游戏中的优化指的是什么？ - 知乎

https://www.zhihu.com/question/22595954/answer/27271824

Milo Yip2016 年度荣誉答主 Tue Sep, 07 01:59

游戏软件的优化和一般软件是有一些区别。

游戏通常是软实时（soft real-time），就是说运行上有时间限制，但没有硬实时般严格。

先谈固定硬件的游戏平台，如游戏机和街机。在这些平台上，通常会设置固定的帧率目标，例如30 FPS（即每帧33.3毫秒）。游戏开发者希望在这个时间限制下，尽量提升游戏的品质，例如更精细的角色和场境、加入更多效果、提升人工智能水平等。优化的目的除了令游戏顺畅，也是提升游戏品质的必要条件之一。

对于PC或手机平台，因为硬件的性能有很大差异，优化就没有一个具体的目标，而是希望尽可能在大部分平台上都能做得最好（虽然PC游戏有几百FPS的情况，但实质上几乎不能增加流畅性）。

从玩家角度，我认为游戏的性能指标大概有这几方面：

1. 平均帧率
2. 流畅性（不要「卡」，专业地说就是少spikes）
3. 互动延迟（输入后至看到反应的时长）
4. 等待时间（读盘、写档、网络连接等）
5. 内存用量
6. 游戏体积
7. 网络流量（主要是移动平台）
8. 耗电量（主要是移动平台）

而在开发的角度来说，我认为优化方法可以分为无损和有损的。无损是指不影响品质，纯粹通过技术上的优化去提升整体性能。而有损是指通过简化、近似化去改善性能，例如简化着色器（shader）、要求美术降低某角色的三角形数目、要求关卡设计师减少一些NPC等。
优化前我们要先进行性能剖析（profiling），找出性能问题的核心，然后再看看有什么方法可以尝试。主要可分为算法上的和底层的优化方法。不详细说明，就举个例子吧。

例如，在二维弹幕射击游戏中，需把大量子弹与飞机做碰撞测试（相交测试）。如果有n颗子弹，m个可被击中的目标，蛮力法需要mn次测试。我们可以看情况，使用一些空间分割的算法，把子弹和目标分配到不同的空间范围里，只需对每个范围里的物体做测试。而在底层方面，我们可以考虑使用多线性、SIMD指令，并考虑到缓存一致性等方面去优化。

上述例子主要是在CPU上进行的游戏逻辑方面的优化，而许多游戏中也需要在CPU/GPU上对图形方面进行优化。在PC/手机平台上，因为瓶颈不固定，游戏开发者通常会尽力优化每一个部分。

@孟德尔和 @Thinkraft提到了Quake的平方根倒数，我引用一篇以前写的文章，测试SSE指令和Quake的实现：

在1999年，id software公司发布了《雷神之锤III竞技场(Quake III Arena)》巨作，此第一身射击游戏有别于前作，以多人连线游戏为主轴，得到空前的成功。
在2002、2003年间，网上出现一段关于该游戏中的源代码讨论，那段代码是这样的：

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = *(long*)&y;                // evil floating point bit level hacking
    i = 0x5f3759df - (i >> 1);            // what the fuck?
    y = *(float*) &i;
    y = y * (threehalfs - (x2 * y * y));  // 1st iteration
//  y = y * (threehalfs - (x2 * y * y));  // 2nd iteration, this can be removed

    return y;
}
```

它是用于计算一个单精度浮点数的平方根倒数（reciprocal square root, 即1/sqrt(x)）。平方根倒数在游戏中经常用到，例如把矢量归一化（normalize）时，就要计算n = v / sqrt(v · v)。
此段代码使用了牛顿法（Newton’s method）去提升精确度，但令人渍渍称奇的是它计算初始估值的这一句：

```
i = 0x5f3759df - ( i >> 1 );
```

它利用了IEEE754浮点数的二进制表示来计算第一个近似值。此方法是谁发明的，魔术数字（magic number） 0x5f3759df 从何而来，暂时也没有确切的证据。但现在已找到比这更优的魔术数字[1]。
然而，本文想带出的是，**虽然此方法如此神奇，在现今的机器上通常不是最理想的。在PC上，自1999年Intel推出的Pentium III，就已经加入了SSE指令集，当中的rsqrtss指令就是能够计算一个单精度浮点数的平方根倒数。此外，rsqrtps则能同时计算四个单精度浮点数的平方根倒数。**

测试

我们可以写一个程序简单测试一下：
（略……）

结果及分析

使用VS2008 (缺省release配置)，在i7 920 2.67Ghz上的結果：

```
   dummy      363.8ms  error= 83.70051795%
standard     1997.4ms  error=  0.00000871%
   quake      586.1ms  error=  0.17520049%
quake2nd      970.1ms  error=  0.00046543%

dummy_ss      109.4ms  error= 83.70051795%
vsqrt_ss     1160.3ms  error=  0.00000871%
rsqrt_ss      108.3ms  error=  0.03087627%
rt2nd_ss      180.6ms  error=  0.00002188%

dummy_ps       26.8ms  error= 83.70051795%
vsqrt_ps      288.4ms  error=  0.00000871%
rsqrt_ps       27.0ms  error=  0.03087627%
rt2nd_ps       53.4ms  error=  0.00002188%
```

standard用了标准库的sqrt()函数，编译器使用传统FPU的运算计算开方和倒数。
quake和quake2nd的确比standard快，但quake的相對误差峰值约是千分之2，误差较大。quake2nd则用接近一倍的运算时间来改善精确度，相對误差峰值降至约百万分之5。

divsqrt_ss使用了SSE运算，准确程度与standard相同，而耗时仅比quake2nd慢一点点。实际上，如果在编译器开启/arch:SSE，standard也会使用SSE运算，产生的代码和divsqrt_ss相约，性能也差不多。

重点来了，**rsqrt_ss的耗时只有quake的18%，而相對误差峰值也更好，约万分之3**。仔细一看，发现它的耗时与dummy_ss相若。換句话说，因为使用了流水綫的潜伏时间，其数据吞吐量和至dummy_ss相若。

那么，再比较使用多一次牛顿迭代的版本。rsqrt2nd_ss的耗时也只有quake2nd的18%。而相對误差值也更好，去到千万分之2的水平。

最后，若真正运用了SIMD的并行运算能力，使用ps后缀的指令又会如何？在此测试中，可以看到性能比ss版本的提升了3至4倍。而rsqrt_ps也因流水綫达至dummy_ps的吞吐量。**rsqrt_ps比quake版本快20倍以上，比standard版本快70倍以上。**

总结

虽然quake里的平方根倒数算法是令人津津乐道的话题，但从应用来说，它并不一定是最好的选择。
……

参考

[1] Lomont, Chris. 'Fast inverse square root.' Technical Report, 2003.http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf