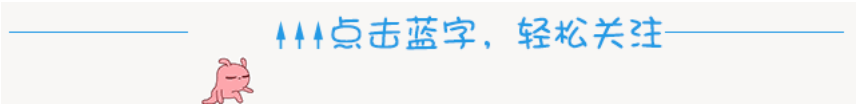
游戏网络开发(五)：浮点数的确定性

2016-10-20 18:07



简介

最近我一直在做一些有关通过确定性的帧同步方案进行网络游戏物理部分模拟仿真的研究。

通过确定性的帧同步方案进行网络游戏物理部分模拟仿真的基本想法是不再直接在网络上发送物体的位置、方向、速度等信息对物理物体的状态进行同步，而是通过在网络上发送玩家的输入信息来隐式的对模拟仿真进行同步。

这是一个非常有吸引力的同步策略，因为网络流量的大小取决于玩家输入信息的大小而不是整个游戏世界的物体状态的大小。事实上，因为这一原因，这一策略已经在即时战略游戏中使用多年了。地图上有成千上万的单位，他们有太多的状态需要通过网络进行传递。

也许你有一个非常复杂的物理模拟，有大量的刚体状态或者有布料和软体模拟需要在两台机器上保持非常精确的同步，这是因为它们的状态会影响到游戏，但是你没有办法承受通过网络发送所有的状态信息。很显然，在这种情况下，唯一可能的解决方案是尝试具有确定性的帧同步方案。

但是，我们碰到一个问题。物理模拟是使用浮点数进行运算的，由于这样或者那样的原因在两台不同的机器上面得到完全一致的浮点数运算结果被认为是非常非常困难的。甚至有人报告说同一台机器在不同的时候进行相同的运算，或者使用同一个程序的debug版本和release版本都会得到不同的结果。还有人说AMD的中央处理器会给出和Intel的中央处理器不同的计算结果，而SSE的计算结果和x87的结算结果也是不同的。这到底是怎么回事呢？浮点数的计算到底是否具有确定性？

遗憾的是，答案不是简单的“是”或“否”，而是一个含糊的答案“可能？”

1

以下是我迄今为止发现的内容：

1

如果你的物理模拟本身具有确定性，通过那种额外做一些工作你就应该能够在同一台机器上通过运行记录的玩家输入信息来重播整个过程，并且得到完全一致的结果。

2

初级编程学习

你用相同的编译器来编译的执行文件并在相同架构的机器上运行这个执行文件以及使用与平台有关的技巧的话，那么有可能能够在不同的机器上对浮点数的计算得到具有确定结果。

的机器上得到完全一致的结果是非常非常大的。

4

但是如果你愿意做大量的工作，让你的编译器“严格”符合IEEE 754编译模型以及限制你所使用的浮点数操作的集合，你或许可以让不同的编译器和不同架构的机器能都对浮点数计算得到完全一致的结果。这通常会导致显著降低浮点计算的性能。

如果你想对这一点进行讨论的话，或者想添加自己的细微差别，请写一个评论！！我认为这个问题绝对没有解决，并且对其他人关于浮点数计算的精确性以及完美重现方面的经验非常感兴趣。特别是如果你成功的在现实世界中的情况下能够在不同的体系结构和编译器上得到二进制的精确结果，请联系我。

2

下面是我在搜索过程中已经发现的资源。。

我们授权给不同客户的技术是基于具有确定性的浮点数计算（即使是在64位条件下也符合这个条件），并且自从2000年以来一直用这个方式工作。

只要你坚持使用同一个编辑器，以及同一套中央处理器的指令集，它就有可能让浮点数的计算有完全的确定性。具体的实现会依据平台而有所不同（举个例子来说，在x86、x64和PPC平台上都会有区别）。

你必须确保内部的精度设置为64比特（不是80，因为只有英特尔实现了这点），并且舍入模式是一致的。此外，在调用外部DLL后你必须检查这一点，这是因为很多DLL（Direct3D、打印机驱动程序、声音库等等）会改变精度或舍入模式而不把它们改回。

这个ISA是IEEE兼容的，如果你的x87的实现不与IEEE兼容，那么它根本就不能算x87。

此外，你在进行浮点数计算的时候不能使用SSE或者SSE2，这是因为它们欠缺让它们具有确定性的规范。”

来源：Jon Watte,GameDev.net forums

http://www.gamedev.net/community/forums/topic.asp?topic_id=499435

“我在GasPowered Games公司工作，我可以告诉你一些第一手的信息，就是浮点数计算是有确定性的。你只需要相同的指令集和编译器，当然用户的处理器必须遵循IEEE754标准，这个标准覆盖了我们所有的个人电脑和360主机。运行DemiGod、超级指挥官1和超级指挥官2的引擎也都依赖于IEEE754标准。更别提市面上所有的支持点对点对战的即时策略游戏也都是依赖于IEEE754标准的。只要你的游戏支持支持点对点对战，每个客户端需要广播它们具体在哪一次更新的时候执行了那些操作并且依赖客户端计算机通过具有确定性的浮点数计算处理器来计算模拟/物理行为的细节。

在应用程序起动的时候，我们调用如下代码：↓↓↓

ntrolfp(_PC_24,_MCW_PC)_controlfp(_RC_NEAR, _MCW_RC)

，在每个时钟周期，我们会对下列这些浮点运算单元的设置进行断言，确保它们的设置改变：

_MCW_RC) == _RC_NEAR);

有一些微软的API函数可能会改变浮点运算单元de 模型，所以在这些微软的API函数调用之后，你需要手动确保这些浮点运算单元de 模型的设置仍然在不同的机器上保持一致。设置这些断言就是为了防止有人要改动浮点运算单元的模型。

我们把编译器的浮点数模型设为Fast /fp:fast（虽然这么做并不是必须的）。

通过使用这种方法在各种中央处理器的个人电脑上，包括AMD的中央处理器和Intel的中央处理器，只要遵循了IEEE标准，我们从未遇到过问题。我们的超级指挥官或者Demigod的顾客也没有在他们的机器上遇到过任何问题，我们在这里讨论的都是有超过一百万用户的游戏（超级指挥官1+扩展包）。如果浮点运算单元不能产生完全一致结果的话，我们肯定会知道，因为这样重播或者多人模式根本就没法正常运行。

在使用一些物理方面的API的时候，我们确实遇到过一些问题，因为它们的代码没有考虑确定性或者完美重现。举个简单的例子来说，一些物理方面的API的计算器，在进行迭代中的时候，迭代的次数可能会依照中央处理器的快慢而不同，可能会在更快的中央处理器上面迭代次数越少。

Elijah, Gas Powered Games http://www.box2d.org/forum/viewtopic.php?f=3&t=1800

如果你把玩家的输入信息存储下来进行录像回放的话，那么如果在不同的中央处理器架构、编译器或者设置了不同的优化选项的机器上，存储下来的玩家的输入信息根本就没有办法正常回放。在MotoGP上，这意味着我们不能在Xbox和PC之间共享保存的录像回放文件。这也意味着，如果我们是在一个游戏的debug版本上保存的录像回放文件，那么这个录像回放文件根本就不能在release版本上正常运行，反过来也是一样。这个在通常情况下可能不是一个问题（毕竟我们肯定不会把调试版本发布出去），但是如果我们要发布一个补丁的话，这要求我们必须使用与原游戏完全一样的编辑器进行构建这个补丁。如果编译器已经想较原始版本编译的时候有所升级，也就是说我们使用一个更新的编译器来编译这个补丁的话，这可能会对足够多的东西进行改变，从而导致通过原始版本保存的录像回放文件根本无法在新的版本上正常的播放。（这个事情你不换编译器也基本不可能）。

这太疯狂了！我们为什么不能让所有的硬件按照一致的方式进行工作？好吧，其实我们可以，如果我们不关心性能的话。我们可以说：“嘿，我亲爱的硬件先生，忘掉你疯狂的乘法和加法指令，只使用最最最基本的IEEE实现就可以了“，以及”亲爱的编译器，请不要试图优化我们的代码来打扰我们“。通过这种方式，我们的程序可以在不同的中央处理器架构、编译器或者设置了不同的优化选项的机器上一致的运行，就是会非常缓慢。

Shawn Hargreaves, MSDNBlog http://blogs.msdn.com/shawnhar/archive/2009/03/25/is-floating-point-math-deterministic.aspx

“战争地带2使用的是帧同步网络模型，这个模型要求在每一个客户端上能够得到完全一致的结果，这包括了尾数的最低位也要一致，否则的话，不同机器上的模拟将开始出现分歧。在每一个客户端上能够得到完全一致的结果是很难实现的，这意味着我们只需要通过网络把用户的输入信息发送出去。所有其他的游戏状态都需要在本地进行计算。在开发过程中，我们 AMD和Intel处理器会对一些数学函数的处理有轻微的不同（这包括sin、cos、tan和他 逆运算），所以我们不得不将它们封装在没有经过优化的函数调用里面，来强制编译器 它们的运算为单精度运算。这样就足以让AMD和Intel处理器保持一致，但是这绝对是 实践才能得来的经验。

Miller, PandemicStudios http://www.box2d.org/forum/viewtopic.php?f=4&t=175

这些会导致不同步的情况在FSW2中得到了修复。我们使用了精确的浮点数运算并且在个人电脑上我们使用了Havok的浮点数运算库来代替指令多数据流。另外整数取模也会是一个问题，这是因为C++标准说这个操作是以“由实现自己定义的”（在这种情况下，多个编译器/平台会有不同的实现）。总的来说，我喜欢那些为了帧同步而开发出来的工具，这些工具能让在FSW2的代码中找到不同步变得非常简单。

Branimir Karadžić,Pandemic Studios

http://www.google.com/buzz/100111796601236342885/8hDZ655S6x3/Floating-Point-Determinism-Gaffer-on-Games

“我所知道的浮点运算不一致主要有三个原因：↓↓↓

1

运算的编译器优化。

2

像乘法累加或正弦波这样的复杂指令。

3

一些指令是x86架构特有的，并不能在其他平台上通用。

好消息是，大部分的问题来源于第三项，可以或多或少地得到自动解决。从做决策的角度出发（“我们应该投入精力来保证浮点运算一致性或者这根本就是徒劳无功的“），我会说，这肯定不会是徒劳的，如果你能举出你会从一致性中得到实际的好处，那么它是值得你付出（连续）的努力来实现这一点。

总结：请使用SSE2或者SSE，如果这一点做不到的话，请将浮点运算的CSR配置为使用64位进行中间计算，同时避免使用32位浮点数。即使只采用后者的解决方案在实践中表现还不错，只要每一个人都能对这一点有认识的话。“

Yossi Kreinin,Consistency: how to defeat the purpose of IEEE floating point

http://www.yosefk.com/blog/consistency-how-to-defeat-the-purpose-of-ieee-floating-point.html

针对这个问题的一个简短的回答是。浮点数的计算是完全具有确定性的，因为根据IEEE浮点标准，每个操作都有明确的规则，只要它符合IEEE标准，浮点数的计算就是完全具有确定性的，但是这并不意味着浮点数的计算能够在不同的机器、编译器、操作系统上能够得到完全一致的重现结果。

一个比较长的针对这些问题的答案可能可以在戴维戈德堡的《每一个计算机科学家都应该知道的浮点数运算》里面找到，这可能是关于浮点数运算最棒的参考文献了。可以跳到IEEE那一节直接看最关键的部分。

，如果你正在相同的初始输入信息上运行同样顺序的浮点数运算，那么整个结果应该是可以重现的。浮点数运算的确切的顺序可以根据你的编译器/操作系统/标准库的不同而，所以通过这种方式你可能会遇到一些小错误。

给浮点数运算单元的输入信息是大致相同但不完全一致。如果你的方法具有数值上的稳定性，那么你可以在一定的误差范围内保证重现性。如果你想要了解比这更详细的信息，可以看下上面链接中给的戈德堡的关于浮点数运算的文章（也就是《每一个计算机科学家都应该知道的浮点数运算》）或者找一篇数值分析的入门文档看一下。

Todd Gamblin, StackOverflow

http://stackoverflow.com/questions/968435/what-could-cause-a-deterministic-process-to-generate-floating-point-errors

C++标准没有为浮点数类型float、double、longdouble指定二进制的表现形式。尽管标准没有做这方面的要求，但是大多数C ++编译器在实现浮点数运算的时候都遵循了一个标准，也就是IEEE 754-1985标准，至少对于float和double类型是这样。这直接导致了这么一个事实：现代中央处理器的浮点数运算单元也支持这个标准。IEEE 754标准指定了浮点数的二进制格式，以及浮点数操作的语义。然而，不同的编译器在对IEEE 754的全部功能的支持程度是有所不同的。这就导致了程序员在用C++编写可移植的浮点数运算代码的时候可能会遇到这样或者那样的陷阱。

Günter Obiltschnig,Cross-Platform Issues with Floating-Point arithmetics in C++
http://www.appinf.com/download/FPIssues.pdf

浮点数运算是严重依赖浮点数运算单元的硬件实现、编译器的具体实现以及编译器所做的优化，还有就是系统的数学计算库（libm）。通过实验，可以看到重现性基本只能在满足使用相同的系统的数学计算库以及使用相同设置的相同编译器的时候出现。

STREFLOP Library
http://nicolas.brodu.numerimoire.net/en/programmation/streflop/index.html

浮点数编程的目标：↓↓↓

1

准确性-产生的结果能够非常"接近"正确的结果。

1

可重现性-在不同的时候运行相同的计算能够得到一致的结果。在不同的构建配置上运行相同的计算能够得到一致的结果。在不同的编译器上运行相同的计算能够得到一致的结果。在不同的平台上运行相同的计算能够得到一致的结果。

3

性能——尽可能的产生最有效率的代码。

这些选项通常情况下是存在冲突的！明智地使用编译器选项可以让你对各种权衡进行控制。

初级编程学习

、C ++编译器：浮点数运算的一致性
://www.nccs.nasa.gov/images/FloatingPoint%5Fconsistency.pdf.

!严格的重现性和一致性是非常重要的话，那么请不要改变浮点数的运行环境设置，只使-model strict（在Linux或者Mac OS平台下）或者使用/fp:strict（在Windows平台下）或

深入C++编译器手册 http://cache-www.intel.com/cd/00/00/34/76/347605_347605.pdf

在fp:strict模式下，编译器不会执行任何有可能干扰浮点数运算准确性的优化。编译器能够正确处理在赋值、类型转换以及函数调用时候的取整，并且内部的取整运算在浮点数运算单元的寄存器中持续使用相同精度。浮点数的异常语义和浮点数运算单元的环境设置敏感性会默认开启。某些优化，比如收缩，会被禁止，这是因为编译器没有办法保证在所有情况下这些优化都有正确性。

Microsoft Visual C++Floating-Point Optimization http://msdn.microsoft.com/en-us/library/aa289157(VS.71).aspx#floapoint_topic4

需要注意的是，浮点数计算的结果可能会在PowerPC和英特尔平台上不完全相同，这是因为在PowerPC上，标量和向量浮点数运算单元的核心是围绕加法乘法一体的方式而设计的。英特尔芯片有独立的乘法器和加法器，这意味着这些操作必须分别单独完成。这就意味着在计算中的一些步骤的时候，英特尔的中央处理器可能会触发一些额外的取整操作，这可能会给计算的乘法阶段引入可能会引入二分之一的舍入错误。

苹果开发者支持 http://developer.apple.com/hardwaredrivers/ve/sse.html

对于所有属于IEEE操作的指令而言（*、+、-、/、平方根、比较，无论它们是属于SSE指令还是x87指令），他们会在相同的控制设置（相同的精度控制以及取整模式、对0的截断等等）和输入下在各个平台上产生完全相同的结果。这个事情对于32位或者64位的处理器都是成立的。。。在x87指令集里面，哪些超越指令比如fsin、fcos等等，可能会根据实现的不同产生一些具有轻微不同的结果。它们指定了一个可以保证的相对误差范围，但是不是完全一致的精确。

英特尔软件网络支持

http://software.intel.com/en-us/forums/showthread.php?t=48339

我的担心来自IEEE-754在不同硬件上的实现之间的区别。我已经知道有关于编程语言的问题，在源代码这一级的实现和在汇编这一级的实现会引入细微的差别。（Mon08）现在，我比较感兴趣的差异是英特尔/ SSE和PowerPC在指令级别上的差异。

D. Monniaux on IEEE 754mailing list

http://grouper.ieee.org/groups/754/email/msg03864.html

如果要做到一致性。。。必须避免使用非754的指令，尽管它们变得越来越普遍，举个简单的例子来说，对平方根连续取两次倒数就不会正确的进行舍入运算，甚至在不同的实现上不会得到一致的结果，此外，x87的超越指令在AMD和英特尔平台的实现是有所不同的。

David Hough on 754 IEEEmailing list

初级编程学习

://grouper.ieee.org/groups/754/email/msg03867.html

]，得到可重复的结果是可能的。但是你不能在没有定义一个编程方法论的情况下做到这。而且这极端的后果远远超过任何的支持者承认——特别是，它实际上与绝大多数的并并没有有效的兼容。

http://grouper.ieee.org/groups/754/email/msg03872.html

如果我们讨论的是可行性，那么事情就会非常不同，在实际项目中期望可重复的结果是不太可能的。我们曾经为这个目标努力过，但愿让我们不用再为这个目标而努力。

Nick Maclaren on 754 IEEEmailing list

http://grouper.ieee.org/groups/754/email/msg03862.html

IEEE 754 - 1985允许实现上可以有许多变化(比如一些值的编码和某些异常的检测)。IEEE754 – 2008对于很多地方已经收紧了，但是一些实现上的变化仍然存在(尤其是对二进制格式而言)。重现性条款建议语言标准应该提供一个编写可重现程序的方法(即如何编写程序，让程序会在这个编程语言的所有实现上产生相同的结果)，并描述了需要做什么来实现可重复的结果。

Wikipedia Page on IEEE754-2008 standard

http://en.wikipedia.org/wiki/IEEE_754-2008#Reproducibility

如果想要语义几乎完全忠实于严格的IEEE – 754关于在最近值取整模式下的单精度或双精度计算，这也包括了在溢出和下溢条件下的计算，可以使用可以使用精度和选项有所限制的实现，同时编程风格上要强制操作数在两次浮点数操作之间写入内存。这会产生一些性能方面的损失。此外，由于在下溢时候会对双精度浮点数进行取整，也会有轻微的差异。

对于目前的个人计算机来说，一个更简单的解决方案仅仅是强制编译器使用SSE单元计算ieee - 754类型。然而，大多数嵌入式系统使用IA32微处理器或微控制器，而不是使用配备了这个单位的处理器。

David Monniaux, Thepitfalls of verifying floating-point computations

http://hal.archives-ouvertes.fr/docs/00/28/14/29/PDF/floating-point-article.pdf

6. 可重复性---

即使是在1985年版本的IEEE – 754标准下，如果两个标准的实现在相同的数据上执行同一个操作，采用同样的舍入模式以及默认的异常处理，那么操作的结果将是相同的。新的标准试图进一步描述如何让一个程序在不同的实现上产生相同的浮点数运算结果。标准中描述的操作都是具有可重复性的操作。

对于推荐的操作，比如库函数或约分操作是不具有可重复性的，因为他们不是所有的实现都会需要。同样的原因，对下溢的依赖以及不精确的标记也是不具有可重复性的，这是因为有两种不同的对于下溢的方法，这主要是为了保持IEEE – 754（1985）和IEEE– 754（2008）之间的一致性。舍入模式具有可重复性。可选的属性不具有可重复性。

如果要想具有可重复性，那么就要避免使用value-changing优化。这包括使用结合律和分配以及当程序员没有显式地使用操作符的时候对乘法和加法操作的融合。

er Markstein, The NewIEEE Standard for Floating Point Arithmetic

://drops.dagstuhl.de/opus/volltexte/2008/1448/pdf/08021.MarksteinPeter.ExtAbstract.1.pdf

初级编程学习

果。因为各种各样的原因，大多数程序会在不同的系统上产生不同的结果。首先，大多数程序会涉及数字的十进制表示和二进制表示之间的转换，而IEEE标准没有完全指定执行这种转换所要遵循的精度。另一方面，许多程序会使用系统库提供的基本功能库，而标准根本就对这些函数没有做任何的规定。当然，大多数程序员都知道这些特性超出了IEEE标准的范围。

许多程序员可能没有意识到，即使是一个程序，只使用数字格式和IEEE标准规定的操作就可以在不同的系统上得出不同的结果。事实上，标准的作者本意就是允许不同的实现获得不同的结果。他们的意图可以很明显的在IEEE 754标准的术语"目的"的定义中中看到："目的可以是由用户显示制定的或者隐式地由系统提供（举个简单的例子来说，表达式的中间结果或参数的过程）。一些语言会把中间计算过程的结果放到目的中去，这超出了用户的控制。尽管如此，这个标准定义了一个操作的结果，也就是目的的格式和操作数的值。"（IEEE 754 – 1985，第7页）。换句话说，IEEE的标准要求每个结果会被正确的舍入到目的的精度并放入到目的里面，但是标准并不要求该目的的精度取决于用户的程序。因此，不同系统可能会把它们计算的结果放入不同精度的目的里面，这可能会会导致相同的程序产生不同的结果（有时候这一点可能会非常显著），尽管这些系统都符合标准。

IEEE 754实现之间的不同

http://docs.sun.com/source/806-3568/ncg_goldberg.html#3098

点击一下

立即阅读相关好文章

》谈贪婪设计丨丨

丨

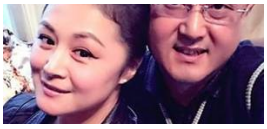
丨

核心设计分析

.....

近期热文

返回搜狐，查看更多



于月仙后事处理完毕，丈夫做法令人痛心，遗产将由他跟…

电子印章

日本女性机器人10万元一个，为何能得到全球追捧？专家…

网传于月仙骨灰在赤峰下葬，墓碑很普通，赵本山在现场…

张哲瀚是谁的儿子 张哲瀚家境背景曝光

谁在领导阿富汗塔利班？有六个关键人物

君越上市

外交部女神张璐升职，因一句神仙翻译走红，这才是年轻…

古代青楼女子的“内行话”，现在变成人们口头禅，让人难以…

我来说两句

1人参与，1条评论

来说两句吧……

登录并发表

搜狐“我来说两句”用户公约

最新评论

狐狐网友 (北京市北京市)

啊

1月2日 18:23 回复 0

没有更多评论了

推荐阅读

评论

1

分享

微信分享

新浪微博

QQ空间

复制链接

初级编程学习