、游戏主机、掌机、移动设备和 AR/VR 独立一体机的硬件架构具有相当大的差异，对于游戏引擎的设计也会有所影响。

（2）设备驱动(Drivers)

该层代表由操作系统或硬件厂商提供的驱动程序，用于管理硬件资源，同时将操作系统和上层引擎隔离开来，使得上层的软件无需理解不同硬件版本的通信差异。

驱动程序相当于硬件和操作系统之间的接口，操作系统只有通过这个接口才能控制硬件设备的工作。如果没有正确安装硬件设备的驱动程序，那么硬件就形同虚设。

驱动程序被称作“硬件的灵魂”，在上古时代，当操作系统安装完毕后，首先要做的就是安装一大堆的硬件设备的驱动程序。当然，诸如硬盘、显示器、光驱、USB 接口、键盘、鼠标等基础硬件是不需要安装驱动程序的，但是对于显卡、声卡、扫描仪 、摄像头、网卡、游戏手柄控制器等就需要安装驱动程序。

具体来说，驱动程序指的是添加到操作系统中的一小段代码，其中包含了硬件设备的信息，计算机根据此信息和硬件设备进行通信。比如当操作系统需要使用声卡播放音乐时，会先把软件生成的相应指令发送到声卡驱动程序，而声卡驱动程序会把指令翻译成声卡可以理解的电子信号指令，才能正常播放音乐。

当然，对游戏玩家和游戏开发者来说，最熟悉的还是显卡驱动程序，特别是 Nvidia 的显卡驱动程序更新频率非常高。Nvidia 还推出了 Nvidia Geforce Experience 软件，可以方便的升级更新显卡驱动，从而获得更好的游戏视觉体验。

（3）操作系统（OS）

该层代表硬件之上运行的操作系统，它的主要作用是协调游戏硬件设备上多个程序的执行。

在 PC 上，操作系统是一直运作的，游戏需要和多个执行中的程序共享硬件。而在游戏主机上，操作系统充其量只是个轻量级的库，并链接到游戏的可执行文件。随着次时代游戏主机的推出（Xbox

One 和 Playstation 4)，游戏主机和 PC 开发的界线正在逐渐模糊。

除了基于 PC 的 Windows 操作系统，还有 UNIX 操作系统，基于 UNIX 的 Linux 操作系统，以及基于 UNIX 的苹果 Mac 操作系统。

当然，对于智能手机来说，目前最主流的则是支持安卓智能手机和平板设备的 Android 操作系统，和支持苹果 iOS 设备的 iOS 操作系统。

相信在不久的未来，我们还将看到专门用来支持 AR/VR 独立一体机设备的操作系统，以及支持智能机器人设备的操作系统。

（4）第三方 SDK(3rd Party SDK)

很多游戏引擎都会借助第三方软件开发工具包（Software Development Kit, SDK）和中间件，并提供基于函数或基于类的应用程序接口（Application Programming Interface, API)。

这些第三方的 SDK 可以用于处理数据结构及算法（如 STL/STLPort, Boost++），图形渲染（如 OpenGL,DirectX),碰撞和物理(Havok,PhysX），角色动画（Granny,Havok Animation,Edge），人工智能（Kynapse），生物力学角色模型（Endorphin 和 Euphoria），AR/VR，AI 人工智能和机器学习等。大家如果对提到的这些第三方 SDK 感兴趣，可以在网上搜索相关的介绍和技术文档。

（5）平台独立层(Platform Independence Layer)

大多数的游戏引擎需要支持不同的平台，因此在游戏引擎的架构中通常有一个平台独立层在硬件、驱动程序、操作系统和其他第三方 SDK 之上，从而将引擎的其他部分和底层平台隔离。

（6）核心系统(Core Systems)

这里所谓的核心系统不是指引擎的核心功能，而是指引擎中一些有用的软件。核心系统层通常支持断言（Assertion)、内存管理、数学库、自定义数据结构及算法等。关于这些高大上的名词，这里就不展开解释了，毕竟我们的第一个目的是先学会怎么用，而不是学会怎么自己制作游戏引擎~

（7）游戏资源管理(Resources/Game Assets)

什么是游戏资源呢？为了让玩家有最理想的视听感受（对于 AR/VR 游戏或应用可能还有其它感官的感受)，相比其它类型的软件，游戏最大的特点就是拥有丰富的外部资源，比如酷炫的视觉效果，栩栩如生的 3D 人物模型和场景模型，引人入胜的背景音乐和互动音效，充满艺术感和情怀的字体等等。相信在不久的将来，我们甚至可以在游戏中提供视觉和听觉之外的资源，比如触觉感受，温度觉，气味等等。

每个游戏引擎都需要有一个特定形式的游戏资源管理器，提供特定的接口，以便访问各种类型的游戏资源，如 3D 模型资源、纹理贴图资源、材质资源、材质资源、骨骼资源、碰撞资源、物理参数、游戏世界/地图、音效资源、字体资源等等。

（8）视觉渲染

在游戏引擎中，视觉渲染部分是最重要的组件，没有之一。其中又包括低阶渲染器(Low-level renderer)、场景图/剔除优化、视觉效果（粒子特性、光照贴图、动态阴影、全屏后期处理效果、颜色校正等)、前端（HUD、游戏内置图形用户界面、游戏内置菜单、游戏内置全景视频等）等。

作为游戏引擎最为核心也是最初始的组成部分（再次强调，没有之一），实时三维视觉渲染的好坏是判断一款游戏引擎功力深厚最核心的指标，没有之一。实际上，在过去相当长的一段时间里，Unity3d 引擎被视为 baby engine 的原因就是内置的实时三维视觉渲染系统并不算特别强大。对于 Unity3d 早期的开发者，只能用它来开发移动设备平台上的所谓休闲游戏，但是对于 3A 级别的游戏大作则一直无能为力。这也是为什么业内人士往往把开发者对 Unity Shader 的精通程度作为鉴定 Unity3d 功力的第一标准。

但是要深入掌握游戏的实时三维视觉渲染系统并不容易，它需要我们掌握 3D 数学、线性代数、数值计算、计算机图形学，OpenGL/DirectX、引擎中的工具管道架构，以及运行时的渲染 API 等等。

幸运的是，对于初学者来说，我们只需要了解如何基于现有的引擎实现所需要的效果。但是如果学有余力，或是有心进行更深入的研究，那么就不得不掌握以上所提到的诸多基础知识。

实际上，对于听起来高深莫测的 AI 人工智能开发也是类似的情况。如果我们只需要知道如何基于已有的工具来满足现实生活中的工作需求，其实并没有那么的复杂。但是如果想具备完善甚至是优化工具的能力，那么就需要更进一步。

一句话来总结，如果只是想要开发一款好玩的游戏，对游戏的视觉效果没有过高的追求，那么其实并不一定需要对游戏引擎的底层有太深的造诣，比如最近火上天的《太吾绘卷》和《中国式家长》，还有早年风靡一时的《愤怒的小鸟》。但是如果想要打造一个以假乱真惟妙惟肖的虚拟数字世界，或是对游戏引擎工具做一些改造工作，那么就需要深入研究这些底层的渲染知识。

（9）性能分析与调试(Profiling&Debugging)

游戏引擎中通常会内置性能分析和调试工具，如内存分析、代码调试等，以便让开发者更方便的对游戏的性能进行优化。需要强调是，对于游戏的性能优化将耗费游戏开发中相当多比例的时间和精力，同时也是决定玩家能否拥有良好体验的关键因素。

游戏的卡顿对于游戏体验是非常致命的，任何一个有过游戏体验的玩家都会明白帧速的重要性。Nvidia 和 AMD 每年在显卡硬件上无止尽的投入重金，一个很重要的原因就是在提升游戏画质和视觉呈现效果的同时，让游戏运行变得更加顺畅。

对于 AR/VR 游戏或应用内来说，性能优化则显得尤为重要。特别是对于 VR 游戏，性能优化是影响玩家用户体验的最关键因素。因为对于 VR 游戏来说，需要以高于 90fps-120fps 的帧速在两个分屏幕中显示游戏画面，而如果优化不给力，很可能导致玩家在 5 分钟之内就产生恶心或晕眩的感觉。

（10）碰撞和物理(Collision&Physics)

游戏引擎中内置的碰撞检测和物理系统用于模拟真实世界中的物理法则，更常用的是刚体动力学模拟。那么什么是游戏中的物理法则呢？其实很简单，对大多数游戏来说，只要掌握经典物理学中的牛顿力学就好。特别是对重力和牛顿三大定律的模拟，是游戏物理系统中最关键的一环。至于经典物理学中跟电磁相关的麦克斯韦方程，以及跟分子运动相关的统计热力学，通常不是关注的焦点。至于更高深晦涩的量子力学和相对论，至少目前的商业引擎并不会涉及到。

因此，简单来说，游戏引擎中的物理系统目前来说指的就是经典物理学中的牛顿力学。

如果初中物理已经忘光光的同学不妨百科一下：

https://en.wikipedia.org/wiki/Newton's_laws_of_motion

https://baike.baidu.com/item/%E7%89%9B%E9%A1%BF%E5%8A%9B%E5%AD%A6/650147

经典牛顿力学是游戏引擎对物理法则的第一个重要简化假设，但实际上游戏中的物理系统还有另外一个重要的简化。到目前为止，大多数的游戏引擎只关注经典动力学中的经典刚体动力学。

什么是刚体？其实很简单，在虚拟的游戏世界中，通常我们所模拟的物体对象是完美的固体，不会变形（液体和气体是经典的反例）。换句话来说，物体的形状是固定不变的，这种假设可以很好的配合碰撞检测系统，从而大幅简化模拟固体动力学所需要的数学计算。

因为有了以上的两个简化，游戏世界中的物体运动 就符合多个约束（constraints），其中最常见的约束是非穿透性，也就是说两个物体不能互相穿透。如果发现物体之间互相穿透时，就需要物理系统提供真实的碰撞响应。这也是为什么游戏中的物理系统和碰撞检测系统往往是紧密联系在一起的。

除了简单的碰撞，游戏中的物理系统还可以允许开发者设置其它类型的约束，从而更好的模拟物体之间的真实互动，比如 hinge（铰链），joint(关节)，ragdoll（布娃娃）等。

目前使用最为广泛的物理引擎是 Nvidia 的 Physx 物理引擎，Unity3d 和 UE4 引擎中内置的物理引擎都是 PhysX。除此之外，很多游戏工作室在诸多的游戏大作中采用了 havok 或 Bullet 物理引擎。

值得一提的是，在 2019 年 3 月的 GDC 2019 大会上，Havok 的 boss 出现在了 Unity3d 的 Keynote 大会上，并宣布将与 Unity 展开更深入的合作，看来 Havok 的强悍已经彻底打动了 Unity3d 引擎团队。

PhysX、Havok 和 Bullet 被公认为最知名的三大物理模拟引擎系统。

关于三大物理引擎的更多信息介绍，可以百科一下：

https://baike.baidu.com/item/PhysX/9272519?fr=aladdin

https://baike.baidu.com/item/Havok

https://baike.baidu.com/item/Bullet/8198766?fr=aladdin

当然，基于经典力学的刚体动力学模拟是传统物理引擎最擅长的，但是其局限也显而易见。

在最近的几年中，行业专家、数学家、物理学家和游戏引擎开发者正在尝试研发超越经典刚体动力学的物理引擎功能，比如 DMM 引擎开始支持对形变体的支持。UE4 引擎对布料、头发的模拟下了大量的功夫。而更深入的研究则针对通用的流体动力学模拟。

（11）骨骼动画（Skeletal Animation）

很多游戏中存在活生生的角色，比如人类、动物、卡通角色和机器人等等。对于此类角色，需要使用动画系统让他们在游戏中变得活灵活现。

传统意义上的动画是把人物或物体的表情、动作和形态变化分解成很多瞬间的图画，再使用摄像机（或是数字摄像机）连续拍摄生成一系列的画面，从而让人类的视觉感觉连续变化。其基本原理和电影电视一样，都是视觉暂留。所谓的“视觉暂留”，指的是人眼在看到一幅画或者一个物体时，在 0.34 秒内所形成的视觉印象并不会消失。利用视觉暂留原理，在一幅画还没有消失前就开始播放下一幅画，就会给人造成流畅的视觉变化效果。

在远古时代的动画时代，曾经有过手绘动画的存在。简单来说，那个时代的动画是用一帧帧相似度比较高的画面连续播放而生成的。经典的手绘动画包括早年的迪士尼系列，中国的传统水墨动画，日本宫崎骏的动画作品等。到了后来的 Flash 时代，我们使用顶点动画中所谓的关键帧让画面看起来具有连贯性，但基本原理还是视觉暂留 。如今大家仍然很熟悉的 gif 动图，也还是基于这个原理。

所谓的顶点动画，其本质就是让每帧动画对应人物或角色模型特定姿态的一个快照，然后通过在关键帧之间插值计算的方法，引擎可以得到平滑的动画效果。

在 Unity3d 的早期版本中，主要也是基于顶点动画的关键帧技术来生成常规的动画，其动画系统被称之为 Animation。但是这类动画对于 3D 的人物角色来说是远远不够的。

从 90 年代开始，以皮克斯为代表的美国动画巨头开始采用计算机技术替代传统的手绘动画。它的基本原理是为角色设置通过互相连接的骨骼组成的骨架结构，然后通过改变骨骼的朝向和位置为模型生成动画。

通过骨骼动画系统，可以更轻松创建人形角色的动画，并让模型具备更灵活复杂的姿态。部分引擎还支持面部动画系统，可以通过音位和情绪来修改骨骼集合，从而呈现出栩栩如生的面部表情和嘴部动作。

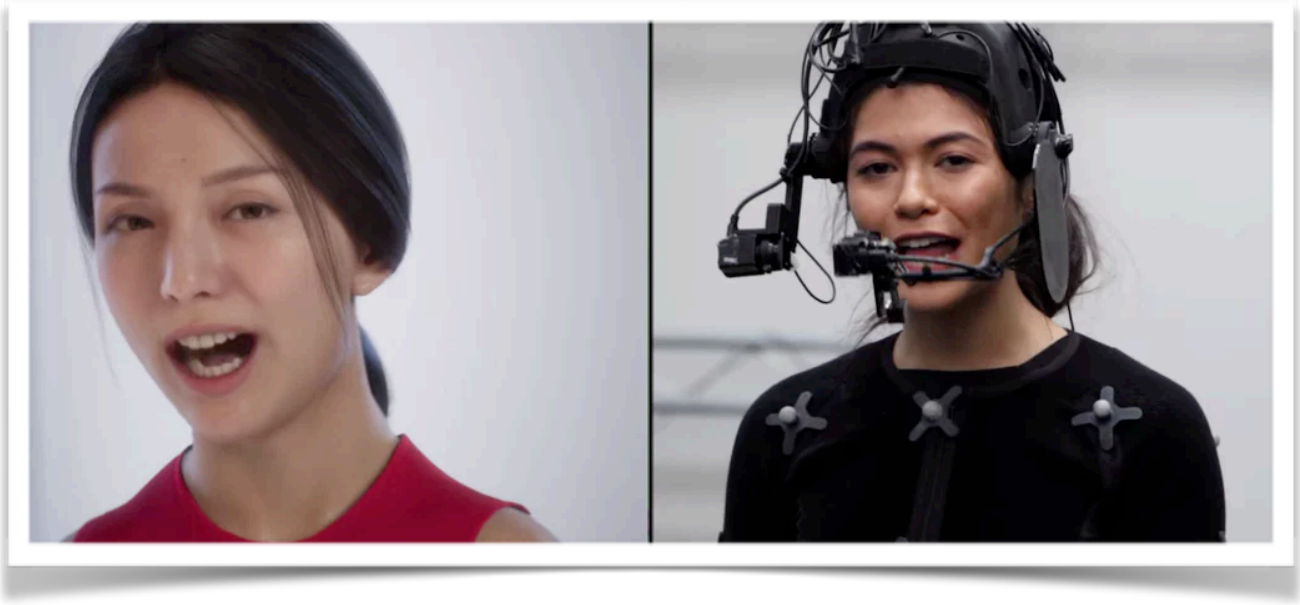
除此之外，使用骨骼动画系统，还可以配合动作捕捉设备和软件，将通过动作捕捉所获取的真实人类的角色动作和表情重现在游戏世界之中。

几乎所有的引擎都支持顶点动画，但只有部分引擎才支持骨骼动画。幸运的是，最新版本的 Unity3d 系统是支持骨骼动画的，其动画系统被称为 Mecanim。

（12）人机接口设备(Human Interface Devices)

任何一个游戏都需要玩家和游戏世界产生互动，而玩家的输入需要使用人机接口设备来实现。

为游戏而设的人机接口设备实在是太多了，最常见的就是我们人人都用过的键盘和鼠标。而其它交互设备则包括摇杆、游戏手柄、轨迹球、Wii 控制器、Sony PS Move、Xbox controller、模拟方向盘、鱼竿、跳舞毯、电子吉他。而对于新兴的 AR/VR 设备来说，还包括了 iPhone 和安卓设备中的加速计和触摸屏、力反馈和震动反馈设备、手柄控制器、仿真枪、跑步机等等。



此外，未来的终极人机接口可能就是《黑客帝国》里面的脑机接口了~

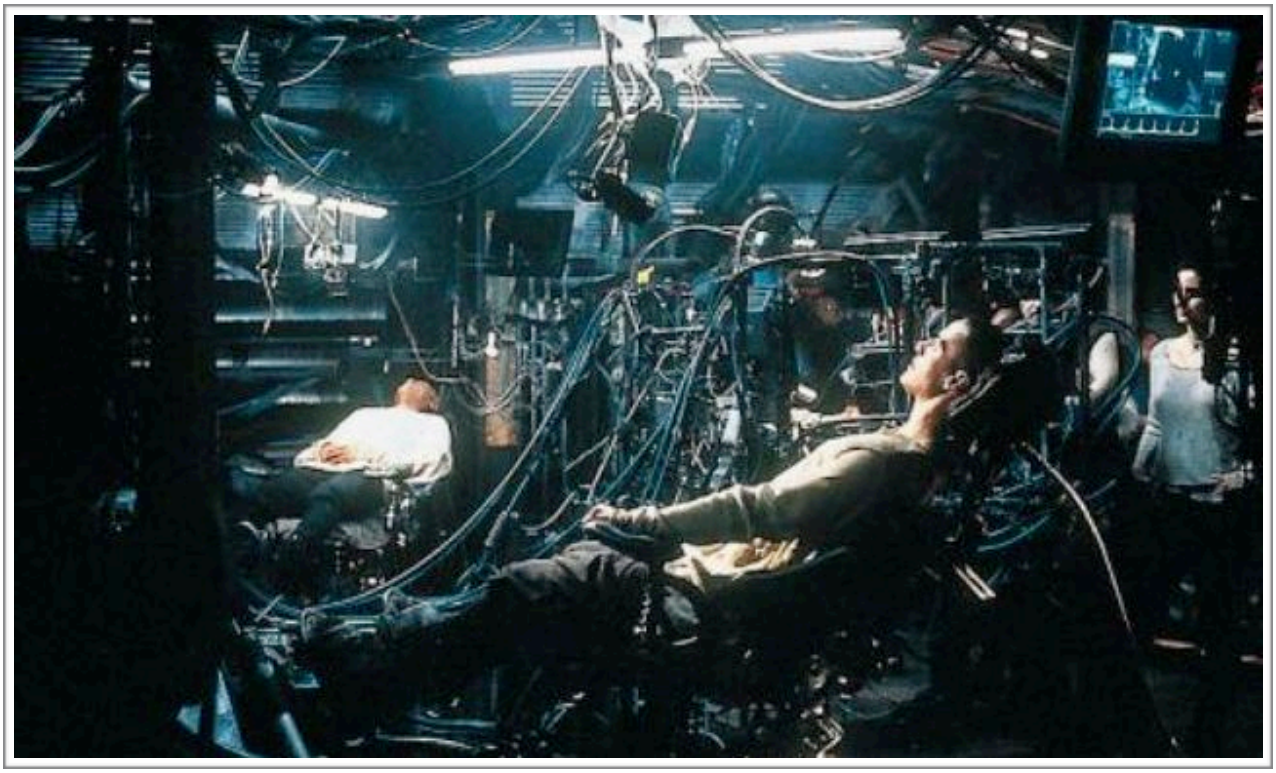
（13）多人在线（Online Multiplayer）

对于支持多人同时在线的游戏，游戏引擎必须提供对应的网络联机功能。

网络联机功能又分为局域网联机和互联网联机，对目前的大多数游戏来说，联机功能指的都是互联网联机。

大家熟悉的联网游戏包括《魔兽世界》、《王者荣耀》、《绝地求生》等。

是否联网并不是一个游戏吸引人的唯一要素，比如《GTA5》、《太吾绘卷》、《使命召唤》等经典单机游戏也得到众多赞誉。但是联网特性对于多人实时在线游戏来说则是不可或缺的，因为玩家之间的互动和社交连接都需要基于这一点。



并非所有的游戏引擎都提供网络联机功能，比如早期版本的 Unity3d 并不提供网络相关的子系统，此前人们不得不使用第三方的插件或自主研发网络功能模块。但幸运的是，最新版本的 Unity3d 中已经提供了 UNET 网络功能模块，可以大大提升联网功能的开发效率。

（14）音效(Audio)

好的游戏背景音乐和音效可以大大提升游戏的吸引力，因此在绝大多数游戏引擎中都会提供音效系统。

对于部分游戏来说，人们甚至仅仅因为其中的背景音乐和互动音效就深深沉迷其中。在游戏音乐性方面表现突出的游戏推荐如下：《仙剑奇侠传》、《魂斗罗》、《大话西游 2》、《梦幻西游》、《极品飞车》、《反恐精英》、《寂静岭 2》、《最终幻想》等。

（15）Gameplay Foundations（游戏性基础）

除了画面、声音和对真实世界的模拟，任何一款游戏都需要具备特定的游戏规则，在游戏引擎中即为游戏性基础系统。其中包含了游戏世界和游戏对象模型、事件系统、脚本系统等。

虽然传统的欧美 3A 大作往往依靠令人震撼的游戏视觉效果吸引了众多玩家，但也有很多经典游戏主要不是拼颜值，特别是任天堂家的众多游戏，最突出的就是游戏性。

主要依靠游戏性而非画质取胜的游戏推荐如下：

《我的世界》、《太吾绘卷》、《超级马里奥》、《口袋妖怪》、《塞尔达传说》、《愤怒的小鸟》、《植物大战僵尸》、《绝地求生》、《文明》等。

当然，需要补充一点的是，并不是说游戏画质不重要，而是说如果游戏性本身足够吸引人，画质其实就没有那么的重要了。

（16）Game-Specific Subsystems（游戏专用子系统）

该系统处于低阶引擎组件之上，用于实现游戏本身的各种特性，包括玩家机制、游戏摄像机、武器系统、载具等。

除了用于实现游戏核心内容的运行时组件，游戏引擎通常还提供了一些工具套件，用于丰富游戏内容。

游戏引擎需要使用各种形式的数字内容，比如 3D 模型、纹理贴图、骨骼动画、音频文件等。因此，在游戏引擎中通常需要内置专用的资源管道，以便从外部的 DCC（Digital Content Creation，数字内容创作）软件中导入相关资源。

很多游戏中使用了酷炫无比的粒子特效，虽然有类似 Houdini 这种第三方工具，但游戏引擎并不能支持这种第三方工具所制作的所有效果。因此，大多数游戏引擎都有内置的粒子特效编辑工具。比如 Unity3d 引擎中就内置了名为 Shuriken 的粒子系统。

此外，虽然使用 3dsmax 或 Maya、Blender 等 3D 建模软件可以导出所需的游戏场景，但大部分的商用游戏引擎也内置了世界编辑器，既可以作为资源管理工具，也可以用于创建游戏世界和场景。

好了，关于游戏引擎的基本构成就先介绍到这里了。

在下一课的内容中，我们仍然会继续做功课，对主流的游戏引擎做一个简单的对比分析~

让我们下一课再见