作者：Luke Hodorowicz

  翻译：乐晨光 (2001/9/28)

  本人对3D也不甚了解，译文动机一则是内容所致兴致昂然，二则锻炼英译中技能。由本人水平及经验有限，文中绝对不乏大量误解与误译，亦恳请读者指出，得以一同提高。

  佳文须共赏，也欢迎大家自由转载 :)

  Introduction （简介）

  让咱们谈谈你如何撰写一份提供优雅性能的3D引擎。你的引擎需要提供的包括：曲面(curved surfaces)、动态光线(dynamic lighting)、体雾(volumetric fog)、镜面(mirrors)、入口(portals)、天空体(skyboxes)、节点阴影(vertex shaders)、粒子系统(particle systems)、静态网格模型(static mesh models)、网格模型动画(animated mesh models)。假如你已经知道如何以上所述的所有功能顺利工作，你也许便能将那些东东一起置入到一个引擎当中。

  等等！在你开始撰写代码前你必须先构思一下如何去架构你的引擎。多数来讲，你一定是迫切地渴望去制作一个游戏，但如果你立即投入便开始为你的引擎撰写代码后，你一定会觉得非常难受，开发后期你可能会为置入新的特效与控制而不得不多次重写大量的局部代码，甚至以失败而放弃告终。花一点时间好好地为你引擎深谋远虑一番，这将会为你节省大量时间，也少一点头痛。你一定不会急切地去架构一个巨型的工程；或许你也会在引擎未完成时而干脆放弃它，然后去干的别的什么事儿。好了，当你掌握学习你所需知识的方式之前，也许你还不能完成那些事儿。将设计真正地完成确实是件美事，为之你会感觉更好，你将为之而耀眼！

  让我们分析一下具备完整功能的3D游戏引擎的需要哪些基本部件。首先，这为具有相应3D经验但且还需一些指引的开发者提供了一些信息。这是一些并不难且能快速掌握但是你必须应用的内容条目。为将你的工作更好地进行下去，这里将对关于“把多大的工作量”与“多少部分”置入一个游戏引擎给出一个总概。我把这些成分称为 系统（System）、控制台（Console）、支持（Support），渲染/引擎 内核（Renderer/Engine Core）、游戏介质层（Game Interface）、以及工具/数据（Tools/Data）。

  Tools/Data （工具/数据）

  在开发过程中，你总是需要一些数据，但不幸的是这并不象写文本文件或是定义一个立方体那么简单。至少，你得需要3d模型编辑器，关卡编辑器，以及图形程序。你可以通过购买，也可以在网上找一些免费的程序满足你的开发要求。不幸的是你可能还需要一些更多的工具可你却根本无法获得（还不存在呢），这时你只得自己动手去写。最终你很可能要自行设计编写一个关卡编辑器，因为你更本不可能获得你所需。你可能也会编写一些代码来为大量的文件打个包，整天面对应付成百上千个文件倒是非常痛苦的。你还必须写一些转换器或是插件将3d模型编辑器的模型格式转换成你自己的格式。你也需要一些加工游戏数据的工具，譬如可见度估算或是光线贴图。

  一个基本的准则是，你可能要为设计工具而置入比游戏本身等量甚至更多的代码。开始你总能找到现成的格式和工具，但是经过一段时间以后你就能认识到你需要你的引擎有很大的特性，然后你就会放弃以前的撰写方式。

  也许目前非常流行利用的第3方工具辅助开发，所以你必须时刻注意你的设计。因为一旦当你将你的引擎发布为opensouce或是允许修改，那也许在某天中会有某些人来应用你的开发成果，他们将其扩展或者做某些修改。

  或许你也应该花大量时间去设计美术，关卡，音效，音乐和实体模型，这就和你设计撰写游戏，工具以及引擎一样。

  System （系统）

  系统(system)是引擎与机器本身做通信交互的部件。一个优秀的引擎在待平台移植时，它的系统则是唯一需要做主要更改（扩加代码）的地方。我们把一个系统分为若干个子系统，其中包括：图形（Graphics）、输入（Input）、声音（Sound）、记时器（Timer）、配置（Configuration）。主系统负责初始化、更新、以及关闭所有的子系统。

  图形子系统（Graphics Sub-System）在游戏里表现得非常直观，如果想在屏幕上画点什么的话，它（图形子系统）便干这事儿。大多数来讲，图形子系统都是利用OpenGL、Direct3D, Glide或是软件渲染（software rendering）实现。如果能更理想一些，你甚至可以把这些API都给支持了，然后抽象出一个“图形层”并将它置与实现API之上，这将给了客户开发人员或是玩家更多的选择，以获取最好的兼容性、最佳的表现效果。

  输入子系统（Input Sub-System）需要把各种不同输入装置（键盘、鼠标、游戏板[Gamepad]，游戏手柄[Joystick]）的输入触发做统一的控制接收处理。（透明处理） 比方说，在游戏中，系统要检测玩家的位置是否在向前移动，与其直接地分别检测每一种输入装置，不如通过向输入子系统发送请求以获取输入信息，而输入子系统才在幕后真正地干活（分别检测每一种输入装置），这一切对于客户开发人员都是透明的。用户与玩家可以非常自由地切换输入装置，通过不同的输入装置来获取统一的行为将变的很容易。

  声音子系统（sound system）负责载入、播放声音。该子系统功能非常简洁明了，但当前很多游戏都支持3D声音，实现起来会稍许复杂一些。

  3D游戏引擎中很多出色的表现都是基于“时间系统”(time)的。因此你需要一段时间来为时间子系统（Timer sub-system）好好构思一番。即使它非常的简单，（游戏里）任何东西都是通过时间触发来做移动变化，但一份合理的设计将会让你避免为实现而一遍又一遍地撰写大量雷同的控制代码……

  配置系统（Configuration）位于所有子系统的顶端。它负责读取配置记录文件，命令行参数，或是实现修改设置(setup)。在系统初始化以及运行期间，所有子系统都将一直与它保持通讯。切换图象解析度（resolution），色深（color depth），定义按钮（key bindings），声音支持选项（sound support options），甚至包括载入游戏，该系统将这些实现显得格外的简单与方便。把你引擎设计得更为可设置化一些，这将为调试与测试带来更大的方便；玩家与用户也能很方便地选择他（她）们喜欢的运行方式。

  Console （控制台）

  哈！我知道所有人都乐意去更风做一个象Quake那样的控制台（console）系统。但这的确是一个非常好的想法。通过命令行变量与函数，你就能够在运行时改变你的游戏或是引擎的设置，而不需要重启。开发期间输出调试信息它将显得非常的有效。很多时间你都需要测试一系列变量的值，将这些值输出到控制台上要比运行一个debugger速度显然要快得多。你的引擎在运行期间，一旦发现了一个错误，你不必立即退出程序；通过控制台，你可以做些非常轻便的控制，并将这个错误信息打印出来。假如你不希望你的最终用户看见或是使用该控制台，你可以非常方便地将其disable，我想没人能看得见它。

  Support （支持）

  支持系统（Support）在你引擎中任何地方都将被使用到。该系统包含了你引擎中所有的数学成分（点，面，矩阵等），（内）存储管理器，文件载入器，数据容器（假如你不愿自己写，也可以使用STL）。该模块任务显得非常基础与底层，或许你会将它复用到更多别的相关项目中去。

  Renderer/Engine Core （渲染/引擎 内核）

  哈~是呀，所有的人都热爱3D图象渲染！因为这边有着非常多的不同种类的3D世界渲染方式，可要为各类拥有不同工作方式的3D图形管道做出一个概要描述也是几乎不可能的。

  不管你的渲染器如何工作，最重要的是将你的渲染器组件制作得基化（based）与干净（clean）。

  首先可以确定的是你将拥有不同的模块来完成不同的任务，我将渲染器拆分为以下几个部份：可见裁减（Visibility）、碰撞检测与反馈（Collision Detection and Response）、摄像器（Camera）、静态几何体（Static Geometry）、动态几何体（Dynamic Geometry）、粒子系统（Particle Systems）、布告板（Billboarding）、网格（Meshes）、天空体（Skybox）、光线（Lighting）、雾（Fogging）、节点阴影（Vertex Shading）和输出（Output）。

  其中每一个部分都得需要一个接口来方便地实现改变设置（settings）、位置（position）、方向（orientation）、以及其他可能与系统相关的属性配置。

  即将显露出来的一个主要缺陷便是“特性臃肿”，这将取决于设计期间你想实现什么样的特性。但如不把新特色置入引擎的话，你就会发觉一切都将变的很困难，解决问题的方式也显得特别逊色。

  还有一件有意义的事便是让所有的三角形[triangles]（或是面[faces]）最终在渲染管道里经过同一点。（并非每次的每个三角形，这里讨论的是三角形列表[triangle lists]、扇形[fans]、带形[strips]、等） 多花一些工作让所有物体的格式都能经过相同的光线、雾、以及阴影代码，这样就能非常便利地仅通过切换材质与纹理id就使任何多边形具有不同的渲染效果。

  这不会伤及到被大量被渲染绘出的点，但是一旦你不当心，它可能会导致大量的冗余代码。

  你也许最终便能发现，实现所有这些你所需的极酷效果可能只占了所有的15%左右的代码量甚至更少。这是当然的，因为大多数游戏引擎并不只是图形表现。

  Game Interface （游戏介质）

  一个3D（游戏）引擎很重要的部分便是------它是一个游戏引擎。但这并不是一个游戏。一个真正的游戏所需的一些组件永远不要将它包含到游戏引擎里。引擎与游戏制作之间的控制介质能使代码设计变得更清晰，应用起来也会更舒服。这虽是一些额外的代码，但它能使游戏引擎具有非常好重用性，通过设计架够游戏逻辑（game logic）的脚本语言（scripting language）也能使开发变的更方便，也可以将游戏代码置入库中。如果你想在引擎本身中嵌入你的游戏逻辑系统设计的话，大量的问题与大量修改一定会让你打消复用这个引擎的念头。

  因此，此时你很可能在思考这个问题：联系引擎与游戏的介质层到底提供了什么。答案就是控制（control）。几乎引擎的每一个部分都有动态的属性，而该引擎/游戏介质层（engine/game layer）提供了一个接口去修改这些动态属性。它们包括了摄像器（camera）、模型属性（model properties）、光线（lights）、粒子系统物理（particle system physics）、声效播放（playing sounds）、音乐播放（playing music）、输入操作（handling input）、切换等级（changing levels）、碰撞检测以及反馈（collision detection and response）、以及2D图形界面的顶端显示、标题画面等相关的东西。基本上来讲如果你想让你的游戏能优雅的实现这些元素，在引擎中置入这个介质层（interface）是必不可少的。

  The Game （游戏）

  在这里，我无法告诉你如何去写你的游戏。这该轮到你发挥啦。如果你已经为你那令人赞异的引擎设计出了一套出色的介质层的话，我想在设计撰写游戏过程中一定会轻松许多。

  3D游戏引擎设计是一项巨大的软件工程。一个人独立完成设计并撰写也并非不可能，但这不只是熬一两个晚上便能搞定的，你很可能会出写出几兆的源代码量。如果你没有持久的信念与激情，你很可能无法完成它。

  当然，别指望你的第一次尝试就能写出完整的引擎，挑一个比较小的项目所需的小规模引擎去实现。按你的方式去努力工作，你就能到达成功。