Skywind Inside » 影子跟随算法（2007年老文一篇）

算法简述

动作类游戏如何在高延迟下实现同步？不同的客户端网络情况，如何实现延迟补偿？十年前开始关注该问题，转眼十年已过，看到大家还在问这类问题，旧文一篇，略作补充（关于游戏同步相关问题还可以见我写于2005年的另外两篇文章，帧锁定算法 和 网游同步法则）：

影子跟随算法由普通DR（dead reckoning）算法发展而来，我将其称为“影子跟随”意再表示算法同步策略的主要思想：

1．屏幕上现实的实体（entity）只是不停的追逐它的“影子”（shadow）。

2．服务器向各客户端发送各个影子的状态改变（坐标，方向，速度，时间）。

3．各个客户端收到以后按照当前重新插值修正影子状态。

4．影子状态是跳变的，但实体追赶影子是连续的，故整个过程是平滑的。

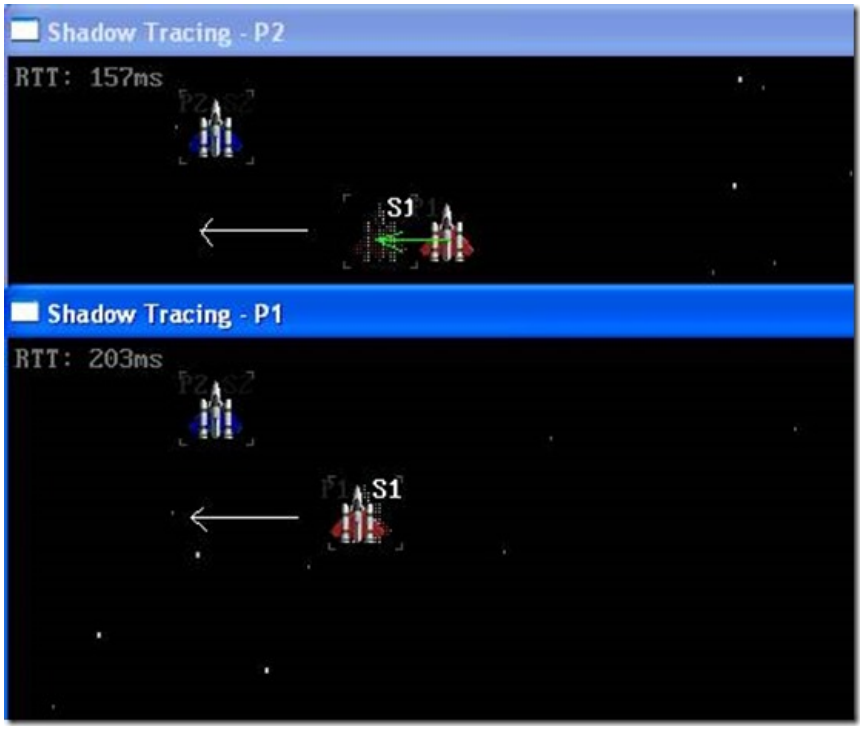

图1 算法演示

前面的1号终端控制红色飞船P1向左飞，并把自己的状态时时告诉服务器

后面的2号终端上接收到飞船P1的影子S1的状态（向左移动），并让P1的实体追赶S1

网络性能指标一：带宽，限制了实时游戏的人数容量

网络性能指标二：延时，决定了实时游戏的最低反应时间

使用该算法可以容易的开发出一款马里奥赛车，或者Counter Strike，详细说明见后：

算法比较：

1．帧间同步：不同客户端每帧显示相同的内容，键盘/时钟数据传到服务器，服务器确认后所有终端做出响应，多用于局域网游戏，比如红警（需要等待客户端），街霸II的网络版（360），网速要求高，复杂度低。参考以及 LockStep和TimeWrap算法，以及我2007年旧文 帧锁定算法。

2．插值同步：不同客户端显示不同步，但是状态同步，常见的Dead Reckoning（或叫导航插值），效果好，但复杂度高。常见于竞速类游戏和 FPS游戏。

算法定义：

1．时间：以贞为单位（FPS=10），一开始由服务器告诉向所有客户端，每5分钟同步。

2．玩家：每个玩家控制自己的实体，并在每贞将状态改变告知服务器。

3．状态：状态数据 = 实体ID + 坐标 + 方向 + 速度 + 时间（贞）。

4．插值：收到新状态包后将根据其运动方向与时间，根据现有时间计算当前的新状态。

5．跟随：实体不停的追踪自己的影子，追上后与影子保持状态同步。

相位滞后：可选参数，实体与影子保持一定距离同步，相当于保持一定车距，这样在控制者突然停止的时候，不容易因为网络延迟跑过了又被拉回来。

惯性移动：可选参数，开始移动或者停止或者改变方向都有加速度，这样就不需相位滞后了。

每次服务器向各个客户端同步时间的时候，由于延迟，所有客户端的时间都是慢于服务器的，这没有关系，只要大家在一定误差范围内以相同的速度增加，就完全没有问题。

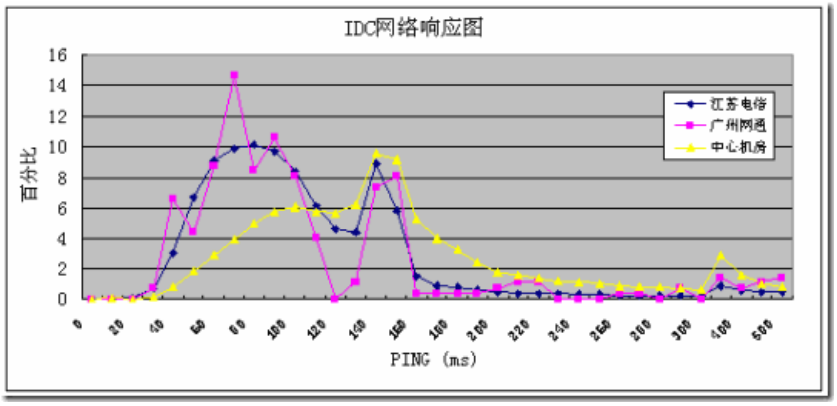

图2 IDC网络响应

在公网平均130ms的Latency下，是不存在“完全的”的同步情况。如何通过消除/隐藏延时，将用户带入快速的交互式实时游戏中，体验完美的互动娱乐呢？

让所有的用户屏幕上面表现出完全不同的表象是完全没有问题的；

把这些完全不同表象完全柔和在一个统一的逻辑中也是完全没有问题的。

需要根据具体情况，分清楚哪些我们可以努力，哪些我们不值得努力，弄明白实时游戏中同步问题关键之所在，巧妙的化解与规避游戏，最终在适合普遍用户网络环境中(200ms)，实现实时快速互动游戏。

案例解析：Counter Strike

实现CS的话，首先我们需要给人物移动加上惯性，比如静止状态突然开始移动，那么需要0.5-1秒的加速过程，而移动中突然停止也需要0.5-1秒的减速过程，这样就实现了无差别同步，不需要相位滞后来避免拉扯影响用户感。

同时开枪射击采用客户端判断，也就是说如果我看见你在墙前面，开枪射中，那么我向服务器发送“我击中你了”，这时有可能真实的你在墙后，那么表现出来的就是我看见我打中你了（减不减血由服务段判断），而你没有看见我，觉得我穿墙打中你了。

图3 CS的同步逻辑

关键状态进行缓存，不然如果别人向前连续跳五次，每次取得状态都取到最高点的话，别人客户端上的影子和跟随的实体会奇怪的持续的飞在天上，所以需要将起跳和落地这两个关键状态缓存，实体追赶时只有追上的第一个状态（一号影子）才能追逐第二个状态（二号影子）。

由此可以在完全时间同步的情况下平滑的跑动、跳跃，开枪射击采用客户端判断后手感得到提高，唯一需要担心的就是外挂，外挂多是实时游戏的代价，只能通过Cheating Death等工具防止了。

案例解析：马里奥赛车

用该算法实现马里奥赛车是很简单的，影子和实体都使用惯性，由于赛车惯性很大，不容易有突变的状态更新，所以效果会比FPS游戏更好。

玩家碰到道具后，马上在屏幕上隐藏该道具的显示并通知服务器，由服务器决断道具属谁，由于刚碰到道具就隐藏所以不会有碰到道具却在一段时间内无法取得延迟现象。

游戏道具系统实现也很容易，比如那个将当前第一名炸毁的道具，它的描述是：原角色+对象角色+约定发生时间。既然知道对象是谁，什么时间发生，那就更本不需要怎么同步了，所有客户端和服务器在该时间让炸弹爆炸就得了，这种手法类似即时战略游戏。

游戏还有一类道具是可以发射的乌龟壳，这个东西属于有弹道的发射物，类似Quake里面的某些武器，需要作一些同步处理，基本特性是服务器判断起决定作用，客户端同步判断，如果客户端与服务器都判断集中，那就集中；如果客户端判断集中而服务器判断没有集中，那会看到该角色似乎被打了一下，但很快又恢复了速度向前冲。

由于赛车本身就具备惯性比较大的特点，因此同步效果是比较好的，可以在更大的延迟情况下表现得和FPS差不多（比如300ms效果相当于FPS的200ms）。

非可靠包：

该“影子跟随算法”支持非可靠传输协议，如果使用非可靠传输，那么我们按照特定频率（如每秒10次）定时发送状态更新，因为协议中每个更新包出了位置外还有速度、方向和时间，甚至还能加速加速度，因此我们丢一个包没有关系，可以根据后来的包重新计算插值。只有关键状态更新时才需要可靠传输，这就避免

了TCP中丢包时RTO指数增长造成的延迟了。

负面情况：

该算法缺点就是无法向“帧同步”算法那样，每次发送按键给服务器，服务器处理后再反馈结果，在局域网中（平均延迟<5ms），这样的效果相当于单机游戏一样即时，游戏性也能很复杂。然而在Internet中（平均延迟130ms，设计基准200ms，每秒最多发送10个数据包）该算法却不能像单机游戏那样有复杂的场景互动，有类似格斗游戏的即时的动作判定。

许多策划在设计实时动作游戏时很多设计我们都难以实现，这样因为策划不容易明白哪些我们能做，哪些我们不能做。即便程序员精通同步理论，策划也经常碰壁。

当多数设计被程序员回复“无法实现”后，策划只有采取一种消极设计（砍掉很多有意思的互动元素），于是网络游戏的表现力到今天还是差单机游戏一大截。

这些问题也并不能因为“影子跟随算法”的提出而得到改进，大于100ms的判定时间，都很难做到即时。

最后，该算法编码复杂度比其他同步策略高，因为服务器需要计算一份影子数据，各个客户端需要计算一份影子数据，还需要计算实体追赶，而这三种计算都需要在同样的时钟下保持一致，这就增加了编码与调试的复杂度。

总结话题：

Internet特点是“高带宽，高延迟”，可以说从本质上Internet就不是为了游戏而设计的。故此Internet绝对意义的同步是不存在的。“影子跟随算法”的核心思想有几个：时钟同步，客户端先行，平滑追赶。通过这三个特性，我们能够在近似时间同步的情况下，模拟各种物体的移动过程，而使用该算法的前提是设计者需要根据各个游戏的特性研究不同的优化技巧，策略因游戏而变。

比如发送状态更新包时，不需要每次都发送，而可以只发送改变的状态。什么时候我们觉得改变了？就是当客户端实体与自己的影子之间的误差大于某特定数值时我们才发送更新包，这样虽然玩家在原地做左右摇摆的小幅度移动，只要没有超出范围，都不需要发送新的状态更新，其他玩家机器上看起来，它是站着不动的。

比如当发现某客户端5秒钟没有相应了，那么就将该人物的影子冻结住，永远不要为了等待某个数据而不让游戏进行下去。

本算法需要客户端与服务器维护相同的时钟，当每5分钟同步的时候，直接根据服务器的时钟替换当前时钟就行了，不需要重新计算所有影子的位置，因为后续的状态数据将会马上刷新这些状态。更不需要将测量到的PING值考虑进去，该算法与PING具体值无关。

当发现策划案子不可行时，寻找近似替代方案，比如减少“一次性的”“决定性的”事件发生，比如延长导弹在空中飞行的时间，比如将敌人加入HP分多次打死，而不是以及毙命，等等，都是大家可以发挥想象的地方。

相关例子：

文章相关DEMO如果有需要的话，可向我索要。

文献参考：

林伟（2007），帧锁定算法

林伟（2005），网游同步法则

Jesse Aronson (1997), Dead Reckoning: Latency Hiding for Networked Games

Yu-Shen Ng (1997), Designing Fast-Action Games For The Internet

Wentong Cai (1999), An auto-adaptive dead reckoning algorithm for distributed interactive sim

Micheal Abrash (1997), Quake’s 3-D Engine: The Big Picture

Nicholas Van Caldwell (2000), Defeating Lag With Cubic Splines