使用A\*平滑消除3D游戏中的画面抖动

作者：谭鹏

学院：计算机学院

专业：计算机科学与技术

导师：陈杰

# 目录

[目录](#_Toc30418)

[一、引言](#_Toc140)

[1.1 背景](#_Toc12158)

[1.2 意义](#_Toc9214)

[1.3 现状](#_Toc20839)

[1.4 其他抖动消除算法简述](#_Toc15260)

[1.5 工作与创新点](#_Toc11297)

[二、相关算法](#_Toc31191)

[2.1 A\*算法](#_Toc27860)

[2.2 \*\*算法](#_Toc4529)

[2.3 \*\*算法](#_Toc13070)

[三、原因与原理](#_Toc17095)

[3.1 抖动与抖动发生的原因](#_Toc3356)

[3.2 抖动消除原理](#_Toc720)

[3.3 算法实现](#_Toc8200)

[四、实验与分析](#_Toc6168)

[4.1 实验结果](#_Toc8370)

[4.2 结果分析](#_Toc21686)

[五、总结与展望](#_Toc11942)

# 摘要

# 一、引言

## 1.1 背景

“游戏”一词泛指棋类游戏例如象棋和《大富翁》；纸牌游戏，例如梭哈和二十一点；赌场游戏例如轮盘和老虎机；军事战争游戏、计算机游戏、孩子们一起玩耍的游戏。在计算机的语境下，“游戏”一词会使我们在脑海中浮现出一个虚拟世界，玩家可以控制人物、动物或玩具。

绝大部分游戏是软实时互动基于代理计算机模拟的例子。

在电子游戏中，会用数学方法来为真实世界的子集建模，从而使这些模型在计算机中运行。显然，这些模型只能是显示或者想象世界的简化或者近似版本，因此，数学模型是现实或者虚拟世界的模拟。

基于代理模拟是指，模拟中多个独立的实体（称作代理）一起互动。

所有的互动游戏都是时间性模拟的，即游戏世界是动态的——随着游戏事件和故事的展开，游戏的状态随着时间改变。游戏也必须响应人类玩家的输入，这些输入是游戏本身不可预知的，这也说明游戏是互动时间性模拟的，大多数游戏会实时回应用户输入，这即为互动实时模拟。

软实时是指即使错过期限却不会造成灾难性的后果。所有游戏都是软实时的。

模拟虚拟世界需要用到很多数学模型。数学模型分为解析式和数值式。例如，一个刚体因为地心引力而以恒定加速度落下，其分析式数学模型可写为：

y(t)=gt2 +v0t+y0

分析式模型可为其自变量设任何值来求值。例如上式，给予初始条件v0和y0、常量g，就能设任何时间t来求y(t)的值。在电子游戏中，用户的输入是不能预知的，因此不能预期对整个游戏完全适用分析式建模。

刚体受地心引力落下的数值式模型可写为：

y(t+t)=F(y(t),y`(t),y``(t),...)

也就是说，该刚体在（t+t)未来事件的高度，可以用目前的高度、高度的第一导数，高度的第二导数及目前的时间t为参数的函数来表示。为实现数值式模拟，通常需要不断重复的计算，以决定每个离散时间的系统状态。游戏也是如此运作的，一个主游戏循环不断执行，在循环的每次迭代中，多个游戏系统，例如人工智能、游戏逻辑、物理模拟等，就会有机会计算或者更新其下一个离散时间的状态。这些结果最后可渲染成图形显示、发出声效或者书橱至其他设备。

游戏引擎这个术语在20世纪90年代中期形成，这与第一人称射击游戏如id software公司的《DOOM》有关。《DOOM》将其软件构架划分为核心软件组件（如三维图形渲染系统、碰撞检测系统和音频系统等）、美术资产、游戏世界、构成玩家游戏体验的游戏规则。这样的划分非常有价值，另一个开发商取得了这样的游戏的授权之后，只需要制作新的美术、关卡布局、武器、角色、游戏规则等，对引擎软件做出很少的修改，就可以把游戏打造成新产品。

现在主流、常见的商业引擎有：Value公司的Source引擎，Epic的Unreal引擎，以及在移动端很常见的跨平台商业引擎Unity3D，Cocos2dx等。另外还有很多游戏公司有自己专用的私有引擎。

一般认为，首个三维第一人称射击游戏是《德军司令部》。这款游戏有美国的id software于1992年制作，他引领游戏产业进入到了令人们兴奋的领域。Id software又相继开发了《DOOM》、《Quake》等游戏。随着20多年的发展，3D游戏已经变得非常常见，基本上已经是随处可见。现如今的游戏大多数是3D游戏。

## 1.2 意义

3D游戏相较于2D游戏来说，从图形学角度来讲，多了一个z轴，使得3D游戏是在空间的层次上进行展示，而2D是在平面的层次上进行显示。2D游戏现实只需要将屏幕的一部分直接投射到屏幕上就可以，但是3D不同，由于3D使得游戏有远近、深浅的概念，简单的投射已经不可行。3D游戏采用了人眼的模式，来展现游戏世界。如果游戏的过程中，游戏画面抖动，将会使玩家感觉自身在抖动，影响玩家在游戏中的判断，降低游戏的可玩性。

由于可玩性是游戏的核心，消除游戏中画面抖动势在必行。采用本文的方法可以提高游戏的体验，减少开发负担。

## 1.3 现状

除了A\*算法以外，漏斗平滑算法也是一种使用较多的平滑算法。

但一般的A\*算法通过比较当前路径栅格的8个邻居的启发式函数值F来逐步确定下一个路径栅格，当存在多个最小值时A\*算法不能保证搜索的路径最优。

漏斗平滑算法在八格子寻路时效率不好。

## 1.4 其他抖动消除算法简述

漏斗平滑算法过程是判断下一个点是在漏斗范围内，就直接移动过去。如果出了范围，那么就设置成一个点，重新一轮漏斗。

## 1.5 工作与创新点

在本文中，通过对传统A\*算法的拓展，实现了一个平滑算法，实现了适用于3D游戏的平滑算法。

# 二、相关算法

## 2.1 A\*算法

A\*算法原本是一种求解最短路径直接搜索方法，是一种启发式搜索算法。

公式表示为：

**f(n) = g(n)+h(n)**

其中， f(n) 是从初始状态经由状态n到目标状态的代价估计，g(n) 是在状态空间中从初始状态到状态n的实际代价，h(n) 是从状态n到目标状态的最佳路径的估计代价。

A\*算法的具体过程为：

1，把起始格添加到开启列表。

2，重复如下的工作：

a) 寻找开启列表中F值最低的格子。我们称它为当前格。

b) 把它切换到关闭列表。

c) 对相邻的格中的每一个？

\* 如果它不可通过或者已经在关闭列表中，略过它。反之如下。

\* 如果它不在开启列表中，把它添加进去。把当前格作为这一格的父节点。记录这一格的F,G,和H值。

\* 如果它已经在开启列表中，用G值为参考检查新的路径是否更好。更低的G值意味着更好的路径。如果是这样，就把这一格的父节点改成当前格，并且重新计算这一格的G和F值。如果你保持你的开启列表按F值排序，改变之后你可能需要重新对开启列表排序。

d) 停止，当你

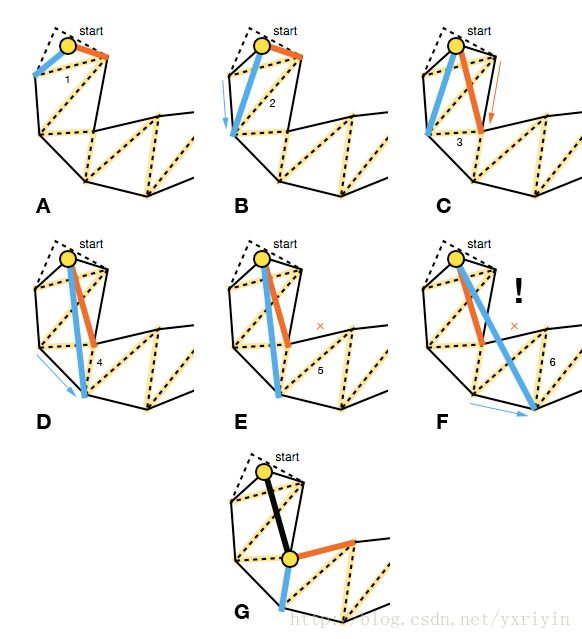
\* 把目标格添加进了关闭列表，这时候路径被找到，或者

\* 没有找到目标格，开启列表已经空了。这时候，路径不存在。

3.保存路径。从目标格开始，沿着每一格的父节点移动直到回到起始格。这就是你的路径。

## 2.2 漏斗平滑算法

漏斗平滑算法的原理在之前已经说，是判断下一个点是在漏斗范围内，就直接移动过去。如果出了范围，那么就设置成一个点，重新一轮漏斗。



## 2.3 弗洛伊德路径平滑算法

弗洛伊德算法是解决任意两点间的最短路径的一种算法，可以正确处理有向图或负权的最短路径问题，同时也被用于计算有向图的传递闭包。Floyd-Warshall算法的时间复杂度为O(N3)，空间复杂度为O(N2)。

弗洛伊德算法的基本思想如下：从任意节点A到任意节点B的最短路径不外乎2种可能，1是直接从A到B，2是从A经过若干个节点X到B。所以，我们假设Dis(AB)为节点A到节点B的最短路径的距离，对于每一个节点X，我们检查Dis(AX) + Dis(XB) < Dis(AB)是否成立，如果成立，证明从A到X再到B的路径比A直接到B的路径短，我们便设置Dis(AB) = Dis(AX) + Dis(XB)，这样一来，当我们遍历完所有节点X，Dis(AB)中记录的便是A到B的最短路径的距离。

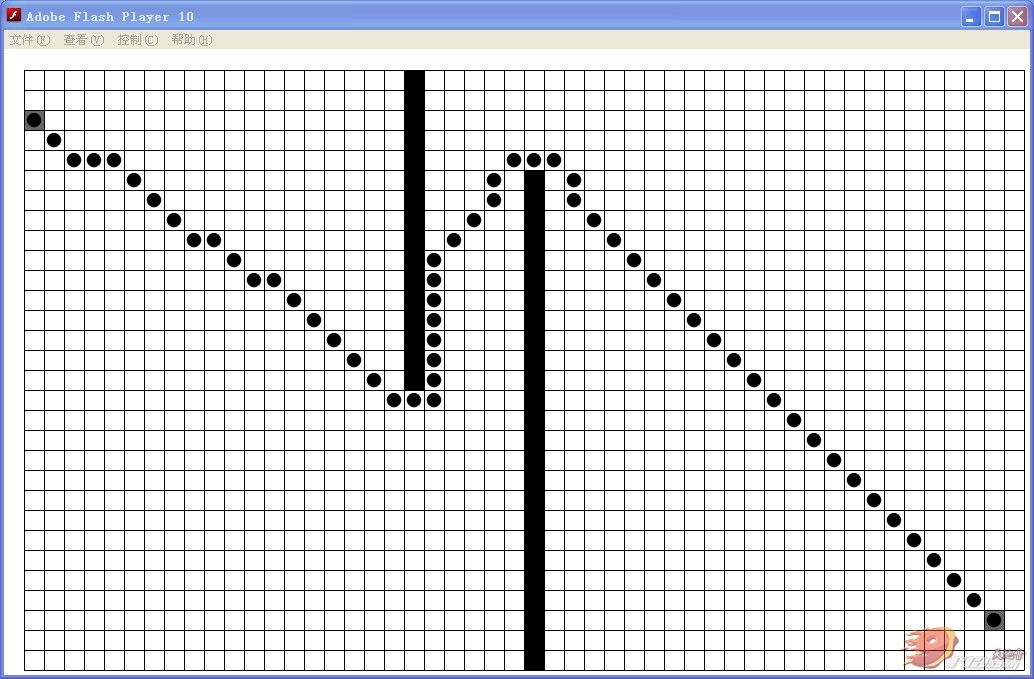
其算法描述：

a.从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

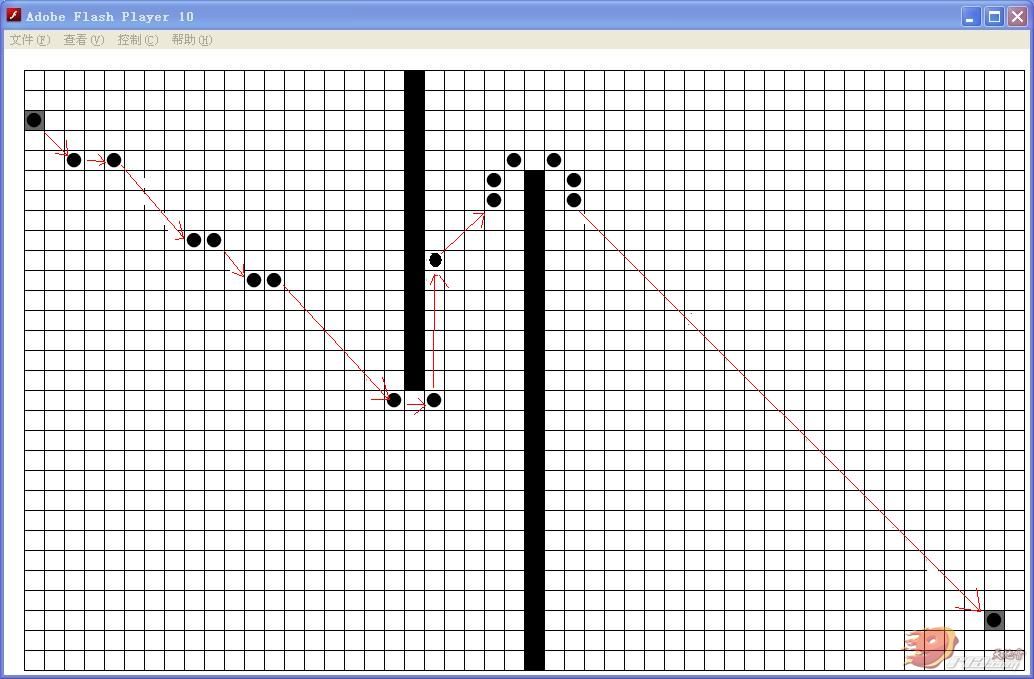
b.对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比己知的路径更短。如果是更新它。

弗洛伊德路径平滑算法是一种在内部应用的算法。

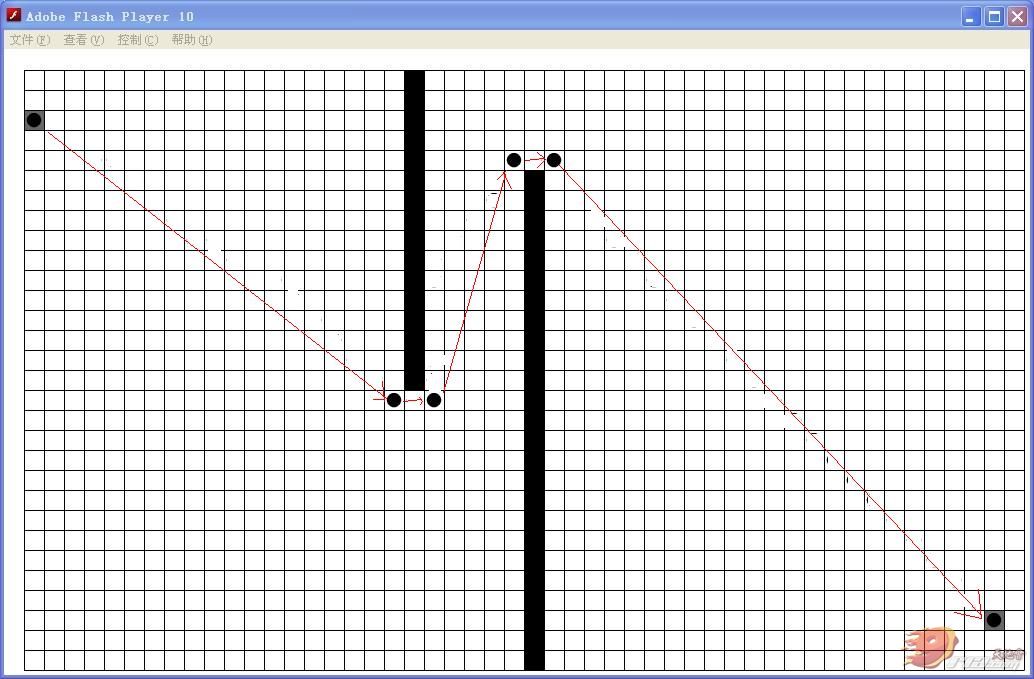
弗洛伊德路径平滑算法应在通过A\*寻路算法得出路径后进行，它的步骤分为两步：一、合并路径数组中共线的节点；二、尽可能地去掉多余拐点。这个过程如下图所示：



去掉共线点



去掉多余拐点



可以看到，使用弗洛伊德路径平滑处理 后的路径正如我们期望的那样，而且大大削减了路径数组中的节点数目。

不难发现，若存在三点A(1,1), B(2,2), C(3,3)，若B与A的横、纵坐标差值分别等于C与B的横、纵坐标差值，则A,B,C三点共线。

仔细观察第三幅图你会发现，若路径中存在节点A,B,C,D,E,F,G，如果A与G之间的连线所经过的节点中没有一个节点是不可移动节点，则我们称A与 G之间是不存在障碍物的。如两节点间不存在障碍物，则可以去掉此两点间其他所有节点。如上例中A-G这些节点，若A与G之间不存在障碍物，则我们可以去掉 A与G之间的B,C,D,E,F节点，最终路径数组中只剩下A与G两个节点。

## 2.4 DDA算法

数值微分法即DDA法(Digital Differential Analyzer)，是一种基于直线的微分方程来生成直线的方法。

算法描述：

设(x1，y1)和(x2，y2)分别为所求直线的起点和终点坐标，由直线的微分方程得：

dx/dy = (y2-y1)/(x2-x1) = m = 直线斜率 = (1)

可通过计算x方向上的增量引起y的改变来生成直线。

xi+1 = xi+ (2)

yi+1 = yi+ \* m (3)

也可通过计算由y方向的增量引起x的改变来生成直线：

yi+1 = yi + (4)

xi+1 = xi + /m (5)

选择x2-x1与y2-y1中较大者作为步进方向(假设x2-x1较大)，取该方向上的增量为一个象素单位( =1)，然后利用式(1)计算另一个方向的增量( =\*m =m)。通过递推公式(2)至(5)，把每次计算出的(xi+1，yi+1)经取整后送到显示器输出，则得到扫描转换后的直线。

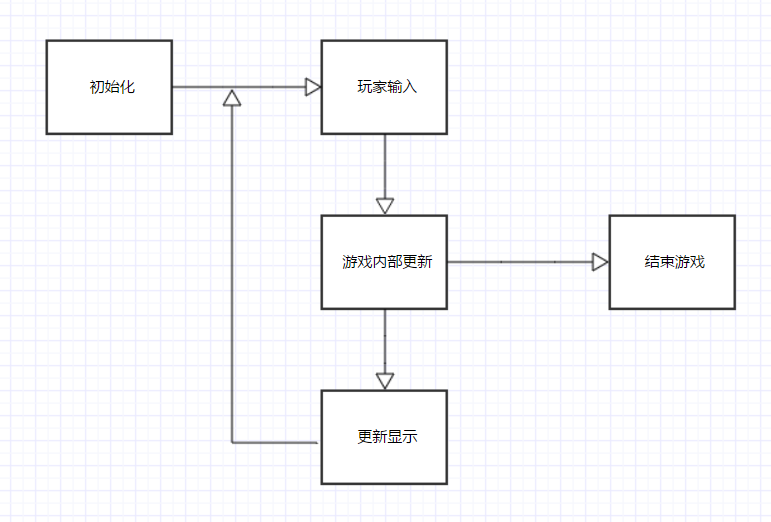
　　之所以取x2－x1和y2－y1中较大者作为步进方向，是考虑沿着线段分布的象素应均匀。

# 三、原因与原理

## 3.1 抖动与抖动发生的原因

在3D游戏开发的过程中，当在游戏内对游戏中的对象进行移动、转向时，游戏的画面会发生剧烈的抖动，会使玩家晕眩，影响玩家体验。

要明白抖动的发生，首先需要明白电子游戏是怎么运作的。首先，需要认识到电子游戏也仅仅是一个可编译和运行的程序。就像其他的程序一样，main函数也是它第一个被执行的函数。游戏的结构如下图所示：



上图包含了五种状态，电子游戏最重要的三种状态在循环中互相连接，这个循环通常称作游戏循环。游戏循环式进行玩家输入，更新游戏内部数据，更新显示的地方。

初始化是尤其开始的地方，通常包含了启动动画以及一个包含了音量、画质等选项的选项菜单，在这个过程中会进行一些必要参数的设定，以及申请内存，设置堆栈，检测游戏环境以及加载显示驱动（比如包含集显和独立显卡的机器来加载独立显卡）。在这些都已经设置完毕后，玩家选择了开始游戏，则跳转到下一个状态。

玩家输入是对玩家所使用的输入设备例如键盘、鼠标、游戏手柄等进行监听，对玩家的输入的信息进行采集，并且在游戏内部中以游戏能够处理的形式进行存储。供后一个状态使用。

更新游戏内部是游戏的心脏，游戏根据玩家的输入，来改变游戏中的设定值，例如对玩家操作的人物进行移动，玩家游戏中的敌人的运动与反应，同时准备好显示所需要的所有数据与图像。

更新显示是将计算后的画面投射到屏幕上显示出来。一般来说，你可以选择对每个物体挨个在屏幕上绘制或者说按照例行办法，先把所有的对象计算好，然后进行一次性的绘制。在屏幕上进行绘制是一个花费时间比较多的过程。所以一次性进行绘制将会有比较好的性能。

结束游戏是玩家选择退出游戏或者玩家通关之后的一种正常状态。一般会有一个结束动画或者提示语来告诉玩家游戏已经结束了。同时也会进行数据的保存，释放内存，消除堆栈等工作。

摄像机指的不是现实社会中用来照相的那个摄像机，但是两者发挥的作用基本上是一样的。电子游戏中的摄像机用另外的一个词来说是“视角”，也就是看游戏世界的角度。展现在玩家屏幕上的画面便是通过摄像机的“照”出来的画面。

由于玩家的画面是由摄像机照出来的，那么画面抖动会有两个原因：游戏世界在抖动或者是摄像机在抖动。 游戏世界我们没有进行直接操作，那么按道理来说不会发生抖动，这个可能性就可以被排除掉。那么抖动发生的原因在于摄像机在抖动。

开发游戏时，当对游戏内部的对象进行移动、旋转操作时时，由于游戏本身的设定，摄像机也会跟随着玩家移动的对象进行移动、旋转，这在酷跑类游戏中很常见。由于游戏是帧驱动的，游戏循环每执行一次，游戏就播放一帧。每一帧中都会执行摄像机位置的变换，使之看向所需要看向地方，摄像机不停变换位置，而且是无规则的。不规则的速度变化会引起抖动。注意这里是相对速度，因为如果绝对速度在变化，但相对速度是一致的，其实你看到的物体是不抖的。

## 3.2 抖动消除原理

既然画面的抖动是由于摄像机的抖动造成的，那么，我们要消除画面抖动，就只需要消除摄像机的抖动就可以了。画面抖动都发生在画面移动时即摄像机朝向或者位置发生变化时。由于现今的游戏通常都是帧驱动的游戏，我们需要在游戏循环每次执行的时候，让摄像机位置、朝向的改变变得平滑也就是使得摄像机看向的位置移动变得平滑，这样，在游戏的过程中，我们所看到的画面是在进行平滑的变换，就如同我们自身走动一般。

抖动消除有一个前提，时间上要是平滑的。也就是说要么是每一帧执行所用的时间是一样那要么就处理成一样的效果。现代游戏是帧驱动的，我们不能保证每一帧所执行的时间是一致的，但是我们可以获得每一帧执行所消耗的时间，在处理时，我们以消耗的时间为影响因子乘如，这样便实现了时间上的平滑。

再者是路径上的平滑，平滑无非两种，一种是延迟，一种是根据以前的速度进行加速度计算，然速度变化变慢。

## 3.3 算法实现

# 四、实验与分析

## 4.1 实验结果

## 4.2 结果分析

# 五、总结与展望