Minecraft 实体运动研究与应用

20211128 lovexyn0827

目录

1 绪论.....1
1.1 研究背景与意义.....1
1.2 本文主要研究思路.....1
1.3 本文所需知识基础.....3
1.4 内容概要.....3
1.5 一些特殊约定.....3
2 实体运动基础.....5
2.1 实体运动基本属性及设定.....5
2.2 实体运算位置及实体间运算顺序.....9
2.3 Minecraft 实体分类.....11
2.4 Entity 类定义的实体模型.....13
2.5 误差和一个重要 Bug.....15
3 实体自由运动相关公式推导.....19
3.1 符号定义及公式推导.....19
3.2 公式的直接及拓展应用.....24
4 实体移动过程及碰撞机制.....29
4.1 基于 Entity.move()方法.....29
4.2 基于 raycast 的弹射物自主移动.....34
4.3 两种移动过程的对比.....38
5 外界因素对运动的影响.....40
5.1 蛛网和浆果丛的减速作用.....40
5.2 粘液块和床的回弹.....40
5.3 蜂蜜块和灵魂沙的减速作用.....41
5.4 滑度机制.....41
5.5 气泡柱的变速作用.....42
5.6 其它特殊方块变速举例.....43
5.7 流体的变速作用.....43
5.8 活塞的推动和变速作用.....44
5.9 潜影盒和潜影贝的推动作用.....48
5.10 实体挤压（推动）.....49

5.11 骑乘.....50
5.12 鱼竿浮标的拉动.....51
5.13 击退.....51
5.14 方块对某些实体的推出作用.....52
5.15 爆炸的变速作用.....53
5.16 区块行为对运动的影响.....57
5.17 TP/折跃门传送.....58
5.18 跨维度运动.....58
5.19 拴绳机制.....59
6 LivingEntity 运动机制.....61
6.1 LivingEntity 的自由运动.....61
6.2 AI 及属性对运动的影响.....64
6.3 鞘翅飞行.....69
7 几类具体实体运动分析.....71
7.1 TNT.....71
7.2 下落的方块.....72
7.3 物品.....74
7.4 经验球.....78
7.5 船.....80
7.6 矿车.....83
7.7 箭矢.....85
7.8 雪球.....87
7.9 恶魂火球.....88
7.10 玩家.....89
8 实体运动应用举例.....91
8.1 改造一种 TNT 大炮.....91
8.2 科学地使用弹射物.....95
8.2.1 珍珠投掷技术.....95
8.2.2 珍珠炮.....96
8.2.3 杀凋大炮.....103
8.3 解释一些奇葩现象.....104
参考文献.....107
附录 A 常见实体运动属性.....108

附录 B 完整公式.....110
附录 C 辅助性工具.....113
附录 D 主要符号及单位.....115
附录 E 定义索引.....116
致谢.....120

1 绪论

1.1 研究背景与意义

实体运动看起来很简单，但是真正了解其具体机制的并不多。就连 wiki 上[1]对实体运动的专门论述也只有这一点（三叉戟和浮漂还是现补的，如图 1-1 所示），不严谨甚至还不完全正确，就比如这里玩家的阻力数据只在 Y 轴上较为准确，其它轴则有很大差距。另外，爆炸、活塞推动等加速方式的具体数据和实体移动过程细节并不普及，网上部分实体运动公式定义不清晰，对实体运动理论的系统论述缺失，这就是我写这篇报告的原因。

运动 [edit | edit source]

Minecraft中的重力与现实世界不同，并不是所有对象的重力加速度都相同。另外，所有实体在运动中都会受到不同大小的"阻力"，不同的实体之间也各有差异。

种类	加速度 格/刻^2	加速度 m/s^2	"阻力" 刻$^{-1}$	终端速度 格/刻	终端速度 m/s
玩家、其他生物和盔甲架	0.08	32	0.02	3.92	78.4
物品 物品、掉落的方块、TNT	0.04	16	0.02	1.96	39.2
船和矿车	0.04	16	0.05	0.76	15.2
扔出的鸡蛋、雪球、药水、末影珍珠[注 1]	0.03	12	0.01	3.00	60.0
射出的箭、三叉戟[注 1]	0.05	20	0.01	5.00	100.0
火球、凋灵之首、末影龙火球、潜影贝导弹	0	0	0	1.90	38.0
鱼竿的浮漂	0.03	12	0.08	0.345	6.9
羊驼唾沫[注 1]	0.06	24	0.01	6.00	120.0

1. ↑ [1.0] [1.1] [1.2] 注意，弹射物的"阻力"在被加速前已经确定并作用，而不是被加速后。因此它们的终端速度值都是很准确的整数。

图 1.1WIKI 关于实体运动的描述

说起来研究过程算是比较困难的，因为网上这方面的现成资料太少了，很多东西得自己弄（代码看到想骂街是什么感觉），加上这个涉及面广（实体运算，游戏运算顺序，区块加载甚至网络方面都不少涉及），运算量大（大部分是手算），研究了将近 10 个月才完成。即使很多内容都经过了反复的研究验证，但出现失误还是难免的，希望大家发现了可以告诉我。

通俗性我已经尽力了，如果还是有看不懂且较为重要的内容大家也可以通知一下，若有修改建议的欢迎大家提出。

1.2 本文主要研究思路

2020 年 10 月中旬：从 TNT 炮开始接触实体运动理论

2020 年 10 月下旬：推导出第一套适用于 TNT 实体的运动公式，但包含 sigma 且只有求初速度、某 gt 末位移及 Motion 的一些基础公式。

2020 年 11 月上旬：成功化简了公式中的求和部分，得出第一套推导过程,不久后引入 k 值

2020 年 11 月下旬：得出另外五类实体对应的公式，调整定义使其更便于使用，初步研究了 Entity 类及 LivingEntity 类，初步研究了一批实体的运算过程。

2020 年 12 月：研究了爆炸机制（可能是正确的，但因为“不合常理”在当时没确定），投掷物的运动，推导出所谓“万能方程”的前身，发现了 LivingEntity 水平阻力系数与 wiki 的不一致。

2021 年 1 月：进一步研究了一批实体的运算过程，推导出运动轨迹方程，研究了流体加速机制，尝试研究鞘翅飞行机制。

2021 年 2 月：进一步研究 Minecraft 源码，编写了一个辅助 Mod，初稿开始，研究了实体间的运算顺序。

2021 年 3 月：进一步研究 Entity 类，重点是 move，pushOutOfBlocks 和 calculateDimension 等方法。进一步研究 LivingEntity 运动机制。研究了另外一些实体的运算过程及服务端运算机制。修正了发现了浮点数误差对爆炸取样点影响的规律。发现了箭矢的一些特性。设计出一种射程约在 13000m 的 TNT 炮。

2021 年 4 月：研究了一些实体的运算流程，尝试了建立更加普适的定义。

2021 年 5 月：推导出鞘翅飞行、TNT 推进相关的一些公式，对 raycast 初步研究，解决了弹射物矫正问题。

2021 年 6 月：对 raycast 形成了较为完整的认知，解释了刷沙机原理，重点研究了物品实体及玩家运算。

2021 年 7 月：进一步研究了玩家运算，解释了在实体脚下循环生成 TNT 相关的问题，大概地研究了矿车的在轨运动机制，进一步研究了船的运动机制，设计出一种可行的珍珠炮方案并搭建。编写了本文的大部分内容，23 日发布初稿，而后进行简单的检查及修正。

2021 年 8 月：对 TNT 归中和珍珠炮相关内容进行了研究，搭建出一个真正有实用价值的珍珠炮，并修正几处相关错误。

2021 年 9 月：对多次弹射的优化问题进行了研究，下旬第一次全面审稿，发现大量错误并修正。

2021 年 10 月：继续检查并修正几十处错误，期间对 raycast 和 36 号方块影响范围等内容进行了重新研究，进一步研究了爆炸接触率计算中取样点的偏移，研究了拴绳机制，设计了两个接近平射的珍珠 Y 轴矫正。

2021 年 11 月：继续检查并修正少量错误，对 1.17 中新特性进行了一些研究，重新研究了实体挤压。更深入地研究了拴绳机制。

1.3 本文所需知识基础

(1) Minecraft 的基础游戏机制，如怪物会掉落物品，活塞会推动实体等。
(2) Minecraft 中一些基础名词，如玩家、方块、实体、生物、粘液块、坐标和爆炸等。
(3) Minecraft 中一些专业概念，如区块、游戏刻、服务端和维度等。
(4) 高中必修阶段及以前的数学概念。最好了解一些线代。
(5) 高中必修阶段及以前的物理概念。

1.4 内容概要

第二节给出一些基本概念，**是阅读下文的基础**

第三节给出用于描述实体在理想运动状态（只受固定加速度和阻力系数阻力作用）下的运动情况的一些常用公式及其应用（包括刻数的判定，以及三个以上运算阶段的实体的处理方法等内容），**强烈建议看完**

第四节描述了 Minecraft 中存在的两大实体移动（就是坐标和碰撞箱移动的过程）方式，个人认为可以区分就行

第五节说明了一些外部因素对实体的运动的影响，大家可以按需阅读或作为一个参考来用

第六节说明了 LivingEntity（玩家生物盔甲架）的运动机制，说明了 AI 控制速度的基本原理，介绍了状态效果等因素对运动的影响，最后简要说明了鞘翅飞行机制

第七节从创建、运算和影响三方面介绍了十类常见实体的运动特点，也是一个参考。

第八节向大家展示了实体运动的一些实际应用，可以按需阅读。

1.5 一些特殊约定

该文档主要面向 Minecraft Java Edition 1.16.4 和 1.16.5，部分旧版本、Java Edition 1.17.1 和基岩版中的差异也有部分提及，但并不是全部。

文中涉及的 Minecraft 截图中判断方向可以参考图中的 F3 界面下的光标，三条垂直的红、绿、蓝色线段分别标示了 X、Y、Z 轴的正方向。

所有“水”如不特殊说明均包括含水方块，所有“水方块”不包括含水方块。除船以外，“与流体接触”相关判断均是以判断发生前最近一次的流体相关运算（调用 updateWaterState()方法）时的状态为准。

文中的“Motion 大小”是指 Motion 各分量的平方和的算术平方根，“水平 Motion 大小”是指 Motion 的 X、Z 分量的平方和的算术平方根。类似地，文中的实体间“距离”是指两实体坐标差值各分量的平方和的算术平方根，实体间“水平距离”是指两实体坐标差值的 X、Z 分量的平方和的算术平方根。

文中经常会有类似于“将碰撞箱向四周/各方向扩大 1m 后得到的新区域”的叙述，这种叙述对应源码中的 Box.expand()方法，具体机制不好通俗说明。如图 1.5，将红石块所示范围相四周（左）和各方向（右）扩大 1m 后得到的区域由灰色玻璃标出（包括红石块本身）。

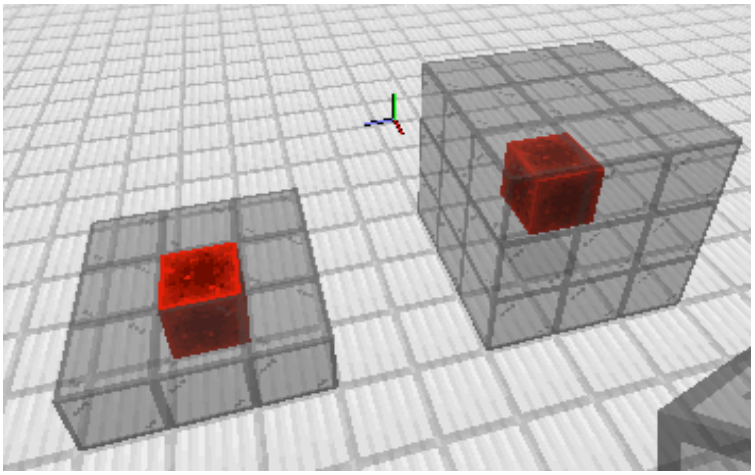

图 1.5 碰撞箱的扩大

若无特殊意义，文中的$t_0 = 1gt$

2 实体运动基础

2.1 实体运动基本属性及设定

在 Minecraft 中，所有实体都有以下**运动相关属性**：

(1) **碰撞箱**（碰撞体积，轴对齐边界箱，AABB）：一个长方体区域，标记实体占用的空间。碰撞箱由两个点指定，其中一个点的任意坐标都不小于第二个点的对应坐标，所指定碰撞箱是那两点之间的一个各面都垂直某一坐标轴的长方体。与实体渲染出来的样子不同，原版 Minecraft 中所有碰撞箱各沿轴视图都是矩形，俯视图都是正方形，而且不会随实体的朝向改变而旋转。

(2) **坐标**：表示实体的位置的一个点，位于实体碰撞箱底面的中心。Minecraft 中的坐标系是一个典型的三维右手直角坐标系，规定东西方向分别为 X 轴的正负方向，规定上下方向分别为 Y 轴的正负方向，规定南北方向分别为 Z 轴的正负方向。与现实不同，实体的坐标变化不一定完全由速度决定，因为实体可以被以某种方式传送。在 Minecraft 中，还存在一个用于指定方块所在单元的**方块坐标**，由精确坐标到方块坐标的转换可以通过向下取整其分量进行（BlockPos.java，<init>）。

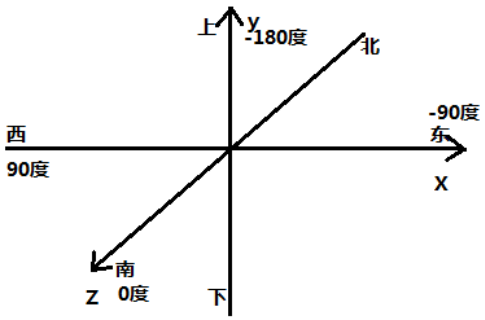

图 2.1 坐标系及偏航角示意图

(3) **Motion**：可认为是一个存储速度及其运算中间量的容纳一个三维向量的容器，在每次加速和阻力作用时其中存储的“准速度”会被改变，在移动时其中的值会被作为位移（就是那一游戏刻的平均速度）使用。可以这样说，Motion 决定速度，但是 Motion 的值并不总是严格等于速度，有时甚至会有很明显的差距，如阻力系数较大和被活塞推动时。说起来大家可能不信，堆叠的矿车被加速时其 Motion 会指数爆炸式增长，在经

过一段距离（与堆叠数量有关，堆叠越多越短，堆叠 100 个时不到 20m）后可以达到无穷大，但是其单轴速度永远不会大于 0.4m/gt。不过，如果真的不喜欢这样，把后文中的 Motion 中大部分换成“速度”问题也不大，能理解就好。

(4) **速度**：表示实体在一定时间内不计传送的坐标值的变化量的一个三维向量。因为 Motion 除速度外还存储速度的运算中间量，所以实体的速度并不等价于 Motion，尽管一般都认为它们是一回事。可以这样认为，Motion 是实体运动的决定，而速度是实体运动的结果。

(5) **眼部坐标**：一个用于某些计算的点，x、z 轴上坐标与实体坐标相同，但 y 轴上会比实体坐标高出来一点，默认为实体高度的 0.85 倍，但也有例外，如箭是 0.13 格，详见**附录 A**。

(6) **俯仰角**（pitch）：用于描述竖直方向的朝向的一个角度，范围在-90 度～90 度，水平方向为 0 度，向上为负，向下为正。后文中用到的俯仰角皆以此为准，无论它是实体的一个属性还是一个纯粹的方向，下同。

(7) **偏航角**（yaw）：描述了实体的水平方向，即东西南北或是南偏东 20 度之类的。其取值范围在-180 度（不含）-180 度(含)之间，正南方对应 0 度，此处起顺时针为正，逆时针为负。

(8) **空气阻力系数**（wiki 上称“阻力”，对应一些公式中的“f”）：实体在自由运动时会在各轴上受到一个大小等于对应轴上速度大小与对应轴上空气阻力系数的乘积，方向与对应轴上速度方向相反的阻力加速度，空气阻力系数表明了实体在自由运动时减速的快慢，其值越大，速度衰减越快。

Minecraft 中与运动相关的**实体运算**主要有以下六种：

(1) **重力作用**（Gravity，简称 **G**）

(2) **阻力作用**（Drag，简称 **D**）

(3) **自身运动控制决定的加速**（Controlled Acceleration，简称 **C**）

(4) **流体加速**（Fluid Acceleration，简称 **F**）

(5) **位移确定**（Displacement Calculation，简称 **Dc**）

(6) **坐标及碰撞箱更新**（Position & AABB Update，简称 **Pu**）

重力作用的含义显而易见，而流体加速在 5.7 节还要详细说明，这里不再赘述。

此处阻力的概念较为广泛，包括了空气阻力、流体阻力和地面阻力这三类在 Mojang 的设定中在某坐标轴上与该实体在该轴上速度大小成正比且反向的作用力。然而，如果试着将这一设定在牛顿力学中的表达式：

$$\frac{dv}{dt} = -fv \tag{2.1.1}$$

$$\frac{v}{v_0} = \lim_{t_0 \to 0}(1 - ft_0)^{\frac{t}{t_0}} \tag{2.1.2}$$

其中 f 是阻力系数。

类推到 Minecraft 中你就会发现这一设定无法准确地被实现，因为 *dt* 和 t_0 不能小于 1gt，更不能无限趋近于 0。实际上，Mojang 对这一运算的实现是将每个阻力作用时受阻力作用的各轴（一般含 X、Y、Z 轴，地面阻力一般只有 X、Z 轴）上的 Motion 减去对应轴上的阻力系数以 gt^{-1} 为单位的数值与对应轴上 Motion 的乘积，即

$$M_{作用后} = M_{作用前} - ft_0 M_{作用前} = M_{作用前}(1 - ft_0) \tag{2.1.3}$$

其中 M 是某一受到阻力作用的轴上的 Motion，f 为对应的阻力系数，$t_0 = 1\text{gt}$。

在第 3 节公式推导中$(1 - ft_0)$通常是一个有特别意义的值，**说明实体受到阻力的本质就是 Motion 被乘以了这个数**，所以在那里我会将这个数字记作 k，称为**速度乘数**，无单位。现有实体中 X、Z 轴上的阻力系数一般相等，这时可以认为在水平方向上的阻力系数就是 X 或 Z 轴上的阻力系数，或者说水平方向上的阻力加速度大小只与阻力系数和水平方向上的合速度大小相关，方向总是与水平方向上的合速度相反。很多实体（**附录 A** 给出了部分例子）各轴上空气阻力系数大小都是相同的，那可以类推，这时可以认为所有方向的阻力系数就是 X、Y 或 Z 轴上的阻力系数，或者说阻力加速度大小只与阻力系数和合速度大小相关，方向总是与合速度相反。

分开来说，**空气阻力**是指实体在空中运动时收到的阻力，包括火球，可能所有可通过指定 Motion 标签自主移动的实体都有这种阻力。**流体阻力**是指实体在与流体接触时受到的阻力，经常与流体种类有关，有时还与浸入深度有关。有些实体没有流体阻力，在流体中受到的也是空气阻力，如 TNT。空气阻力与流体阻力通常不共存。**地面阻力**较为复杂，包括滑度作用、灵魂沙、粘液块和蜂蜜块对上方实体的减速以及 TNT 等实体在着地时受到的较大阻力，这些将在第 5 节的前几小节和第 7 节对应小节中详细说明。最后补充一下，即使这几类阻力在这里被归为了一大类，它们在实际运算中的位置

也不一定相邻，这在一些计算中可能会造成难以确定实体类型和 k 值的问题，解决方案可以参考 3.2 节。

阻力系数的单位是负一次方刻，即 gt^{-1}，这个单位有两种理解。第一个是把它写成$(m/gt^2)/(m/gt)$，也就是把它理解成阻力加速度和速度的比值。第二种是把它直接就看成 gt^{-1}，即一定时间内速度减少量在原速度中的占比。

自身运动控制造成的加速主要包括生物的 AI 控制的加速、玩家控制的加速、火球类实体（见 2.3 节）的 Power 标签指定的加速以及矿车在充能铁轨上的加速等可以认为是实体在自身运算过程中“自己要”产生的加速。这将在 6.2 节和第 7 节对应小节中详细说明。

可能有些出乎意料，部分弹射物的坐标和碰撞箱移动是在位移确定并进行了一些其它运算后才会进行的。乍一看中间那些运算很可能会导致一些不希望出现的问题，但实际上并非如此。一方面，大部分实体（可能包含使用 move 方法（见 4.1 节）移动的所有实体，也就是烟花火箭、浮漂和其它所有非弹射物的可移动实体）的坐标和碰撞箱移动阶段在位移确定且进行完应有的预处理（如碰撞判定）后就被立即执行了，可以认为这两个阶段同属一个阶段，也就是**移动**（Movement，简称 **M**）；另一方面，很多时候（如实体在均匀介质中运动时，不考虑与外界方块和实体的某些交互时等）在位移确定且进行完应有的预处理后就立即执行坐标和碰撞箱移动与原顺序是等价的，也就是说此时仍然可以认为坐标和碰撞箱移动与位移确定同属一个阶段，即移动。

根据移动、阻力和重力作用的顺序，可以分六类，即 MDG（移动->阻力->重力）、DMG、DGM、MGD、GMD 和 GDM。部分实体的重力加速度或阻力系数为零，可认为对应阶段不存在，于是又派生出六类实体：DM、MD、GM、MG 和 M。同一非实体同一状态（如是否在水中）下的运算顺序一般不变，具体可以参考**附录 A**。其中部分类型没有对应实体。

重力加速度等只是加速度的一些特例，实际上为了在一些时候方便研究，可以将所有**加速度**（Acceleration，简称 **A**）统一。上面涉及到的几类实体分别可以改称为 MDA（移动->阻力->加速）、DMA、DAM、MAD、AMD、ADM、DM、MD、AM、MA 和 M。

总结：加速度的本质是加法，阻力的本质是乘法。

2.2 实体运算位置及实体间运算顺序

众所周知，原版 Minecraft 服务端（含客户端内置的服务端）的主线运算是单线程的，也就是说 Minecraft 服务端不会在同一时间运算多个维度、进行多个阶段或者运算多个实体。所以，可以给出 Minecraft 服务端每 tick 的运算流程大概如下：

这张图是高度简化的，有兴趣的话可以翻一下 Fallen_Breath 的 B 站主页看一下相对完整的版本[2]或直接翻阅源代码。

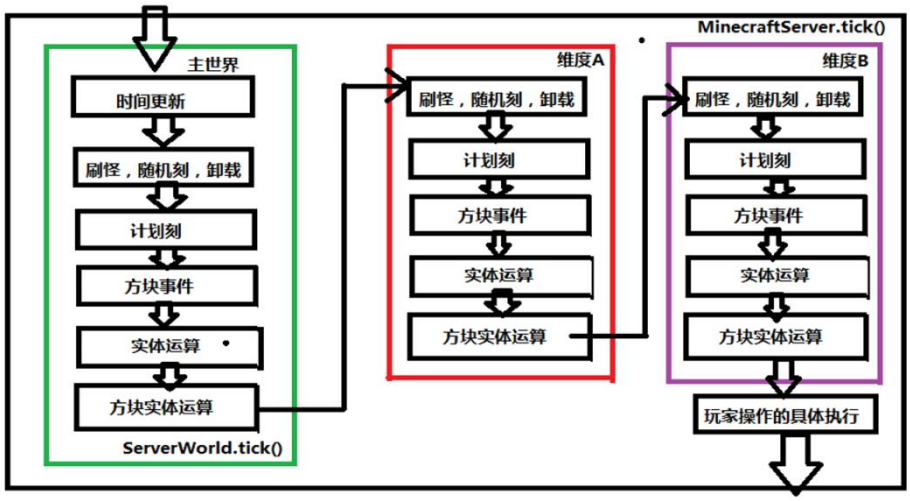

图 2.2.1 服务端每 tick 运算流程图

定义：

根实体：没有骑乘任何实体的实体。

同位实体：直接骑乘同一实体的实体，所有根实体互为同位实体。

实体的加载包括创建新实体、重新加载已卸载区块中的既有实体和从某一维度传送到另一维度。

研究 ServerWorld.tick():355-418 、 ServerWorld.tickEntity() 和 ServerWorld. tickPassenger()方法，可以得到以下几点规律：

(1) 第一个被运算的实体永远为根实体，因为骑乘者的运算依赖于根实体的运算。

(2) 所有根实体中，最近一次被加载到所在维度较早的实体先运算。

(3) 任意实体若有骑乘者，则骑乘者会紧随该实体之后运算。

(4) 一个实体本身及其所有骑乘者完成运算后下一个应运算的同位实体才（就）会开始运算，如果没有还未运算的同位实体，该实体骑乘的实体

的下一个同位实体才（就）会开始运算，依次循环，直到所有实体运算完毕。

(5) 一组互为同位实体的实体中最近一次开始骑乘该实体较早的实体先运算。

(6) 所有实体运算完成前不能向世界实体运算列表加入新实体，期间创建的所有实体会在该维度整个实体运算阶段完成后按计划加入的顺序加入。但是，向区块的实体列表的加载可以正常进行。也就是说，在这段时间内中，被加入的实体在该游戏刻不会被运算，但是通常可以在其它运算中被取到（如可以被炸毁，可以被推动等）。

(7) 被标记 removed 的实体和末影龙的组成部分（EnderDragonPart）不会被运算。

看起来比较难懂，那可以举一个例子，假设某维度有按最近一次加载时间编号 1～20 的 20 个实体，编号越小的最近一次加载越早，它们的骑乘关系及运算过程如图 2-2-2 所示，其中最上方都是根实体，骑乘者有一个额外编号标明其最近一次开始骑乘所骑乘的实体的时间顺序，红色数字标明运算顺序：

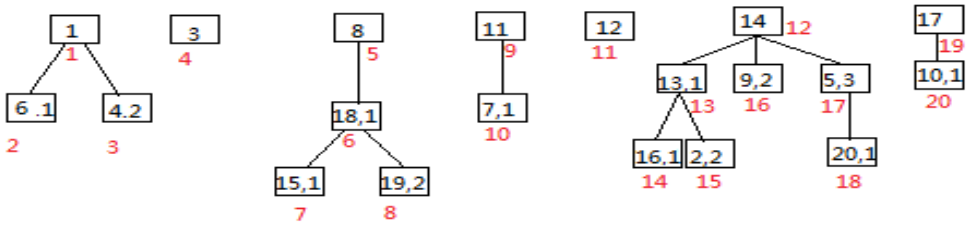

图 2.2.2 实体运算顺序示意图

可以看出，实体由骑乘关系组成了一个树形结构，该树形结构中一个分支运算完成后下一个同级分支就会开始运算，且分支内的运算顺序是由“根”到“叶”的。

掌握实体运算顺序在很多情况下都是有用的，如果没有这样的实体运算顺序，现在的很多 TNT 炮就无法稳定工作，原因可以发挥一下想象力（推进 TNT 没炸完炮弹就运算飞走了）。如果还是看不懂，那记住下面这一条就可以满足绝大部分需要了：

在不涉及实体间骑乘、区块重载和跨维度时，一个维度中较早创建的实体先运算。

服务端玩家实体是一个特例，它在实体运算阶段只进行一些触发进度一类的运算，服务端运算中运动部分则是发生在各维度的一般运算（计划刻、实体运算和方块实体运算等）之后。

2.3 Minecraft 实体分类

为了更好地指代多种实体，这里将基于源码中的继承关系对 Minecraft 中实体进行简单分类。

先供上 wiki 上对实体的定义：

实体（Entity）包括在 Minecraft 中所有动态的、移动中的对象。

Entities encompass all dynamic, moving objects throughout the Minecraft world.

不过个人认为还可以这样定义实体：

实体是 Minecraft 中一个独立存在的、实时更新的对象。

其中独立性标明了实体不是像方块实体那样依赖于其它对象而存在，运算的实时性又排除了方块一类没事不运算的对象，应该还行吧。wiki 上那一条定义个人认为并不准确，因为并不是所有实体都是可移动的。

大部分实体都可以归到四大类中：弹射物、LivingEntity、装饰物和矿车。

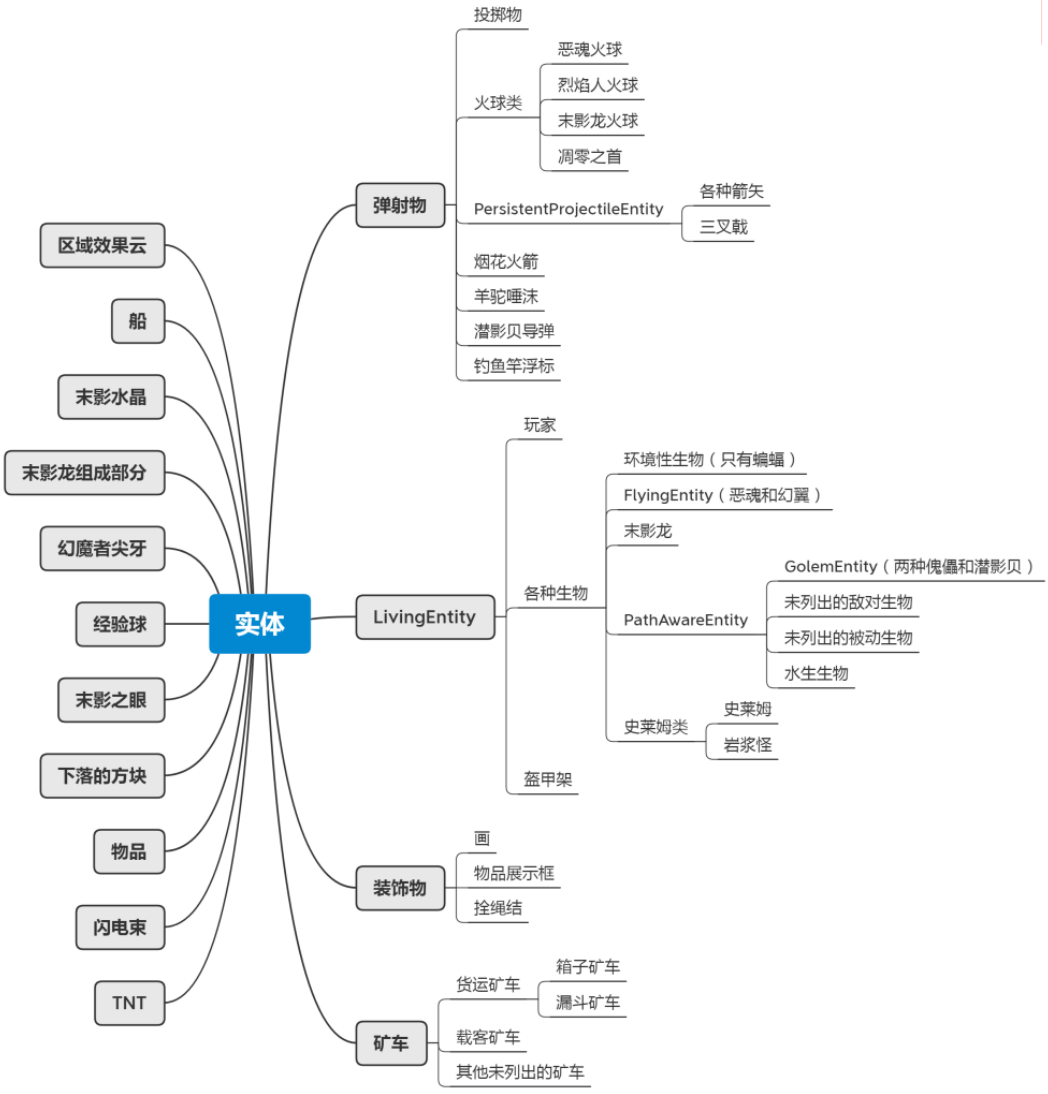

图 2.3 实体分类示意图

(1) **弹射物：**是一类正常情况下能以某种方式被发射出去的实体，该类别中所有实体都可以通过指定 Motion 标签自主移动。其中可以由单次右键使用物品投掷的归为**投掷物**；没有重力且可以通过指定 Power 标签给予加速度的实体统称**火球类**；在落到地面后有插入地面的行为且不会立即消失的统称 **PersistentProjectileEntity**（直译为“持久性弹射物实体”）；另外还有四种实体难以归入任何一类，在此不再细说。

(2) **LivingEntity：**是一类有生命值、属性值及使用状态效果的能力的实体，非得要翻译就是“活的实体”，除潜影贝、NoAI 标签为 true 的生物和 NoGravity 标签为 true 的盔甲架外都可以通过指定 Motion 标签自主移动。其中，玩家在类层次中还可分为客户端主玩家（ClientPlayerEntity）、

客户端上存在的其它玩家（OtherClientPlayerEntity）和服务端玩家（ServerPlayerEntity），这些细节将在 7.10 节中再详细说明。生物是一类具有 AI 的 LivingEntity，更深层次的分类较接近大家的一般认知，图中完全可以说明。我并不确定可不可以认为 PathAwareEntity 就是可寻路的生物，因为其它生物也有一定的寻路机制。至于盔甲架……你没看错，它的很多行为很接近玩家和生物，如存在生命值，可以装备盔甲，甚至有一定的附魔和状态效果使用能力，所以 Mojang 顺便就把它归到 LivingEntity 来了。

(3) **装饰物**：是一类与方块网格对齐，在移动或有非零的 Motion 时被破坏的实体。

(4) **矿车**：更不必多说。

(5) **末影龙的组成部分**：尽管它们仍是 Entity 类的子类，它们严格意义上来讲可能并不是一类实体，因为它是末影龙的一部分（不独立存在）且不会被运算，应该只是一个用于标示末影龙各部分范围并可以与外界进行一些交互的工具。说句题外话，我猜测之后它可能会被用于分割一些较复杂的实体的碰撞箱用于更精确的碰撞判定

(6) **余下的实体**：可以通过指定 Motion 标签自主移动的有船、经验球、末影之眼、下落的方块、物品和 TNT。

2.4 Entity 类定义的实体模型

该段内容会涉及一些源码，但非常重要，最好能看完。

就是不看代码估计大家也有不少人猜到所有实体的基类是 Entity。刨去一些 getter、setter 和形式上的代码，该类主要包含所有实体共有的一些属性（坐标、Motion 和朝向等）、大部分实体都有的行为（被下界门传送、被保存和检查方块碰撞等）以及一些不便放到其它任何类的内容。

下面给出 Entity 类中与运动相关的主要的字段：

entityId：实体的一个整数 ID，即**实体 ID**。每次游戏启动（服务端重启或客户端重启）后游戏会将一个计数 MAX_ENTITY_ID 置为 0，后来每次 Entity 对象被创建时该值会被加一并取加一之后的计数作为所创建 Entity 对象的 entityId。在客户端实体的 entityId 会被二次修改以与服务端保持一致，服务端玩家重生时也会将新玩家对象实例的 entityId 与设为与重生前一致，但是计数永远不会进行除上述“加一”操作之外的变更。1.17 中该字段被更名

为 networkId，说明了它最大的用途还是两端同步，但是在一些比较偏的地方它确实也被某些运算用到了，甚至有些还被某些装置利用了。

pos：坐标

velocity：用于存储可通过指定 Motion 标签自主移动的实体的坐标变化趋势及其运算中间量，也就是 Motion

yaw：偏航角

pitch：俯仰角

entityBounds：碰撞箱

onGround：表示该实体**着地**，在使用 Entity.move()（详见 4.1）移动实体时由以下代码确定（Entity.java:546-547）

```
this.verticalCollision = movement.y != vec3d.y;
this.onGround = this.verticalCollision&&movement.y< 0.0D
```

即实体在位移趋势向下时发生了竖直方向的碰撞。其中，movement 为该次移动的实际位移趋势（经活塞推动、蛛网浆果从减速和潜行相关调整后的结果），vec3d 为 movement 在经过碰撞调整后的结果。实体在某次移动中撞到地面又凭借水平位移趋势移下了平台或移到了一个坑上面后就会被错误地认为着地，被活塞向上推动的实体也不会被认为着地，个人认为这应是一个 Bug，但修复它可能要修改沿轴运动顺序并造成更严重的问题。

horizontalCollision：表示实体最近一次移动中在水平方向上发生了碰撞，即实体在 x，z 轴上实际位移趋势和实际坐标变化量在有不可忽略（差值绝对值大于 10^{-5}）的差别。在基于 Entity.move()方法的移动过程中确定。

verticalCollision：表示实体最近一次移动中在垂直方向上发生了碰撞，确定方式见 onGround。

stepHeight：存储实体能直接上去（如玩家和大部分生物直接走上去台阶或玩家被活塞从侧面推上完整方块）的高度。

noClip：表示实体不会与其它实体和方块发生碰撞，有时会无效。该字段为 true 的实体有末影龙的整体轮廓、恼鬼、客户端上存在的其他玩家、NoGravity 或 Marker 标签为 true 的盔甲架、旁观者玩家、潜影贝导弹、因忠诚附魔返回中的三叉戟、区域效果云和卡在方块中的物品。

movementMultiplier：位移乘数，一个三维向量，详见 5.1。

NO_GRAVITY：只是一个键，本身不存储值，但可以用于从 dataTracker 取来自 NoGravity 标签的表示实体不受重力影响的一个布尔值。

pistonMovementDelta：存储了这一游戏刻内实体被活塞移动的总位移的一个三维向量，用于限制同一刻活塞直接推动的距离，其各分量绝对值不

会大于 0.51，没有被活塞推动过时全为 0，被活塞推动时在 Entity.java：677-688 被计算并使用。

pistonMovementTick：最近一次被活塞推动时的时间，用于计算 pistonMovementDelta。

FallDistance：存储实体下落的高度用于摔落伤害等计算，可能会因为一些原因被改动而不准确。

实体基础运算在 Entity.tick()和 Entity.baseTick()方法中定义，大部分实体（例外：TNT、下落的方块、画、物品展示框、末影水晶、拴绳结、幻魔者尖牙和矿车）都会通过 super.tick()来一层一层地指向 tick()方法在 Entity 类中最基础的版本，其流程如下：

1 向客户端更新发光状态（385-387）
2 baseTick()，默认如下：
2.1 若正在骑乘的实体被移除，则停止骑乘（394-400）
2.2 骑乘冷却更新
2.3 记录当前的水平速度和朝向备用(402-404)
2.4 下界传送门相关更新（405）（tickNetherPortal()）
2.5 疾跑粒子（406-408）
2.6 流体相关更新（410-412）（updateWaterState()）
2.6.1 清空上一次更新流体时的液面高度记录（980）
2.6.2 更新水流变速（981）
2.6.3 更新熔岩变速（982-983）
2.7 火焰相关更新（413-428）
2.8 熔岩相关更新（点燃和 FallDistance 减半）（430-433）
2.9 虚空（-64 以下）相关更新（直接移除或受虚空伤害）（435-437）
2.10 向客户端更新着火状态（439-441）
2.11 标记已经过第一次更新（443）

我们主要关心 2.4 和 2.6，这分别将在 5.18 和 5.7 节处讲解。实际上，很多时候如果把整个实体基础运算压缩得只剩这两环节问题也不是很大。

2.5 误差和一个重要 Bug

因为在计算机上所谓实数只是一个有限位数（坐标，Motion 使用的双精度约有 15.65 位精度）的浮点数。所以当出现无法用浮点数精确表示的数时，就会产生浮点误差。浮点误差在大部分情况下基本可以忽略，毕竟在值

不是很大，运动时间较短时，误差只在数十万分之几的数量级，而且 Minecraft 设计和源代码中很多地方也对浮点误差进行了专门处理，如很多方法中存在的是否小于 10^{-4} 或 10^{-7} 而非是否等于 0 的判定、有些地方将绝对值过小的 Motion 或 Motion 改变量替换为 0 以及基于 Entity.move()的碰撞判定也会留出 10^{-7}m 的容错空间等。

在精度要求很高时，例如涉及到取整和比较操作或数据本身就很小时，浮点数误差会导致一些诡异的问题，需要注意。下面是一些例子：

(1) 珍珠炮中珍珠高度计算时有 10^{-6}m 左右的偏差

(2) TNT 归中时坐标即使在理论上完全重合，坐标也可能有所差异

(3) 爆炸接触率计算中取样点的异常偏移（只是预言）

根据产生时机，可以将浮点数误差分为编译时产生的浮点数误差和运行时（运算中）产生的浮点数误差。

前者误差由存储常数所用的浮点数种类决定，在常数为整数和 0.5、0.25、1.25 和 0.0625 一类数字时不会差生误差，产生时大小大概在 10^{-9} 或 10^{-15} 的数量级，但有时会因误差积累达到较大值，尤其是阻力系数和速度乘数被存储为单精度浮点数时。这一误差是可以在某种程度上消除的，即在高精度计算中使用真实值而非近似值，这一真实值可以在一些浮点数转换工具中转换得到或从原始的 class 文件中找到。表 2.5 中给出了几个例子，而且附录 A 中也进行了相关说明。

后者误差大小因为坐标和 Motion 使用了精度较高的双精度浮点数而在坐标绝对值不是很大的位置显得很小，实际上世界边境处无非也就是 10^{-9} 的数量级，不经长时间累积很难被发觉，但它是无法消除的。

表 2.5 几个数值对应的真实浮点值

文中给出数值	对应的单精度浮点数真实值
0.99	0.9900000095367431640625
0.98	0.980000019073486328125
0.05	0.05000000745058059692382812 5
0.04	0.03999999910593032836914062 5
0.03	0.02999999932944774627685546875
0.6	0.6000000238418579101562 5
0.989	0.98900002241134643554687 5

浮点数误差不是完全随机地出现的，有一定的规律：

(1) 坐标绝对值越大的地方由运算造成的浮点数误差越明显

(2) 坐标绝对值接近 2 的整数次幂的位置浮点数误差更可能发生
(3) 速度为 0 的移动和加速度为 0 的加速不会产生浮点数误差
(4) 把属性设定为一个定值的过程中不会发生浮点数误差，除非那个定值本身就有误差
(5) 在相同位置，相同的影响造成的结果是相同的，除非存在人为的随机性，即参与运算的各元素及运算过程确定时运算结果也确定
(6) 浮点数误差的发生情况通常关于原点呈中心对称

这些只是一些关于浮点数误差的非常基础的结论，如需要深入研究，可以查阅相关资料或国际标准。

出于效率原因，Minecraft 中大部分三角函数不会被准确地求出，而是在 MathHelper 类中通过查表（间隔为$\frac{360}{65536}°$的正弦函数表）得到的，求弦函数的平均误差约在 10^{-5} 的数量级，需要留意。

上面也提到过，在很多时候，绝对值小于 10^{-5} 或 10^{-7} 的数值会被替换为 0，这本是 Mojang 应对浮点误差的一种策略，但有些时候确实可能造成一些奇怪的问题，这里算是给出一个考虑方向。

除去各种误差，另一件必须指明的事是 Minecraft 中查询与给定范围相交的实体算法中存在一个严重 Bug。

给定一个 AABB（一个各面都垂直某一坐标轴的长方体区域），查询与该 AABB 相交的实体，则只有与该 AABB 的切比雪夫距离小于 2m 的区段（Section，又称子区块，Subchunk）中，或者说与该 AABB 向各方向扩大 2m 的范围不计边界地相交的区段中的实体会被“看到”，其它区段中的的实体在此时是“看不到”的。对于较小的实体，这一般不会造成问题，但如果实体的高度超过 2m（不含，下同）或宽度超过 4m，问题就会开始显现，它们超出所在区段超过 2m 的部分在某种意义上是不可见的。如图 2.5 所示，绿色方框标示的实体坐标点位于最中央的区块（俯视图只能画出来区块），则只有与给定的 AABB 与红色区域相交时该实体才会被看到，或者说该实体黄色部分与灰色部分别是可见与不可见的。

在实际操作中，如果处理大型实体时出现了活塞无法推动它们等异常现象，这将是一个主要的考虑方向。Fallen_Breath 曾发了一个视频来演示这一 Bug[3]，在这一特性较为明显的地方文中也进行了专门讲解。

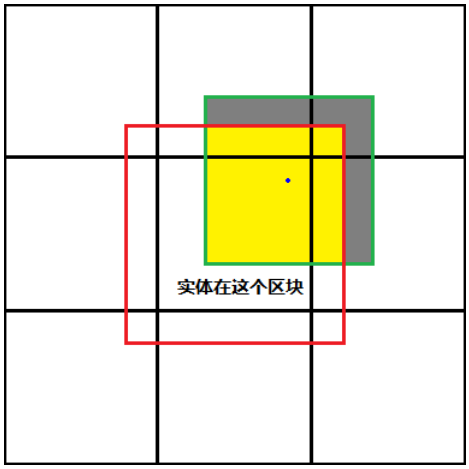

图 2.5 getEntities()方法的 Bug 示意图（俯视）

3 实体自由运动相关公式推导

3.1 符号定义及公式推导

首先，再次强调一下那个很让人迷惑的事实，Motion 不是速度，至少不完全是。

同一维度中，对于任意有效实体，定义如下：

该实体的第 *n* gt（下文未特殊说明时 n 为正整数）自该实体第 *n* 次更新开始时开始，第 *n*+1 次更新即将开始而未开始时（如果期间实体被移除，则为该实体被移除前开始的最后一次运算结束时）结束。

在该实体的第 1 gt 即将开始而未开始时，实体在 x、y、z 轴上的坐标分别为x_0、y_0、z_0，在 x、y、z 轴的上 Motion 分别为v_{x0}、v_{y0}、v_{z0}。一游戏刻长度

$$t_0 = 1gt \tag{3.1.1}$$

在该实体的第 n gt 结束时，实体在 x、y、z 轴上的坐标与x_0、y_0、z_0的差分别为x_n、y_n、z_n，在 x、y、z 轴的上 Motion 分别为v_{xn}、v_{yn}、v_{zn}。

在该实体的的第 *n* gt 内，实体在 x、y、z 轴上的位移分别为

$$\Delta x_n = x_n - x_{n-1} \tag{3.1.2}$$

$$\Delta y_n = y_n - y_{n-1} \tag{3.1.3}$$

$$\Delta z_n = z_n - z_{n-1} \tag{3.1.4}$$

其值与t_0的比值分别为该实体在其第 *n* gt 中在 x、y、z 轴上的平均速度。

该实体每 gt 会受到加速度及阻力作用，当各轴上加速度及阻力作用都视为每 gt 仅进行一次且加速度和阻力系数均不变时，实体在 x、y、z 轴上的 Motion 在加速度作用后的值与作用前的值的差值与t_0的比值（加速度）分别为a_x、a_y、a_z，实体在 x、y、z 轴上的的 Motion 在阻力作用后的值与作用前的值的比值（速度乘数）分别为k_x、k_y、k_z。

以上定义中各量在无需或无法指定具体坐标轴时用于指定坐标轴的角标可省略，位移符号中不是角标但用于指定坐标轴的字符的以 d 替换。即x_0、y_0、z_0统一记作d_0，v_{x0}、v_{y0}、v_{z0}统一记作v_0，x_n、y_n、z_n统一记作d_n，Δx_n、Δy_n、Δz_n统一记作Δd_n，a_x、a_y、a_z统一记作a，k_x、k_y、k_z统一记作k。

为实现速度乘数与 **wiki** 中阻力系数的转换，有

$$k = 1 - ft_0 \tag{3.1.5}$$

无论何时。

以 MDA 型实体为例：

对任意自然数 n_0，有

$$v_{n_0+1} = kv_{n_0} + at_0 \tag{3.1.6}$$

令 n_0=0，则可以得出

$$v_1 = kv_0 - gt_0 \tag{3.1.7}$$

$$v_2 = k^2v_0 - gt_0k - gt_0 \tag{3.1.8}$$

$$v_3 = k^3v_0 - gt_0k^2 - gt_0k - gt_0 \tag{3.1.9}$$

$$\vdots$$

$$v_n = k^nv_0 - gt_0\sum_{i=0}^{n-1} k^i \tag{3.1.10}$$

$$\because (1-k)\sum_{i=q}^{r} k^i \tag{3.1.11}$$

$$= k^q + k^{q+1} + \cdots + k^{r-1} + k^r - k^{q+1} - k^{q+2} - \cdots - k^r - k^{r+1}$$

$$= k^q - k^{r+1}$$

$$\therefore \sum_{i=q}^{r} k^i = \frac{k^q - k^{r+1}}{1-k} \tag{3.1.12}$$

令式(3.1.12)中 q=0，r=n-1，代入式(3.1.10)中，得

$$v_n = k^nv_0 + \frac{at_0(1-k^n)}{1-k} \tag{3.1.13}$$

由运算过程可知

$$\Delta d_n = v_{n-1}t_0 \tag{3.1.14}$$

将式(3.1.13)代入其中，有

$$\Delta d_n = k^{n-1}v_0t_0 + \frac{at_0{}^2(1-k^{n-1})}{1-k} \tag{3.1.15}$$

易知

$$d_n = \sum_{i=1}^{n} \Delta d_i t_0 \tag{3.1.16}$$

将式(3.1.15)带入其中，有

$$d_n = v_0t_0\sum_{i=1}^{n} k^{i-1} + \frac{at_0{}^2n}{1-k} + \frac{at_0{}^2}{1-k}\sum_{i=1}^{n} k^{i-1} \tag{3.1.17}$$

令式(3.1.12)中 q=0，r=n-1，并将其代入式(3.1.17)中，有

$$d_n = \frac{v_0t_0(1-k^n)}{1-k} + \frac{at_0{}^2[k^n + n(1-k) - 1]}{(1-k)^2} \tag{3.1.18}$$

解得

$$v_0 = \frac{d_n {t_0}^{-1}(1-k) - a t_0 n}{1-k^n} + \frac{a t_0}{1-k} \tag{3.1.19}$$

令式(3.1.15)中$\Delta d_n = 0$，解得

$$n = 1 - log_k \left[1 - \frac{v_0(1-k)}{a t_0}\right] \tag{3.1.20}$$

可以证明，将求得 n 向下取整[4]（取不大于 n 的最大整数），代入式(3.1.18)中，可求得运动折返点（若有）

令$n \to +\infty$，则有$k^n \to 0$（0<k<1）

则使Δd_n绝对值最大的

$$\Delta d_{max} = \frac{a {t_0}^2}{1-k} \tag{3.1.21}$$

若此时 a=0，使d_n绝对值最大的

$$d_{max} = \frac{v_0 t_0}{1-k} \tag{3.1.22}$$

若

$$a_x = a_z = 0, a_y = -g \tag{3.1.23}$$

$$k = k_x = k_y = k_z \tag{3.1.24}$$

则有

$$\Delta x_n = k^{n-1} v_{x0} t_0 \tag{3.1.25}$$

$$\Delta y_n = k^{n-1} v_{y0} t_0 - \frac{g {t_0}^2 (1-k^{n-1})}{1-k} \tag{3.1.26}$$

$$\Delta z_n = k^{n-1} v_{z0} t_0 \tag{3.1.27}$$

$$x_n = \frac{v_{x0} t_0 (1-k^n)}{1-k} \tag{3.1.28}$$

$$y_n = \frac{v_{y0} t_0 (1-k^n)}{1-k} - \frac{g {t_0}^2 [k^n + n(1-k) - 1]}{(1-k)^2} \tag{3.1.29}$$

$$z_n = \frac{v_{z0} t_0 (1-k^n)}{1-k} \tag{3.1.30}$$

易知实体第 $n$gt 中位移俯仰角正切值

$$tan\alpha_n = -\frac{\Delta y_n}{\sqrt{\Delta {x_n}^2 + \Delta {z_n}^2}} \tag{3.1.31}$$

将式(3.1.25)、(3.1.26)、(3.1.27)代入式(3.1.31)，得

$$tan\alpha_n = \frac{g t_0 (k^{n-1} - 1)}{k^{n-1}(1-k)\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} - \frac{v_{y0}}{\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} \tag{3.1.32}$$

由式(3.1.28)整理得

$$\frac{v_{x0}t_0(1-k^n)}{1-k}=x_n \tag{3.1.33}$$

将式(3.1.33)与式(3.1.30)相乘，得

$$zv_{x0}=xv_{z0} \tag{3.1.34}$$

即运动轨迹始终在同一垂直 xOz 平面的一平面 θ 上；

由式(3.1.28)、(3.1.30)分别解得

$$n=log_k\left[1-\frac{x(1-k)}{v_{x0}t_0}\right] \tag{3.1.35}$$

$$n=log_k\left[1-\frac{z(1-k)}{v_{z0}t_0}\right] \tag{3.1.36}$$

分别代入式(3.1.29)中得

$$y=\frac{v_{y0}(1-k)+gt_0}{v_{x0}(1-k)}x-\frac{gt_0{}^2}{1-k}log_k\left[1-\frac{x(1-k)}{v_{x0}t_0}\right] \tag{3.1.37}$$

$$y=\frac{v_{y0}(1-k)+gt_0}{v_{z0}(1-k)}z-\frac{gt_0{}^2}{1-k}log_k\left[1-\frac{z(1-k)}{v_{z0}t_0}\right] \tag{3.1.38}$$

可以证明，式(3.1.37)对应曲面与平面 θ 的交线=式(3.1.38)对应曲面与平面 θ 的交线=式(3.1.37)对应曲面与式(3.1.38)对应曲面的交线，且运动轨迹点全部在该曲线上

设初速度仰俯仰角为 α，偏航角为 β，大小为，则各轴分量

$$v_{x0}=-v_0\,cos\,\alpha\,sin\,\beta \tag{3.1.39}$$

$$v_{y0}=-v_0\,sin\,\alpha \tag{3.1.40}$$

$$v_{z0}=cos\,\alpha\,cos\,\beta \tag{3.1.41}$$

分别代入式(3.1.28)、(3.1.29)、(3.1.30)中，有

$$x_n=-\frac{v_0\,cos\,\alpha\,sin\,\beta\,t_0(1-k^n)}{1-k} \tag{3.1.42}$$

$$y_n=-\frac{v_0\,sin\,\alpha\,t_0(1-k^n)}{1-k}-\frac{gt_0{}^2[k^n+n(1-k)-1]}{(1-k)^2} \tag{3.1.43}$$

$$z_n=\frac{v_0\,cos\,\alpha\,cos\,\beta\,t_0(1-k^n)}{1-k} \tag{3.1.44}$$

下面认为

$$v_{y0}=-a_0t_0\,sin\,\alpha+v_{y0}^{'} \tag{3.1.45}$$

其中a_0是合初始加速度（如爆炸推进加速度），v_{y0}'是加速前的 y 轴 Motion，由此可得

$$y_n=-\frac{t_0\left(-a_0t_0\,sin\,\alpha+v_{y0}^{'}\right)(1-k^n)}{1-k}-\frac{gt_0{}^2[k^n+n(1-k)-1]}{(1-k)^2} \tag{3.1.46}$$

可以证明

$$a_0 t_0 = \frac{(1-k)\sqrt{{x_n}^2 + {z_n}^2}}{cos\,\alpha\, t_0(1-k^n)} \tag{3.1.47}$$

代入式(3.1.46)并解得

$$tan\,\alpha\sqrt{{x_n}^2 + {z_n}^2} = \frac{g{t_0}^2(1-k^n)}{(1-k)^2} - \frac{g{t_0}^2 n - v'_{y0}t_0(1-k^n)}{1-k} - y_n \tag{3.1.48}$$

对所有有效实体，式(3.1.12)、(3.1.16)、(3.1.31)通用；若式(3.1.23)和(3.1.24)成立 (3.1.34) 、(3.1.39) 、(3.1.40) 、(3.1.41)、(3.1.45)式也通用。

对于 DA 型实体，式(3.1.13)通用

对于 DF 型实体，有

$$v_n = k^n v_0 + \frac{at_0(k-k^{n+1})}{1-k} \tag{3.1.49}$$

对于 M（速度不变，仅移动），N（无法移动）型实体，有

$$v_n = v_0 \tag{3.1.50}$$

所有其它运动形式对应的公式可由式(3.1.13)，(3.1.49)或(3.1.50)与下表中各式结合推导出。

表 2.1 运动形式对应 tick 末位移与 tick 末 Motion 关系式

类别	公式	类别	公式
MDA	$\Delta d_n = V_{n-1}t_0$	AMD	$\Delta d_n = k^{-1}v_n t_0$
DMA	$\Delta d_n = kv_{n-1}t_0$	ADM	$\Delta d_n = v_n t_0$
DAM	$\Delta d_n = v_n t_0$	M	$\Delta d_n = v_0 t_0$
MAD	$\Delta d_n = v_{n-1}t_0$	N	$\Delta d_n = 0$

个人认为，式(3.1.13)、式(3.1.16)的推导大概就是一个数列求和题，或者说就是初中的证明规律的那种题，后面的推导可以说很无脑，硬解就好了，有时候代入消元还是挺管用的。

最后需要说明，完全准确的实体运动公式可不是这样就能推出来的，因为存在不可避免的浮点误差，但是这套公式已经足以在几乎所有情况下使用了。

3.2 公式的直接及拓展应用

在上一节给出关于 MDA 实体的 31 个等式中，只有 9 套是最常用的：

(3.1.13)在某一刻末 Motion 和加速度已知的情况下，求任意刻实体运算结束时的 Motion。这在一些 TNT 炮设计中选择 TNT 数量的过程中是比较有用的，因为这个 Motion 直接参与了射出的初速的决定。

(3.1.15)在某一刻末 Motion 和加速度已知的情况下，求任意刻内的平均速度，比较重要。

(3.1.18)在某一刻末 Motion 和加速度已知的情况下，求任意刻末的位移，重要性可想而知。

(3.1.19)在某一刻末位移、加速度和运动时间已知的情况下求出初速度，重要性可想而知。

(3.1.20)在某一刻末 Motion 和加速度已知的情况下，求实体的折返时间，进而求出折返点（如最高点，向流体上游抛出的物品的折返点等）。

(3.1.21)在时间不受限的情况下求出实体的理论最大速度，可以用于稳定的或周期性的加速过程中（如重力，行走，划船等）实体最终速度的估计，非常重要。

(3.1.22)在时间不受限的情况下求出实体的理论最大位移，但实际应用较少。

(3.1.32)求出实体的合速度俯仰角，但实际应用不多，有时可用与碰撞判定相关计算

(3.1.34)、(3.1.37)、(3.1.38)给出了实体在空中平滑化的运动路径的解析式，可用于快速模拟实体运动轨迹，但实际应用不多，推这个主要就是想知道实体运动轨迹的解析式究竟是什么。图 3.2.1 中可以看出，运动轨迹在开始时接近为倾斜直线，而后发生较大转折，最后无限趋近于水平初速决定的一个竖直线。

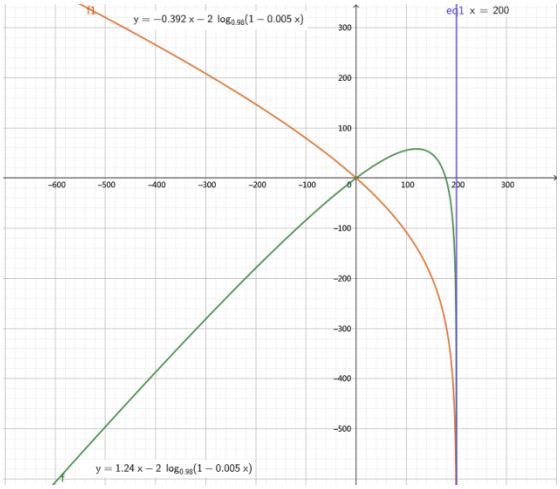

图 3.2.1 5m/gt 初速 37 度斜向正东发射的 TNT 运动轨迹（平滑化）

(3.1.48)一个“万能方程”，可以快捷地求出某种瞬时加速装置（如 TNT 炮）的最远落地位移等数据。

直接应用别的估计都好说，这里就只谈一下 *n* 的确定。*n* 说简单了就是从自己选定的一个时间点（多数为实体创建或开始自由运动时）以来经过的刻数，或者说就是实体开始的运算周期数。但是，因为实体的运算过程问题，在计算位移时有时 *n* 会有 1 的偏差。例如，TNT 实体的爆炸是在位置更新（移动）后才进行的，TNT 的初始引信时间为 80gt，在引信时间为 0 时尝试爆炸会成功，从图中很容易想出它从创建到爆炸计算了 80 周期的运动，所以要想求出 TNT 的爆炸点可以选取 *n* 为 80；但是，如果是想知道一个末影珍珠在某一时刻触发了哪个拌线钩则需要将经过的刻数减 1 作为 *n*，因为它检查方块网格碰撞（触发绊线）是在位置更新前就进行了，此时它移动的次数会比它开始的运算周期数少 1。这一类问题较复杂，对每种实体详细的说明在这里就显得太多余了，不过第七章中的介绍应该会解决多数常见的此类问题。

另外，如果可以保证实体的运动状态没有改变且实体一直存在，3.1 节公式中 n 的值可以取任意整数，而不局限于正整数。非整数值的 n 一般是没有意义的。

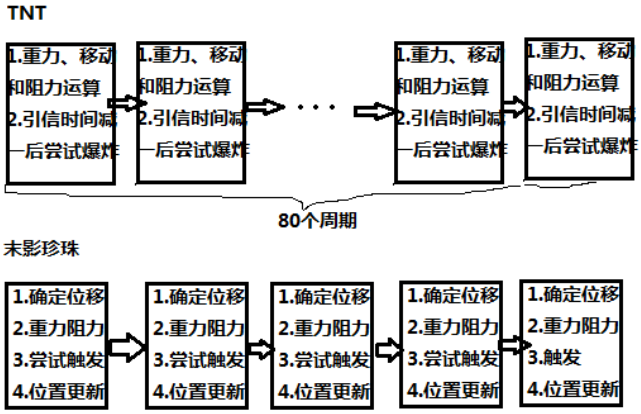

图 3.2.2 TNT 和末影珍珠的运算流程简图

前面说的 MDA、AMD 等由三个步骤组成的运动只是理想情况，一般只适用于每 gt 只考虑至多一次加速、至多一次阻力作用和至多一次移动的情况（尽管已经满足大部分需求了），很多时候，特别是在计算玩家运算的时候经常会有实体在同一 gt 多次受到加速度、阻力甚至可能有多次移动的情况。一个典型的例子就是逆水行舟，根据源码，可以得到其在水中时水平方向上运动运算流程如下：

(1) 受到水流推动

(2) 受到大小为 $0.1gt^{-1}$ 的流体阻力，即 k 值为 0.9

(3) 受到沿视线的水平投影方向 0.04 或 $0.005m/gt^2$ 的动力

(4) 移动

也就是说，在它的运算过程中出现了两次加速，不能直接套用公式。

若要解决这个问题，我们不妨暂时忽略移动这一步骤，先将其下一刻的水平某轴上 Motion 用一个式子表示出来，其中a_f为流体加速度在该轴上的分量，a_c为动力加速度在该轴上的分量，k 为步骤三后与步骤三前该轴上 Motion 的比值，v_0、v_1分别为原 Motion 与下一刻该轴上的 Motion：

$$v_1 = k(v_0 + a_f t_0) + a_c t_0 = kv_0 + (a_f k + a_c)t_0 \tag{3.2.1}$$

现在下一刻的 Motion 已经被表示成一个关于v_0的一次代数式，可以发现这与前面公式推导时 DA（先阻力后加速）型实体的相似。所以，我们将式$a_f k + a_c$作为 a，k 作为 k 带入 DA 型式的 Motion 公式，也就是式(3.1.13)中，有

$$v_n = k^n v_0 + \frac{t_0(a_f k + a_c)(1 - k_n)}{1 - k} \tag{3.2.2}$$

易知这里

$$\Delta d_n = v_n t_0 \tag{3.2.3}$$

这与 DAM 和 ADM 型实体相同。因为上面的推导过程中我们假定v_1是由v_0先通过阻力作用，再经过加速得到的（也就是先乘以 k，再加上at_0），所以这里我们选择 DAM 型实体，此后余下的公式也可以类似地代入使用。

有时候，多个加速度或阻力都是连在一起的（中间有移动过程时也认为它们是连在一起的），这时我们甚至可以不用求v_1的表达式，直接用加法合并加速度，用乘法合并阻力的 k 值就可以了。例如，某个实体每 gt 运算顺序是 $MA_1A_2A_3D_1D_2D_3$，这时可以将 $A_1A_2A_3$ 过程中的加速度相加，将 $D_1D_2D_3$ 中速度乘数（k 值）相乘分别得到 a 和 k。可以发现这个实体的运动过程在化简后实际上就是 MAD，将求得 a 和 k 带入对应公式中即可。

实体的移动过程可能在多个加速或阻力作用过程中间，可能不能直接带入任何与位移相关的现有公式，这时就需要对现有公式进行变形或直接推出一套新公式才能代入使用。再举一例，某实体的运动运算过程为 $A_1D_1MA_2DA_3$，我们想要从某个初速度求得任意时刻的位移，也就是要推导出(3.1.18)式的对应版本。利用前面的方法我们可以很轻松的得到这种实体的加速度和阻力，以及

$$\Delta d_n = v_{n-1}t_0k + a_1t_0^2k \tag{3.2.4}$$

此时完全可以推导出对应公式。

有些时候阻力和加速度作用的周期并不一定是 1gt，如计算玩家向正前方跑跳时的平均水平速度时周期是玩家从跳起到再次落地的时间，此时可以对 n 的取值除以周期并向上取整代入导出的公式以求得周期末的数据，借此求得具体时刻的数据。不过此时计算位移的意义已经不大，通常还是实验更有效。但如果是希望求解某个受到相对于时间有周期性的加速度和阻力系数影响的实体的最终速度（平均的或是稳定后周期内的具体变化情况），可以假设一个初始 Motion 为v_0，并求出一个周期后的 Motion 关于v_0的表达式$f(v_0)$，令$f(v_0) = v_0$，解得v_0即为运动稳定后的某个周期的初始 Motion，进而可以求得运动稳定后的任意周期内的实体速度变化情况。

还是开头那个例子，先假设一个周期从玩家即将跳起时开始，到玩家第二次即将跳起时结束，长度为n_0。除n_0和初始水平 Motion v_0外，还有下列几个量影响周期末水平 Motion：

(1) 玩家在空中的加速度a_a，玩家疾跑时为 $0.026\mathrm{m/gt^2}$与前向加速的系数（一般为 0.98）的乘积

(2) 玩家在地面的加速度a_g，玩家疾跑时数值为玩家的 generic_movement_speed 属性值与前向加速度系数的乘积加 $0.2\ \mathrm{m/gt^2}$

(3) 玩家在空中的速度乘数k_a，固定为 0.91

(4) 玩家在地面的速度乘数k_g，等于判定所得滑度与 0.91 的乘积

易知周期内玩家在空中的运动时长为$(n_0-1)\ gt$，地面上移动时长为 1gt，结合玩家的运动运算顺序可以导出周期末水平 Motion：

$$v_{next}=(v_{0+}a_gt_0)k_gk_a^{n_0-1}+\frac{a_at_0\left(k_a-k_0^{n_0}\right)}{1-k_a}+a_gt_0\#(3.2.5)$$

令$v_{next}=v_0$，解得

$$v_0=\frac{a_at_0\left(k_a-k_a^{n_0}\right)}{(1-k_a)(1-k_gk_a^{n_0-1})}+\frac{a_gt_0k_gk_a^{n_0-1}}{1-k_gk_0^{n_0-1}}\#(3.2.6)$$

接下来，可以得出玩家在周期内的水平 Motion 变化情况。

可以发现，$1-k^n$似乎是一个有特殊意义的值。实际上，在一些实体运动的过程中，$1-k^n$等于原来的一半的意义还是比较丰富的（不止这些）：

(1) 在加速度和阻力恒定的从无初速开始的加速运动中，这时 Motion 约是最大 Motion 的一半，合加速度约是最大合加速度的一半，由实际位移与相同时间以最大速度运动的位移之差是这个差值的最大值的一半。

(2) 在没有加速度但有初速且阻力固定的运动中，这时阻力加速度约是最大阻力加速度的一半，移动的位移约是最大位移的一半，Motion 约是初始 Motion 的一半。

不敢说这个的实际用途有多大，但这确实有助于我们理解这些公式。

4 实体移动过程及碰撞机制

4.1 基于 Entity.move()方法

首先需要说明，这是几乎所有实体（包括玩家、除恼鬼外大部分生物、交通工具、TNT、物品、落沙和被活塞潜影盒推动时的弹射物等）共有的运动形式，通用性极强，所以很重要，有些内容可能不很好懂，但最好还是了解一下。部分实体重写了该方法，如潜影贝重写了该方法使其在被潜影盒推动时尝试传送。

此处输入位移均指作为参数传到 move()方法中的位移向量。

研究相关源代码（net.minecraft.entity.Entity:517）,可知这种实体移动过程如下，其中 2-4 步合称“**位移趋势预处理**”，**输入位移**（即作为参数传入该方法的位移，也称**位移趋势**）经预处理得到“**实际位移趋势**”：

1 若实体的 noClip 属性为 true，直接移动碰撞箱及坐标，跳过其它步骤（518-520）

2 若移动源为活塞，进行相关调整（在 adjustMovementForPiston()方法中定义，主要为限制位移和坐标轴）（522-527）

3 尝试应用蛛网和浆果丛对本次移动的减速作用（529-534）

4 进行潜行相关调整（在 adjustMovementForSneaking()方法中进行，默认为不调整，潜行且着地的玩家在处理自身 Motion 产生的移动时会避免从高度大于 stepHeight，即 0.6m 的边缘掉落）（536）

5 对方块，世界边界和固体实体的碰撞检查（adjustMovementForCollisions()）（537）

5.1 初步检查（709）

5.2 对能直接移上去的方块相关的修正（714-727）

6 如果剩余的位移趋势大于 10^{-7}m，移动碰撞箱和坐标（需注意，碰撞箱和坐标无直接关系）（538-541）

7 更新 horizontalCollision（水平碰撞），verticalCollision（垂直碰撞），onGround 状态（545-547）

8 进行摔落相关运算（此处由实体的 fall()方法定义）默认如下（548-550）

8.1 若已落地且 FallDistance>0,进行摔落伤害运算并置零 FD（916）

8.2 否则，增加 FallDistance（921）

9　将 X、Z 轴中发生碰撞（实际位移未达到实际位移趋势）的轴上 Motion 归零（552-558）

10　若发生竖直方向碰撞（顶头或落地）则进行实体着陆方块（见第 5 节开头）的 onEntityLand()方法定义的一个操作（560-567）

10.1　空气等大部分方块默认为置零 y 轴 Motion（Block.java:374）

11　若着地且未潜行，则尝试产生下方方块在实体从上方经过时的效果（调用对应方块的 onSteppedOn()方法）（565-567）

11.1　默认无操作，其它方块如红石矿石进行了重写（Block.java：335）

12　行走声音相关（569-597）

13　检查方块网格碰撞（checkBlockCollision()）（600）

14　下方方块的减速（影响 Motion）作用（608-609）

15　此时若在火或熔岩中，则被引燃（610-619）

现在，这里主要讨论 5 和 13，余下需要讲解的内容将在后文中讲解。

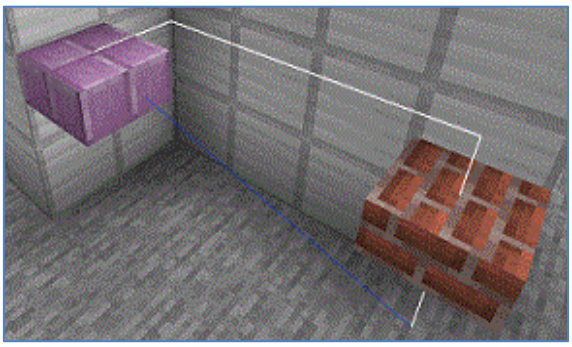

图 4.1.1 碰撞检查中实体移动路径示意图

在 5 中，游戏首先会生成一个可能的碰撞范围，包括路径附近的方块、世界边界（仅在实体于边境内时）和固体实体（仅包含不直接或间接共同骑乘统一实体的实体）。然后，游戏会尝试进行一次初步的碰撞检查，即检查实体在按沿各轴方向运动时有没有发生碰撞。实体碰撞箱相交的物体会被排除，这意味着实体在卡到方块和固体实体内部时可以轻松地出来，但不能重新进入。检查一个轴的过程大概可以理解为为：从某一起点 A（原坐标或上一个轴的终点）在对应轴上将坐标加上输入位移趋势在该轴上分量的数值得到另一点 B，检查实体碰撞箱从 A 到 B 运动过程中有无碰撞，若有，将碰撞点作为该轴的终点和下个轴检查的起点，否则将 B 作为该轴的终点和下个轴检查的起点。

在 1.14 及以后，沿轴检查时，Y 轴总是被最先检查，然后是 XZ 轴中实际位移趋势绝对较大的轴（若相等，则是 X 轴），最后是余下的一个轴；1.14 之前则总是按 YXZ 的顺序进行。这一改动会使一些本来具有方向性的

设备不再依赖朝向或者直接失效。例如，假设存在一款想沿轴方向发射 TNT 的 TNT 炮，但由于只进行了在两侧用几个方块阻挡炮弹点燃时的随机运动的矫正所以推进方向会略微偏离预计的沿轴方向。那么在 1.14 前，这一推进速度偏差在预计的方向所在轴为 Z 轴时会因为炮弹首先尝试沿 X 轴检查碰撞并撞到方块被消除，但所在轴方向为 X 轴时这一推进速度偏差则一般不会被消除。在 1.14 之后，由于游戏可以判断出发射方向并先沿这个方向进行碰撞检查，所以这一推进速度偏差无论目标方向在哪个轴上都不会被消除。

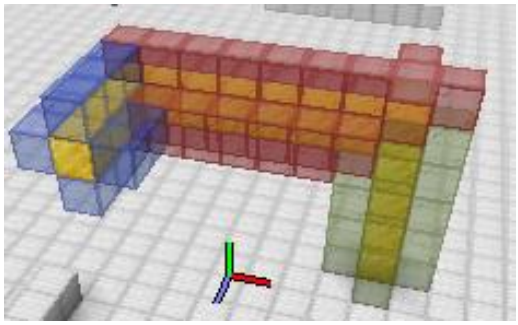

图 4.1.2 金块管道示意图

可以这样理解，虽然一次移动从坐标变化的角度看来是瞬时的，但是这种移动从碰撞判定的角度却完全可以分成三次独立的**沿轴移动**。也就是说，如图 4.1.1 所示，这种实体移动过程中可以认为实体是沿白线而非蓝色直线运动的，假如选取大小合适的实体，再赋予合适的速度，实体可以在图 4.1.2 中管道内部（金块标注的部分，实际上应为空气）由下向上穿过管道。

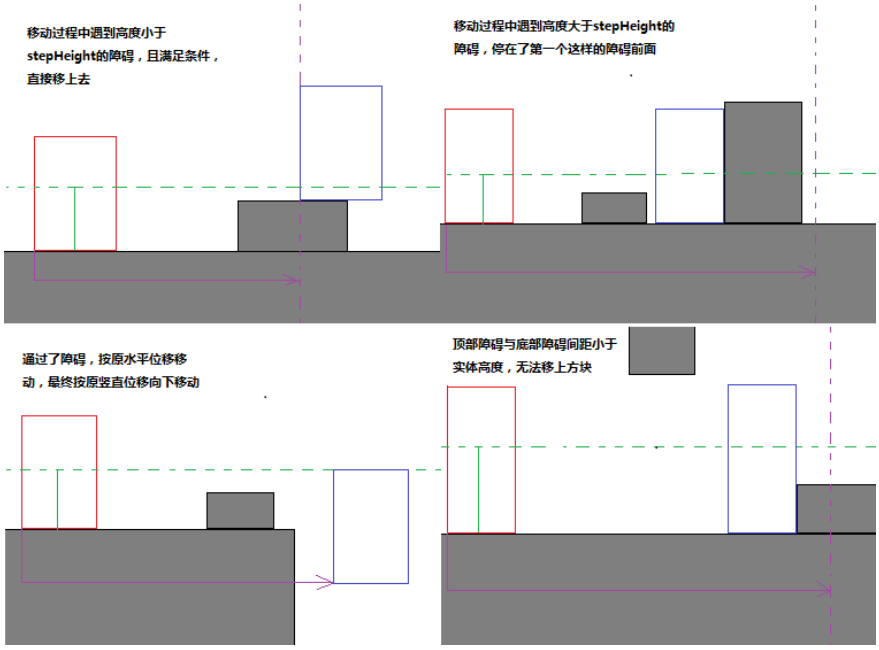

图 4.1.3 实体直接移上方块相关调整示意图

然后，游戏会进行实体直接走上方块的相关调整，大概就是在保证不会出现实体走过去高度（顶部 Y 坐标与实体原坐标的 Y 坐标的差值）大于 stepHeight 的障碍和穿过小于实体碰撞体积的洞的现象且水平位移不多于输入位移的情况下尽可能让水平位移更远，并在不使实际 Y 轴位移多于输入趋势的情况下的让实体尽可能着地。好吧，通俗地完全解释还是比较难的，大家直接按经验来也好，不过还是给出一些例子，图 4.1.3 中红框和蓝框为移动前后的实体碰撞箱，灰色部分是固体方块，绿色实线标明了实体的 stepHeight，绿色虚线标明了实体一次最高能移上的高度，紫色实线标明了实体的实际位移趋势。前三幅都较好理解，第四幅有时可能算是一个 Bug，需要注意。理论上，所有实体都可以有这个能力，但大多数实体的 stepHeight 被设为 0 从而没有外在表现，未被骑乘或未被控制的猪和未被骑乘或未被控制的赤足兽为 0.5，可以迈上或被推上半个方块；客户端玩家和大部分生物则是 0.6，可以迈上或被推上部分不完整方块和船；末影人、溺尸、铁傀儡、马类（HorseBase 及其子类）、服务端玩家（目的可能是实现被活塞推上方块的机制）、被骑乘且被控制的猪和被骑乘且被控制的赤足兽为 1，可以迈上或被推上完整方块。

脚手架和熔岩有一些特殊机制，这使得它们对不同实体可以有不同的碰撞箱。脚手架倒好说，估计大家都试过了。对于熔岩，这一设定使它对赤足兽有了一个 0.5m 高的碰撞箱，这是为了使赤足兽可以稳定地走在上面。

在碰撞运算中，流体会被忽略，这使路径上的流体不会对该实体产生影响，除非在移动结束后实体仍有部分碰撞箱与其相交。

固体实体是一类完全阻挡部分实体移动的实体，目前对大多数实体存在固体性的实体主要有船和活的潜影贝。除了它们以外，isPushable()方法返回 true 的实体（包含矿车、船、鹦鹉、未被骑乘的马类、活的不是旁观者而且未处于可攀登方块所在方块网格内的大部分 LivingEntity）对矿车和船也是固体的。另外，共同直接或间接地骑乘同一个实体的实体间相互不表现出固体性。

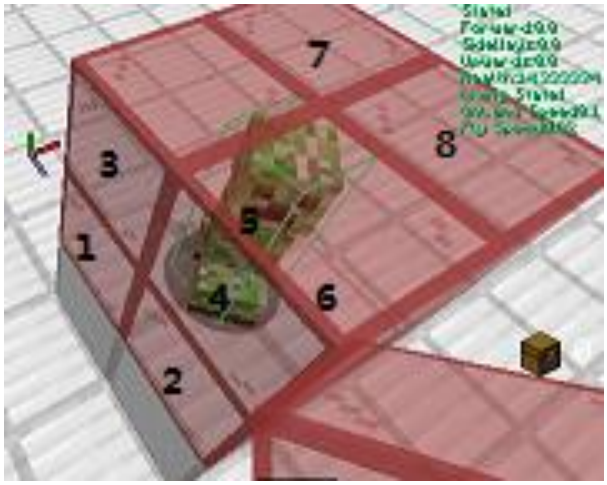

图 4.1.4Creeper 举例图

第 13 步中，如果与实体的碰撞箱从各方向向内缩 0.001m 后的区域相交的方块网格所在区块均被加载，实体会尝试调用那些方块网格中的方块定义的 onEntityCollision()方法，以实现与相交的方块的交互，如气泡柱变速、仙人掌的伤害运算和促使绊线检查是否被触发（只是促使它检查是否被触发，真正触发还需要它的检查范围内，也就是所在方块内部最下方 0.15625m 内同时存在有效实体碰撞箱）这一运算被称为**检查方块网格碰撞**。检查顺序为从西北到东南，从下到上，但一般不必关心。具体实现代码可以参考 Entity.java:835,checkBlockCollision()。举个例子，图 4.1.4 中的苦力怕站在这四个铁块的正中央，绿色线框是它的碰撞箱，红色玻璃标记的 8 个方块网格都与苦力怕的碰撞箱相交，黑色数字标明了对它们进行检查的顺序；铁块所在网格范围在微量缩小后并不与苦力怕相接触，认定为不相交。

4.2 基于 raycast 的弹射物自主移动

这一类移动方法中决定运动的碰撞判定部分主要用于除鱼竿浮标和烟花火箭外的弹射物的自主移动，烟花火箭虽然在决定运动碰撞判定中使用Entity.move()方法，但是在确定究竟撞到了什么方块或实体时仍会使用这种碰撞检查过程。

raycast 是指检查一个点沿一个有向线段移动的过程中是否发生或者发生了怎样的碰撞的过程。在解释 Minecraft 中的 raycast 前，要先引入“方块网格”的概念。

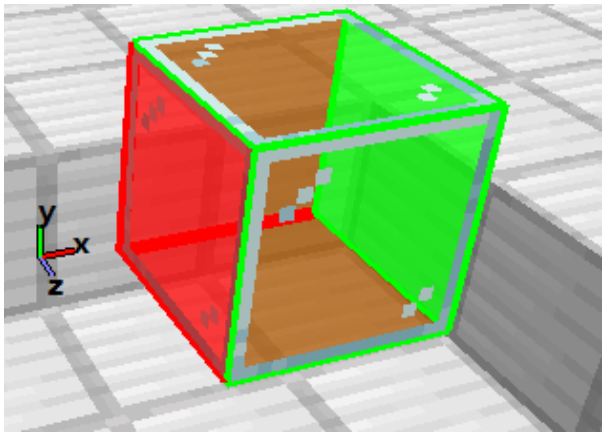

图 4.2.1 方块网格示意图

在一个维度中，每个方块坐标对应一个**方块网格**，一个方块网格范围内各点对应方块坐标相等，或者说将各轴坐标向下取整得到同一个点的所有点对应一个方块网格。一个方块网格大小等于一个完整方块，内部只能有一个方块，没有任何点不属于任何一个方块网格，也没有任何点属于多个方块网格。图(4.2.1)中的玻璃方块占据了一个方块网格，方块内部以及标红的面（西面、北面和下面，不计边界）、棱（与西北偏下的顶点直接相邻的棱，不计端点）和西北偏下顶点属于该方块网格，其它面、棱和顶点不属于该方块网格，各轴正方向在图中以 F3 界面下准星的形式指出。

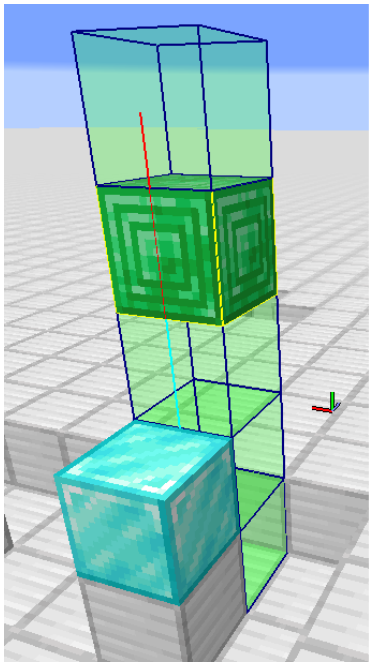

图 4.2.2 raycast 示意图一

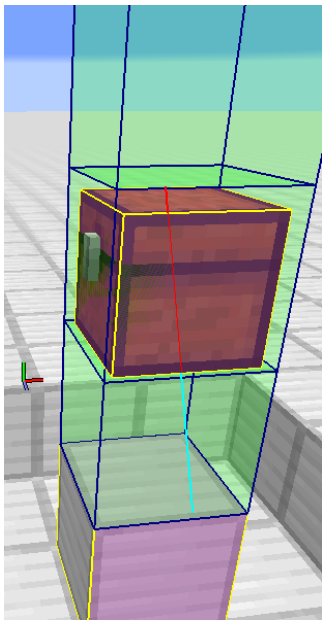

图 4.2.3 raycast 示意图二

在对于方块的 raycast 的过程中，原线段向两端延长一个微小值后得到的新线段（小到可以忽略，即每一端延长总长度的 10^{-7} 倍）经过的所有方块网格中的方块会被依次检查。

目前版本中，对单个方块的 raycast 主要有以下特点：

(1) 线段在方块内部或穿过了方块表面，游戏认定发生了碰撞

(2) 如果线段的起点或终点在方块表面而且其他部分都在方块外部，游戏会认定碰撞没有发生
(3) 如果线段经过且只经过了其表面，除非线段只有起点或终点经过了其表面，游戏认定发生了碰撞
(4) 如果线段起点终点重合或近似重合，游戏总会认定碰撞没有发生
(5) 如果线段没有经过该方块的任何部分，游戏会认定碰撞没有发生

举个例子，图 4.2.2 中的线段以铁块顶端作为起点，竖直向上擦过钻石块和绿宝石块的表面，到达较起点高 3.5m 的终点。在这个过程中，线段经过的是钻石块的南面，不属于钻石块所在的方块网格，所以钻石块并不会被纳入检查范围；线段经过了绿宝石块的北面，也就是说经过了绿宝石块所在的方块网格，所以绿宝石会被检查，线段在绿宝石块的下方就被阻挡了。图 4.2.3 中绿色（棱为蓝色）方框标明了被检查的方块所在网格的轮廓，黄色方框标明了被检查方块的碰撞箱（若有），青色线条是有向线段中未被阻挡的部分，红色线条是有向线段中被阻挡的部分，其分界处便是碰撞点。

再举一例，图中线段仍是从铁块顶面出发，沿一个斜向上的方向擦过箱子表面，最终到达终点。过程中，因为微量伸长后的线段经过了铁块所占据的方块网格，所以铁块会被检查，但由于只有起点经过铁块表面，铁块并不会阻挡有向线段；尽管线段擦过箱子也是南面，但线段明显经过了其所在网格，所以箱子阻挡了线段。图中标注意义同上。

相对于上面说明的，由这一特性造成的一个更明显的问题就是以这种方式移动的实体有时不会被墙、栅栏、栅栏门、打开的潜影盒、short 属性为 false 的活塞头和 36 号方块超出其所在方块网格的部分（如栅栏的最上方 0.5 格）所阻挡，除非线段也经过了它们所在的方块网格，这一差别理论上也应该可以应用在弹射物矫正中。

在 1.16.1 及之前部分弹射物（如投掷物）检查的是方块的 Outline Shape（即光标对准时显示的黑框）的碰撞。这会使一些碰撞错误地发生（MC-73844），如撞到打开的栅栏门或草；也会使一些碰撞被错误地忽略，如 36 号方块的碰撞。

在基于 raycast 的弹射物运动过程中，游戏首先确定一个以原坐标为起点，以原坐标与位移趋势确定的目标位置为终点的有向线段并用它作为路径对方块进行 raycast。如果发生碰撞，则将路径截短到碰撞点。然后，游戏会尝试获取路径附近的所有 collides()方法返回 true 的实体（例外：区域效果云、闪电束、物品、末影之眼、末影龙的较大轮廓、烈焰人火球、末影龙火球、

凋零之首、幻魔者尖牙、经验球、箭矢、投掷物、鱼竿浮漂、带有 Marker 标签的盔甲架、烟花火箭和羊驼唾沫）的碰撞箱向六个方向各扩大 0.3 米后的立方体，并对这些长方体进行另一套 raycast，机制与对单个方块的 raycast 相似，只是仅在长方体内部的线段会被认为未发生碰撞。如果多个实体碰撞箱拥有公共表面且碰撞点就在这一公共表面，理论上最近一次加载到该维度最早的实体会被碰撞。

因为获取待测实体列表时会先筛选出准备从中取出实体的区段（Subchunk），且只有与以移动路径为体对角线且底面平行坐标平面的一个长方体沿个方向扩大$3+\frac{\text{弹射物碰撞箱宽度}}{2}$的区段会被选中，所以如果实体足够大（高度或宽度的一半大于$3+\frac{\text{弹射物碰撞箱宽度}}{2}$），就会有一部分区域的碰撞被错误地忽略。但是在生存模式中几乎不需要考虑该问题，因为只有恶魂的碰撞箱大小达到了标准且可以与弹射物发生碰撞，而且只有在位置恰到好处，也就是高度接近坐标所在区段的上界时才会造成很少一部分碰撞的丢失。

总的来说，可以这样理解，一次移动中方块碰撞是检查该实体坐标点的直线(而非 Entity.move()中沿轴的折线)运动轨迹上有没有方块阻挡，若有，在碰撞点处截短轨迹；实体碰撞则是检查一个以坐标为中心 0.6*0.6*0.6 的立方体在轨迹上是否发生了碰撞，若有，在碰撞点处截短轨迹。

如果多次移动的轨迹共线且期间路径附近的方块和实体没有发生改动，这几次移动与其合并后的一次移动大致是等价的。

需要注意，有些实体即使 collides()方法返回 true 也不能与该弹射物发生碰撞，如投掷物，羊驼唾沫，烟花火箭还要求以下条件（ProjectileEntity.method_26958）：

(1) 离开发射者，即第一次碰撞箱不与发射者接触且以碰撞箱附近没有与发射者直接或间接地骑乘同一个实体的实体的碰撞箱之后，详见 ProjectileEntity.method_26961()。
(2) 目标实体不是旁观者，也没有死亡或被移除
(3) 目标实体与发射者不共用一个最底层骑乘实体

1、3 满足一个即可，2 必须满足。

也有一些类重写了该方法，在上述三类弹射物的基础上增加或是放宽了一些限制。PersistentProjectileEntity（箭矢三叉戟）的版本在原来的基础上使其不能在落地前第二次击中某一实体（想一下穿透附魔）；火球凋零头和潜影贝导弹的版本在原来的基础上使 noClip 属性为 true 的实体(如带有

NoGravity 标签的盔甲架)被忽略；钓鱼竿浮漂则不仅是原条件情况下可与实体碰撞，物品实体即使 collides()方法返回 false 也是可以的。

至于线与立方体的碰撞检查实现，可以这样说，游戏首先会检查该线与该立方体朝向起点的三个面所在平面的交点，然后检查该交点是否在该面的范围内（边上及与边上距离可以忽略的也算），如果是，那么这个点是有效点。最后游戏会在有效点中选取离起点最近的作为碰撞点，若没有，则判定没有碰撞（Box.java:244）。

同样，这里的方块碰撞检查也是忽略流体的，结果同上。另外，我们还发现这里没有对世界边境进行检查，这就可以解释投珍珠穿越边境的操作了。

要知道，上面研究的方法主要是弹射物的共有部分，即碰撞判定，完整的移动部分是各种实体各自实现的，还需要结合其运算流程研究，详见 7.7-7.9。

4.3 两种移动过程的对比

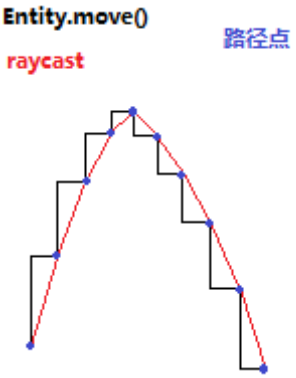

图 4.3 路径点曲线

个人认为，两种移动方式有以下几点主要差别：

(1) Entity.move()方法在碰撞判定中检查的是实体碰撞箱的碰撞情况，实体中只检查固体实体；基于 raycast 的弹射物自主移动对方块是检查的坐标点的碰撞情况，对符合条件的实体是检查的一个另类的“碰撞箱”的碰撞情况。

(2) Entity.move()方法在碰撞判定中检查的是实体按沿轴折线移动碰撞情况，而基于 raycast 的弹射物自主移动对方块是检查的是直线轨迹上的碰撞情况。

(3) Entity.move()方法会导致 X、Z 轴中发生碰撞的轴上 Motion 的归零，Y 轴上发生碰撞时通常也会导致其 Motion 分量的归零，但基于 raycast 的默认并不会，具体处理取决于实体类型。

(4) Entity.move()方法会正确地处理碰撞箱超出其所在方块网格的方块，如墙和栅栏，但基于 raycast 的有时并不能检查到超出部分的碰撞。

(5) Entity.move()方法会触发下方方块的一些效果，如红石矿石发光，但基于 raycast 的默认并不会。

(6) Entity.move()方法会使 onGround（着地）状态更新，但基于 raycast 的并不会。

(7) Entity.move()方法中实体总会检查方块网格碰撞，但在 1.16.1 以前，一部分弹射物（已确认有投掷物，不包括使用 move()方法移动的和箭矢）自主运动时并不会进行这些处理。

(8) Entity.move()不能检查到未加载区块中的碰撞，除非某次移动绝对沿轴，但基于 raycast 的总是可以。

无论哪种移动过程都对 TPS 不敏感，所以单纯的服务端卡顿不会导致碰撞检测发生错误，所以除了高 PING 或开有外挂的玩家外，没有“卡穿墙了”这一说。实际上，即使取消掉客户端上玩家的碰撞判定，在玩家速度不够快时仍不能穿墙。

另外，碰撞检查过程中一般不会发生导致实体卡到墙里或错误地发生碰撞的浮点数误差，因为方块的碰撞箱边界坐标都在浮点数集内。即使产生了浮点数误差，碰撞判定中也有一定的容错机制，确切地说来容错极限是 10^{-7}m，已经足够了。所以一般来说，碰撞检查应该还是比较靠谱的。

5 外界因素对运动的影响

在开始说明前需要先解释一下：

实体所处方块是指实体的实体 blockPos（即实体坐标处的方块坐标）指定位置的方块。

实体下方方块是指实体坐标下方略大于 0.5 格对应方块坐标处的方块。

实体着陆方块一般指实体坐标下方略大于 0.2 格对应方块坐标处的方块，如果原标准对应方块为空气且下方是栅栏、墙或栅栏门，则以后者为准。

5.1 蛛网和浆果丛的减速作用

这一类减速的基础是 Entity 类中的 movementMultiplier 字段。**movementMultiplier** 是一个三维向量，在其有效时，基于 move()方法的移动中的的输入位移趋势各轴分量会在碰撞检测前被乘以 movementMultiplier 中对应轴上分量的值，并将 Motion 置为 0，movementMultiplier 置为无效值。它在 Entity.move()方法的第 13 阶段更新。

我们可以发现，movementMultiplier 只在实体被创建时和 slowMovement 方法被调用时赋值。前一种情况下，取值必定为无效值；后一种情况下也只有两个值可取，分别是浆果丛(0.8,0.75,0.8)和蛛网的(0.25,0.05,0.25)。需注意，赋值是在检查方块网格碰撞时进行的，意味着只有当实体在上次移动后有一部分碰撞箱位于那两种方块所处的微量缩小后的方块网格内且该实体有相关检查（即调用了方块的 onEntityCollision()方法），下次使用 Entity.move()移动时才会受到那两种方块的减速作用。所以，除烟花火箭和鱼竿浮标外的弹射物在一般情况下不会受到这种减速方式的影响。

蜘蛛有一个特判，使得它不会受到蛛网的减速。创造模式和旁观模式飞行中的玩家不会受到该减速作用的影响。

5.2 粘液块和床的回弹

Block.onEntityLand()方法的行为默认是将该实体 y 轴 Motion 设为 0，粘液块和床对其进行了重写以实现其弹起实体的效果。需要注意，由于该方法只会被 move()调用，大部分弹射物自主运动时不会以此方式被影响。不过，理论上，如果能以某种方式调用弹射物的 move()方法（如活塞或潜影贝推动），对弹射物仍是可以造成影响的。某些实体有一些在移动后判断是否着地，如果着地继续反转速度的机制，目的就是禁用弹起，详见第 7 节。

在实体 y 轴 Motion 小于 0 且实体未潜行时，粘液块会将 LivingEntity 的 y 轴 Motion 直接反转，或将非 LivingEntity 实体的 y 轴 Motion 乘以 0.8 之后反转（SlimeBlock.java：33,bounce()）需要注意，这里的反转不是相对上一刻的反转，而是相对调用方法时，因为从上一刻到现在 Motion 已经做了一些其它修改（如重力）。

床的反弹原理与粘液块相似，只是在粘液块的基础上将回弹后 y 轴 Motion 再乘上 0.66 而已。

5.3 蜂蜜块和灵魂沙的减速作用

这一类变速是在基于 Entity.move()方法的实体移动的第 14 阶段中进行的，原理是在实体所在方块坐标处不为水方块、气泡柱或 velocityMultiplier 为 1 的方块时将实体水平两轴上 Motion 乘以其下方方块定义的 velocityMultiplier，否则乘以所在方块定义的 velocityMultiplier。

目前只有蜂蜜块和灵魂沙定义了该值为 0.4，其余全为 1。在 LivingEntity 拥有灵魂疾行附魔时，灵魂沙的 velocityMultiplier 对该实体来说为 1。对于各种飞行状态（包括鞘翅飞行）下的玩家，velocityMultiplier 总为 1。

抛开实现原理不论它也与下面滑度不同，因为它在实体未着地时也有效。

该值仍是只在 Entity.move()方法中被使用，从而无法影响大部分弹射物的自主运动。

5.4 滑度机制

滑度，说简单了，就是描述一个方块有多滑的一个值，值越大说明方块越滑，等于 1 时意味着方块绝对光滑。一个方块上方实体若满足一定条件，Motion 会在每刻额外乘以该方块的滑度值。

只有着地（onGround 为 true）的船、掉落物、经验球和着地且不在流体中的 LivingEntity 会受到滑度影响。着地的 TNT 和矿车也有类似于受到固定（分别为 0.7 和 0.5）滑度作用的行为，但实际上与此并没有任何关系，详见第七节。

目前原版滑度只有四个不同值，0.6、0.98、0.989 和 0.8，分别归属于普通方块、冰类、蓝冰和粘液块。

对于大部分 LivingEntity，滑度总是以其坐标下方略多于 0.5 格对应的方块坐标处的方块为准；掉落物、经验球、恶魂和幻翼是以其坐标下方 1 格对应的方块坐标处的方块为准；船则是以直接支撑它的所有方块的滑度的

平均数为准，且其在地面上时水平轴上阻力全部取决于此。图中所有实体得到的滑度都为 0.989（蓝冰）。

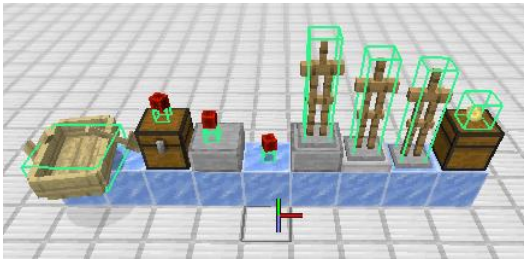

图 5.4 滑度示意图

对于组成有周期性的地面，可以用一个平均滑度近似地计算加速度一定时的最终速度和无加速度时的最终位移等数据。平均滑度可能是每周期参与滑度计算的所有方块滑度的几何平均数。不过，照窃梦者的研究[5]，船的平均滑度应该是每周期参与滑度计算的所有方块滑度的算术平均数，目前不好确定哪个更接近实际。

另外，滑度相关的运算不是在 move()方法中进行的。对于 LivingEntity，其运算位于 travel()方法中，移动执行前；对于掉落物和经验球，其运算位于 tick()中，移动执行后；对于船，其运算位于 method_7458()中，最终在 tick()中移动前被调用。这可能造成取到滑度的位置与实际作用滑度的位置不同，需要了解。

5.5 气泡柱的变速作用

再检查方块网格碰撞时，如果一个除烟花火箭外的可通过指定 Motion 标签自主移动的实体微量缩小后的碰撞箱有一部分在气泡柱所在的方块网格中，该实体就会被气泡柱改变 Motion，或者说被推动。

在气泡柱上方为空气方块时，每个向上的气泡柱会把实体的 y 轴 Motion 设为 1.8 和处理前 Motion+0.1 中较小值，向下的气泡柱会把实体的 y 轴 Motion 设为-0.9 和处理前 Motion-0.03 中较大值。在气泡柱上方不为空气方块时，每个向上的气泡柱会把实体的 y 轴 Motion 设为 0.7 和处理前 Motion+0.06 中较小值，向下的气泡柱会把实体的 y 轴 Motion 设为-0.3 和处理前 Motion-0.03 中较大值。也就是说，气泡柱中的实体在一定的 Y 轴 Motion 范围内受到的加速度是恒定的，否则速度会被限制到一个最值。

在气泡柱上方为空气方块时，其上方的船只有在连续与气泡柱接触 60 刻（无卡顿的 3 秒）后才会进行真正的变速，且处理上方直接是空气方块的气泡柱的机制略有不同，详见 7.4。

这种变速有累加作用，不涉及反向气泡柱的情况下，参与的气泡柱越多，变速越快。具体些，每个气泡柱方块都会把第二段中提到的过程重复一遍，顺序可以参考 4.1 节中方块网格碰撞相关说明。

5.6 其它特殊方块变速举例

在基于 Entity.move()方法的移动过程的第 11 步中，如果实体未潜行且已经着地，实体的着陆方块指定的 onSteppedOn()方法会被调用。粘液块是唯一使用这种方式直接影响实体的运动的方块，在实体经过其上方（即调用其 onSteppedOn()方法）且实体的 Y 轴 Motion 绝对值小于 0.1 时，它会将该实体的 X、Z 轴上 Motion 等比变为原来 0.4+0.2|M_Y|倍，其中 M_Y 为实体的 Y 轴 Motion 以 m/gt 为单位的数值。减速后与减速前速度的比值应在区间[0.4,0.42)之间，即实体受到一个以 gt^{-1} 为单位的阻力系数在区间(0.58,0.6]的地面阻力。这一范围已经算是较小的了，加上实体的 y 轴 Motion 趋于稳定或以较短（2-4gt）周期变化，最终的速度一般而言是接近稳定的。

检查方块网格碰撞时，若实体贴在蜂蜜块的侧面但不位于内部，未着地且拥有大小大于 0.08m/gt 的向下 Motion，实体的 Y 轴 Motion 会被锁定为-0.05m/gt。如果满足上述条件且处理前实体的 Y 轴 Motion 小于-0.13m/gt，实体的水平方向的 Motion 也会被减少到原来的$-\frac{0.05}{Motion_{Y,\ 处理前}}$倍。这一过程可以叠加，即每一个被检查的蜜块都亏重复上述操作。过程中实体的 FallenDistance 会被归零。

5.7 流体的变速作用

这种变速是在方法 Entity. updateWaterState()方法中发生的。除水生生物、创造飞行中的玩家、部分不进行实体基础运算的实体和直接骑乘船的实体外，几乎所有实体都会进行流体相关的变速运算。

首先，游戏会检查实体碰撞箱向内缩小 0.001m 后得到的区域中所有区块是否已经被加载（确切来说，至少为边界加载），如果有任一区块不符合要求，整个过程会被终止。

然后，游戏会取实体碰撞箱向内缩小 0.001m 后得到的区域中所有方块网格中的**流体流向向量**并对它们做加权平均。流程大致如下：

1 取 h=0，向量 **v**=**0**；

2 依次检查范围内的流体，东边晚于西边，南边晚于北边，上边晚于下边。

2.1 取实体浸入该处流体的深度 d，即流体顶部高度减去实体 Y 轴坐标

2.2 若 d≥0,认为该处流体有效

2.3 将 h 设为 d 与原有 h 中的较大值

2.4 在流体有效的前提下，若 h<0.4，将该处流体流速向量与 h 的积加到 **v** 中，否则将该处流体流速向量与 h 的积直接加到 **v** 中

3 将 **v** 乘以有效流体的个数的倒数

4 如果影响的实体不是玩家实体，将 **v** 缩放使其大小为 1

一个流体的流向向量由其附近的同类流体的与该流体的高度差及该流体的 falling 状态决定，某水平方向上高度差越大，该方向上分量就越大，模恒为 1，除非不流动（四周没有同类流体或四周同类流体等级相等）。如果一个流体有为 true 的 falling 状态，它的流向向量向下的分量应远大于（大约 8 倍）水平方向的分量。获取一个流体流速向量的具体代码如下在 FlowableFluid.getVelocity()方法中。实际上，平面的情况下最简单的确定一格流体的流向向量的方法就是看这格流体纹理的流动动画，因为在不涉及向下的水流时流体的流动动画与其流向向量是完全同向的。另外附录中提到的 MCWMEM 也提供了流体流向向量的显示。

最后，游戏会将平均向量乘以给定的合加速度大小加到 Motion 上。这一合加速度大小在水中为 0.014m/gt²，在下界的熔岩中为 0.007m/gt²，在其它维度的熔岩中约为 0.002333333m/gt²。

5.8 活塞的推动和变速作用

除旁观者、所有没有碰撞箱的实体、区域效果云和带有 Marker 标签的盔甲架外，所有实体均能被活塞的推动影响，你甚至可以推动幻魔者召唤的尖牙，如果足够及时你甚至可以把它通过传送门推到另一维度。

在活塞伸出或缩回时，它会为对每一个被移动的方块（包括活塞头）创建一个 id 为 moving_piston 的方块（又称 36 号方块、B36）和对应的方块实体。B36 不保留被推动的方块的滑度等属性，大部分属性与石头类似。

在每一个方块实体的运算过程中（TE 阶段，在实体运算后，可以参考 Fallen_Breath 的 MC 运算流程图[2]），该方块实体会尝试对一定范围内的实体进行一系列移动及变速操作。可以总结出如下的流程：

(1) 第 1gt 的 BE（方块事件处理）阶段，活塞开始移动，被移动的方块对应的 B36 被创建，此时其 progress（进度）属性值为 0.0,碰撞箱范围与原碰撞箱一致。

(2) 第 1gt 的 TE（方块实体运算）阶段，各 B36 方块实体被运算，开始运算时 progress 值为 0.0，碰撞箱范围向移动方向移动了 0.0m。该方块实体的运算即将结束时，碰撞箱会向前移动 0.5m。

(3) 第 2gt 的 TE 阶段，各 B36 方块实体被运算，开始运算时 progress 值为 0.5，碰撞箱范围向移动方向移动了 0.5m。

(4) 第 3gt 的 TE 阶段，各 B36 方块实体被运算，但未影响实体，运算后方块到位，各 B36 本身被移除。该方块实体的运算即将结束时，碰撞箱会向前继续移动 0.5m，到达最终位置。

任何有效 moving_piston 方块实体在每 gt 都会尝试将与其目前的碰撞箱在运算结束时即将新新扫过范围相交（不计边缘）的实体沿移动方向移动至距其目前的碰撞箱前方 0.01m 处。0.01m 的额外位移与被移动的方块数量和推动的刻数无关。活塞在直接移动实体过程中实体的 Motion 不变。在这里每个 B36 产生的最大位移不超过 0.51m。如果实体卡在了一个形状比较奇葩的方块中（如楼梯和炼药锅），该方块的 B36 对该实体的移动过程将非常复杂，大概是需要把碰撞箱拆成几个立方体（不是逐像素），并把每一个立方体单独作为一个方块计算位移大小。这只涉及到极少一部分问题，此处暂时从略。

粘液块的 moving_piston 方块实体会将与其目前的碰撞箱范围或下一 gt 碰撞箱范围相交（不计边缘）的实体沿移动方向的轴上的 Motion **直接设为**大小为 1m/gt，与移动方向同向的一个值，其它轴不变。这意味着如果一个实体某轴上速度大于 1m/gt，同向弹射甚至会使其减速。虽然 TNT、经验球和下落的方块在自己的运算中落到地面不会被弹起，但是被活塞推动的粘液块仍可以给它们加速，这是由于粘液块弹射是不排除这些实体的。实际上，给末影水晶等根本无法自主移动的实体增加 Motion 甚至也是可以的，只是没什么用处罢了。这一类利用“实体撞方块”和“方块撞实体”的区别的情况在活塞推动中的应用是比较普遍的，最常见的两个例子就是盾构机和珍珠矫正。

收回的活塞头的 moving_piston 方块实体在每 gt 会尝试将即将被卡到活塞底座与活塞头中间的缝隙中且受到了该 B36 的直接推动或位于活塞底座内部且受到了该 B36 的直接推动的实体沿与活塞收回方向相反的方向移动到距活塞方块网格 0.02m 处。在每 gt 中的推动进行后，实体会被额外多推动 0.02m，所以在收回过程结束后，实体碰撞箱边缘正常情况下应距离活塞 0.02m。这一过程中实体的 Motion 不变。在这里每个 B36 产生的最大位移不超过 0.51m。

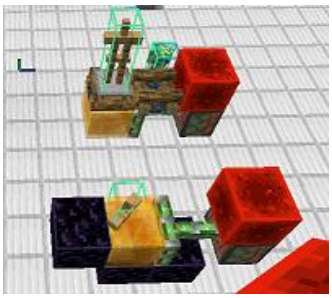

图 5.8.1 蜂蜜块粘动示意图

水平移动的蜂蜜块的 moving_piston 方块实体在水平移动时会将其上方的着地的实体沿移动方向移动。准确些，在蜜块上面放块半砖，在蜜块顶部与一个比半砖顶部略高的位置之间我们取一个长方体，则所有与该长方体范围相交（不计边界）且坐标点在某一水平面上的正投影在该长方体在同一水平面上的正投影内部或边界上的实体如果着地就会被移动。或者说，如图 5.8.2，如果实体碰撞箱与紫色区域相交且坐标点位于绿色区域内部，实体才可能会被影响。这意味着，除直接站在被移动的蜜块上的实体外，站在蜜块内部且在顶部露出了一部分碰撞箱的实体和站在蜜块上方的不高于 0.5m 的方块上的实体也会被移动（如图 5.8.1 中的两个盔甲架在活塞收回时都会被移动）。每个实体在该方块实体被运算时会被推动 0.5m，注意没有前两种中的额外的 0.01m 的推动。这一过程中实体的 Motion 不变。

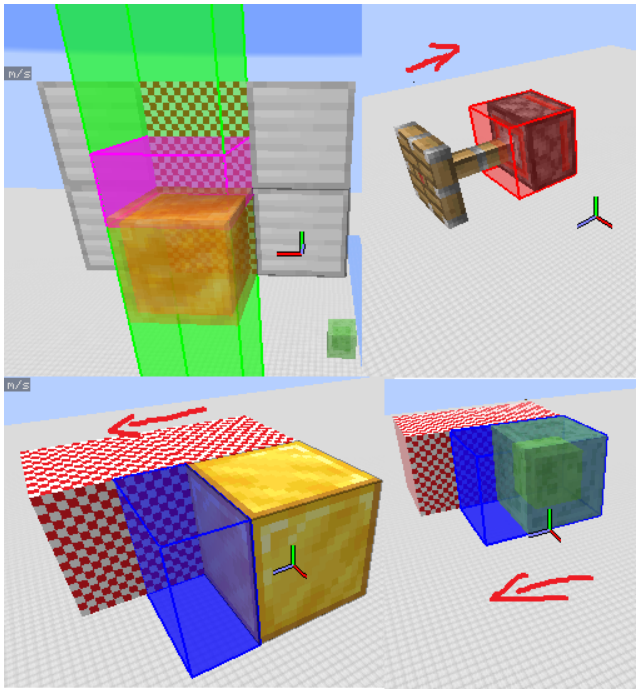

图 5.8.2 B36 的影响范围示意图

单个 B36 的运算中，上述四种影响的产生顺序为：粘液块弹射、一般推动、活塞头挤压和蜜块粘动。

在分析活塞推动过程时，一般不能只找准一个 B36 去分析（还真被这个坑过），还得分析临近的 B36 的影响，这就牵扯出来一个方块实体运算顺序的问题。其实吧，这个在不卸载区块时和活塞更新顺序完全相同。

因为活塞直接造成（不包括粘液块的弹射）的所有移动都是基于 Entity.move()方法的移动，所以这些移动都有能被蛛网削减，进行的是对碰撞箱的碰撞检查，在水平轴上发生碰撞时发生碰撞的轴上速度会被置零等基于 Entity.move()的移动的共性。

活塞造成的总位移会在 move()方法中受到限制，所以无论如何，每个坐标轴上每 gt 由活塞推动造成的相互抵消后的位移绝对值不会大于 0.51m。在 1.11 及之前的版本中是没有这一机制的，意味着如果被活塞推动的实体再某次推动后如果被推到同一 gt 中下一个被运算的 B36 的推动范围内，这个实体在 1gt 内就会被移动两次。如此循环，可以构造一个推动实体的活塞长链，理论上可以将实体在 1gt 内移动到无限远的地方，但是因为一个与实体管理相关的 Bug 实际上每 gt 最多只能移动 16m。[6]在 1.12 之后，由于这一机制的加入，这种推动装置失效。

在某些旧版本中，尤其是 1.9.4 之前，活塞的推动机制存在一些很严重的 Bug，如碰撞箱出现部分缺失、合并以及等栅栏断开连接等[6]。在 1.15 及以前版本，活塞在收回的最后一 gt 还会在活塞底座后面露出来一段（4px，0.25m 长）活塞臂，导致后面的实体被那一部分推动（不过我认为这不一定算是一个 Bug，说实体是被震动的也不是不行）。目前，虽然大部分 Bug 已经修复，现在 B36 的碰撞箱的有些行为还是比较奇怪的：

伸出的活塞头 B36 的碰撞箱会在它后面自带一个本来应该有的无头活塞的碰撞箱，无论这个活塞底座是否真正存在。乍一想非常奇怪，但这是很正常的，因为这时原来的活塞底座已经被 B36 覆盖，没有这个碰撞箱倒会出现很多 Bug。这个多出的碰撞箱不影响推动位移的计算。

在实体被活塞直接移动期间，如果 B36 的移动方向和实体移动方向一致且 progress 不为 1（第 2 次运算完毕前），B36 的碰撞箱会被忽略，除非它是伸出的活塞头的 B36，这时它自带的无头活塞碰撞箱会被保留，活塞头本身对应碰撞箱会被忽略。如图 5.8.3，放置两排水平的粘液块，间隔两格，并在下方一列粘液块上方生成一只远古守卫者（高度为 1.9975m），最后同时激活两个活塞，守卫者会穿过下方一列粘液块被下弹。过程中，远古守卫

者被上方粘液块在直接向下推动时没有检测同为下推的下方粘液块 B36 的碰撞，从而被推到下方一列粘液块 B36 碰撞箱内部，进而在以后的移动中不再对其进行碰撞检查并直接穿过它们。

给定一个 B36 影响范围，与影响范围扩大两格后的范围相交的区段内的实体才会被影响，原因在 2.5 节中已经进行了说明。如图 5.8.3 中的凋零骷髅，如果将它放在 y=31 处的一个箱子正上方，则在 y=34 处的活塞将不能正确地推动凋零骷髅，尽管推动过程中凋零骷髅的碰撞箱会与 B36 的碰撞箱相交。

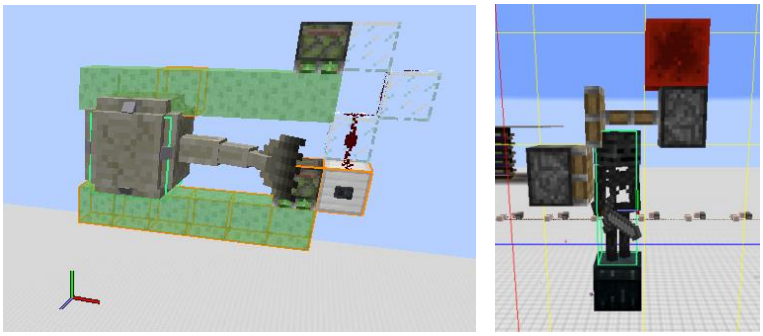

图 5.8.3 活塞推动的两个特性演示

最后补充一下，0tick 瞬推不能移动或弹射实体，因为活塞对实体的影响依赖于 B36 方块实体的运算，但是 B36 方块实体在脉冲撤销时会被提前移除。但是，瞬推可能也伴随着收回过程，这时 B36 是可能对实体造成影响的。例如，虽然 0tick 活塞不能推动实体，但在一个推动和收回周期后实体会因为活塞头的收回移至活塞头前方 0.02m 处。如果活塞收到的脉冲持续时间为 1gt，实体可以被粘液块弹射，但移动不会完整地进行。

5.9 潜影盒和潜影贝的推动作用

被潜影盒和潜影贝推动与被被活塞推动的石头推动非常相似，主要的差别有：

(1) 到位比活塞慢得多（活塞为 2gt，而潜影盒潜影贝为 10gt）

(2) 到位后“盒盖”的位移为 0.5m 而不是 1m

(3) 原碰撞箱内部的实体总是会被移动

(4) 在一刻内推动某个实体的位移不受限

另外潜影贝的推动会忽略其他潜影贝和 noClip 为 true 的实体。

5.10 实体挤压（推动）

在你尝试穿过一群动物时，虽然不会发生那种撞到方块时发生的实打实的碰撞，但穿过的过程却还是比较困难；当你想让一个村民坐到船上时，你通常会选择把它挤过去：这就是**实体挤压**。

对于 LivingEntity，实体挤压发生在运算接近末尾处，travel()方法执行后；对于矿车和船，实体挤压也是发生在运算接近末尾处。

在服务端一个 LivingEntity 运算过程中与另一个 LivingEntity 发生实体挤压需要满足以下条件：

(1) 两方碰撞箱相交（不含边界，且要考虑 2.5 节中提到的 Bug）
(2) 两方都不为盔甲架、蝙蝠或旁观者玩家
(3) 两方的 noClip 都不为 true
(4) 两方没有直接或间接地骑乘同一实体
(5) 被推动的一方未睡眠
(6) 被推动一方都没有死亡
(7) 被推动一方坐标所在方块坐标处方块都不为可攀登方块（见 6.1）
(8) 队伍设置允许碰撞（原版生存可忽略）

其中，前 4 条和第 8 条如未被满足，实体挤压不会发生，其余条件若不满足，实体挤压仍然可能发生，但挤压只能在双方中的一方的运算中进行，变速幅度相对较小。

对于 Living 和船，在满足条件后，游戏会取得两个实体在 X、Z 轴上坐标差的绝对值中较大值作为一个另类的“距离”，以 m 为单位时数值记作 d，实体坐标间距离为 r，那么这次挤压对双方造成的 Motion 改变量以 m/gt 为单位的数值 axz 满足：

$$a_{xz}=\begin{cases}0, d<0.01\\ \dfrac{0.05r}{\sqrt{d}}, 0.01\le d\le 1\\ \dfrac{0.05r}{d}, d>1\end{cases} \tag{5.10.1}$$

也就是说，在两个实体间距离过近（没有一轴大于 0.01m）时这两个实体不相互挤压，然后在 0.01≤d≤1 时，加速度随距离递增，最大能达到 $0.05\sqrt{2}\ \mathrm{m/gt^2}$，再然后 d>1 时加速度在实体坐标间连线方向不变时不变，具体大小与连线方向有关，沿轴方向时最小，对角线方向最大。不过由于要保证两碰撞箱相交，d 永远不会大于两实体碰撞箱的平均半径。

实体挤压造成的加速度的方向只与实体坐标间连线的水平投影方向有关，或者说只与实体间的相对位置有关，总是沿该连线水平投影向两端加速，与实体速度，朝向等因素是无直接关系的。实体挤压不影响 Y 轴 Motion，也就是说从实体上面踩下去不会受到实体挤压的阻挡。

Entity 类中定义了 pushSpeedReduction 字段，大概是用于表示实体在对实体挤压造成的变速的抗性大小，或者说被削减加速与总加速度的比值，目前似乎未被使用。

如果实体被骑乘，那么该实体不会在实体挤压过程中变速，除非该实体为矿车。

这一加速度在堆积的实体足够多的情况下大小可与爆炸加速匹敌（理论上接近无限），可以使生物瞬间获得极大的速度，这足以使玩家跳跃到数百米外[7]。

矿车的推动与 LivingEntity 略有不同，实体坐标间距离为 r，那么这次推动对矿车一方造成的加速度以 m/gt² 为单位的数值 axz 满足：

$$a_{xz}=\begin{cases}0, r<0.01\\ 0.05, 0.01\le r\le 1\\ \dfrac{0.05}{r}, r>1\end{cases} \tag{5.10.2}$$

也就是说，矿车挤压造成的加速度大小与方向无关。另外，这一挤压过程中非矿车实体的变速大小为矿车的 1/4。

船的挤压与 LivingEntity 相似，但是因为它有那种实打实的碰撞，挤压一般只能发生 1-2gt，而且在普通方块上时阻力较大，所以变速很小。

大量实体（500 个以上）在挤压时会有一部分实体朝几个方向被发射出去，并在几条直线上排列，排列所在直线似乎与实体的创建顺序有关，而且距离随实体 ID 似乎成负相关，目前还个人没有较好的解释。

5.11 骑乘

骑乘一般来说是一个实体附着在另一实体上的状态。一个实体只能骑乘一个实体，但一般可以被多个实体骑乘。需要了解，鹦鹉附着在玩家肩膀上并不属于严格意义上的骑乘，因为此时它根本就不是一个实体，它只保留了一个 NBT 标签而不是 Entity 实例。

在骑乘发生的每一 gt 中，骑乘的实体运算前，其 Motion 会被归零；骑乘的实体运算后它会被 TP 到该实体与被骑乘实体共同指定的一个位置。该位置在默认为比被骑乘实体的坐标高被骑乘实体高度的 0.75 倍的一个点，

不同实体组合会有不同。船不仅会修改骑乘者的坐标，方向角也会随之被修改。

LivingEntity 在骑乘到其它实体时，每 gt 运算完成后还会归零下落高度，所以落地船（其它实体仍会为骑乘者运算摔伤）之类比较另类的减伤方案还是可行的。

猪和赤足兽被骑乘且骑乘者在控制它时还会有一个特殊且死板的 AI，这时它会以一定加速度移向骑乘者的视线方向。

实体存在一个 60gt 的骑乘冷却，意思是实体在从某个实体上下来后 60gt 内不能骑乘其他实体。另外，潜行中的实体无法骑乘其他实体。

实体被骑乘时碰撞箱大小一般不变，也就是说骑乘不影响底层实体的碰撞检查，但骑乘在某实体上面的实体因为是靠 TP 移动的所以不进行碰撞检查，可以卡到方块中。

5.12 鱼竿浮标的拉动

鱼竿浮标是一类弹射物，在它击中符合条件的实体（详见 4.2）时会勾住该实体。鱼竿浮标在拉回时对被拉动实体的加速度大小在以 m/gt^2 为单位时数值上等于以 m 为单位时被拉动实体与鱼竿使用者坐标间距离的十分之一相同，方向在被拉动实体与鱼竿使用者坐标的连线上，朝向使用者，只作用 1gt。这差不多就是说，如果一个 LivingEntity 在空中被鱼竿拉动，不计 AI 它差不多最终就移动到鱼竿使用者下方。

5.13 击退

在玩家和生物攻击一个非旁观者玩家或生物或玩家使用带有击退附魔的物品攻击任何可被有效攻击的实体时被攻击的实体的 Motion 会发生改变，也就是被击退。

在没有击退附魔时，被击退的 LivingEntity 的 Motion 的 XZ 轴上分量会被减半并被加上一个大小为 $0.4m/gt^2$(1-generic_knockback_resistence)，方向在实体坐标间水平距离小于 0.01m 时是随机的，否则是沿坐标连线的水平投影的加速度。在被攻击实体着地时 Y 轴 Motion 会改为当时实体 Y 轴的 Motion 的一半与 0.04m/gt 的和，但会被限制在 0.04m/gt 及以下。很多实体的 generic_knockback_resistence（击退抗性）属性值为 1，说明实体完全不会被击退，详情可以参考 wiki。不过，还有一部分 LivingEntity（如潜影贝、盔甲架和末影龙）因为各种原因在这种情况下无法被击退。被有横扫之刃附魔的玩家击退的 LivingEntity 的行为与此类似。

在拥有击退附魔的玩家攻击 LivingEntity 时，实体会在受到上一段描述的初步击退的基础上将水平轴上 Motion 减半，并加上一个沿玩家视线的水平投影方向向前（竖直时与保持偏航角不变平视时相同），大小为附魔等级乘以 0.5m/gt 的向量，并受到与上面类似的击退抗性影响。在拥有击退附魔的玩家攻击其它实体时，实体会将水平轴上 Motion 加上一个沿玩家视线的水平投影方向向前（竖直时与保持偏航角不变平视时相同），大小为附魔等级乘以 0.5m/gt 的向量。此时如果玩家在疾跑，那么疾跑会终止。

不仅是直接攻击，间接攻击（射出箭矢、雪球、点燃 TNT 和苦力怕爆炸等）也会造成击退。举个例子，一个 LivingEntity 点燃了一个 TNT，那么即使这个 TNT 产生的爆炸被方块完全阻挡，接触率为零（此时如果 TNT 是红石点燃的则不会对玩家造成击退），只要玩家还处于爆炸范围内，玩家就会被击退，而且方向与相对坐标有关。这一特性已经在一个视频中被介绍[8]，可以用于在无政府服务器寻找玩家的基地。

5.14 方块对某些实体的推出作用

当物品、经验球或玩家的一部分碰撞箱位于方块内部（不含边界）时，方块会将它们从内部推出。这虽然也是那些实体运算的一部分，但我觉得还是把它们放到这里更好。

对于物品和经验球，首先，游戏会选择一个距离卡住实体的方块所在方块网格（也就是实体碰撞箱中心所在方块网格）的表面最近的一个沿轴方向，如果都相等则为上方。然后，游戏会检查该方块网格的那个那个方向上紧邻的方块网格中是不是完整方块，如果不是，就选取这个方向作为推动方向，否则选取上方作为推动方向。再然后，实体的各轴 Motion 会被削减为原来的0.75倍。最后，实体沿轴方向的速度会改为一个大小在0.1～0.3m/gt之间，方向沿选定的方向的一个值。

物品实体卡在方块中时在移动过程中不进行碰撞检查，所以无论障碍有多厚都可以被一直推动直到被彻底推出或 despawn。如果试着把一个物品实体卡在大箱子中间，这个实体会不停的在这两个箱子间移动，这与 MC-4 的成因不同，但也不排除是一个 Bug。经验球实体由于在在方块中时推出过程中还会进行碰撞检查，所以只能被推出单个方块。

玩家的与此类似，但完整方块的判断被改为方块是否能造成窒息的判断，而且永远不会被向上或向下推动，推动速度恒为 0.1m/gt，而且没有那个各轴 Motion 会被削减为原来的 0.75 倍的操作。

5.15 爆炸的变速作用

除旁观者和有 Invisible 标签的盔甲架外，这种变速（改变 Motion）作用几乎对任何实体都有效，甚至对区域效果云都有效。但有一些实体的 Motion 不控制实体的运动，爆炸并不能使它们运动起来。

需要引入几个概念，部分名称来自 Minecraft 中文 Wiki：

爆炸中心：一个表示爆炸位置的点。由 TNT 产生的爆炸的爆炸中心经实验验证是在坐标上方 0.06125m 处（准确点，0.061250001192093m）；由凋零产生的爆炸中心位于其眼部坐标；由末影水晶、苦力怕和 TNT 矿车产生的爆炸中心位于其坐标；由床和重生锚产生的爆炸中心位于玩家右键的那个方块所在方块网格的中心；由火球类弹射物产生的爆炸中心位于其碰撞前最后一刻的坐标。

爆炸威力：由 Explosion.power 存储，是一个单精度浮点数，决定了爆炸的范围，对方块的破坏能力以及对实体的某些影响（如伤害值，但不影响加速度上限）。一次爆炸只有一个强度。Java 版常见的爆炸强度有 TNT 的 4 级，末影水晶和闪电苦力怕的 6 级，凋零生成时的 7 级，苦力怕的 3 级，恶魂火球和凋零之首的 1 级。TNT 矿车的爆炸威力是动态决定的，与速度或摔落高度有关，且有一定的随机性，最低为 4 级，最高可达到 11.5 级，是生存模式中可获得的最大爆炸威力。

爆炸伤害半径（简称**爆炸半径**）：爆炸能影响到的实体的距离上界，坐标与该爆炸间距离大于该值的实体不会受到该爆炸的任何直接影响，单位为 m 时其数值为爆炸威力的两倍。爆炸范围是一个以爆炸中心为中心，半径等于爆炸半径的球。

爆炸接触率：近似地描述一次爆炸中实体实际未被方块阻挡的朝向爆炸中心的体积与无方块阻挡时本应直接朝向爆炸中心的体积之比，在每次爆炸中方块被破坏前被计算。该值可由 Explosion.getExposure()方法得到，原理为在目标实体碰撞箱中按一定规律选取几个**取样点**，利用 raycast 检查该点与爆炸中心间连线是否被方块碰撞箱（现在是按完全阻挡运动的那种，蛛网和流体一类不算，1.16 之前的版本依据 Outline Shape，即鼠标对准时显示的黑框）阻挡，最后将未被阻挡的点数与总点数的比值作为接触率。正常情况下，在爆炸中心与实体间没有任何有碰撞箱的方块时该值为最大值 1；爆炸中心与实体间完全被方块碰撞箱阻挡或实体没有碰撞箱（碰撞箱大小为 0）时为 0。图 5.15.1 为 TNT 实体的典型选点情况，图中存在 27 个取样点：

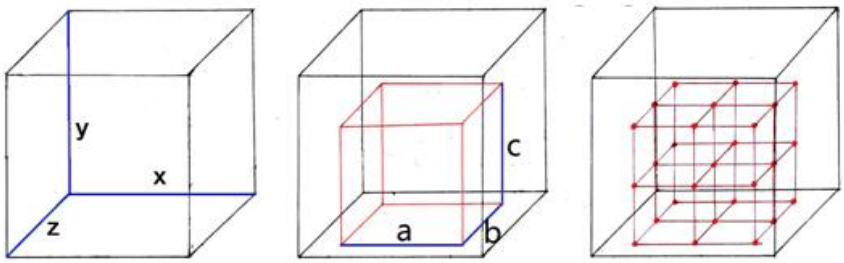

图 5.15.1 取样点示意图

具体来说，正常情况下计算接触率的取样点可以由如下过程得出：

对于一个平行 X、Y、Z 轴的棱长分别为 x、y、z 的实体碰撞箱，取一个平行 X、Y、Z 轴的棱长分别为$\frac{x}{2x+1}$、$\frac{y}{2y+1}$、$\frac{z}{2z+1}$的长方体 A，用尽可能多个仅通过平移就可以与长方体 A 完全重合的“较小（实际上也可能与下面的长方体 B 等大）”长方体紧凑地构成一个下底面中心与实体碰撞箱下底面中心重合的“较大”长方体 B，使得任意两个构成 B 的较小长方体仅在表面拥有公共点或没有公共点且不超出实体碰撞箱，则每个用于构成 B 的“较小”长方体的顶点所在位置有且仅有一个取样点。

可能有些难懂，不过也没关系，这个并不大适合理论计算，实验往往更实用。一方面，人工计算这个确实复杂，复杂些差不多得算上半个小时才出结果，附录提到的 MCWMEM 中也有可视化检测线和监测爆炸接触率的功能。另一方面，大家可能也发现这里已经两次提及“正常情况”了，因为这里的特殊情况（Bug 或者说特性）真的很常见，或者说基本上都是特殊情况。

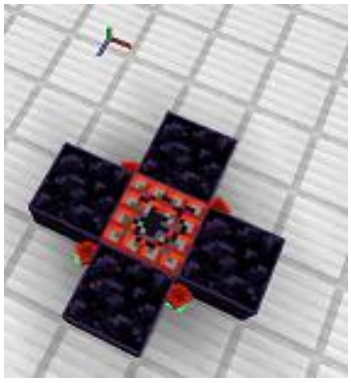

图 5.15.2TNT 爆炸对掉落物影响实验

根据的 DawNemo 的研究[9]，有以下三条：

1. 宽度在区间(0,0.5m)的实体的所有取样点会向东南方等量偏移，都有 3/4 的取样点被移出了碰撞箱。图 5.15.2 几个红石块物品都挤在两个黑曜石间的角落里，在点燃图中的 TNT 后，西北方的红石块会被炸死，因为其

部分取样点（全部的 1/4）向东南发生了偏移移出了碰撞箱并移到了 TNT 所在的方块未被黑曜石阻挡，从而得到了不为零的接触率，进而得到了较高伤害。

2. 宽度在区间(0.5,1m)的实体的所有取样点会向东南方等量偏移，但不可以移出碰撞箱。尽管没有上一条那么 buggy,这对一些装置还是有影响的。例如，TNT 的取样点会向东南偏移约 0.006487m，这会导致某些 TNT 炮的炮弹向西北发射（此时可能有更多取样点接近爆炸中心）时平均射程最大；但是如果 TNT 完全不被方块阻挡，或阻挡地没有那么恰到好处，这一影响就没有了。
3. 宽度在区间(1m,+∞)且不为 0.5 的倍数的实体的所有取样点会向西北方等量偏移，但不可以移出碰撞箱。以 m 为单位的宽度位于区间[0.5x,0.5x+0.5)中的实体中宽度越接近 0.5x+0.5 偏移量越大。

也就是说，宽度不为 0.5m 的整倍的实体的取样点普遍存在偏移，且小于 1m 时向东南（坐标轴正向）偏移，否则向西北（坐标轴负向）偏移。

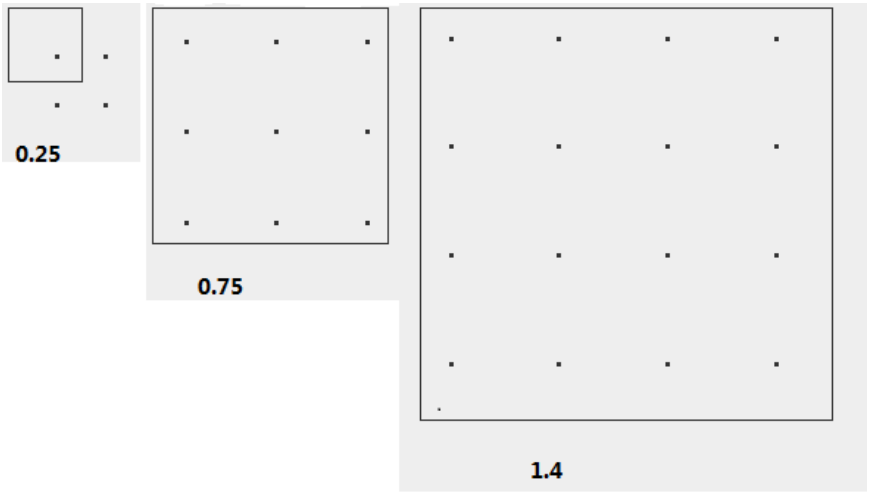

图 5.15.3 碰撞箱俯视图（示取样点）

图 5.15.3 中为使用可视化工具生成的一些例子，图中给出的是碰撞箱的俯视图，数字是以 m 为单位的宽度，方向遵循“上北下南，左西右东”。

实际上，偏移量能用一个这样的公式计算：

$$\Delta=\frac{1}{2}(1-x)\left(1-\frac{\lfloor 2x+1\rfloor}{2x+1}\right) \tag{5.15.1}$$

个人猜测因为浮点数误差和取整操作，宽度为 0.5m 的整倍且为 2 的整数次幂的实体个人猜测可能会在坐标接近 2 的整数次幂的地方发生取样点的向东或向南的偏移；宽度为 0.5 的整倍且不为 2 的整数次幂的实体可能会

在任何位置发生取样点的向东或向南的偏移。目前仅由理论得出，尚无实验证据。

爆炸影响力：用于描述一次爆炸对某个实体的影响大小，由以下公式得出：

$$爆炸影响力 = \frac{实体坐标与爆炸中心间距离}{爆炸半径} \cdot 爆炸接触率 \tag{5.15.2}$$

然后，在每次爆炸中，方块被破坏前，坐标在爆炸范围内且眼部坐标（TNT 是个特例，它是按实体坐标）不与爆炸中心重合的实体的 Motion 会沿爆炸中心到实体眼部坐标的连线（TNT 是个特例，它是沿爆炸中心与其坐标连线）向远离爆炸中心的方向增加爆炸影响力的数值（单位为每/gt）。（Explosion.java：186-220）例如，图中的珍珠的 Motion 变化量大小与线段 B 的长度有关，而加速方向与直线 A 共线，但这一不一致性是否是是有意而为之的不很确定。

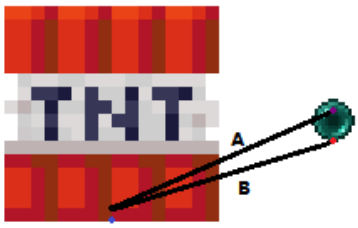

图 5.15.4 爆炸加速的大小与方向确定方式示意图

堆叠多个爆炸可以对附近的实体造成极大的加速度（可以大于 1000m/gt²，理论上接近无限）。但需要注意，在堆叠的 TNT/TNT 矿车的爆炸中，由于每一个 TNT 的爆炸都会进行关于对其它每个 TNT 造成的影响的运算，所以**堆叠的 TNT/TNT 矿车的爆炸造成的卡顿时间与堆叠数量的平方大概成正比**，这意味着大量堆叠 TNT/TNT 矿车进行推进可能会造成极大的瞬时卡顿。但是 JE1.12 及以后**堆叠的末影水晶在引爆时爆炸会被合并**（实际上是末影水晶被爆炸破坏时本身不会产生爆炸，直接移除），整体上只有一个爆炸，基岩版和 JE1.10 及之前似乎没有这一设定。

另外尽管也有声音和部分效果，烟花火箭和末影龙火球的爆炸在代码层面并不属于一般的爆炸（不由 Explosion 类实现），而且本身不会造成实体变速，此处不再论述。（不过话说回来烟花火箭爆炸的实现跟一般的爆炸的实现还挺像。）

5.16 区块行为对运动的影响

在实体离开强加载区块后其运算会暂停，此时实体的速度等属性不再主动改变，移动也不再主动发生。如果在一段时间后重新加载那个区块，实体会一般正常地恢复其运算。如果期间外部环境没有发生影响到该实体的改变（如本应顺利通过的路径上突然生成了一个方块），这和连续地运算的结果是相同的。

但是，如果实体所在区块被彻底卸载（也就是被保存了），实体的一些属性可能会丢失（如 movementMultiplier、跳跃的冷却时间和幻魔者尖牙的消失计时等），这通常会导致实体在区块重载后的行为与连续的运算出现差距。

与之类似的一个比较明显的问题是游戏会拒绝从磁盘或 NBT 读取绝对值大于 10 的 Motion 分量（你可以试一下用指令生成一个以 20m/gt 的速度运动的箭，这在 1.16.x 原版不会成功），这意味着如果一个实体在某个坐标轴上的 Motion 大于 10m/gt，区块重载后该轴上的 Motion 会被置零，这在大中型炮类中有时会是一个严重的问题。

闪电束和鱼竿浮标不能被保存，所以当区块被彻底卸载后这些实体会丢失。1.17 中这一特性可能已经被修复。

1.17 以后由于实体管理机制的调整，实体的存储被独立出来，某一区块即使没有被完全卸载（在内存中但不可访问），该区块中实体也会被卸载并丢失部分数据。

移动过快的实体（至少 100m/gt）可能会直接移出加载范围而不被区块卸载过程保存，这时实体会丢失。

无论是 raycast 还是 Entity.move()，移动过程都不要求路径上的所有区块都以某种形式被加载。对于珍珠炮而言，如果要使珍珠正常地飞往目的地，只加载路径点（坐标待过的点）所在区块即可。

据传在之前的一些版本中，raycast 不会检查路径上未被加载的区块中方块的碰撞，但是经测试，1.16.4 中这一特性似乎已经被修复，而且路径上所有区块都会被瞬间地加载到 33 级（边界加载）。在位移不绝对沿轴时，基于 Entity.move()方法的碰撞检查似乎仍不会检查未加载区块中的碰撞，也不会加载途中的区块。

5.17 TP/折跃门传送

严格意义上，TP 和折跃门传送并非实体的运动，这里提一下只是说明这类传送不会造成实体的速度及方向角改变，还有折跃门传送是在方块实体运算阶段进行的，竟有要求最底层被骑乘实体没有下界传送门门冷却这一奇葩要求。

5.18 跨维度运动

在服务端，除 TNT、下落的方块、末影龙、凋灵、鱼竿浮标、画、物品展示框、末影水晶、闪电束和拴绳结外，大部分与其它实体没有骑乘关系的实体都能通过下界传送门进入另一维度，除末影龙、凋灵、鱼竿浮标、画、物品展示框、闪电束和拴绳结外，大部分分与其它实体没有骑乘关系的实体都能通过末地传送门进入另一维度；但在客户端实体不会进行跨维度运动，实体的传送会在一段时间后（取决于 ping）以原维度中实体被移除，新维度中实体被创建的方式进行，不过这只与渲染和实体对玩家的挤压有影响，一般无需在意。

切换维度的过程中，除玩家实体外所有实体都会被重新创建，也就是说它们在区块保存时会丢失的属性在这里大多也会丢失，但不同的是实体大于 10m/gt 的 Motion 分量不会被归零。除此之外，虽然坐标可能被缩放了，但是 Motion 大小不会被缩放，所以可以利用传送门制作一类远射程大炮。

在实体运算时，游戏首先会选取一个用于传送该实体的传送门方块。对于除投掷物外的所有实体，这个方块是实体在最近一次检查方块网格碰撞的过程中检查的最后一个下界传送门方块，如果没有就选倒数第二次检查方块网格碰撞的过程中检查的最后一个下界传送门方块，以此类推。说简单点，该方块通常为在某 gt 中实体移动（一般为自身移动，如被活塞推动则为活塞推动）后与实体相交（与整个方块网格微量缩小后的区域，而不只是下界传送门本身）的 X、Y、Z 坐标最大的下界传送门方块。可以发现，在这些实体高速移动的过程中途中的下界传送门方块并不会被检查，就说比较容易获取的，箭矢直接穿过下界传送门而不被传送的概率超过一半，所以如果要制造跨维度的 TNT 大炮需要对准位置使炮弹在某一 gt 的碰撞箱与选定的传送门方块网格相交。投掷物在以上过程中没有找到时还会尝试进行一次 raycast 取寻找下界传送门方块，但这并不能起效，以后不排除修复。如果找到了用于传送的传送门，游戏会检查实体的传送冷却是否已经结束，如果未结束，实体永远不会被传送；如果已经结束，实体在 1gt（一般实体）或 80gt

（玩家）后会发生传送。如果期间有一刻不与传送门相交，传送延迟会增加4gt，直到达到最大值，即初次进入下界传送门时的传送延迟。发生传送时游戏会按照维度变化将坐标的X、Z分量缩放到原来的8倍（下界到主世界）或1/8（主世界到下界或下界到末地），并在其附近按一定机制寻找合适的位置传送实体，这在Wiki上已经有比较完整的论述了，在此不再多说。如果两维度下界传送门门方向不一致，实体的偏航角就会发生+90 度的偏转，水平两轴 Motion 会互换，有时还有符号的反转（实际上还是偏转+90 度，和偏航角偏转同步）。

在实体在执行 checkBlockCollision()方法（通俗点，一般是在基于Entity.move()的移动过程的第13步和一些实体的运算过程中）时，实体会寻找与其碰撞箱发生相交的方块网格中包含的末地门，如果找到了，实体会被立即传送到末地的100.5，50.0，0.5处，速度不变，朝向改为正西平视。玩家的高度可能比这个要低一格，不知道为什么。由于这一传送没有延迟，所以当实体同时进入末地门和下界传送门时总是会被末地门传送。

5.19 拴绳机制

在 Minecraft 中，大部分“牲畜类”生物都可以被拴绳拴住。当一个实体被拴绳拴住时，根据拴绳的长度，即生物坐标与玩家坐标或拴绳结所在方块网格中心的间距（也就是实体坐标与拴绳固定点间的距离），该实体会在其运算进行接近末尾时到受到不同的影响。

在拴绳长度不大于6m时，拴绳不会直接给实体任何加速度，这时实体会尝试利用AI走到拴绳固定点附近并停止AI移动。

在拴绳的长度大于6m但不大于10m时，拴绳会尝试拉动被拴住的实体。具体来说，设实体坐标与拴绳固定点坐标在X、Y、Z轴上的差值分别为x、y、z，r为拴绳长度，则拉动过程中实体的Motion改变量在各轴上分量绝对值以m/gt为单位的数值可表示为：

$$|a_x| = 0.4\frac{x^2}{r^2} \ \#(5.19.1)$$

$$|a_y| = 0.4\frac{y^2}{r^2} \ \#(5.19.2)$$

$$|a_z| = 0.4\frac{z^2}{r^2} \ \#(5.19.2)$$

拴绳给实体的加速度在各轴上分量朝向拴绳固定点在该轴上相对于实体坐标的方向。

对此简要分析后可知，这一加速度方向并不严格地从实体坐标指向拴绳固定点，实际上，加速度方向会更倾向于沿实体与拴绳固定点坐标差值较大的轴，只有当实体与拴绳固定点坐标连线沿轴或处于一个类似“角平分线”的位置时加速度方向才是从实体坐标指向拴绳固定点的方向。在加速度沿轴时各轴合加速度大小达到最大值 0.4m/gt，方向偏离坐标轴越多，大小越小。

此时实体的 AI 移动控制会被暂时禁用，但实体被拉回拴绳固定点 6m 内以后仍有可能进行 AI 加速。

当拴绳长度大于 10m 时，拴绳会断裂。

以使用拴绳数字悬挂生物为例，如果忽略实体的水平方向的运动，那么实体会在拴绳长度大于 6m 时在运算末尾处将 Y 轴 Motion 加上 0.4m/gt，否则保持自由落体，这时实体会不断地在一定位置附近振动，且拴绳不会断裂。在使用经典力学的理论分析中，这种情况下实体应该做一种阻尼振动，动能逐渐衰减，趋近于 0，时间足够长后实体将静止在使得拴绳长度等于 6m 处。但是由于 Minecraft 中的时间存在一个最小单位，即 gt，实体不会静止，即使实体满足了传统力学中的静止条件，实体仍会在运算中按照情况一次性受到所在的游戏刻内应受到的总的加速度而非连续地在每一瞬间进行累积。

但是，仍有方法让生物在拴绳拴住的情况下在空中完全静止，那就是让拴绳的加速度和重力加速度相平衡，此时实体与拴绳固定点的高度差约为水平距离的 0.4938 倍。

另外拴绳运算过程中 FallDistance 会正常运算，即摔落距离会被持续累积，这回造成在空中长时间振动的生物在落地时看似莫名其妙地摔死，需要当心。

6 LivingEntity 运动机制

LivingEntity 是实体中的一大类，包括了玩家、生物和盔甲架。虽说这里搞理论计算没有弹射物和 TNT 实用，但见外国跑酷理论已经有所发展了[10]，我们也不能落后对吧。本节内实体若无特殊说明均指 LivingEntity。视线水平投影方向如无特殊说明均指保持偏航角不变平视时的视线方向。

给出一些基础定义：

流体深度阀值：一个长度，在实体眼部高度不小于 0.4m 时为 0.4m，否则为 0。

AI 加速度：在 AI 加速过程中实体受到的加速度的一个决定量，与实体前/右向加速度系数决定实体的真正水平移动加速度，但**通常不等于真正的加速度**。

地面移动加速度：实体在地面运动时或有满级深海探索者且在水底地面运动时的 AI 加速度。

飞行移动加速度：实体在空中运动（不算鞘翅飞行）时的 AI 加速度。值一般固定为 0.02m/gt²，疾跑的玩家为 0.026m/gt²。

可攀登方块：minecraft:climbable 标签内的方块和下方有同侧梯子的活板门。

6.1 LivingEntity 的自由运动

与其它大部分实体不同，LivingEntity 的运动没有被直接放到 tick()方法中，其运动大多是在 travel()方法中进行的。

默认的 travel()方法可以分成四大部分：实体与水接触时，赤足兽以外的实体与熔岩接触但并不与水接触时，鞘翅飞行时和其它情况。如果实体有缓降状态，所有重力加速度变为给出数值的 1/4。为防止 Y 轴 Motion 落入$\pm 0.003\mathrm{m/gt}$之间而被归零，流体中的部分重力作用（包括第一种情况的第 9 步、第二种情况的第 3.2 步）中会有特殊处理，将受到重力后 Y 轴 Motion 在$\pm 0.003\mathrm{m/gt}$之间的实体的 Y 轴 Motion 直接设为-0.03m/gt。

在第一种情况中运算过程如下（LivingEntity.travel(),1936-1970）：

1 取 0.8（未疾跑时）或 0.9（疾跑时）作为 j_0

2 获取实体拥有的深海探索者附魔的有效等级 h，最大为 3，没有这一附魔时为 0，实体若未着地则减半。

3 如果没有海豚的恩惠效果，取$j = j_0 + \frac{h}{3}(0.546 - j_0)$，否则直接 j=0.96

4 取$g = 0.02(1 - \frac{h}{3}) + \frac{h \cdot Ms}{3}$，其中 Ms 为实体的地面移动加速度以 m/gt² 为单位的数值（见 6.2）

5 以 g 为 AI 加速度数值进行 AI 加速

6 以当前 Motion 作为位移趋势进行基于 Entity.move()的移动

7 如果移动过程中发生了水平碰撞且坐标所在方块网格处为可攀登方块，将 y 轴 Motion 设为 0.2m/gt

8 将 X、Z 轴 Motion 乘以 j，将 y 轴 Motion 乘以 0.8

9 如果实体没有 noGravity 标签，受到 0.005m/gt² 的重力加速度

10 如果水平方向发生了碰撞则取一个底面中心与现在碰撞箱底面中心在同一竖直线上，底面中心的 Y 坐标为移动前的坐标上方 0.6m 的与该实体碰撞箱大小相等的长方体区域，将该区域沿现在 Motion 指定的偏移量和方向移动，若区域内没有任何固体方块和流体，则将 Y 轴 Motion 设为 0.3m/gt

此时实体的运动类型是 FCMDG（即流体加速->AI 加速->移动->阻力->重力，各字母具体含义见 3.1）在实体没有深海探索者附魔是时实体的流体阻力为 0.2gt⁻¹ 或 0.1gt⁻¹，取决于实体是否在疾跑。在实体拥有深海探索者时，实体的 AI 加速度会随等级递增，最高与该实体的地面行走速度相同；阻力也随等级递增，最高为 0.464gt⁻¹，与在普通方块地面上的阻力相同，加上运算顺序可以证明**实体在拥有最高等级的深海探索者附魔时在水下的地面上行走的速度和在地面行走的速度相同**。有海豚的恩惠效果时阻力最低，达到 0.04gt⁻¹。最后一步我怀疑是与实体上岸有关，因为如果去掉这一步实体就无法上去在水面上方 0.5m 以上（台阶、附魔台或是灵魂沙）的岸边，而正常情况下则完全可以。

在第二种情况中运算过程如下（LivingEntity.travel(),1971-1991）：

1 以 0.02m/gt² 作为 AI 加速度进行 AI 加速

2 以当前 Motion 作为位移趋势进行基于 Entity.move()的移动

3 如果实体浸入熔岩深度不大于流体深度阀值：

3.1 水平轴上受到 0.5gt⁻¹ 的阻力，Y 轴上受到 0.2gt⁻¹ 的阻力

3.2 如果实体没有 noGravity 标签，受到 0.005m/gt² 的重力加速度

4 如果实体如果实体浸入熔岩深度大于流体深度阀值，各轴上收到 0.5gt⁻¹ 的阻力

5 如果实体没有 noGravity 标签，继续受到 0.02m/gt² 的重力加速度

6 同第一种情况的第 10 步

你没想错，足够深熔岩和水中相比熔岩中实体的下落速度确实比水中快。

好了，说正事，**这里实体的运算流程是 $\mathbf{FCMG_0DG_1}$**（实体浸入熔岩深度不大于流体深度阀值）**或 FCMDG**（实体如果实体浸入熔岩深度大于流体深度阀值），别的倒没什么好说的。

第三种情况在 6.3 节还要详细说明，在此从略。

第四种情况可以说是最常见，也是最常用的情况了，在实体在上面三种情况之外时被用到，流程如下（LivingEntity.travel(),2036-2055）：

1 取实体下方（坐标下方略多于 0.5m 处）方块的滑度 s

2 在实体着地时，取 j=0.91s，否则 j=0.91

3 在实体着地时，取$a = \frac{0.216Ms}{s^3}$作为 AI 加速度进行 AI 加速，其中 Ms 为地面移动加速度，否则取实体的飞行加速度作为 AI 加速度进行 AI 加速

4 如果坐标所在方块网格处为可攀登方块，实体的 X、Z 轴中单轴 Motion 会被限制在$\pm 0.15\mathrm{m/gt}$之间，Y 轴 Motion 小于-0.15m/gt 时会被设为 0.15m/gt。如果实体是潜行中的玩家，且可攀登方块不是脚手架，实体的 Y 轴 Motion 会被归零。

5 以当前 Motion 作为位移趋势进行基于 Entity.move()的移动

6 如果移动过程中发生了水平碰撞或在尝试跳跃且坐标所在方块网格处为可攀登方块，将 Y 轴 Motion 设为 0.2m/gt。

7 Y 轴 Motion 的一些运算

6.1 如果拥有漂浮效果将 Y 轴 Motion 改为 $0.01 \cdot$ *漂浮等级*$+0.8 \cdot$ 原*Y 轴* Motion

6.2 否则如果这是在客户端上的运算，而且实体所在区块未加载（连边界加载都不是，此时只有玩家可能会被运算），实体的 Motion 会被设为 -0.1m/gt（Y 轴坐标大于 0 时）或直接归零（Y 轴坐标不大于 0 时）

6.3 如果实体上述两步中条件均未被满足且实体的 NoGravity 标签为 false，受到 $0.08\mathrm{m/gt^2}$ 的重力加速度。

8 将水平轴上 Motion 改为原来的 j 倍，Y 轴 Motion 改为原来的 0.98 倍。

这时实体的运算流程为 CMGD。可以发现，空中运动时 Y 轴的阻力为 $0.02\mathrm{gt^{-1}}$，与 Wiki 相符；而水平轴上为 $0.09\mathrm{gt^{-1}}$，明显与 Wiki 不符。另外，滑度还可以决定实体的 AI 加速度和阻力系数，其中 AI 加速度与滑度的立方成反比，阻力系数与滑度负相关。

恶魂和幻翼有一套不同的方案：可以说是默认的删减版：

在实体接触水时，每 gt 以 0.02 m/gt² 的 AI 加速度进行 AI 加速，然后以当前 Motion 作为位移趋势进行基于 Entity.move()的移动，最后各轴的 Motion 乘以 0.08，完成。

在实体接触熔岩但不接触水时，每 gt 以 0.02 m/gt² 的 AI 加速度进行 AI 加速，然后以当前 Motion 作为位移趋势进行基于 Entity.move()的移动，最后各轴的 Motion 减半，完成。

在其它情况，实体的 AI 加速 a 和阻力系数的决定与默认类似，顺序相同，但没有漂浮效果和可攀登方块相关的运算。

除此之外，有些运动运算被安排在了 travel()方法外部，具体的流程如下：

1 实体基础运算
2 跳跃冷却更新
3 实体绝对值小于 0.03m/gt 的 Motion 分量被归零
4 若已经死亡，将前向和侧向加速度系数都改为 0
5 AI 或输入更新
6 跳跃相关判断及运算
7 前向和侧向加速度系数乘以 0.98
8 鞘翅耐久更新
9 执行 travel()方法
10 激流相关更新
11 实体挤压

其中第 9 步应该是要使实体运动的停止更加干脆利索。

6.2 AI 及属性对运动的影响

Minecraft 中几乎所有生物都有 AI，玩家的控制在某种意义上也可被称为 AI，盔甲架没有 AI，也不会自己控制自己移动在此不再考虑。最近红石科技搬运组在搬运一个讲解实体 AI 的视频系列（目前更新中），如有需要推荐去看一下。

Minecraft 中，生物 AI 主要由 Goal、Task、GoalSelector 和 EntityNavgation 四个类和 BodyControl、JumpControl、LookControl 和 MoveControl 四个直接控制生物的类组成。

Goal 和 Task（可以翻译成目标和任务）表示的是生物的一个整体的行为，或者说就是某种非条件或条件反射，再或者说就是实体将长期拥有的习

惯，大部分是从生物创建以来一直就有的，但有时候也会在生物创建后发生改变，如骷髅的武器被替换，猫被驯服时。

GoalSelector（可以翻译成目标选择器，但请勿与命令中的目标选择器混淆）会根据各 Goal 的优先级等信息安排 Goal 对生物各个方面的控制。

EntityNavgation（可以翻译成实体的导航，好吧，很像机翻）负责为生物进行寻路，是比 MoveControl 和 JumpControl 高级的运动控制。

Move/Jump/LookControl 通过修改生物的 AI 加速度和加速度系数较直接地控制着生物的运动。

B 站上红石科技搬运组搬运了一系列（疑似停更，只有一期但还是很有价值）关于生物 AI 的讲解视频，讲得很通俗，如果需要了解生物 AI 建议关注一下。

在 LivingEntity 类中，定义了几个关于运动的字段：

(1) **movementSpeed**：决定实体的地面移动加速度的一个实数，等于地面移动加速度以 m/gt^2 为单位的数值。对于玩家总为 generic_movement_speed 属性值，对于生物由种类、AI 状态和 generic_movement_speed 属性共同决定。

(2) **flyingSpeed**：决定实体的飞行移动加速度的一个实数，等于飞行移动加速度以 m/gt^2 为单位的数值。默认为 0.02，疾跑的玩家为 0.026，被骑乘且被控制的猪、马和赤足兽由 AI 状态和 generic_movement_speed 属性共同决定，具体来说是 movementSpeed 的 0.1 倍。

(3) **forwardSpeed**：表示实体**前向加速度系数**的一个实数，可以为负数，在每 gt 中 travel()方法执行前会被乘以 0.98。

(4) **sidewaysSpeed**：表示实体**侧向加速度系数**（准确点说是右向的）的一个实数，可以为负数，在每 gt 中 travel()方法执行前会被乘以 0.98。

(5) **upwardSpeed**：实体**向上的加速度系数**，也是实体向上的 AI 加速度以 m/gt^2 为单位的数值，可以为负数，在每 gt 中 travel()方法执行前会被乘以 0.98。

在 forwardSpeed、sidewaysSpeed 和 upwardSpeed 的平方和不大于 1 时，实体进行 AI 加速时会给实体加上一个朝向实体视线的水平投影方向，大小等于 AI 加速度与 forwardSpeed 的乘积的加速度、一个方向为实体视线的水平投影方向的正右方，大小等于 AI 加速度与 sidewaysSpeed 的乘积的加速度以及一个竖直向上的，大小数值上等于 upwardSpeed 的加速度。如果 AI 加速将要进行时 forwardSpeed、sidewaysSpeed 和 upwardSpeed 的平方和

大于 1，实体会保持比例不变缩放这三个 speed（得到三个对应的新值，对应字段的内容不变）使其平方和等于 1，再使用缩放后的值进行上述加速。大部分实体通常会选择向前直走来到达目的地，所以侧向及向上的速度系数一般为 0。在运动过程中，由于加速度与阻力的作用，实体的水平方向 Motion 会很快地趋近于一个最大值。

在上一节中已经说明，实体脚下的方块滑度越大，加速度就越大，阻力也就越小，那么，就可能存在一个平衡点，使得实体的最终速度达到一个极值。

在无流体的地面上时，陆生生物实体和玩家的水平方向运算流程可表示为 CMD（AI 加速->移动->阻力），也就是 AMD，假设 sidewaysSpeed 和 upwardSpeed 都为 0，将上一节的结论代入对应公式，得出最终速度（由每 gt 的位移表示）的表达式及 Ms·forwardSpeed=1m/gt 时的图像，其中 Ms 为地面移动速度：

$$\Delta d_{max} = \frac{0.216Ms \cdot forwardSpeed}{s^3 - 0.91s^4} \tag{6.2.1}$$

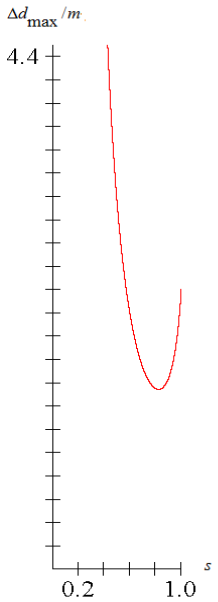

图 6.2 Δd_{max} 与 s 之间关系曲线

对其关于 s 求导，得

$$\Delta d_{max}{}'_s = -\frac{0.216(3s^2 - 3.64s^3)Ms \cdot forwardSpeed}{(s^3 - 0.91s^4)^2} \tag{6.2.2}$$

求得其[0,1]间的零点只有一个，约在 $s=\frac{75}{91}$（约为 0.82418）处，此时最终速度大小最小，约为 1.54331Ms·forwardSpeed。另外还可求得 s 等于各常见方块滑度时的情形：

(1) s=0.6 时，最终速度大小约为 2.20264Ms·forwardSpeed；
(2) s=0.8 时，最终速度大小约为 1.55101Ms·forwardSpeed；
(3) s=0.98 时，最终速度大小约为 2.2103Ms·forwardSpeed；
(4) s=0.989 时，最终速度大小约为 2.23265Ms·forwardSpeed。

但是粘液块上的实际行走速度要远小于 1.55101Ms·forwardSpeed（约 0.58807Ms·forwardSpeed），因为粘液块本身就会给实体减速，具体情况可参考 5.6 节。

MoveControl 类中为生物提供的最主要的用于进行 AI 加速的方法是 **moveTo(double x, double y, double z, double speed)**，这个方法需要指定一个目的地(x, y, z)和一个决定速度的实数 speed。执行后实体的地面移动加速度以 m/gt² 为单位的数值和 forwardSpeed 会被设为 generic_movement_speed 属性值与 speed 的乘积，并在简单判定后决定是否需要跳跃。最后，如果到达了目的地就将 forwardSpeed 设为 0。

研究所有生物的 AI 加速情况似乎不大现实，这里暂时不再说明。

除了 AI 加速之外，玩家和生物的跳跃也可以归入 AI 控制的运动的范围内，不过这里只研究比较一般的情况。在实体尝试跳跃时，游戏会检查下列条件是否成立：

A. 实体在水中
B. 实体在熔岩中
C. 实体着地
D. 实体浸入对应流体（实体接触的流体，如果同时接触水和熔岩则选定熔岩）深度不大于流体深度阀值
E. 实体跳跃冷却时间已经结束

如果满足下列全部条件，则可以执行从地面上跳起的行为（即将Y轴Motion以m/gt为单位的数值改为0.42+0.1跳跃提升等级，并将跳跃冷却时间重设为10gt，此时如果实体正在疾跑，将实体的Motion沿视线的水平投影方向加上0.2m/gt）：

(1) E成立
(2) C成立
(3) D成立或A、B均不成立

写成布尔代数式就是$\mathrm{CE}(\overline{\mathrm{A}}\cdot\overline{\mathrm{B}}+\mathrm{D})$

如果下列条件成立，则尝试执行从水中向上游的行为（即将Y轴Motion加上0.04m/gt，蜜蜂为将Y轴Motion加上0.01m/gt）

(1) A成立

(2) C、D中至少有一个不成立

写成布尔表达式就是$A \cdot (\overline{C} + \overline{D})$

如果满足下列条件，则尝试执行从熔岩向上游的行为（即将Y轴Motion加上0.04m/gt，蜜蜂为将Y轴Motion加上0.01m/gt，岩浆怪较为复杂）

(1) B成立

(2) C、D中至少有一个不成立

(3) 写成布尔表达式就是$B \cdot (\overline{C} + \overline{D})$

如果跳跃运算进行时发现实体中断了对跳跃的尝试，跳跃冷却会立即结束。

实际上目前不是很确定这些条件是正确的，因为找不到合适的实验方案来验证。

上面说到，generic_movement_speed 属性与实体的 AI 加速度的确定有关，进而影响着实体的行走速度。（玩家速度与它成正比，生物速度与它的平方成正比，不知道这一差异是否为有意为之的。）余下的部分主要将说明一下这一属性的确定方式。

实体属性值的确定一般有两个元素参与：基础值（base value）与修饰符（attribute modifier）。其中，基础值是一个实数，在 Wiki 上已经比较详细地给出了[11]（不过不要过于相信下一列的移动速度）。修饰符是一个获取属性值时对基础值进行的一个运算步骤（不改变基础值本身），Wiki 上在那一页同样也说明了，较为常见的有：

(1) 实体在疾跑时 generic_movement_speed 属性值被增加为原来的 1.3 倍。

(2) 实体每级速度效果能使其 generic_movement_speed 属性值增加 20%。

(3) 实体每级迟缓效果能使其 generic_movement_speed 属性值减少 15%。

(4) 一些幼年生物的 generic_movement_speed 属性值较大，如幼年僵尸是成年僵尸的 1.5 倍，幼年猪灵是成年猪灵的 1.2 倍等。

(5) 在实体在灵魂沙上面且有灵魂疾行附魔时，generic_movement_speed 属性值加上$0.03 \cdot (0.65 + 0.35\ \textit{附魔等级})$。

易知generic_movement_speed属性值的最终值与前三种修饰符的作用顺序无关。第四种修饰符总是最先被应用。

另外generic_movement_speed属性值存在上下限，分别为1024和0

6.3 鞘翅飞行

我不是很确定研究鞘翅飞行是否很有实用价值，但见它那么常用也就提一下吧。

在实体使用鞘翅飞行时，其运动完全由其视线方向决定，水平速度最终会与视线方向的水平投影同向，不再受 AI 加速，其它加速也因位置原因一般很少受到。期间如果实体发生水平方向的碰撞，实体会受到一个由速度和碰撞位置决定的碰撞伤害；如果实体着地，鞘翅飞行会终止。

鞘翅飞行时，实体的运动运算流程如下（LivingEntity.travel()，1992-2035，为方便理解做了一定改动）：

```
Vec3d v = this.motion;
Vec3d facing = this.getRotationVector();    //视线方向的单位向量
float pitch = this.pitch * 0.017453292F;    //俯仰角
double cosp = Math.sqrt(facing.x * facing.x + facing.z * facing.z);     //俯仰角的余弦
double vxz0 = Math.sqrt(v.x * v.x + v.z * v.z);     //水平方向合速度
double c1 = facing.length();    //永远等于1
float n = MathHelper.cos(pitch);    //又是一个余弦，用意不明
n = (float)((double)n * (double)n * Math.min(1.0D, c1 / 0.4D)); //不懂Mojang
v = this.motion.add(0.0D, 0.08 * (-1.0D + (double)n * 0.75D), 0.0D);    //重力
double q;
if (v.y < 0.0D && cosp > 0.0D) {    //平时或视线斜向下且Y轴Motion小于0
    //水平轴加速，Y轴受到大小0.1cos^2p gt^-1的阻力
    q = v.y * -0.1D * (double)n;
    v = v.add(facing.x * q / cosp, q, facing.z * q / cosp);
}

if (pitch < 0.0F && cosp > 0.0D) {  //视线斜向上
    //Y轴向上加速，水平轴受阻力
    q = vxz0 * (double)(-MathHelper.sin(pitch)) * 0.04D;
    v = v.add(-facing.x * q / cosp, q * 3.2D, -facing.z * q / cosp);
}

if (cosp > 0.0D) {  //视线不竖直
    //从开始到现在的水平轴的速度变化量削减0.1倍
    v = v.add((facing.x / cosp * vxz0 - v.x) * 0.1D, 0.0D, (facing.z / cosp * vxz0 - v.z) * 0.1D);
}

this.motion = v.multiply(0.9900000095367432D, 0.9800000190734863D, 0.9900000095367432D);
this.move();
```

图 6.3.1 相关代码图

这里我们主要研究视线平视或斜向下的情形,不考虑其它加速，此时：

$$a_y = gt_0(0.75\,cos^2\,p\,itch - 1) \tag{6.3.1}$$

$$k_y = 0.98(1 - 0.1\,cos^2\,p\,itch) \tag{6.3.2}$$

我们也可以看出，在 Y 轴上，实体的运动明显是 ADM 型，带入对应公式，有

$$\Delta y_{max} = \frac{gt_0(0.75\,cos^2\,p\,itch - 1)(98 - 9.8\,cos^2\,p\,itch)}{9.8\,cos^2\,p\,itch + 2} \tag{6.3.3}$$

由源码也可以得出:

$$a_{xzmax} = -\frac{0.09\Delta y_{max}\,cos^2\,p\,itch}{t_0 k_y} \tag{6.3.4}$$

$$k_x = k_z = 0.99 \tag{6.3.5}$$

其中a_{xzmax}为实体在水平方向上的最大合加速度，

我们发现，鞘翅飞行的实体在 X、Z 轴上的运动也是 ADM 型，代入对应公式，又有：

$$\sqrt{\Delta_x^2 + \Delta_z^2} = \frac{891gt_0(cos^2\,pitch - 0.75\,cos^4\,pitch)}{9.8\,cos^2\,pitch + 2} \tag{6.3.6}$$

这时，我们得到了在视线水平或斜向下时由视线俯仰角得到最终水平合速度以及最终竖直速度的方法。至于斜向上的情况个人能力不足，只能写段小程序做出图像，没有得出表达式。

下面是实体在由静止飞行 72000gt 后水平速度与俯仰角的关系图：

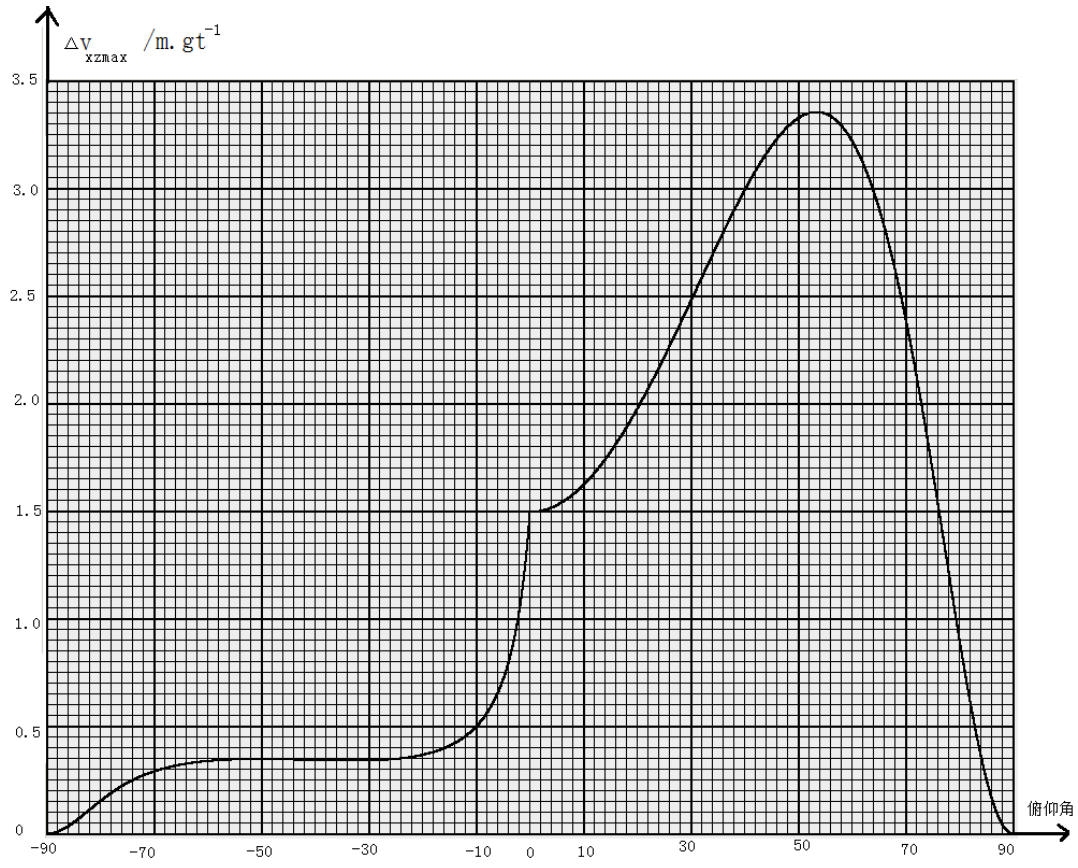

图 6.3.2 水平速度与俯仰角的关系图

在表达式(6.3.6)中求出$\sqrt{\Delta_x^2 + \Delta_z^2}$关于$cos^2\,pitch$的导函数，求得其在定义域上仅有一个零点$\frac{\sqrt{67.8}-3}{14.7}$，也就是说图中那个最高点对应俯仰角正切的平方就是这个值，可以求得此时俯仰角约为 53.366 度。将其带入式(6.3.6)，求得此时实体飞行速度约为 3.3888m/gt,即 67.776m/s，与实验值 3.38879125m/gt

高度接近，也明显高于 52 度俯冲的速度 3.38383913m/gt，所以，Wiki 上那个说 52 度俯冲最快的那些个地方又得改了。

如果希望更加详细地了解一些其它方面的实验数据、仰视时的情况和一些实际问题，可以查阅这两个专栏[12]。

7 几类具体实体运动分析

截止到 1.16.4，Minecraft Java 版中实体共有 94 种，在这里很难也没有必要一一对它们进行说明，这里就挑选 10 种最常见也最典型的实体从创建、运算、对外界影响和移除四方面进行较为详细的说明。其中，创建这一方面，所有实体都可以通过指令创建，可进行非常灵活的控制，除未指定时各属性默认为 0 或 false 外，不存在固定的机制，下面不再说明。

以后如果说某实体在哪里生成，就是说实体刚生成时的坐标在哪里。

7.1 TNT

TNT 实体永远是实体运动研究中关注的焦点（尽管机制特别简单），不仅是因为自身的运动重要，更因为它是目前最重要的爆炸源，可以破坏方块，推动或杀死其它实体。

除了使用指令生成外，TNT 可以由以下方式创建：

(1) 点燃 TNT 方块。在 TNT 方块收到激活后的第一次方块更新时、被打火石点燃、被着火的箭矢击中时以及被爆炸破坏时会消失，然后一个 TNT 实体会在该 TNT 方块所在方块网格底面中心处生成，该实体会拥有一个沿 Y 轴向上，大小 0.2m/gt 的初始向上 Motion，和一个随机方向大小 0.02m/gt 的初始水平 Motion，前三种情况下初始 Motion 为 80，被爆炸破坏时初始 Motion 为 10～39 间一个整数。

(2) 使用发射器发射。此时行为与用红石点燃发射器正对方向的一个 TNT 方块无异。

TNT 的运算过程如下（TntEntity.tick()，50-74）：

1. 如果没有 noGravity 标签，受到大小为 0.04m/gt^2 的重力加速度
2. 以当前 Motion 为位移趋势进行基于 Entity.move()的移动
3. 各轴 Motion 乘以 0.98，也就是说受到大小为 0.02gt^{-1} 的空气阻力
4. 如果着地了 X、Z 轴 Motion 乘以 0.7，Y 轴 Motion 减半并反转
5. 将引信时间减一
6. 如果引信时间等于 0：

6.1. 将自身标记为移除

6.2. 在坐标上方 0.06125m 处产生一个 4 级爆炸

7. 流体相关更新（含流体推动）

由此可见，**TNT 实体的运动运算流程是 GMDF**（重力->移动->阻力->流体），重力加速度大小恒为 $0.04m/gt^2$，空气阻力系数恒为 $0.02gt^{-1}$。第四步中，若 TNT 着地了，水平 Motion 乘以 0.7 是说 TNT 实体会受到了 $0.3gt^{-1}$ 的地面阻力，而 Y 轴速度减半反转（此时如果撞到一般方块而且没有与气泡柱相交 Y 轴 Motion 应为 0）应该是为了防止 TNT 实体被粘液块和床弹起，不过这对气泡柱下拉 TNT 实体着地时也有较小影响。TNT 实体的爆炸检查置于运动运算（流体加速除外）后意味着 TNT 在爆炸时已经进行的运动运算（流体加速除外）周期数与初始引信时间相同。

TNT 创建时的初始 Motion 会使 TNT 实体跳起，并在 4gt 后达到高于初始坐标 0.38423872m 的最高点。如果点燃时它在平地上，它会在 9gt 后落地，落点距初始坐标 0.16625224m，然后继续运动，直到第 41gt 才静止，此时距初始坐标 0.20256773m。如果它在足够高的空中，它的水平速度不会衰减到 0，最终会在下落 73.45407581m 后爆炸，如果在引燃前在其上方用完整方块挡住，则 TNT 最终会在下落 79.84488503m 后爆炸，水平方向上偏移了 0.80164885m。

最后，重要的事情说两遍不算多，**TNT 的爆炸中心在其坐标上方 0.06125m 处**。

7.2 下落的方块

在受重力影响的方块下方失去支撑后的第一次更新后 2（一般重力方块）或 5（龙蛋）gt 后的计划刻阶段一个下落的方块实体会在该方块所在方块网格中心生成，没有初始 Motion。需要知道，因为某些不明的打算，该方块此时仍然存在。

重力方块的运算流程大概如下（FallingBlockEntity.tick()，98-192）：

1 检查创建下落的方块的那个方块是否被替换（活塞是否无头等仅状态替换不算），如果是，将自身移除并结束运算，否则移除该方块

2 如果没有 noGravity 标签，受到大小为 $0.04m/gt^2$ 的重力加速度

3 以当前 Motion 为位移趋势进行基于 Entity.move()的移动

4 如果是混凝土粉末且现在坐标所在方块网格中存在水，复原为方块状态，然后立即固化

5 如果 Motion 大于 1m/gt，而且是混凝土粉末，进行一次移动前后两点间检查固体方块和流体源的碰撞箱的 raycast，如果被水源阻挡，在阻挡射线的水那里复原并固化。

6 如果坐标超出了可放置方块的上下限且已下落的时间在 100gt 以上或坐标仍在可放置方块范围内且下落的时间已经超过 600gt，游戏规则允许实体掉落物品，而且没有着地，则掉落成物品并移除自身

7 如果着地了 X、Z 轴 Motion 乘以 0.7，Y 轴 Motion 减半并反转

8 如果已经着地且当前位置方块可以被落沙替换且复原后的方块能稳定存在（如铁轨不能浮空），在坐标所在方块网格中复原，否则掉落成物品

9 各轴 Motion 乘以 0.98，也就是说受到大小为 $0.02gt^{-1}$ 的空气阻力

可以看出，**下落的方块实体的运动运算流程是 GMD**，空气阻力和重力加速度与 TNT 实体相同，另外请注意下落的方块没有进行流体加速。第一步个人猜测是修复一种活塞单维度刷沙机用的，因为落沙实体的创建是在 NTU（计划刻）阶段，第一次运算是在 EU（实体运算）阶段，而活塞伸出正好位于这两个阶段中间（BE，方块事件阶段），所以如果没有这一个是否被替换的检查而直接删除方块，利用 0t 活塞就可以推走沙子并完成一次复制了。第 4-5 步 Mojang 可谓是想尽了办法让混凝土粉末落沙“不错过”途中的任何水方块，算是…不过特性还是有点，如射线可以穿过流体顶部的缝隙等。第 7 步我认为是多余的，因为此后该实体就复原或者掉落成物品了，不会再进一步移动，不排除是某个版本的遗留代码。第 8 步中可以被落沙替换的方块需要满足以下条件：

(1) 不能作为可下落方块的支撑方块

(2) 能被玩家放置方块替换

Wiki 以前上说落沙在很高的高空落下时可能会在触及地面前就掉落成物品了（具体出处已不可考），这在 MinecraftJE1.0 及以前确实如此，那时下落的方块会在生成 100gt 后被移除，但是至少在 1.7.10 之后就是现在的情况了，也就是一般不会那么短时间就掉落。

现存的末地门刷沙机的原理是在同一游戏刻内第 3 步移动的过程中使实体同时做到与末地门相交且在合适的地方着地，然后虽然末地门传送走了落沙实体（在新维度创建了新实体并把旧实体标记为移除），但落沙实体这一游戏刻的运算并不会因此终止，甚至还可以正常地复原，所以就复制了两

份沙子——复原的一份和被传送的一份。不过，因为落沙实体无法通过下界传送门被传送，所以地狱门刷沙机并不可行。

7.3 物品

获取物品实体的途径有很多，常见的有：

(1) 破坏方块。这时游戏会等可能地选取将被破坏方块的方块网格从各方向向内缩 0.25m 后的区域中的一个点并在该点处生成一个物品实体，然后分别给实体 X、Z 轴上一个-0.1～0.1m/gt 之间的的初始 Motion，给 Y 轴上一个固定向上且大小为 0.2m/gt 的初始 Motion。如果该方块掉落了多个物品实体，它们的位置和初始 Motion 可以不同。如果方块是被爆炸破坏的，那么这些物品实体在开始运算前会被尝试合并一次。

(2) 杀死实体。这时游戏会在实体坐标处生成一个物品实体，然后分别给实体 X、Z 轴上一个-0.1～0.1m/gt 之间的的初始 Motion，给 Y 轴上一个固定向上且大小为 0.2m/gt 的初始 Motion。

(3) 由部分实体掉落。下面给出的几例杀死实体类似，只是初始位置一般要比杀死实体高一点。这一高度差对于被修剪的雪傀儡是 1.7m，对于合成面包的农民是 0.5m，对于被修剪的羊和刚成年的海龟是 1m，对于被拾起的箭和三叉戟是 0.1m。

(4) 由发射器或投掷器发射。选取发射器或投掷器的中心，向发射器或投掷器的朝向偏移 0.7m，再向下偏移 0.15625m（发射器或投掷器朝向为东西南北四个方向之一时）或 0.125m（发射器或投掷器朝向为上下两个方向之一时），在此处生成一个物品实体，给它一个方向为发射器或投掷器的朝向，大小为 0.2～0.3m/gt 中（朝向 Y 轴时恒为向上 0.2m/gt）一个值的初始 Motion，最后再在各轴上分别（各轴上可以不相同）加上一个分布满足平均数为 0，标准差为 0.045m/gt 的正态分布的随机 Motion 作为误差。正态分布意味着误差实际上有极其微小的概率远大于标准差，需要注意。

(5) 村民共享食物、赠送礼物和猪灵丢出交易品。首先游戏会选定一个目标坐标和出发坐标，目标对村民来说位于目标实体坐标处，对猪灵来说位于目标实体坐标上方 1m 处，出发坐标则总是投掷者眼部坐标下方 0.3m 处。然后，游戏会在出发坐标处创建并将物品 Motion 设置为从出发坐标指向目标坐标，大小数值上为出发坐标和目标坐标间距离的 0.3 倍的一个向量。注意此时不存在随机性。

物品实体的运算流程大致如下（ItemEntity.tick()，77-153）：

1 进行实体基础运算（见 2.4 节）

2 若拾起冷却不为 32767 则减一

3 若在流体中且浸入高度大于 0.10138889m，受到对应流体的浮力作用和流体阻力，否则，如果没有 noGravity 标签，受到大小为 0.04m/gt² 的重力加速度

4 如果卡在了方块或固体实体中或与固体实体相接触，则禁用碰撞检查（noClip 设为 true）并尝试进行从方块中被推出的运算，否则重新启用碰撞检查

5 如果没有着地，或者水平 Motion 大小大于$\sqrt{0.00001}$m/gt，或者实体 ID 与实体已存在的刻数（最初为 0）的和为 4 的倍数，则：

5.1 以当前 Motion 为位移趋势进行基于 Entity.move()的移动

5.2 如果实体着地了，取 k=0.98 与坐标下方一格处方块滑度的积，否则 k=0.98

5.3 将 X、Z 轴 Motion 乘以 k，Y 轴 Motion 乘以 0.98

5.4 如果着地了且当前 Y 轴 Motion 小于 0，将其减半并翻转

6 如果实体此次移动过程中进入了新的方块网格且实体已被加载的刻数为 2 的倍数，或如果实体此次移动过程中没有进入新的方块网格且实体已存在的刻数为 40 的倍数，尝试合并一次

7 增加已存在的刻数计时

8 再次进行流体相关运算（含流体加速）

9 如果已存在的刻数大于 6000 则标记为已移除

这里可以看出，不考虑浮力和从实体方块推出的过程，**物品实体的运动运算顺序为 F_1GMDF_2**，也就是说它在每刻中分别在实体基础运算和第 8 步时进行了两次流体加速，这是比较罕见的，目的可能是提高物品运输的速度。第 5 步比较有趣，这意味着物品实体在着地且 Motion 接近 0 时（如在地面上滑动时）每 4gt 才运算一次移动和阻力，而且运算的具体时间与其实体 ID（或者说就是创建的序号，同一时间内创建的一般连续）和已存在的的时间有关，这已经被应用于无线红石。[13]注意 5.4 步中的反转速度由于有速度小于 0 的判断并不会造成粘液块无法弹起物品实体，用意未知，不排除 SBMojang。

在物品实体进入足够深的水中时，它不会受到重力，在 X、Z 轴上会受到一个大小为 $0.01gt^{-1}$ 的流体阻力，这会和空气阻力叠加，而且在 Y 轴 Motion 小于 0.06m/gt 时还会受到一个竖直向上大小为 $0.0005m/gt^2$ 的浮力加速度。

在物品实体进入足够深的熔岩中且未与水接触时，它也是不会受到重力，在 X、Z 轴上会受到一个大小为 $0.05gt^{-1}$ 的流体阻力，这会和空气阻力叠加，而且在 Y 轴 Motion 小于 0.06m/gt 时还会受到一个竖直向上大小为 $0.0005m/gt^2$ 的浮力加速度。

尽管水中的掉落物会自己飘较远的距离，但这并不是随机加速度作用的结果，实际上根本就没有这回事。水中的物品的较远距离漂浮就是创建时有初始速度和阻力系数较小的结果。在一格的静止的水底生成的物品实体的运动相关部分数据如下（X、Z 轴没有必要研究）：

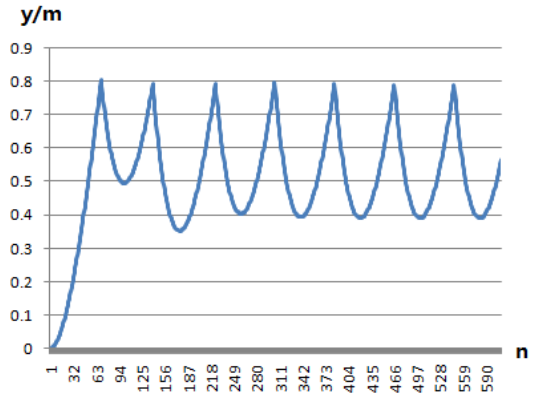

图 7.3.1 位移与时间关系曲线

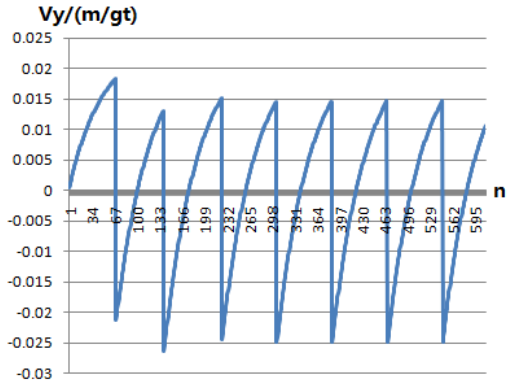

图 7.3.2 速度与时间关系曲线

可以看出，实体的速度发生了几次突变，这是实体在入水深度过浅时受到重力加速度作用导致的。另外，高度呈现出明显周期变化，但始终不至于再次着地。

图 7.3.3 这一种物品运输很多人都做过，这里来研究一下。

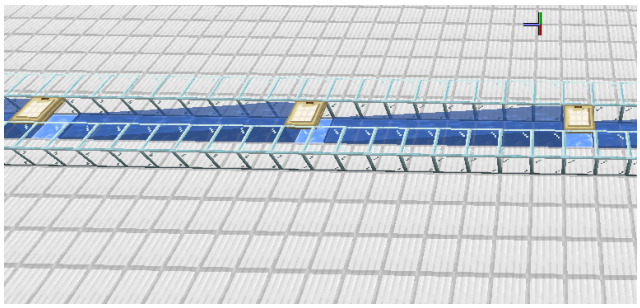

图 7.3.3 物品运输

在物品实体静止从水底生成通过 125m 长的这种物品运输装置并最终停止的过程中物品实体的水平速度及位移随时间变化规律如下（红线为纯铁块地面，蓝线为纯蓝冰地面）：

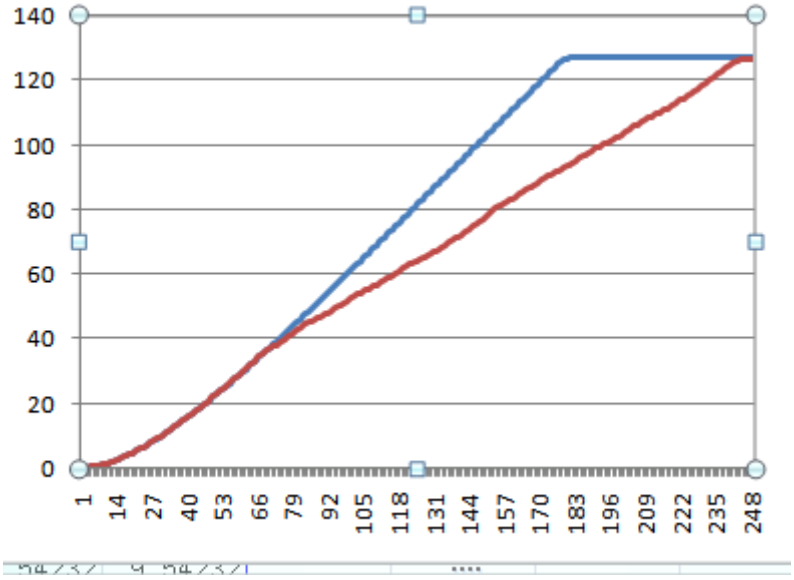

图 7.3.4 位移与时间关系曲线

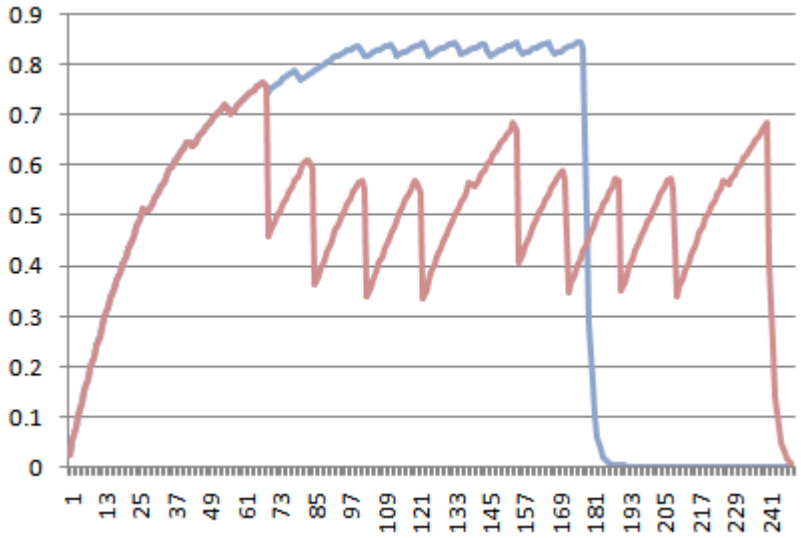

图 7.3.5 速度与时间关系曲线

由此看出，使用铁块作为地面时，实体稳定后的速度随时间在大约 0.35m/gt-0.7m/gt 之间来回变化，但没有明显的周期性，而且物品实体加速到稳定所需的时间和位移分别约是 20gt 和 8m，此后至结束时物品的平均速度约为 0.527m/gt，或者说不卡顿时为 10.54m/s。使用蓝冰作为地面时，71gt（34m）前速度与铁块地面相似，71gt 后速度有较大提高，稳定时平均速度大约在 0.832m/gt 左右，约合无卡顿的 16.64m/s。进一步实验表明，将两段水流中间附近的地面换成蓝冰就可以达到与纯蓝冰水道相近的速度，但其他地方由于实体不会着地，受不到滑度影响，所以将地面全部换成冰无法进一步提高运输速度。

可能得问了，物品与流体到底啥关系，在这里差不多光说流体的影响了。实际上，物品的运动研究主要还是在流体和冰道运输以及物品收集，但冰道运输只是滑度和粘液块弹射的结合，没有特殊内容，而收集装置设计上还是靠实验比较好，真正的计算并无一般规律，在此从略。

7.4 经验球

获取经验球实体的方法主要有：

(1) 部分实体掉落。在这种情况下，一些经验球会在实体坐标处生成（交易除外，这时是在坐标上方 0.5m 处）并在 X、Z 轴上分别被随机赋予 ±0.2m/gt 之间的一个初始 Motion，在 Y 轴上被随机赋予一个 0～0.4m/gt 的初始 Motion。

(2) 破坏一些方块。此时一些经验球会在被破坏的方块所在方块网格中心处生成，初速度情况与第一种相同。

(3) 从熔炉掉落。此时一些经验球会在玩家坐标处生成，初速度情况与第一种相同。

(4) 附魔之瓶撞击方块。此时一些经验球会在发生碰撞前最后一刻的坐标处生成，初速度情况与第一种相同。

经验球的运算流程大致如下（ExperienceOrbEntity.tick()，49-111）：

1. 进行实体基础运算（见 2.4 节）
2. 拾起冷却则减一
3. 取眼部坐标下方 1/9m 处的一个点，若该点所在的方块网格中的流体为水且浸没了该点，在 X、Z 轴上受到大小 $0.01gt^{-1}$ 的流体阻力，在 Y 轴上受到 $0.0005m/gt^2$ 的浮力加速度并把 Y 轴 Motion 限制在 0.06m/gt 及以下，否则，如果没有 noGravity 标签，受到大小为 $0.03m/gt^2$ 的重力加速度
4. 如果坐标处方块网格中有熔岩，在 X、Z 轴上赋予一个满足三角分布的，随机在±0.2m/gt 间的 Motion 及一个 0.2m/gt 的 Y 轴 Motion
5. 如果卡在了方块或卡在固体实体中或与固体实体相接触中，尝试进行从方块中被推出的运算
6. 如果 20 减去距上一次更改目标以来的刻数大于实体 ID 与 100 的模

6.1 如果没有选定的目标、原目标距离大于 8m 或已被放弃则寻找坐标 8m 内最近的非旁观者玩家作为新目标玩家

7. 如果现在的目标玩家是旁观者就放弃该目标
8. 选取目标玩家坐标与眼部坐标连线中点为目标点，如果目标点与自身坐标距离小于 8m，沿从自身坐标向目标点方向，受到大小为$0.1 \cdot (1 - \frac{距离}{8})^2 m/gt^2$的加速度
9. 以当前 Motion 为位移趋势进行基于 Entity.move()的移动
10. 如果实体着地了，取 k=0.98 与坐标下方一格处方块滑度的积，否则 k=0.98
11. 将 X、Z 轴 Motion 乘以 k，Y 轴 Motion 乘以 0.98
12. 如果着地了，将 Y 轴 Motion 乘以 0.9 并翻转，以此防止被粘液块弹起
13. 如果已存在的刻数大于 6000 则标记为已移除

由此得出，**经验球的运算流程主要为 FGMD**，运动机制与物品实体略有相似。第 4 步中实体的那个加速度可理解为经验球会慌不择路地逃离熔岩，不过成功率不大高。第 6、7 步中目标的选择不排除玩家是否已经下线或是否仍在该维度，所以如果玩家在经验球靠近的时候下线或者被传送到另一维

度且坐标与在原维度时坐标接近，经验球就会在玩家下线的位置或与在另一维度玩家的坐标相同的原维度的坐标处继续飞行而不更换目标。两个字，专一！（不过应该还是特性，据说 1.17 已修复）第 8 步中经验球靠近玩家是利用加速度进行的，所以在到达目标位置后如未被玩家拾取就会出现刹不住车的情况，进而就在目标点附近来回飞行，直到被玩家拾取或被清除。

7.5 船

船的获取方式应该只有两种，而且都没有初始 Motion

(1) 使用物品形式的船放置。此时游戏会选取玩家正对的点，并检查如果创建一个将碰撞箱从各方向向内缩 0.1m 后的船会不会被阻挡。如果不会，一只船会在该点处生成，朝向偏航角与玩家相同

(2) 使用发射器发射。此时如果发射器正对的位置有水，游戏会选取一个从发射器所在方块网格中心沿发射器朝向偏移 1.125m 后再向上偏移 1m 后得到的一个点，在这里生成一只朝向与发射器朝向一致的船，发射器朝下或朝上时船总是朝东。类似地，如果发射器正对的位置下方有水，游戏会选取一个从发射器所在方块网格中心沿发射器朝向偏移 1.125m 后所得的一个点，在这里生成一只朝向与发射器朝向一致的船，发射器朝下或朝上时船总是朝东。

注意，第一种方法中船的有机会卡到方块中，第二种则完全不进行碰撞检查，由此可以把船卡到方块中，第二种方法甚至可以轻松地制造堆叠的船。

船的运算流程大致如下（BoatEntity.tick()，235-312）：

1 更新船的位置及滑度记录
2 如果船连续在水下 60gt，移下所有骑乘者
3 实体基础运算
4 客户端与服务端间位置同步相关运算
5 根据现在的位置决定阻力系数 f，设 $k=1-ft_0$
6 将水平速度和偏航角的角速度分别乘以 k
7 如果没有 noGravity 标签，受到大小为 $0.04m/gt^2$ 的重力加速度
8 如果在水中，Y 轴受到一定的阻力和微量加速度用于产生上下浮动效果
9 如果玩家按住了且只按住了 A、D 键之一，向对应方向将偏航角角速度加上$1°/gt$
10 将偏航角加上偏航角角速度与 1gt 的积

11 如果玩家没有按住 W、S 键且按住且只按住 A、D 键之一，取 0.005m/gt² 作为前向加速度；如果玩家只按住了 W 键，取 0.04m/gt² 作为前向加速度；如果玩家只按住了 S 键，取-0.005m/gt² 作为前向加速度；如果玩家同时按住了 W、S 键，取 0.035m/gt² 作为前向加速度

12 给船一个朝向船的视线的水平投影方向，大小为前向加速度大小的加速度

13 以当前 Motion 为位移趋势进行基于 Entity.move()的移动

14 进行气泡柱相关的运算

14.1 如果连续接触了向下的气泡柱达到 60gt，使船下沉（Y 轴 Motion 设为 -0.7m/gt）并移下所有骑乘者

14.2 如果连续接触了向上的气泡柱达到 60gt，给船 2.7m/gt（有玩家骑乘时）或 0.6m/gt（无玩家骑乘时）的 Y 轴 Motion

14.3 如果同时接触了两种气泡柱，以东南方为准，其中东方优先

15 再次检查方块网格碰撞（checkBlockCollision()）

16 如果可能，尝试使与碰撞箱向四周扩大 0.2m 后得到的区域相交（不计边界）的可被挤压的实体（船上下方的实体不被检查）上船，否则尝试对其进行挤压

相对来说比其它的复杂一些，不过差不多可以理解。**船在 X、Z 轴上的运算顺序主要为 FDCM**，在第三节已经初步说明了。竖直方向上比较特殊，**当船在空中运动时 Y 轴上是不受空气阻力的，运算顺序为 GM**，所以只要船的初始位置足够高，船就可以无限加速，直到速度太大以至于远距离碰撞检查把服务器卡到崩溃（至少每 gt 下落几十万米不掉刻是不成问题的）。第 15 步中多出的那次检查用意暂时不能确定，只是想增加仙人掌的伤害也说不定。

船的阻力系数在不同位置的大小如下表：

表 7.5 船的阻力系数

状态	水平方向（单位 gt^{-1}）	竖直方向（单位 gt^{-1}）
入水瞬间	0.95	0
水面	0.1	0.25
浸没于流水	0.1	0
浸没于	0.55	0.25

静水 空中	0.1	0
无水的 地面上	将下方直接接触的方块滑度做平均，然后 1 减去这个平均值就是阻力系数的数值	0

由阻力和动力大小结合对应公式可以容易地推算出：只按住 W 键控制时，在水面上和空中，船的最终水平速度趋近于 0.4m/gt；在普通地面上，船的水平速度最终趋近于 0.1m/gt；在浮冰和霜冰上，船的水平速度最终趋近于 2m/gt；在蓝冰上，船的水平速度最终趋近于 3.63636m/gt。但是在粘液块地面上并不能这样推算，因为粘液块本身有个减速作用。实验表明，在粘液块上，船的水平速度最终趋近于 0.05907m/gt。存档中有几条冰道，本来计划测试一下混合冰道上的最终速度（按窃梦者的研究[5]应该是可以理论计算的），但因电脑性能问题未能完成，希望哪位好心人帮忙测试一下。

船的运动（5～13 步）只在服务端和客户端中的一个地方运算，另一端的 Motion 为零，坐标滞后。当船被玩家骑乘时，船的运动是在客户端运算的，否则是在服务端运算的，所以当船在快速行进时玩家突然下船就会导致两端的坐标停止同步从而出现大小为 1gt 的位移的偏差。又因为玩家刚下船时服务端的船的 Motion 仍为 0，所以服务端无法自行用船拥有的 Motion 让船继续运动调整这个误差，进而导致船的真碰撞箱留在了玩家下船的地方。服务端认为玩家在就在碰撞箱的上面，但在客户端碰撞箱随船已经移走了，它认为玩家可以下落，但是服务端一看不对就又将客户端玩家 TP 回来，其它如向外移动的请求也随之被驳回，于是玩家就被卡在了船的服务端的碰撞箱中。解决方案也简单，那就是破坏掉那只船，只要找到客户端的船攻击它既可。另外，角速度的同步大致也是这样的，所以就有了另一个类似的 Bug[14]。这样的关于两端同步的 Bug 还有很多，最经典的就是 MC-4，这里不再多说。

实体被动上船需要满足以下条件：

(1) 船上没有玩家
(2) 船上的直接骑乘者总数小于 2
(3) 实体没有骑乘其它实体（不能脚踏两只船）
(4) 实体的宽度小于船的宽度（1.375m）
(5) 实体是水生生物外的生物
(6) 实体没有潜行且骑乘冷却时间已结束
(7) 队伍规则允许实体挤压

了解了这些，还有第 16 步中给出的那个范围，估计得少跟村民费不少劲。第 7 条可能是一个 Bug，以后可能修复。

实际上船还有一个摔坏的机制，似乎是要求船在 20gt 内从至少 3m 高的地方摔落下来 3 次，这是几乎不可能做到的。

7.6 矿车

矿车的运算流程比较复杂，而且也已经有很多详细的研究资料了[15]，下面仅列出几点零星的，主要是比较偏的研究结果。

矿车的主要获取方法如下，创建的矿车均无初速：

(1) 在轨道类方块上使用矿车物品，初始位置位于轨道所在方块网格底面中心上方 0.0625m 处。

(2) 使用发射器向轨道类方块发射，初始位置会略微偏向发射器所对的方向。

矿车脱轨移动时运算流程大致如下（AbstractMinecartEntity.tick(),258-360）：

1 在 Y 坐标小于-64 时破坏自身

2 下界传送门相关更新（tickNetherPortal()）

3 如果没有 noGravity 标签，受到大小为 0.04m/gt² 的重力加速度

4 将 X、Z 轴 Motion 限制在最大速度（±0.4m/gt）以内

5 如果着地了，受到 0.5gt^{-1} 的地面阻力，Y 轴 Motion 减半，然后以当前 Motion 为位移趋势进行基于 Entity.move()的移动，否则先以当前 Motion 为位移趋势进行基于 Entity.move()的移动，然后受到 0.05gt^{-1} 的空气阻力。

6 寻找与碰撞箱向东西南北四个方向扩大 0.2m 后的区域相交（不计边界）的可被挤压的实体，如果可能使它乘上矿车，否则尝试将其挤开。

7 流体相关运算

不考虑达到速度大小上限，在着地时，矿车的运算顺序为 DMF，否则为 GMDF。上述运算过程中 3-5 步是脱轨运动特有的。

大量堆叠的矿车每刻运算结束时的水平 Motion 会以指数爆炸的方式递增，最终可以达到正/负无穷大（浮点数上溢的结果）。堆叠的矿车越多，所需时间越短，堆叠 300 矿车时从矿车错位开始到 Motion 达到无穷大仅需不到一秒的时间。听起来很吓人，但实际上因为速度限制，这并不会使速度也达到无穷大。实际上，在地面上时，由于强制限速这种矿车的运行速度最大也只有$4\sqrt{2}$m/gt，说不定速度限制就是因为这个引入的。但是，一旦 Motion

开始这个指数爆炸过程，在区块彻底卸载前 Motion 就不会下降，所以矿车就会在水平方向上保持匀速直线运动（MC-14）。在区块重载后，Motion 也会很快地恢复无穷大，然后继续做上述运动。原因很可能隐藏在 pushAway()方法中，因为在速度被限制到运算结束可能加速能加速的阶段就只有这一个了。现在，这一 Bug 已经被报告给 Mojang 了（MC-14），之后也许会修复。

实体被动骑上矿车需要满足以下条件：

(1) 实体是船或者除蝙蝠、铁傀儡之外的生物
(2) 实体没有骑乘或被骑乘
(3) 矿车是空的可骑乘矿车
(4) 矿车的水平 Motion 大于 0.1m/gt
(5) 实体没有潜行且骑乘冷却时间已结束
(6) 队伍规则允许实体挤压

容易得到，船是没有被排除的，所以平时船套矿车严格意义上来说是矿车套船。第 4 条估计大部分人都不了解，知道这一条拿矿车运输村民也简单多了。

矿车在移动过程中如果发现坐标处方块网格或坐标所在方块网格下方处存在铁轨就会以最短位移被 TP 到铁轨指定的轨迹上，其中下方铁轨优先（联想一下连续浮空铁轨）。铁轨指定的轨迹对于直轨是距离方块网格底面 0.0625m 处的一个水平截面上一对相对的棱中点的连线，对于弯轨是距离方块网格底面 0.0625m 处的一个水平截面上一对相邻棱中点的连线，对于斜轨是方块网格一个体对角面的两短边中点连线正上方 0.0625m 的线段。矿车在铁轨上时总是沿这些指定的轨迹行进。图 7.6 是一个典型的例子，注意铁轨底部均高出地面 0.0625m：

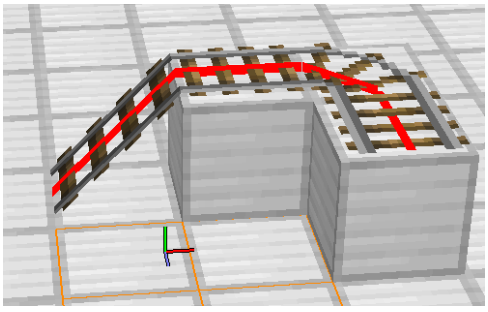

图 7.6 一段铁轨指定的轨迹

矿车在轨道上的运动很大一部分是靠 TP 来完成的，所以一不注意就会卡在方块内部，特别是弯轨，非常 Buggy。

TNT 矿车在以大于 0.1m/gt 的水平 Motion 撞到方块时、从超过 3m 的空中落下时和引信时间结束时会发生爆炸，前两种和第三种情况下最大爆炸威力分别与 Motion 和 FallDistance 有关，而且有随机性。因为爆炸检查时普通矿车的基础运算已经完成，堆叠的矿车可以轻易地把最大爆炸威力提高到最大值 11.5，这时实际爆炸威力在 4～11.5 之间等可能地选取。

7.7 箭矢

可以说，研究箭矢的运动也许不是最有用的，但一定是研究人数最多的，估计几乎每个 Minecraft 玩家都或多或少地练习过箭矢的发射技术。

箭矢的常见获取方法有:

(1) 玩家发射。此时箭矢会在玩家的眼部坐标下方 0.1m 处生成，然后被赋予大小以 m/gt 为单位时数值为蓄力刻数（2-20 间一个整数）的 0.15 倍，方向朝向玩家视线方向的初始 Motion，最后再在各轴上分别（各轴上可以不相同）加上一个分布满足平均数为 0，标准差为初始 Motion 大小的 0.0075 倍的正态分布的随机 Motion 作为误差。正态分布意味着误差实际上有极其微小的概率远大于标准差，但是，因为浮点数表示数值下限不够小，使箭矢因误差反向发射仍是不可能的。最后，箭矢的 Motion 会被叠加上玩家的水平 Motion，如果玩家未着地还会叠加上玩家的 Y 轴 Motion。

(2) 生物发射。这种情况与玩家发射类似，不同的是箭矢的初始 Motion 大小总是 1.6m/gt，而且误差的标准差是随难度改变的，简单、普通和困难难度下分别为玩家的 10、6 和 2 倍。生物的箭矢发射方向与它们的视线方向无直接关联，应该是只与目标位置有关。这种发射不会叠加发射者的 Motion。

(3) 发射器发射。此时箭矢会在发射器所在方块网格中心沿发射器朝向偏移 0.7m 外的地方生成，初速度为 1.1m/gt，误差标准差为玩家的 6 倍。

箭矢的运动运算流程大致如下：（ArrowEntity.tick(),102-119、PersistentProjectileEntity.tick，123-249）：

1 实体基础运算，更新离开发射者状态

2 如果当前坐标位于当前坐标所在方块网格中方块碰撞箱内部（对于边界的判定似乎有方向性），判定为卡到了方块内部

3 如果被判定为卡到了方块内部且卡进去的方块被替换为其它方块，检查以坐标为中心棱长 0.12m 的立方体中是否没有任何方块和固体实体的

碰撞箱，如果真没有就将现有 Motion 各轴分量分别乘以一个[0,0.2)之间的数（各轴上可以不同）、重新认定没有卡到方块内部并结束运算，否则，若累积卡在方块中的刻数超过 1200 则移除自身并结束运算

4 将现有坐标加上现有 Motion 与 1gt 的乘积确定目标坐标

5 进行基于 raycast 的碰撞检查以检查目前坐标到目标坐标的路径上有没有发生碰撞，若有则进行对应处理

6 如果实体与水接触，受到 $0.4gt^{-1}$ 的流体阻力，否则受到 $0.01gt^{-1}$ 的空气阻力

7 如果没有 noGravity 标签，受到大小为 $0.04m/gt^2$ 的重力加速度

8 更新坐标和碰撞箱

9 检查方块网格碰撞（checkBlockCollision()）

10 如果箭矢为药箭且进入方块的连续时间超过 600gt 就清除效果

可以得出，**箭矢的运动运算顺序主要是 FDcDGPu**（流体->位移计算->阻力->重力->位置更新），可以当成是 **FMGD**。

在箭矢与方块发生碰撞时首先会触发方块被弹射物击中时的行为，如火矢箭点燃 TNT、激活标靶方块、击落紫颂花等，然后箭矢的暴击、穿透、从弩发射等属性会丢失，Motion 会被设为方向为从方块网格的西北偏下的顶点到击中方块前箭矢最后的位置的一个向量，并将坐标按与原向量相反的方向移动 0.05m，除非 Motion 为 0。等到运算进行到第 3 步，如果条件合适，箭矢就会以一个与撞击位置及撞击时 Motion 有关的，带有一定随机性且不超过现有 Motion 的 0.2 倍的新 Motion 继续运动。

如果箭矢撞击到了可以被该箭矢伤害的实体，实体会受到大小为箭矢合速度大小以 m/gt 为单位时的数值与 Power 标签的乘积的向上取整结果的伤害，在箭矢有暴击属性时，这一伤害会加上[1,原伤害一半+1]中一个整数，但不会超过 2147483647 点。如果箭矢有冲击属性且 Motion 不趋近于 0，被击中实体的 Y 轴 Motion 会被设为 0.1m/gt，水平 Motion 会沿箭矢水平 Motion 方向设为 0.6m/gt 与冲击等级的积。对于玩家射出的箭，Power 标签与弓的力量附魔有关，无力量附魔时为 2，有力量 I 时为 3，以后每级增加 0.5。箭矢伤害和速度相关意味着给箭足够大的速度就可以对被攻击者造成极高的伤害，这已经被应用于屠龙炮等设备，8.2.3 节对此略有说明。

箭矢每次进行基于 Entity.move()方法的移动后会进行一遍类似上面第 3 步的检查，只是在发现碰撞箱时不增加计时而已。

另外，很反常识，箭矢在熔岩中的阻力竟然比在水中小，因为箭矢只有在水中时才受到流体阻力，在熔岩中受到的阻力和空中相同。

虽然说这是箭矢的分析，但三叉戟的运动与之相比很相似，只是在水中的阻力系数也是 $0.01gt^{-1}$，而且不存在第 10 步。忠诚三叉戟返回时不再进行碰撞检查（noClip 设为 true）并在击中目标后$\frac{1}{0.05\text{ 附魔等级}}$gt内返回。

7.8 雪球

获取雪球实体的方法主要有：

(1) 使用雪球物品。这种情况与玩家使用弓发射箭矢类似，只是初始 Motion 大小为 1.5m/gt，随机加速度分布的标准差为 0.01125m/gt。
(2) 使用发射器发射。这种情况与发射器发射箭矢相同。
(3) 由雪傀儡投掷。这种情况与生物发射箭矢相似，只是误差的标准差恒为玩家的 12 倍。

雪球实体的运算过程大致是（ThrownEntity.tick()，48-97）：

1 实体基础运算和更新离开发射者状态
2 进行基于 raycast 的碰撞检查以确认目前坐标到目标坐标的路径上有没有发生碰撞
2.1 如果撞到的方块是下界传送门就计划下一刻进行下界传送门门相关运算时被传送，是折跃门方块就尝试立即被传送
2.2 否则进行撞到的方块或实体对应的运算
3 检查方块网格碰撞（checkBlockCollision()）
4 根据目前 Motion 确定移动的目标位置
5 如果实体与水接触，受到 $0.2gt^{-1}$ 的流体阻力，否则受到 $0.01gt^{-1}$ 的空气阻力
6 如果没有 noGravity 标签，受到大小为 $0.03m/gt^2$ 的重力加速度
7 更新坐标和碰撞箱

与箭矢类似，**雪球的运动运算顺序仍主要是 FDcDGPu**，在不考虑触发绊线勾、气泡柱加速等事件的情况下可以当成是 **FMGD**。

第 3.1 步中，虽然实体进行了额外检查，但是因为两种传送门都没有碰撞箱，在这里的 raycast 中无法被检测到，所以速度足够大时仍会错过传送门。说不定这个特性就是 Bugjump 修复 MC-73844 时修出来的。

由于雪球的坐标和碰撞箱是在最后更新的，所以在一些关于坐标的计算中需要将 n 值减去 1 后才可以代入公式使用。

在雪球自己击中（不含活塞推动击中）方块时会触发方块在弹射物击中时的行为并移除自身。在雪球击中实体时会对实体造成 0（一般实体）或 3 点（仅烈焰人）伤害并移除自身。0 点伤害除不会对被攻击实体的生命值造成影响外，其它性质与一般的攻击相同，如击退仍会发生、击中末影水晶时水晶仍会被破坏等。

不仅是雪球，玩家掷出的各种其它投掷物的运动机制都与此类似，但是附魔之瓶和药水的重力分别为 $0.07m/gt^2$ 和 $0.05m/gt^2$，初始 Motion 大小分别为 0.7m/gt 和 0.5m/gt，具体情况略有偏差。

7.9 恶魂火球

恶魂火球实体的获取方法应该只有恶魂发射。此时，游戏会选取会选取恶魂的碰撞箱中心上方 0.5m 的一个点，并沿恶魂的视线方向偏移 4m 得到起始点，并在这个点处生成一个火球，并设定它每 gt 受到一个从出发点指向目标实体的碰撞箱中心，大小为 $0.1m/gt^2$ 的加速度，没有初始 Motion。

恶魂火球的运算流程大致如下（ExplosiveEntity.tick()，62-97）：

1 进行实体基础运算，更新离开发射者状态
2 检查方块网格碰撞（checkBlockCollision()）
3 根据当前 Motion 确定位移
4 Motion 加上设定的加速度（由 Power 标签决定）
5 如果在水中，受到受到 $0.2gt^{-1}$ 的流体阻力，否则受到 $0.05gt^{-1}$ 的空气阻力
6 更新坐标和碰撞箱

由此得出，**恶魂火球的运算顺序为 FDcCDPu**,在总是接触到水或总是接触不到水的情况下可以当成是 **FMCD**。

通常都认为，恶魂火球是做匀速直线运动的，但其实不然，从火球加入到现在火球一直都不是匀速直线运动的。恶魂火球的初始 Motion 为 0，其速度是由其运算过程中的第 5、6 步决定的，在设定的恒定加速度和阻力的共同作用下，大小很快趋近于一个定值，而合加速度会以指数方式衰减，半衰期大约是 13.51gt（换句话说就是说恶魂火球在不到三秒的时间内就会加速到最大速度的 90%）而且因为不受重力，速度方向不变，所以看起来像匀速直线运动。

在火球被玩家击中时，它的 Motion 会被设为与攻击者（对于弹射物的攻击，攻击者就是发射者）视线方向同向，大小等于 1m/gt 的向量，加速度

会被重新设为沿攻击者视线方向，大小 0.1m/gt² 的一个向量，而且主人会被改为攻击者。也就是说，恶魂火球被打回去后的运动与被打回去前运动除打回去的一刻坐标相同外并无直接关联。

7.10 玩家

玩家在初次生成或死亡重生时没有初始 Motion，位置位于重生点附近的地面上；在下线后重新上线时会被生成在上次下线的地方，Motion 与上次下线时服务端玩家的 Motion 相同，除非某轴上 Motion 大于 10m/gt。

玩家的运动运算流程已经在第 6 节进行了说明，这里主要讲玩家的特殊运动机制。

众所周知，玩家的运动运算主要在客户端进行，只有在有必要时（活塞推动和触发反作弊等）才在服务端进行运算。

在客户端，玩家每秒的运算次数与帧率有关，每帧玩家最多运算 10gt 的运动，所以在帧率低于 2FPS 时玩家的运算就会被减缓，在帧率至少在 20FPS 以上才能保证玩家实体的正常运算。在客户端上玩家是随一般实体一并被运算的，顺序与服务端相似。

在服务端，玩家的运动是在每刻中所有维度的一般运算都完成后，自动保存和玩家输入处理前运算的，运算速率只与 TPS 相关。

如果某时玩家在两端中某一端运算，另一端一般不主动同步玩家的 Motion，只是将玩家以 Entity.move()移动到另一端指定的位置。例如，当客户端玩家在空中下落并尝试行走时，服务端玩家在计算的同时也发现玩家在下落，所以也有 Y 轴 Motion。但是，服务端并不去同步客户端上玩家控制造成的速度，只是按客户端决定的位移把玩家移动到对应的位置，如果移动是正确的就类似于 TP 到客户端的位置。

在客户端运算玩家的大部分运动有利于减少服务器 TPS 和延迟对操作体验的影响，也可能在某种程度上减少了服务器的负担，但是有部分人利用这一特性开发了一系列的外挂，常见的有飞行挂和速度挂。虽然 Minecraft 对玩家的输入进行了一定的检验（ServerPlayNetworkHandler.onPlayMove()，768-874），也就是反作弊，但这种检验主要只是防止玩家速度远大于服务端玩家 Motion、长时间非法浮空和非法穿墙等情况，并在前两种情况下将玩家 TP 回移动请求前玩家的坐标，但这并不足以防范大部分现有外挂。

关于客户端与服务端上的玩家运动情况，这两篇文章[16]也进行了一些分析。

玩家在运算时还会尝试主动拾起物品。具体来说，如果玩家在骑乘实体，与玩家的碰撞箱向四周扩大 1m 的范围相交的可拾起实体会被拾起；如果玩家没有在骑乘实体，与玩家的碰撞箱向四周扩大 1m 并向上下扩大 0.5m 的范围相交的可拾起实体会被拾起。

下面主要研究玩家在地面上时的直线向前运动情况。

玩家在进行 AI 加速时前向的和侧向速度系数只有 5 个值可取：没有按住对应移动控制键或按住对应相反的移动控制键（一般为 WS 或 AD）时为 0，按住对应的控制键且未潜行或使用物品时为±0.98，按住对应的控制键且在潜行或使用物品时为±0.294。根据附录 B 中式 B.5.5 可知，玩家的最终速度与加速度成正比，所以其它条件相同且只按下了前后或左右方向的控制键时，使用物品或潜行时的速度是正常情况的 0.3 倍。由于在各方向速度系数平方和大于 1 时会进行规格化，所以在正常行走时同时按住前后和左右方向的控制键时合速度不会明显加快，但是在物品或潜行时这样做会导致速度快$\sqrt{2}$倍。地面移动加速度以 m/gt^2 为单位的数值为 generic_movement_speed 属性值，玩家的基础值为 0.1。

可以计算出玩家行走时在普通方块地面上最终水平速度约为 4.317m/s，在普通冰面上的最终速度约为 4.242m/s，蓝冰上的最终速度约为 4.464m/s，粘液块上的最终速度约为 1.176m/s，空中的最终速度约为 4.356m/s。

由于在疾跑时跳跃会有一个向前的加速，所以跑跳速度会更快，地面滑度较高时会更明显。但因为在空中的速度慢于玩家在普通地面疾跑的速度（5.612m/s），所以应该存在一个临界初始跳跃速度，使得平均水平速度在跑跳后反而下降。

8 实体运动应用举例

只纸上谈兵还是不行的，下面给出几种大家可能比较熟悉的实体运动理论的应用。

8.1 改造一种 TNT 大炮

首先在这定义四种效率：

爆炸半径利用率：数值为每个爆炸爆炸中心与被推动实体坐标间距离与爆炸半径比值的平均数。

接触率利用率：每个爆炸对被推动实体的接触率的平均数

加速度利用率：需要的方向上的加速度分量与总加速度的比值

飞行时间利用率：只对TNT炮存在，值为$\frac{1-0.98^n}{1-0.98^{80}}$，其中n为TNT炮中炮弹开始飞行时的余剩引信时间，其它炮种取1即可。

爆炸利用率：上述四种利用率的乘积

相信图8.1.1中的TNT大炮相信大家基本都见过或做过，它建造与使用都很简单，但是它的效率低（不到30%），射程近（50m左右），而且落点不固定。这里就来试着对它魔改一番，使其做到高效，准确，且射程足够远。下面定个小目标：

(1) 爆炸利用率达到80%以上

(2) 多次发射落点不会出现任何偏差

(3) 可以打击至少10000m外的地面目标

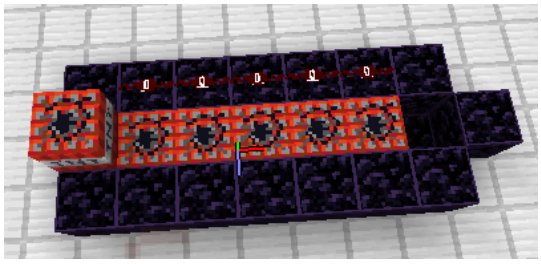

图 8.1.1 TNT 大炮

貌似有点难度，我们一步一步来。

首先看一下原设计，容易发现它有一部分TNT离炮弹较远，利用效率低，不妨给它们推过来。最容易想到的推进方法就是用水推动，在炮膛后面放一格水就好了（在Wiki上，这叫推进力集中器）。这里我们会使水源尽量靠近被发射的TNT，因为如果太长下游的流水就较浅，推力较小，就无法把这些

TNT都推过去，后果严重的话不排除炸膛。然后，很容易想到这里垫住被发射的TNT的方块太高了，阻挡住了大部分（对于最近的那个是2/3）的接触率判定射线，造成接触率利用率较低，这也是一个致命缺陷。另外，准确性在两边用方块挡住，并使用红石计时点燃炮弹后就可以大幅提高了。于是就有了下面的一个新设计，因为有水流保护，爆炸炸不坏，这里就不用黑曜石那么令人瞎眼的方块了。

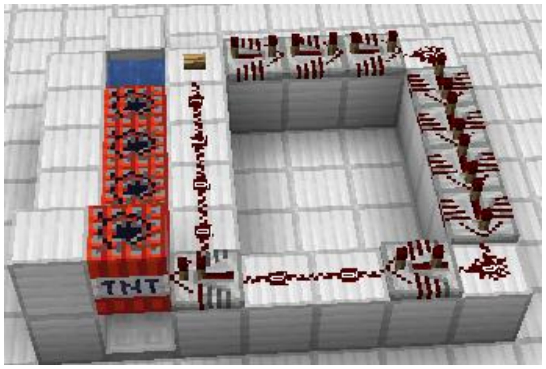

图 8.1.2 新方案(1)

测试一下，方块似乎垫得太低了，这些TNT根本打不起来炮弹，所以，再改。我们发现，直接摆放TNT堆不了多少，所以在新方案中TNT源被换成了发射器。

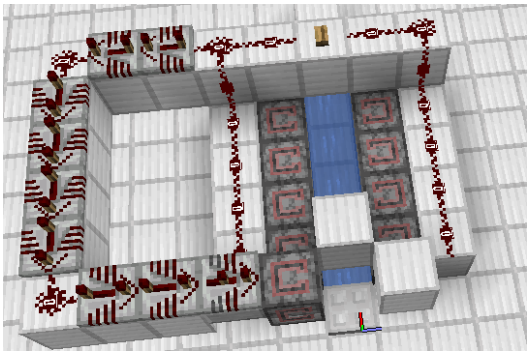

图 8.1.3 新方案(2)

终于有一个能用的了，这次射程大约是在177m左右，效率差不多是在54.7%左右，偏移（由TNT碰撞箱略小于1m造成）已经不到一米了。但是，这离我们的目标还有较大距离。

首先看射程。10000m的射程意味着什么呢？由初始Motion公式可以得出，即使飞行时间达到了80gt（理论最大），初始水平Motion大小也要达到

249.58m/gt左右，这意味着即使爆炸率利用率为100%（尽管不可能），也至少需要250个TNT作为推进。

同时，我们是要使TNT落地的，这需要Y轴的初始Motion恰到好处（大约1.9m/gt左右），假设TNT在被推进时的Y轴Motion为0，爆炸点与炮弹的水平距离为1m，使用“万能方程”计算得出，即使飞行时间为80gt，推进时炮弹坐标与爆炸中心的高度差也需要低至0.008147m，这要比所有的方块高度都低，所以暂时不考虑使用方块垫住炮弹了，得让炮弹悬空（悬空推进TNT并不靠谱，因为它们在移动后就会发生爆炸把余下的推进TNT炸飞，可能导致炸膛）。很幸运，经过寻找，我们发现在TNT无初速下落2gt后会达到一个恰到好处的点，这时TNT的坐标仅比某一个像素的顶部高度高0.0058m，也许可以试一下。再次寻找一段时间后，我们可以发现，如果让炮弹从一个高度0.9375m（15px）的方块（如蜂蜜块）上无初速落下，2gt后让推进TNT在附近的一个高度为0.75m（12px）且正好能防爆的方块（如附魔台）上爆炸，这时炮弹坐标与爆炸中心的高度差仅有0.00705m，如果TNT飞行时间足够，这个高度差应该是够小了。

在让被推进的TNT悬空的同时，效率的问题已基本解决了。现在，原先拉低效率最厉害的接触率利用率由于没有方块阻挡已经达到100%，而爆炸半径利用率也早已达到了87.2%左右，剩下就看飞行时间利用率了。

最后是归正。如果炮弹和推进TNT都被完全归正，即Motion和坐标都是确定的了，这个问题也就解决了。经过一段时间的设计，一个炮弹矫正装置设计产生了。

运行时序如下表：

表 8.1 运行时序表

刻数	阶段	事件
1	NTUBEEUTE	发射器激活，炮弹被发射 推动炮弹的活塞开始伸出，B36被创建 用于调整TNT初始Motion的TNT爆炸 炮弹更新，向一个死角移动，但被阻挡，失去各轴Motion B36更新，推动炮弹
2	E	炮弹再次更新，更新后引信时间还剩78gt

	U TE	B36再次更新，炮弹离开蜂蜜块上方，水平方向上到位
3	E U TE	炮弹再次更新，下落到距网格底部0.8975m处，引信时间还剩77gt B36到位并被移除
4	E U	炮弹再次更新，下落到距网格底部0.8183m处，引信时间还剩76gt
5	E U	主要的推进TNT爆炸，加速炮弹 炮弹再次更新，开始飞行，此后总飞行时间76gt

不过由于技术及时间原因，到现在还是没有搭建出那种时序比较紧凑的设计，存档中搭建的原理与此类似，但是炮弹飞行时间不是76gt而是74gt。

可以计算出，炮弹被推进时的Y轴Motion是0.0392m/gt，接着由“万能方程组”计算出射程约为10425m，已经达到目标了。再然后计算出所需初始水平Motion约为268.7716m/gt，进而计算出所需TNT数量约为308个。经过测试该TNT炮的实际射程为10453m，比理论值较大的原因是TNT数量是尽可能多取的。再看效率，现在已经达到了84.414%，过关。至于归正，炮弹归正已经做好了，推进TNT源及归正可以使用8.2.2节中提到的320TNT归中阵列，位置一定时也是不存在随机性的（浮点数误差造成的随机性也可以通过优化底层归中消除），同样过关。

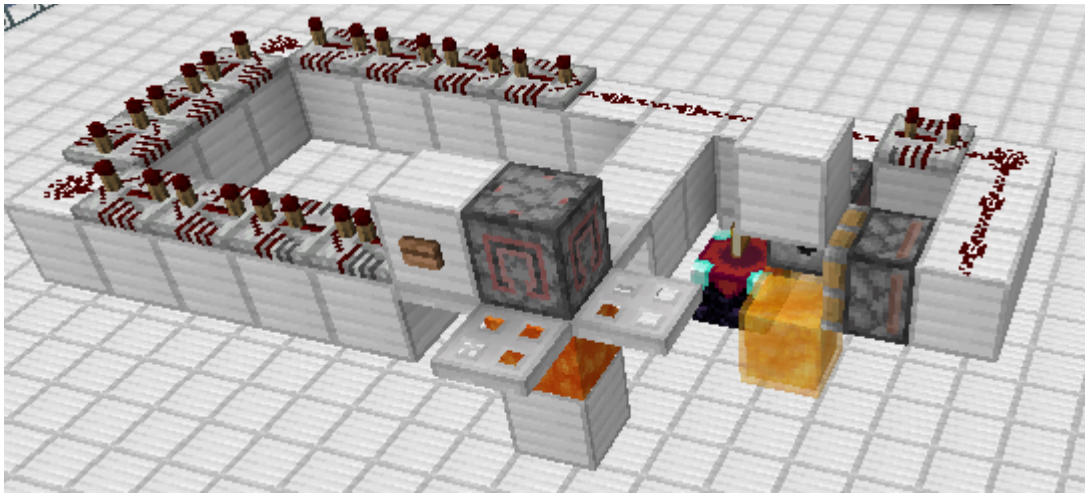

图 8.1.4 新方案(3)

然而，这远不是最优解。可以回想一下，TNT的爆炸高度是TNT坐标上方0.06125m，与地毯的高度只差了0.00125m。利用这个高度差，可以设计出一种推进TNT和炮弹距离1m射程达到60000m以上的TNT炮。

实际上，我们也可以跳出这个思维定式，不再追求多小的高度差，而是在水平和竖直方向上分别加速，这时大炮的射程是可以接近无限的。

无限射程的TNT炮种也有很多变种，这这些TNT炮通常使用TNT矿车作为推进、在弱加载区块堆积TNT，推进弱加载区块的TNT或利用末地传送门不强加载区块的特性在末地堆叠TNT来获取足够的推力。换句话说，只要电脑性能够好且实体不会导致区块过大（1.15后限制为4GB）理论上就能无限地获取推力。如不采用在水平和竖直方向上分别加速的方案，这种TNT炮炮弹的Y轴Motion通常会很大，此时可以设方块阻挡炮弹或用活塞推粘液块弹射炮弹来减少Y轴Motion来防止这种大炮成为“一飞冲天炮”。

目前有一种被称为“电磁炮”的较新炮种，通过发射一系列激活时序不同（可以在同一刻不同顺序激活，也可以在多个游戏刻内激活）的TNT，使靠前的TNT被后面的TNT的爆炸多次推动（可以是加速，也可以减速），最终得到一系列在一条直线上但位置不同的爆炸。

8.2 科学地使用弹射物

8.2.1 珍珠投掷技术

比如说，投掷末影珍珠时我们经常会希望将其尽可能扔到远处，我们可能会套用课本猜测末影珍珠的最佳投射角度是 45 度，但这在 Minecraft 中错误的。因为，珍珠在受到重力的同时还会受到一个阻力，导致其运动轨迹并不是抛物线（解析式已经在第三节给出）。事实上，根据程序计算，在珍珠落点高度与掷出点高度差忽略不计时掷出距离和俯仰角的关系如图 8.2.1。

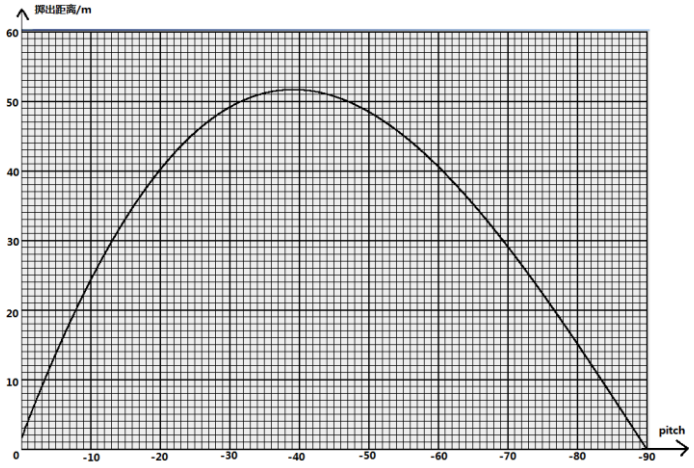

图 8.2.1 掷出距离和俯仰角的关系

可以看出，在俯仰角约-39°时掷出距离达到最大，大约是 53m。

但是这一最佳抛出角度不是不变的，在目标点高度增大时或初始 Motion 增大时（尽管对于投掷珍珠这是不会考虑的），珍珠的最佳掷出角也会减小，也就是说得把头抬得更高。

不过，最靠谱的办法还是用运动机制与末影珍珠一致的雪球和鸡蛋进行初步试验，基本能保证 90%保险，但并非 100%。因为珍珠运动是离散的且珍珠在自己移动的过程中发生碰撞时会将玩家传送到碰撞前最后一 gt 的坐标处而不是碰撞点处，所以即使看到测试实体勉强安全落地了珍珠也有可能把玩家传送到空中，而且真正投掷珍珠时也可能因初始 Motion 的随机性出现问题。

8.2.2 珍珠炮

在一些大型服务器中，可能会因为交通问题而修建珍珠炮。珍珠炮的那个规模乍一看挺吓人，但是如果只看它的原理就会觉得简单很多了。不过，这里并不是讲解图中那样一个简单到没法用的珍珠炮，这里将以一个发射角范围可以达到 360 度，最大 TNT 量为 1280（应该还可以加），最远射程约 14000m（接近目前珍珠高度选取的极限了）的珍珠炮为例对珍珠炮的基本原理进行简单的讲解。存档中还有一个发射角范围约为 53 度，最大 TNT 量为 120，最大射程为 8800m 左右的微型矢量珍珠炮，大家也可以自行研究。

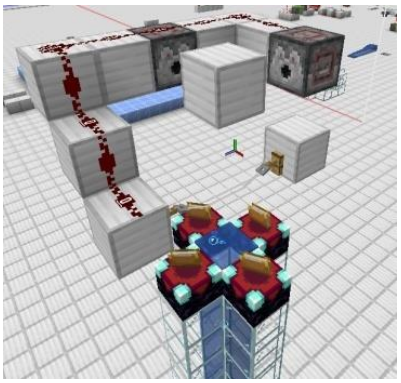

图 8.2.2.1 一个过于简易的珍珠炮

个人认为珍珠炮最关键的部分有四个：珍珠矫正，TNT 复制及归中、区块加载以及控制部分。其中，由于控制部分无关实体运动的技术太多，在此从略，而区块加载只简单提及。

末影珍珠在被投出时除了玩家视线方向的一个初始 Motion 外，还会被叠加上一个随机的 Motion（详见 7.8），这会使珍珠的运动无法被准确地预

知，从而导致其在被 TNT 加速后的方向无法完全确定，造成的落点误差可能会达到数千米。所以，对珍珠进行矫正很有必要。

末影珍珠的水平方向矫正的根本原理是末影珍珠在自己移动时和被活塞推动时的移动方式不同。在末影珍珠被活塞推动时是进行的基于Entity.move()方法的移动，这时的碰撞检查检查的是末影珍珠的碰撞箱有没有与外界发生了碰撞，并且在末影珍珠发生碰撞时不仅不会将其移除并传送玩家，还会将它发生碰撞的轴上的 Motion 归零。而在末影珍珠自己移动时碰撞检查是检查的坐标点的碰撞，这时发生碰撞才会将珍珠移除。也就是说，只要珍珠的坐标点连线不与方块和实体发生碰撞（也就是对坐标点上的连线进行 raycast 的过程中不被阻挡）就可以。珍珠水平方向上的矫正一般是选取一些碰撞箱刚好的方块推动珍珠，使其撞到另一个碰撞箱刚好的方块上，并在该过程中保证珍珠的坐标点不会卡到方块或实体中。

末影珍珠在竖直方向上的矫正的原理通常是粘液块弹射是直接设置 Motion 的，而且也有类似普通方块 B36 的直接推动作用，在不至于造成单轴活塞推动位移限制时推动后珍珠 Y 坐标确定在粘液块 B36 碰撞箱上方 0.01m 处。在末影珍珠被粘液块弹起 62gt 后珍珠就会返回，这些时间是不足够等待 TNT 爆炸的，需要把珍珠重新弹起才可以。需要注意，如果珍珠开始被推动时坐标过低（低于粘液块 B36 碰撞箱顶部 0.5m 或更多），因为单轴活塞推动位移限制，珍珠的实际位移不会超过 0.51m 而无法消除初始位移的随机性（有时也会利用这个特性保留位置的多样性）。

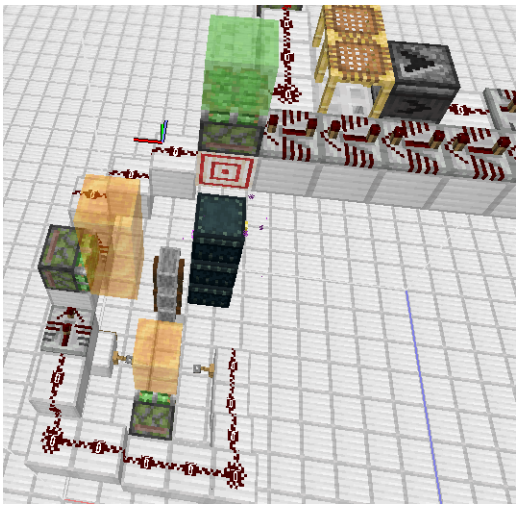

图 8.2.2.2 珍珠矫正装置

如图，这是一种适配 1.16.2+版本（稍作修改后应该可以适配 1.14+）的珍珠矫正装置。在珍珠被投出后，绊线勾被触发，然后珍珠被推到砂轮的那

个轮子上，Z 轴随机性被消除，坐标正好被卡在两个方块网格的交界处，Motion 为 0。再然后，由于珍珠的碰撞箱仍有一部分位于蜜块正前方，末影珍珠又被推到了末影箱侧面，X 轴的随机性也随之消除。再然后，在珍珠通过标靶和活塞时，由于其坐标点路径是在它们的东北棱上，也就是说没有位于它们的方块网格内部，所以可以通过。最后，活塞推动粘液块弹起珍珠，将其 Y 轴 Motion 设为 1m/gt，并将其移动到粘液块 B36 碰撞箱上方 0.01m 处，Y 轴随机性也被消除。至此，矫正完成。其中，Y 轴矫正大家可以试着分析一下，说不定会有新发现。

在 1.16.1 及更早版本，由于投掷物自主运动时不会检查方块网格碰撞，这种珍珠矫正中的绊线不会被触发。所以，需要在一个可以促使绊线检查自己的激活状态而自身又不会激活绊线的实体作为辅助。如图 8.2.2.3，为了解决这个问题，通常可以放置一个会检查方块网格碰撞而且与珍珠将要触发的绊线所在方块网格相交但不与该绊线的检查范围（方块网格最下方 0.15625m）相交的实体。为了装置的稳定，这一实体一般会选取盔甲架或船等难以推动的非生物实体。在 1.16.1 及之前的版本中，因为投掷物对方块的碰撞判定是根据 Outline Shape（准星对准时显示的黑框）进行的，投掷物会撞到在准星对准时会显示黑框但实际上无碰撞箱的方块（包括绊线，还有草和蛛网等），所以需要卡准珍珠的路径，使得珍珠某一时刻能与绊线的检查范围相交且坐标点路径不通过绊线。也是因为这个，投掷物不会检查 36 号方块的碰撞，所以可以做出另一种珍珠矫正，此处不再演示。

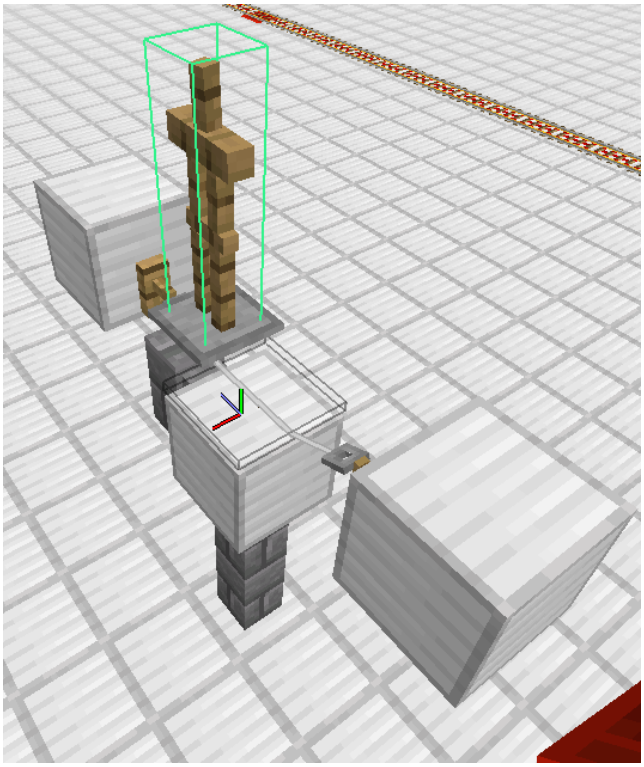

图 8.2.2.3 1.16.1-适配的珍珠检测装置

只有如果只有矫正，珍珠炮无非就是帮玩家卡到整数坐标的一个沙雕红石装置，甚至都不可以说是“炮”了。给珍珠极大的速度的奥秘就隐藏在 TNT 复制归中装置中。在 TNT 爆炸时，每个 TNT 最大能给珍珠加上 0.90125m/gt 的初始 Motion（不是 1m/gt 的原因是珍珠可以撞上 0.3m 以内的 TNT 并传送玩家，被传到那鬼地方可不好玩）。

现在流行的大型 TNT 复制机单元的基本原理是利用红石线指向状态改变不造成方块更新制造 BUD 状态的 TNT，在活塞开始推动的瞬间将它们复制一份。刚复制好的 TNT 是零散分布的，我们要使它们集中到同一个地方。几个十几个 TNT 归中很容易，但珍珠炮上归中成千上万个 TNT 就不见得那么容易了。这个可能没有很一般的方法，不过，基本原理都还是近似的：首先通常会在 TNT 下方放置栅栏门将结合上面 TNT 的 B36 的推动作用被复制的 TNT 两两地合并，然后通常会用活塞推动，粘液块弹射（在远距离归中时还会用到爆炸推进）等方式结合合适的阻挡使其一步一步地合并成一大堆 TNT。一个具体些的方案可以在存档中找到。

目前较为流行的TNT归中的具体方案主要的原理是利用粘液块弹射给每个TNT一个分别沿X、Z轴方向的Motion并使TNT自然下落并落地，期间TNT在移动的过程中撞到方块归正坐标并归零Motion。由于TNT只在侧面撞

到方块时不会发生摩擦减速，所以正常情况下这些TNT最终会挤在某个墙角处，水平方向上位置确定，Motion固定为0。又因为各TNT的Y轴上受到的弹射、阻挡因素影响的程度都相同，初始Motion又没有随机性，归正后Y轴上的位置和Motion也是确定的。

举个比较简单的例子，图8.2.2.4中是一个10TNT复制归中阵列，魔改自星芒本尊视频中的复制阵列[17]，弹射部分灵感来自LgYn小凉音[18]，工作流程大致如下，其中各TNT从左到右编号0-9：

(1) 在激活侦测器A后，最上方粘性活塞会拉回红石块，导致红石粉指向改变BUD充能TNT，然后活塞推动TNT使之受到更新被点燃同时创建其B36，完成复制过程。
(2) 由于活塞B和被推动的TNT的推动作用，TNT实体获得一个向左的位移趋势，但除2、4、7、9号TNT外都被栅栏门的上半部分阻挡。在推动完成后，0号和5号TNT仍在红框中的粘液块上方，其它TNT都正常地移动到了打开的栅栏门上方。因为与TNT碰撞箱相交的方块不会被检查，那些栅栏门在Y轴上不阻挡TNT，所有TNT下落的高度都一样。
(3) 由于被推动的TNT的推动作用，0和5号TNT也被移至打开的栅栏门上方，其它TNT因为关闭的栅栏门的阻挡移动未能成功。
(4) 在TNT有部分碰撞箱位于蓝框标注的粘液块所在的那一层时，TNT第一次被粘液块弹射，获得垂直纸面向外的Motion。
(5) 在TNT通过打开的栅栏门完全落出红框中粘液块所在的那一层之前最后一gt，TNT第二次被粘液块弹射，获得了一个沿Z轴的Motion。
(6) 在被弹射后下一gt，TNT发生移动，由于TNT先在Y轴上移动，所以在TNT进行水平移动前TNT已经完全落出红框中粘液块所在的那一层，不会撞到右边的粘液块。
(7) 所有TNT依次撞上两列白色玻璃，X、Z轴上位置确定，Motion归零。
(8) 所有TNT落到附魔台上，等待爆炸。

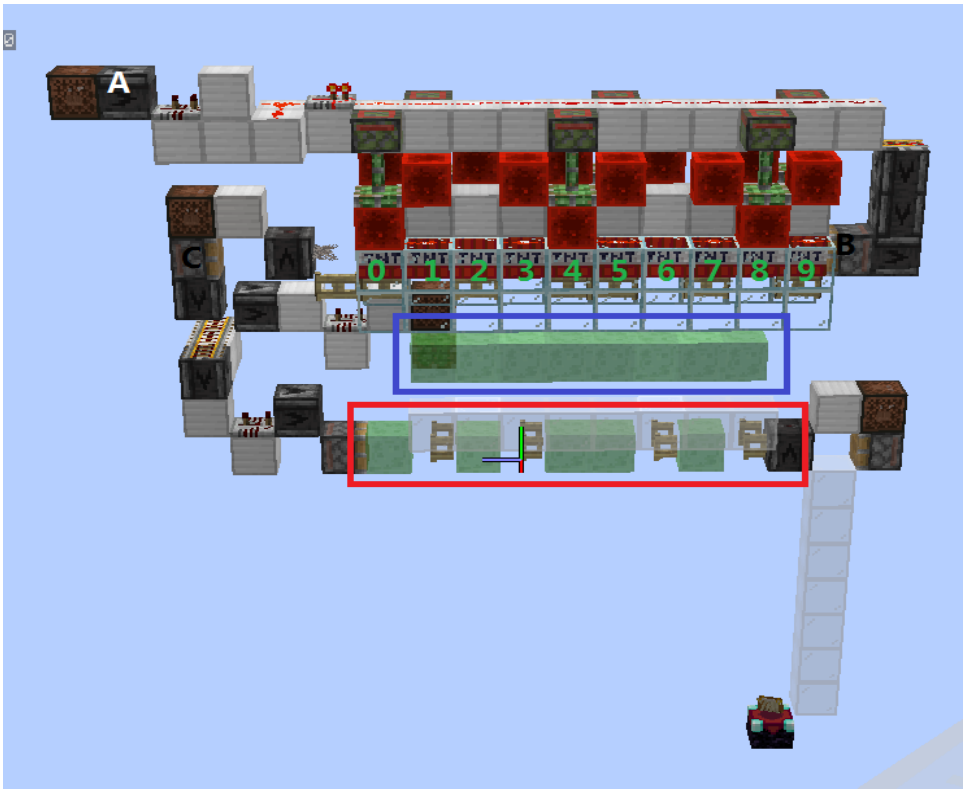

图8.2.2.4 TNT归中阵列的一个单元

照目前的测试，这一TNT复制归中阵列是有概率炸膛的，有些TNT会卡在蓝框中的粘液块上方。原因应该是TNT实体生成时朝向玻璃的初始Motion过小，在TNT落出玻璃所在高度范围后仍未撞到玻璃消除该方向上的Motion，然后继续运动至下排粘液块上方无法继续下落。不过解决起来也很简单，在玻璃下方继续摆放树叶等不被粘动的方块就可以了，图中为了使结构更清晰略去。

为了更进一步地压缩复制归中阵列，提高效率，目前几乎所有流行的TNT复制归中阵列都会采用将2x2个除方向外基本相同的TNT复制阵列的部分组成一个较大阵列的方案，四个部分复制的TNT会被初步归中到以一个固定位置为中心的3x3网格中的四个角，然后再用活塞推动和粘液块弹射等手段将它们最终归中到同一位置。存档中给出了这一类TNT归中阵列的一个较为实用的例子，可以一次性复制320个TNT并归中至同一位置。如果不计浮点数误差，这些TNT的位置应该是近似相同的，但是，在第二次归中，也就是归中那四小堆TNT的过程中，由于TNT是从两个不同的位置被推过来的，且运算中有一些值不在浮点数集内，浮点数误差是有可能发生的，而且爆炸加速实体只需要各轴坐标差平方和的算术平方根为零就会进行，这意味着部分TNT在浮点数误差较大时（如坐标绝对值很大或接近2的整数次幂时）可

能会被炸飞导致炸膛。不过目前来看，这些TNT各轴坐标发现的最大的差值只在10^{-8}m的数量级，TNT的Motion变化量大小理论上也不过是在10^{-7}m/gt左右，不会引起炸膛，但可能会产生推进方向的一点与位置相关的随机性。如果真的要解决这个问题，可以改进第二次归中装着让TNT发生两次碰撞，但是这会在一定程度上缩短归中后对TNT进一步处理的时间。

只用这样的归中阵列对于大型珍珠炮确实是不足的，按照粗略计算，使用存档中的TNT阵列结构，即使让TNT归正完成后立即爆炸，上限也只有600TNT左右。

TNT飞行途中多次弹射可以有效增加单阵列TNT数量。可以证明，在B36的直接推动作用可以忽略不计时，多次弹射的弹射时间间隔相同时弹射距离达到最远。也可以推导出多次弹射TNT的最远位移公式和最短耗时公式，式中m为弹射次数，k=0.98，$v_0 = 1\mathrm{m/gt}$，$t_0 = 1\mathrm{gt}$：

$$x = \frac{mv_0t_0(1-k^{\frac{n_0}{m}})}{1-k} \#(8.2.2.1)$$

$$n = mlog_k\left[1-\frac{x(1-k)}{mv_0t_0}\right] \#(8.2.2.2)$$

但这仍无法满足大型珍珠炮的需求。所以，大型珍珠炮中经常会使用多阵列复制TNT，并使TNT在另外一些TNT的爆炸推动下向中间靠拢，最终同时引爆。

不过实际上，我们需要复制两堆这样的 TNT，因为通过控制两堆 TNT 的数量（通过按一定规律开启或关闭复制单元）及位置（通过使用打角器），我们可以控制发射的珍珠的初始 Motion 大小及方向，进而控制珍珠的落点。利用矢量分解，可以将珍珠的初始水平 Motion 分解到两个水平方向上，从而比较轻松地求得获取需要的速度需要的 TNT 数量。一般来说，我们会希望这两个水平方向是垂直的，这样运算就会简便很多。公式形式与珍珠炮的具体设计有关，且推导较为简单，这里暂时从略。

除此之外，与 TNT 炮类似，珍珠炮的 TNT 爆炸中心与珍珠眼部坐标的高度差需要尽可能小，过大的 Y 轴 Motion 会使珍珠在高空中长时间下不来，从而延长珍珠的到站时间，降低平均速度。目前使用起来比较方便的几个可以达到平射的珍珠 Y 坐标的小数部分有 0.34875m、0.59875m、0.78625m 和 0.84875m，最常用的是 0.34875m，可以通过使 TNT 在 0.5 或 1.5m 高的平台上爆炸，并利用弹射时珍珠坐标较低时不消除位置多样性但可以加速珍珠的特性二次弹射珍珠接近，此时珍珠 Y 坐标的小数部分为 0.3456984188109，珍珠的眼部坐标与爆炸中心的高度差约为-0.003051m（不计浮点误差）。因

为推进方向是略偏向下的，所以这一高度下的射程是受限的，虽然可以通过将第二次弹射提前 1gt 将珍珠 Y 坐标的小数部分卡到 0.35776585772253 并适当调整爆炸时间抛射珍珠解决，但这样就会大幅增加到站所需时间和常加载的区块数量…一个两难的选择。

珍珠炮主体造好了，拿着珍珠开到满档立马就去测试，结果发现珍珠投出去老半天了都没有反应，这是因为珍珠飞到了弱加载或者边界加载甚至暂时无法访问的区块中，导致其运算暂停无法继续飞行。需要强加载珍珠坐标所能到达的所有区块，保证珍珠一直被运算。在 1.14 之后，由于瞬时区块加载线失效，为珍珠炮加载的区块需要一直被加载，所以我们会希望使珍珠运动的飞行时长尽可能小，也就是速度尽可能快，从而避免一直被加载的区块过多造成永久卡服。

现在珍珠的落点是地面上，而且有很多地点因为珍珠运动是离散的且珍珠在自己移动的过程中发生碰撞时会将玩家传送到碰撞前最后一 gt 的坐标处而不是碰撞点处不能直接到达。为了解决这个问题，可以通过计算等比调整两堆 TNT 个数直到珍珠有一 gt 位于目标位置附近（当然也可以适度调整目标位置）。

由于篇幅和技术有限，这里只能给出一些关于珍珠矫正、TNT 归中和区块加载的一些非常基础的知识，如果是真的希望建造一个大型矢量珍珠炮，可以去观看 LgYn 小凉音发布的一系列教程视频（更新中）[19]。

最后，说明一点，珍珠炮虽然被称为珍珠炮，利用它发射雪球和鸡蛋也是可行的，因为它们的初始 Motion 确定方式和运动机制与珍珠是完全相同的。实际上，在测试珍珠炮时为了避免被反复传送，雪球还是用得比较多的。

8.2.3 杀凋大炮

前面提到了，箭矢的伤害与速度有关，所以我们给箭矢足够的速度，理论上它的伤害是完全可以秒杀末影龙和凋零的。这里以一个 80TNT，配合力量 V 能造成 344 点伤害，凑合着能秒杀凋零的炮为例来对这一炮种进行简要介绍。需要知道，由于末影龙会削减大部分伤害，这个炮每箭只能给它造成 87 点伤害，所以并不足以屠龙。

这种炮最关键的就是箭矢矫正和 TNT 复制归中，由于后者在前一节已经进行简要说明，此处不再重复。

箭矢矫正有一种方案与珍珠矫正类似，可以保留箭矢的暴击和穿透等属性，但这种方案更适合水平发射高速箭矢，存档中已给出，这里从略。

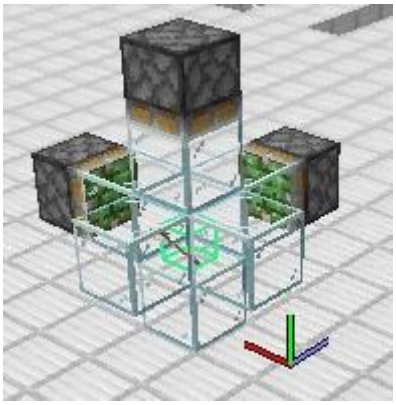

图 8.2.3 箭矢矫正方案

如图 8.2.3，存档中采用的箭矢矫正方案是来回向方块上推动箭矢的方案。通过轮流激活三个粘性活塞让玻璃把箭矢往另一边的玻璃上推动，使箭矢在玻璃上发生碰撞，Motion 被真正归零（需要注意卡到方块中的箭矢 Motion 不一定为 0）。这种矫正由于箭矢撞到了方块，所以不能保留箭矢的暴击及穿透属性，但这不是什么大问题，毕竟暴击本来就是不稳定的，不能光指望暴击提伤害，对吧。注意箭矢要尽可能在最下面的玻璃顶面的中心，也就是说不能使箭矢碰撞箱卡到方块中，否则矫正会失败。

等到矫正完毕后，TNT 即将爆炸时将箭矢推到 TNT 附近，使爆炸加速箭矢完成一次开炮。

因为三叉戟与箭矢运动机制相似，这种箭矢炮也可以发射三叉戟。

8.3 解释一些奇葩现象

Bugjump 的 Bug 和特性（有用叫特性，没用叫 Bug）实在是太多了，在此举三个例子。

第一个：请试着这样做：在地面上放置一个潜影盒，然后在上面放置一个盔甲架，再然后再盔甲架上方放一个蜘蛛网，最后打开潜影盒，你会发现盔甲架似乎陷下去了。没错，它确实陷下去了。因为在潜影盒推动盔甲架时，其位移会因蜘蛛网减到原位移的 1/20，而潜影盒的打开却不受影响，从而导致盔甲架卡到潜影盒内部。又因为基于 Entity.move()方法移动的实体时不检查实体卡进去的方块的碰撞，而盔甲架的移动就是基于 Entity.move()方法的，所以它不检查那个潜影盒的碰撞，它就陷下去了。

图 8.3TNT 销毁核心部件

实际上，如果把潜影盒换成被活塞推动的方块，把盔甲架换成任何基于Entity.move()方法移动且可以被推动的实体这个特性应该都是可以复现的。所以，就有了这样一个 TNT 销毁机的核心部件（很沙雕）：

第二个：在一片足够大的蓝冰上放一只船，按住 A 或 D 使船的角速度增到足够大，然后突然下船，让一个能上船的生物上船，那么那个生物的头就会一直转圈。这个特性我第一次是在 EDDxample 的一个视频[14]上见到的，觉得很有趣。至于原因，还记得第 7.5 节提到的船的角速度同步问题吗？就是这个导致的。

在玩家下船后，船的角速度还被保留着，但偏航角已经与服务端同步了。船更新骑乘者的偏航角时是直接按船的角速度而非船的真实朝向进行的，所以就有了那样的一个效果。

第三个：放置一个开启的循环命令方块，设置命令为/execute at @a run summon tnt 使其在玩家坐标处每 gt 生成一个 TNT，然后玩家快速地向上飞行，通过测试最终速度约在 45.9m/gt 左右浮动。但是请想一下，玩家实体被加载的时间总是早于 TNT 的，所以玩家的运算按一般实体的理论理应早于 TNT，又因为 TNT 生成的坐标是计划刻阶段（命令方块执行所在阶段）玩家的坐标，根据 TNT 的运算顺序，TNT 的爆炸中心坐标为生成坐标上方 0.02125（即-0.04+0.06125）m（即-0.04m+0.06125m）处，但是玩家如果真的有这么大的速度，它在运算时应该已经移动到 45.9m 之外了，不会被爆炸影响，这就出现问题了。

实际上，如果不炸玩家，炸其它实体（TNT 和不受爆炸变速的实体除外），你会发现虽然被炸的实体仍总是快速地向高空飞行，但它们的最终速度会在不到 8m/gt 的一个值（是可以求解的，这里从略）附近浮动而达不到

玩家的速度。这就让我们开始考虑这一问题是否是玩家实体运算的特殊性导致的。

如果看得仔细些就会发现前面提到过服务端玩家实体的运动运算是在各维度的一般运算全部完成后才进行的，这样问题似乎就能解释了，因为爆炸发生时时玩家还未被运算，坐标还在 TNT 坐标附近，可以被爆炸赋予较大加速度。综合 TNT 爆炸前的运动、玩家的重力加速度和玩家的空气阻力系数，可以求解玩家的理论最终速度约为 45.99625m/gt，与实验值接近。

参考文献

[1] https://minecraft.fandom.com/zh/wiki/%E5%AE%9E%E4%BD%93
[2] https://t.bilibili.com/447368865911693871?tab=2
[3] https://www.bilibili.com/video/BV1Gv411t7sb?p=4
[4] https://www.bilibili.com/video/BV1jb411V7tG
[5] https://www.bilibili.com/read/cv6074690
[6] https://www.bilibili.com/video/BV1tx41117TX
[7] https://www.bilibili.com/video/BV1ea4y1s7JC
[8] https://www.bilibili.com/video/BV1Ez411v7iY
[9] https://www.bilibili.com/read/cv12166270
[10] https://www.mcpk.wiki/wiki/Main_Page
[11] https://minecraft.fandom.com/zh/wiki/%E5%B1%9E%E6%80%A7
[12] https://www.bilibili.com/read/cv2532349
https://www.bilibili.com/read/cv3728017
[13] https://www.bilibili.com/video/BV1RB4y1P758
[14]]https://www.bilibili.com/video/BV16t41147EM
[15] https://github.com/Xiaoyuan-xyz/The-Principle-of-Redstone-Circuits/blob/master/%E9%93%81%E8%BD%A8%E5%8F%8A%E7%9F%BF%E8%BD%A6%E7%B3%BB%E7%BB%9F.md
http://www.rmcteam.org/machinery-circiut/railway/model_rail_i_perfect.html
http://www.rmcteam.org/machinery-circiut/railway/model_rail_ii_perfect.html
http://www.rmcteam.org/machinery-circiut/railway/cart-experiment.html
http://www.rmcteam.org/machinery-circiut/railway/railay_system_basics.html
等
[16] https://www.bilibili.com/read/cv7209627
https://www.bilibili.com/read/cv7593366
[17] https://www.bilibili.com/video/BV1o5411t7WL
[18] https://www.bilibili.com/video/BV1sV411r7Tt
[19] https://www.bilibili.com/video/BV1Mb4y1y7it

附录 A 常见实体运动属性

下面总结一些常见的实体的运动相关属性，其中运算顺序含义见 3.1 节，A 为重力之外的某种加速。为了方便实际应用，这里给出的是实体的速度乘数，也就是公式中用到的“k”，阻力系数并未直接给出。

表 A1 常见实体运动属性表

实体	重力 /(m/gt²)	速度乘数（k 值）	高度 /m	宽度 /m	眼部高度/m	运算顺序
TNT	0.04	0.98	0.98	0.98	0.15	GMD
下落的方块	0.04	0.98	0.98	0.98	0.833	GMD
空中的物品	0.04	0.98(1)	0.25	0.25	0.2125	GMD
空中的经验球	0.03	0.98(2)	0.5	0.5	0.425	GMD
空中的船	0.04	水平 0.9，Y 轴 1	0.5625	1.375	0.5625	GDM
空中的矿车	0.04	0.95	0.7	0.98	0.595	GMD
空中的箭矢/三叉戟	0.05(3)	0.99(3)	0.5	0.5	0.13	MDG
空中的雪球/鸡蛋/末影珍珠	0.03(3)	0.99(3)	0.25	0.25	0.2125	MDG
空中的药水瓶	0.05(3)	0.99(3)	0.25	0.25	0.2125	MDG
空中的附魔之瓶	0.07(3)	0.99(3)	0.25	0.25	0.2125	MDG
空中的恶魂和末影龙火球	0	0.95(3)	1	1	0.85	MAD
空中的烈焰人火球/普通凋零之首	0	0.95(3)	0.3125	0.3125	0.265625	MAD
空中的鱼竿浮标	0.03	0.92	0.25	0.25	0.2125	GMD
空中的蓝色凋零之首	0	0.73(3)	0.3125	0.3125	0.265625	MAD
空中的羊驼唾沫	0.06(3)	0.99(3)	0.25	0.25	0.2125	MDG

无有效目标的潜影贝导弹	0.04	1	0.3125	0.3125	0.265625	GM

（续表）

实体	重力/(m/gt²)	速度乘数（k 值）	高度/m	宽度/m	眼部高度/m	运算顺序
空中或地面上的玩家（除体积和眼部高度外也适用于大部分生物和盔甲架）	0.08	水平 0.91(3)，Y 轴 0.98(3)	站立时 1.8，睡觉和濒死时 0.2，鞘翅飞行和游泳和激流飞行时 0.6，潜行时 1.5	站立、潜行、鞘翅飞行和游泳和激流飞行时 0.6，睡觉和濒死时 0.2	睡觉时为 0.2，游泳、鞘翅飞行和激流飞行时为 0.4，潜行时为 1.27，站立时为 1.62	MGD
水中的玩家（除体积和眼部高度外也适用于大部分生物和盔甲架）	0.005	水平取决于深海探索者等级，Y 轴 0.8(3)	同上	同上	同上	MDG
较深熔岩中的玩家（除体积和眼部高度外也适用于大部分生物和盔甲架）	0.005	0.5	同上	同上	同上	MGD

(1)：如需高精度计算水平运动，建议 X、Z 轴上使用这里给出的数值的单精度浮点真实值，可在表 2.5 中找到

(2)：如需高精度计算竖直运动，建议 Y 轴上使用这里给出的数值的单精度浮点真实值，可在表 2.5 中找到

(3)：如需高精度计算竖直运动，建议使用这里给出的数值的单精度浮点真实值，部分可在表 2.5 中找到

附录 B 完整公式

这是第 3 节中部分公式对于不同实体类型的各种形式，除 MDA 类实体对应各式和求出Δd_n、d_n、v_0、Δd_{max}和d_{max}的各式以及“万能方程组”基本确定无误外，其它公式不排除仍存在推导错误或录入错误，但缺少检验方法，希望有人帮忙检查一下。

MDA（移动->阻力->加速）

$$\Delta d_n = k^{n-1} v_0 t_0 + \frac{a {t_0}^2 (1-k^{n-1})}{1-k} \tag{B.1.1}$$

$$d_n = \frac{v_0 t_0 (1-k^n)}{1-k} + \frac{a {t_0}^2 [k^n + n(1-k) - 1]}{(1-k)^2} \tag{B.1.2}$$

$$v_0 = \frac{d_n {t_0}^{-1} (1-k) - a t_0 n}{1-k^n} + \frac{a t_0}{1-k} \tag{B.1.3}$$

$$n = -log_k \left[\frac{1}{k} - \frac{v_0 (1-k)}{a t_0 k} \right] \tag{B.1.4}$$

$$\Delta d_{max} = \frac{a {t_0}^2}{1-k} \tag{B.1.5}$$

$$d_{max} = \frac{v_0 t_0}{1-k} \tag{B.1.6}$$

$$tan\alpha_n = \frac{g t_0 (k^{n-1} - 1)}{k^{n-1}(1-k)\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} - \frac{v_{y0}}{\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} \tag{B.1.7}$$

$$\begin{aligned} y &= \frac{v_y (1-k) - g t_0}{v_{x0}(1-k)} x - \frac{g {t_0}^2}{1-k} log_k [1 - \frac{x(1-k)}{v_{x0} t_0}] y \\ &= \frac{v_y (1-k) - g t_0}{v_{z0}(1-k)} z - \frac{g {t_0}^2}{1-k} log_k \left[1 - \frac{z(1-k)}{v_{z0} t_0} \right] \end{aligned} \tag{B.1.8}$$

$$tan\,\alpha \sqrt{{x_n}^2 + {z_n}^2} = \frac{g {t_0}^2 (1-k^n)}{(1-k)^2} - \frac{g {t_0}^2 n - v'_{y0} t_0 (1-k^n)}{1-k} - y_n \tag{B.1.9}$$

DMA（阻力->移动->加速）

$$\Delta d_n = k^n v_0 t_0 + \frac{a {t_0}^2 (k-k^n)}{1-k} \tag{B.2.1}$$

$$d_n = \frac{v_0 t_0 (k-k^{n+1})}{1-k} + \frac{a {t_0}^2 k [k^n + n(1-k) - 1]}{(1-k)^2} \tag{B.2.2}$$

$$v_0 = \frac{d_n {t_0}^{-1} (1-k) - a t_0 k n}{k-k^{n+1}} + \frac{a t_0}{1-k} \tag{B.2.3}$$

$$n = -log_k\left[\frac{1}{k} - \frac{v_0(1-k)}{at_0k}\right] \quad (B.2.4)$$

$$\Delta d_{max} = \frac{at_0{}^2k}{1-k} \quad (B.2.5)$$

$$d_{max} = \frac{v_0t_0k}{1-k} \quad (B.2.6)$$

DAM（阻力->加速->移动）

$$\Delta d_n = k^n v_0t_0 + \frac{at_0{}^2(1-k^n)}{1-k} \quad (B.3.1)$$

$$d_n = \frac{v_0t_0(k-k^{n+1})}{1-k} + \frac{at_0{}^2[k^{n+1}+n(1-k)-k]}{(1-k)^2} \quad (B.3.2)$$

$$v_0 = \frac{d_nt_0{}^{-1}(1-k) - at_0n}{k-k^{n+1}} + \frac{at_0}{1-k} \quad (B.3.3)$$

$$n = -log_k\left[1 - \frac{v_0(1-k)}{at_0}\right] \quad (B.3.4)$$

$$\Delta d_{max} = \frac{at_0{}^2}{1-k} \quad (B.3.5)$$

$$d_{max} = \frac{v_0t_0k}{1-k} \quad (B.3.6)$$

MAD（移动->加速->阻力）

$$\Delta d_n = k^{n-1} v_0t_0 + \frac{at_0{}^2(k-k^n)}{1-k} \quad (B.4.1)$$

$$d_n = \frac{v_0t_0(1-k^n)}{1-k} + \frac{at_0{}^2k[k^n+n(1-k)-1]}{(1-k)^2} \quad (B.4.2)$$

$$v_0 = \frac{d_nt_0{}^{-1}(1-k) - at_0kn}{1-k^n} + \frac{at_0k}{1-k} \quad (B.4.3)$$

$$n = -log_k\left[\frac{1}{k} - \frac{v_0(1-k)}{at_0k^2}\right] \quad (B.4.4)$$

$$\Delta d_{max} = \frac{at_0{}^2k}{1-k} \quad (B.4.5)$$

$$d_{max} = \frac{v_0t_0}{1-k} \quad (B.4.6)$$

AMD（加速->移动->阻力）

$$\Delta d_n = k^{n-1} v_0t_0 + \frac{at_0{}^2(1-k^n)}{1-k} \quad (B.5.1)$$

$$d_n = \frac{v_0 t_0 (1-k^n)}{1-k} + \frac{a{t_0}^2[k^{n+1}+n(1-k)-k]}{(1-k)^2} \tag{B.5.2}$$

$$v_0 = \frac{d_n {t_0}^{-1}(1-k) - a t_0 n}{1-k^n} + \frac{a t_0 k}{1-k} \tag{B.5.3}$$

$$n = -log_k\left[1 - \frac{v_0(1-k)}{a t_0 k}\right] \tag{B.5.4}$$

$$\Delta d_{max} = \frac{a{t_0}^2}{1-k} \tag{B.5.5}$$

$$d_{max} = \frac{v_0 t_0}{1-k} \tag{B.5.6}$$

$$tan\alpha_n = \frac{g t_0 (k^n - 1)}{k^{n-1}(1-k)\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} - \frac{v_{y0}}{\sqrt{{v_{x0}}^2 + {v_{z0}}^2}} \tag{B.5.7}$$

$$tan\,\alpha \sqrt{{x_n}^2 + {z_n}^2} = \frac{g{t_0}^2(k-k^{n+1})}{(1-k)^2} - \frac{g{t_0}^2 n - v'_{y0} t_0 (1-k^n)}{1-k} - y_n \tag{B.5.8}$$

ADM（加速->阻力->移动）

$$\Delta d_n = k^n v_0 t_0 + \frac{a{t_0}^2(k-k^{n+1})}{1-k} \tag{B.6.1}$$

$$d_n = \frac{v_0 t_0 (k-k^{n+1})}{1-k} + \frac{a{t_0}^2 k[k^{n+1}+n(1-k)-k]}{(1-k)^2} \tag{B.6.2}$$

$$v_0 = \frac{d_n {t_0}^{-1}(1-k) - a t_0 k n}{k-k^{n+1}} + \frac{a t_0 k}{1-k} \tag{B.6.3}$$

$$n = -log_k\left[1 - \frac{v_0(1-k)}{a t_0 k}\right] \tag{B.6.4}$$

$$\Delta d_{max} = \frac{a{t_0}^2 k}{1-k} \tag{B.6.5}$$

$$d_{max} = \frac{v_0 t_0 k}{1-k} \tag{B.6.6}$$

附录 C 辅助性工具

大部分实验内容来自 Minecraft Java 版 1.16.4，极少部分来自 Java 版 1.16.5 的 MCP，在 1.16.4 的实验中用到了以下辅助 Mod：

(1) fabric-carpet-1.16.4-1.4.17+v201111
(2) mcwmem-1.16.4-20210714
(3) chunkmap-20210718
(4) malilib-fabric-1.16.4-0.10.0-dev.21+arne.1
(5) minihud-fabric-1.16.4-0.19.0-dev.20201103.184029
(6) tweakeroo-fabric-1.16.4-0.10.0-dev.20201110.132827

mcwmem 全称为 Minecraft World Manipulation Enchantment Mod（Minecraft 世界控制增强，似乎名称起的太大了），是我临时赶工期开发的一个用于进行世界数据访问和修改的 Mod，内容很乱，目前功能较不完善且仅支持 1.16.4 和 1.16.5，但是够这次使用，可以在以下地址下载。

MOD:

https://github.com/lovexyn0827/Minecraft-World-Manipulation-Enchantment-Mod/releases/tag/1.16.x-20210731

README:

https://github.com/lovexyn0827/Minecraft-World-Manipulation-Enchantment-Mod/blob/main/README.md。

实体运动研究中不能过于相信贴图（或者说是客户端上显示出的实体以及碰撞箱），因为贴图位置比在服务端上实体的真正位置通常是滞后的，研究过程中太相信贴图那么可以认为第一步就输了，而这个 Mod 个人认为最大的功能就是正确地显示服务端上实体的碰撞箱位置，防止被客户端欺骗。

chunkmap 是我在 7 月 9-18 日开发的一个用于可视化区块的加载、生成和区块加载票的 Mod，目前功能趋于完善。

最新版本下载地址：

https://github.com/lovexyn0827/Chunkmap/releases/tag/v20210718

MCBBS 上的介绍贴：

https://www.mcbbs.net/thread-1225181-1-1.html。

文中涉及所有源代码由 minecraft fabric 中 genSources 生成，mapping 使用 yarn-1.16.4+build.1-v6，如需要可以自行反编译。

配套存档可在原帖链接中下载。

用到的一些小程序（请在 classpath 中包含反混淆的 Minecraft 及其运行库）：

https://github.com/lovexyn0827/Entity。

附录 D 主要符号及单位

这里仿照国际单位制选取时间和长度作为连个基本单位。定义时间是用于描述某一过程经过的运算周期的量，长度是用于描述两个点之间距离的量。常用单位及其换算关系如下，其中格与米是等价的，大家可以按照个人喜好使用，文中以米为准：

表 D1 主要符号表

量	单位名	符号	简要的定义	换算
长度	米	m	完整方块的棱长	1m
	格	b	完整方块的棱长	=1b
	像素	px	方块最小渲染单元的边长	=16px
	区块		一个区块俯视图的边长	=1/16 区块
时间	游戏刻	gt	一个游戏刻代表的时间	1gt
	无卡顿的秒	s	在 TPS 为 20 时的一秒	=1/20s
	红石刻	无	一档红石中继器的延迟	=1/2 红石刻
速度	米每刻	m/gt		1m/gt
	米每秒	m/s		=20m/s
加速度	米每平方刻	m/gt^2		$1m/gt^2$
	米每平方秒	m/s^2		$=400m/s^2$
阻力系数	负一次方刻	gt^{-1}		

当然，这些单位的使用没有必要那么严格，把仅有 m 和 gt 组成的单位都看成 1 也行。

附录 E 定义索引

A

AI 加速度 59

B

爆炸半径 52
爆炸半径利用率 90
爆炸接触率 52
爆炸利用率 90
爆炸伤害半径 52
爆炸威力 52
爆炸影响力 54
爆炸中心 52

C

侧向加速度系数 63

D

地面移动加速度 59
地面阻力 7

F

FallDistance 14
flyingSpeed 63
forwardSpeed 63
horizontalCollision 13
方块网格 33
方块坐标 5
飞行时间利用率 90
飞行移动加速度 59
俯仰角 6

G

根实体 9
固体实体 31

H

horizontalCollision 13
滑度 40

火球类 12

J

加速度 8
加速度利用率 90
检查方块网格碰撞 32
接触率利用率 9

K

可攀登方块 59
空气阻力 7
空气阻力系数 6

L

LivingEntity 12
流体加速 6
流体流向向量 42
流体深度阀值 59
流体阻力 7

M

Motion 5
movementMultiplier 39
movementSpeed 63

N

noClip 13

O

P

PersistentProjectileEntity 12
pistonMovementDelta 14
pistonMovementTick 14
碰撞箱 5
偏航角 6

Q

前向加速度系数 63
取样点 52

R

raycast 32

S

sidewaysSpeed 63
stepHeight 13
实际位移趋势 28
实体 ID 13
实体基础运算 14
实体挤压 47
实体所处方块 39
实体下方方块 39
实体着陆方块 39
输入位移 28
速度 6
速度乘数 7

T

同位实体 9
弹射物 12
投掷物 12

U

upwardSpeed 63

V

verticalCollision 13

W

位移趋势 28
位移趋势预处理 28
位移确定 6

X

向上的加速度系数 63

Y

沿轴移动 30
眼部坐标 6
移动 8
着地 13

Z

沿轴移动	30
眼部坐标	6
移动	8
着地	13

致谢

感谢 B 站用户粗缯大布裹生涯帮助优化该报告的排版，并指出文中几处错误。