上古时代的游戏并不会使用例如PhysX等物理引擎，例如Quake和Doom，开发者们都会自己编写简单的碰撞检测模块来完成角色移动逻辑。虽然碰撞检测需要的物理算法很简单，但想让游戏操作起来更加顺畅，往往需要非常多的细节处理逻辑。这些特殊的移动处理逻辑叫做collide and slide算法，经过了10多年的积累沉淀，这套逻辑已经非常成熟，被应用到各种类型的游戏上。

好奇的同学可能会问，既然有了PhysX物理引擎，为什么不直接用它来完成角色移动呢？

原因有很多，这里列举几个比较典型的

* 讨厌的tunneling effect，用过物理引擎的同学可能会遇到，如果角色的移动速度过快，它很可能会穿透墙。所以，角色的最大速度往往是被限制的，这并不能满足游戏设计需求。即使不出现tunnel effect，角色碰到了墙角，会出现抽搐抖动，甚至移动到不可知的位置。
* 不能直接控制角色，想让物理引擎的刚体移动，需要施加impluse或者force推力，这并不能让角色移动到想要到的位置。
* 摩擦力问题，想让角色站在斜坡上，需要设置无限大的摩擦力，但这回导致角色无法在斜坡上移动。
* 不受控制的跳跃，在类似波浪线有起有伏的地形上移动，不可避免的会腾空。

以上这些情况如果使用物理引擎几乎是无法避免的，所以目前几乎所有的游戏都会自定义自己的移动模块，模块的复杂程度根据游戏的类型规模有着天壤之别，运动类游戏和第一人称射击游戏的移动模块往往是最复杂的。而第一人称射击类游戏的移动模块更具有通用性，经过多年发展，已经比较成熟，所以本文参考UE4中的代码，抽取其中核心逻辑，向大家介绍collide and slide算法。

在了解UE4的移动逻辑之前，我们先熟悉下碰撞的基础接口

UE4移动中碰撞检测主要使用PhysX的[Geometry Queries](http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/3.3.4/Manual/GeometryQueries.html#sweeps)（几何查询）功能

* 射线检测RayCasts
* 重叠检测Overlaps
* 渗透深度计算Penetration Depth
* Sweeps检测
* InitialOverlaps检测

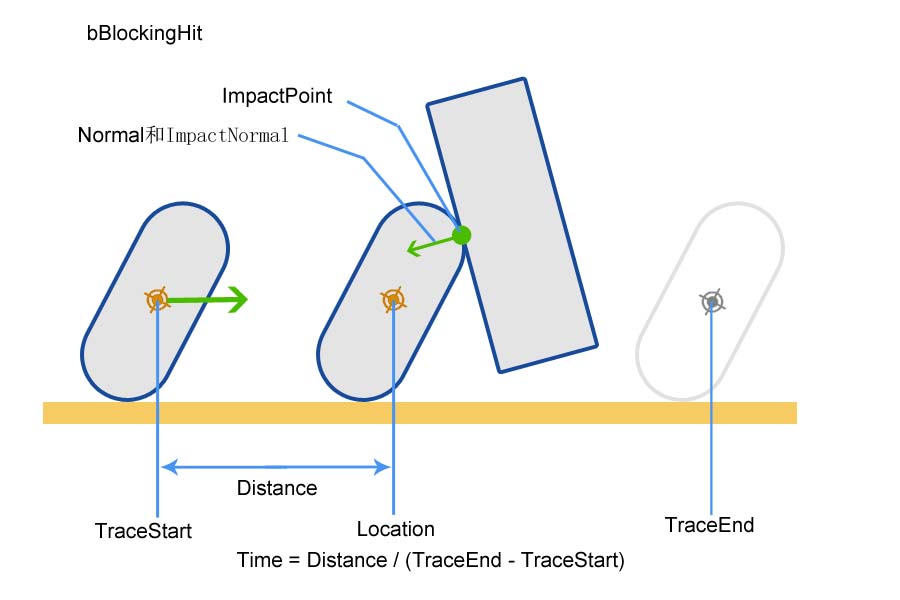
UE4把查询后返回的hit封装成了FhitResult，FhitResult的结构如下

### FHitResult

|  |  |
| --- | --- |
| bBlockingHit | 是否发生碰撞 |
| bStartPenetrating | 是否在检测开始就有渗透情况 |
| Time | 碰撞后实际移动距离除以检测移动距离 |
| Distance | 碰撞后实际移动距离 |
| Location | 碰撞后最终位置 |
| ImpactPoint | 碰撞接触点 |
| Normal | 碰撞切面法向量 |
| ImpactNormal | 碰撞切面法向量（非胶囊体和球体检测与Normal不同） |
| TraceStart | 检测开始位置 |
| TraceEnd | 检测结束位置 |
| PenetrationDepth | 渗透深度 |

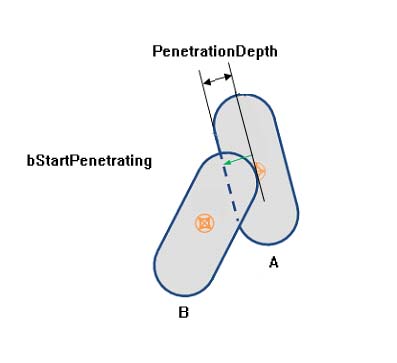
我们可以借助以下两种移动中常见的情况熟悉一下这些参数，

###### 第一种是常见的胶囊体Sweep查询



查询开始结束分别是TraceStart和TraceEnd两个位置，如果碰到了障碍，bBlockingHit就是true，胶囊体最终会停在Location位置，它移动的距离是Distance，Time是一个0到1的值，表示实际移动距离比查询距离。还有一些可能会用的参数，比如碰撞接触点ImpacePoint，碰撞切面法向量Normal和ImpactNormal

###### 第二种常见的情况通常是InitialOverlaps，开始位置检测到了重叠，



这时候bStartPenetrating是true，通过渗透深度计算可以获得PenetrationDepth，这个参数对于处理移动中穿透的情况非常重要

仔细观察的话可以发现上面胶囊体的Sweep就是一次简单的移动过程，UE4将这个过程进一步封装成了SafeMoveUpdatedComponent，它是UE4移动最关键的函数，几乎所有的移动都要靠它来完成。它的主要功能有以下几点

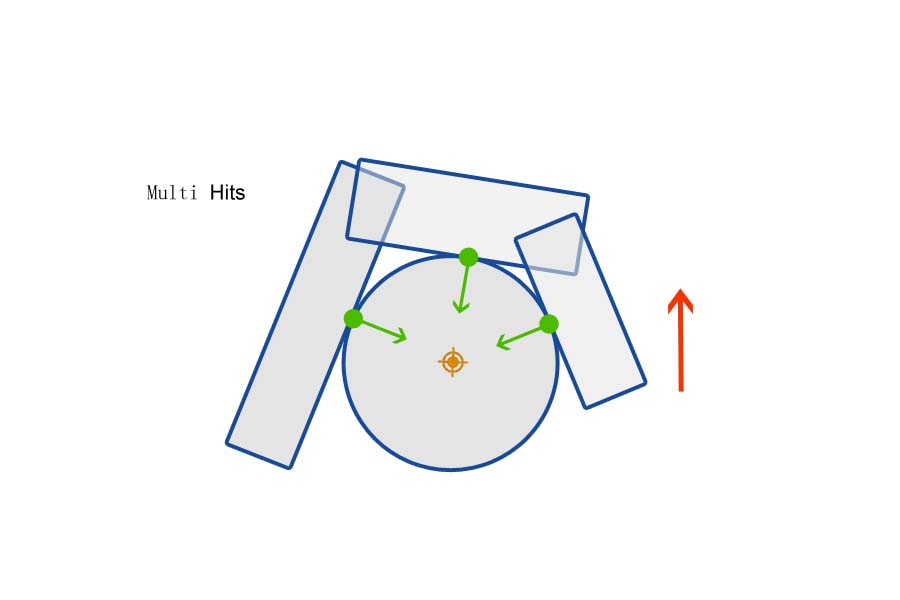
* 筛选Hit
* SetLocation并递归更新子组件
* UpdateOverlap，Overlap检测
* 解决渗透的情况，bStartPenetration
* 返回检测结果Hit

下面分别介绍一下这些功能，注意下面的符号▽△用于表示函数的开始和结束

## ▽▽▽▽▽▽ SafeMoveUpdatedComponent ▽▽▽▽▽▽

#### ▽▽▽▽ UPrimitiveComponent::MoveComponentImpl ▽▽▽▽

###### 调用SweepMulti获取合理的Hit

* 调用SweepMulti得到的所有Hit需要拉回微小的距离(缩小hit.time)，避免因为浮点数精度的问题导致跟碰撞物重叠
* 如果检测到多个block hit，优先选择不是在初始位置就检测到block的hit，否则的话选取跟运动方向最相反的hit 

如上图是俯视图，圆形是胶囊体，方形是碰撞物，红色箭头是运动方向，胶囊体同时跟3个障碍物发生的碰撞，得到了3个hit，也就是图中的3个绿色剪头，按照筛选规则，选取跟红色箭头方向最相反的，也就是中间的绿色箭头的hit。

###### SetPosition以及相关操作

* 调用SetWorldLocationAndRotation
* 更新ComponentToWorld Transform矩阵
* 更新父组件和递归更新子组件
* 更新导航网格数据，Bounds边界
* 更新RenderTransform以及 PhysicsTransform

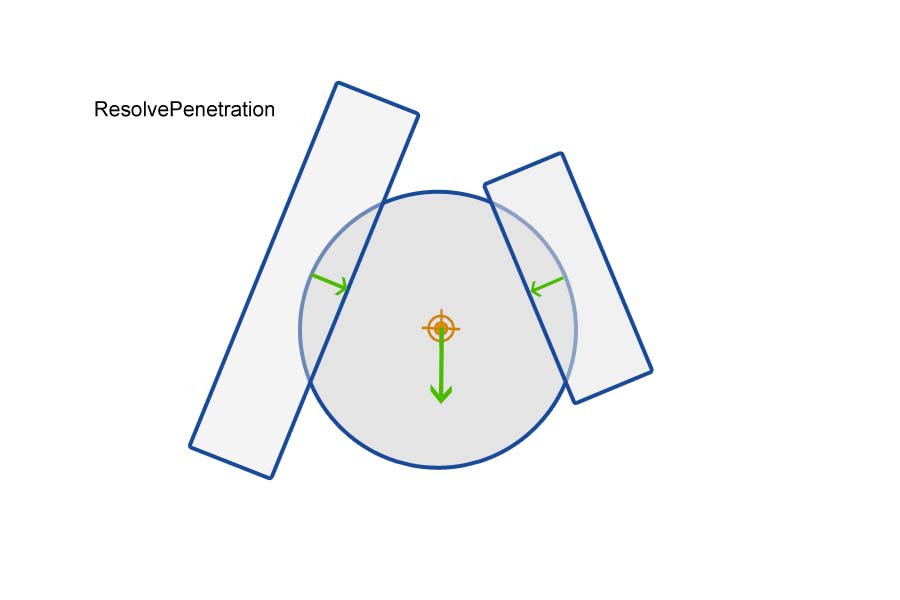
###### 调用UpdateOverlap 更新重叠状态

* 调用OverlapMulti，获得检测结果
* 更新Overlap Components列表，删除不再Overlap的Component，新增新的Component。
* 更新子Component的 Overlap Components列表
* 更新PhysicsVolume（比如进入离开水域）

#### △△△△ UPrimitiveComponent::MoveComponentImpl △△△△

如果调用MoveComponentImpl返回的hit结果bStartPenetrating是true，需要调用ResolvePenetration解决穿透的问题

#### ▽▽▽▽▽▽ ResolvePenetration ▽▽▽▽▽▽



上图是俯视图，圆形代表胶囊体，方形是障碍物，胶囊体跟左边的障碍物穿透了，比较直观的解决方法是将它按照左边重叠的绿色箭头拉回，拉回的距离就是上面提到的PenetrationDepth变量，如果拉回过程中又跟右边的障碍物穿透了，这时候会得到右边的绿色箭头，左右两边的箭头叠加，也就是向量相加，会得到中间向下的箭头，按着这个方向拉回，就会避免穿透问题。如果调整位置成功了，还需要再次尝试最开始的移动。

#### △△△△△△ ResolvePenetration △△△△△△

## △△△△△△ SafeMoveUpdatedComponent △△△△△△

SafeMoveUpdateComponent可以看做是底层碰撞检测和上层移动逻辑的中间层，是基础的移动单元，接下来我们要介绍的移动逻辑，看似复杂，其实都是由这些移动单元构成的。整个移动逻辑的主函数是PerformMovement，我们还是按照函数的调用顺序梳理一遍它的主要逻辑。

## ▽▽▽▽▽▽ PerformMovement ▽▽▽▽▽▽

1. 根据输入向量InputVector计算加速度向量Acceleration
2. 随着被骑乘物MovementBase（比如电梯，载具）移动
3. 将冲力Impulse和推力Force作用于速度Velocity，一般用于击退和径向运动
4. 根据不同的运动状态运动

* MOVE\_None (不做运动)
* MOVE\_Walking （踩地面上运动）
* MOVE\_NavWalking （踩导航网格上运动）
* MOVE\_Falling （在空中受重力加速度）
* MOVE\_Flying （不受重力加速度的运动）
* MOVE\_Swiming （在水中运动）
* MOVE\_Custom （自定义运动，比如插值运动）

先看下MOVE\_Walking

### ▽▽▽▽▽▽ PhysWalking ▽▽▽▽▽▽

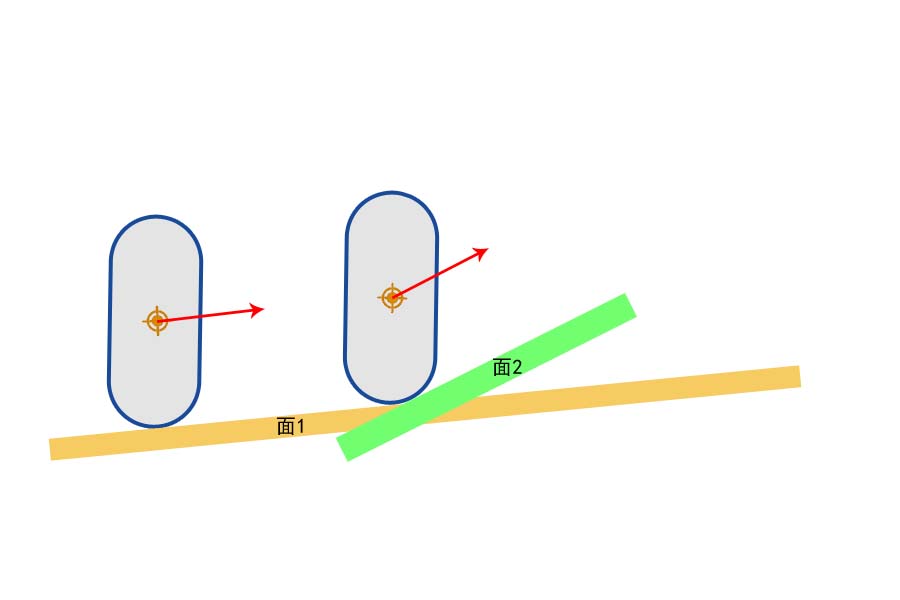
首先将速度和加速度的垂直方向分量设为0，方向始终保持在水平面上

#### ▽▽▽▽▽▽ CalcVelocity ▽▽▽▽▽▽

1. 计算速度，先设置为RequestedVelocity（寻路组件PathFollowingComponent根据路径不断设置该速度）
2. 加速度是0的时候，将受到减速度BrakingDeceleration和摩擦力的影响而减速
3. 加速度不是0的时候，摩擦力将会影响速度方向改变快慢
4. 计算速度向量Velocity += Acceleration \* DeltaTime
5. 最后，如果支持RVOAvoidance，将会根据RVO重新计算速度，避免跟其他角色重叠在一起，效果就像被弹回来。

#### △△△△△△ CalcVelocity △△△△△△

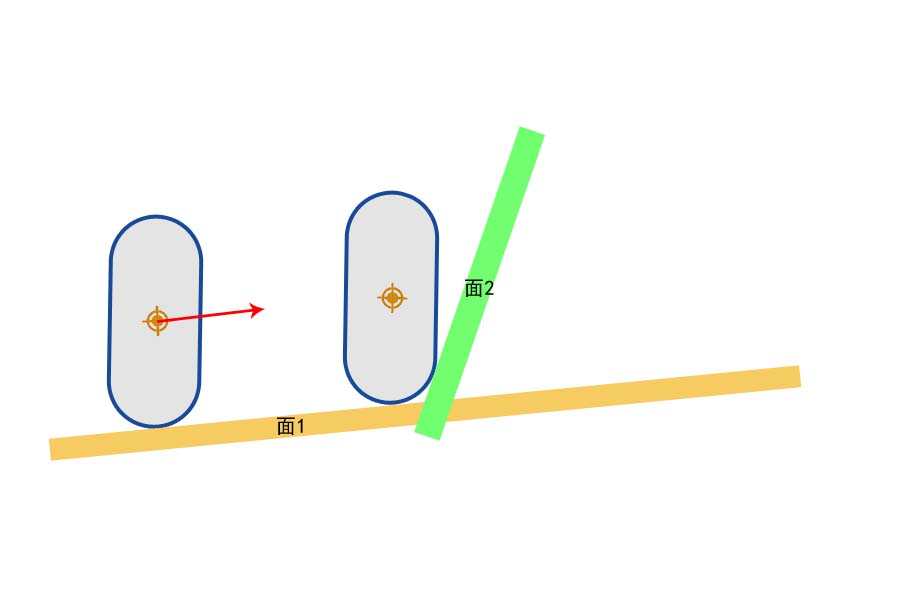
#### ▽▽▽▽▽▽ MoveAlongFloor ▽▽▽▽▽▽



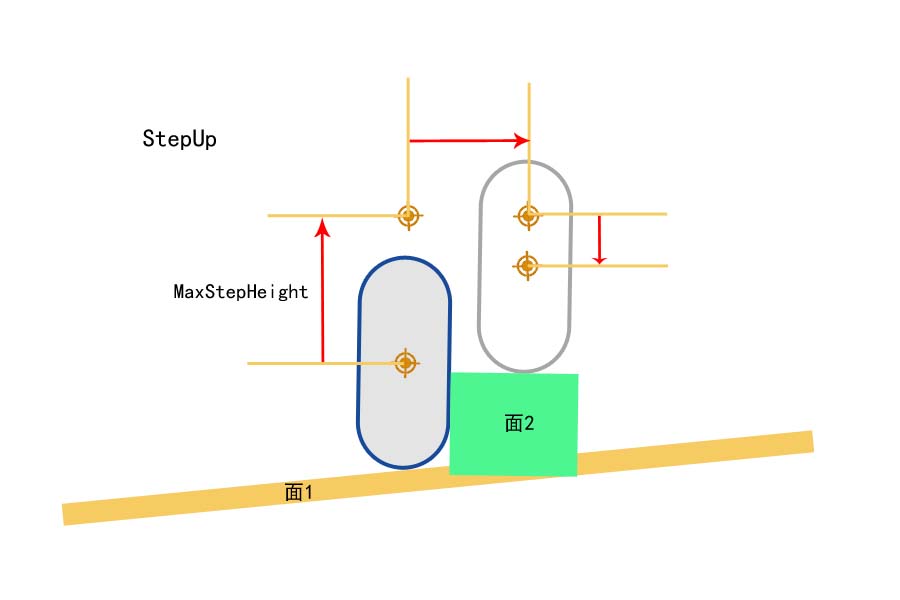
计算移动向量Delta = Velocity \* DeltaTime

根据地面坡度调整移动向量方向，如上图需要改为沿着面1坡度的方向，也就是红色箭头的方向，调用SafeMoveUpdatedComponent

如果返回Hit结果是block，如上图碰到了面2，通过返回的Hit的Normal参数检测到面2的斜面坡度较缓，这时可以将剩下的移动向量改为沿着面2移动，再次调用SafeMoveUpdatedComponent，如果返回的Hit结果还是block或者面2非常陡峭（如下图所示），可以开始尝试调用StepUp上楼的逻辑



##### ▽▽▽▽▽▽ StepUp ▽▽▽▽▽▽



理想情况下的上楼梯过程如图所示，它是由3次移动构成，首先向上移动MaxStepHeight高度，然后向前移动（向前移动过程中如果检测到block，需要调用SlideAlongSurface），最后向下移动，落到面2上面。当然，存在很多情况会导致StepUp失败，比如移动过程中检测到穿透Penetration，最终无法落到一个合理的落脚点（比如面2比较陡峭），都会导致调用StepUp失败，在这种情况下，我们需要调用SlideAlongSurface，贴着面走

##### △△△△△△ StepUp △△△△△△

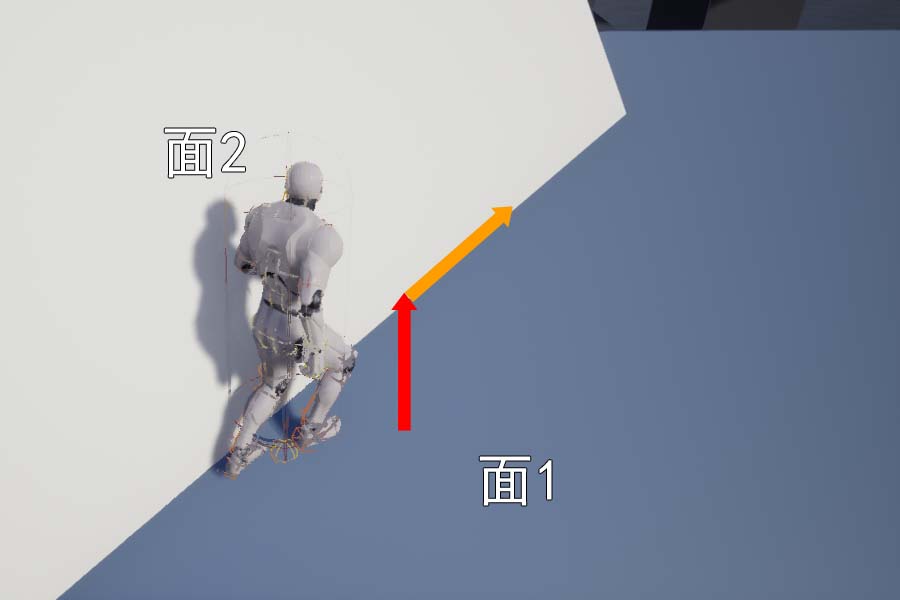
在调用SlideAlongSurface贴着面走之前，需要调用HandleImpact，处理碰撞发生后带来的副作用

##### ▽▽▽▽▽▽ HandleImpact ▽▽▽▽▽▽

发送MoveBlockedBy事件，如果开启bEnablePhysicsInteraction，可以给与刚体一个反推力

##### △△△△△△ HandleImpact △△△△△△

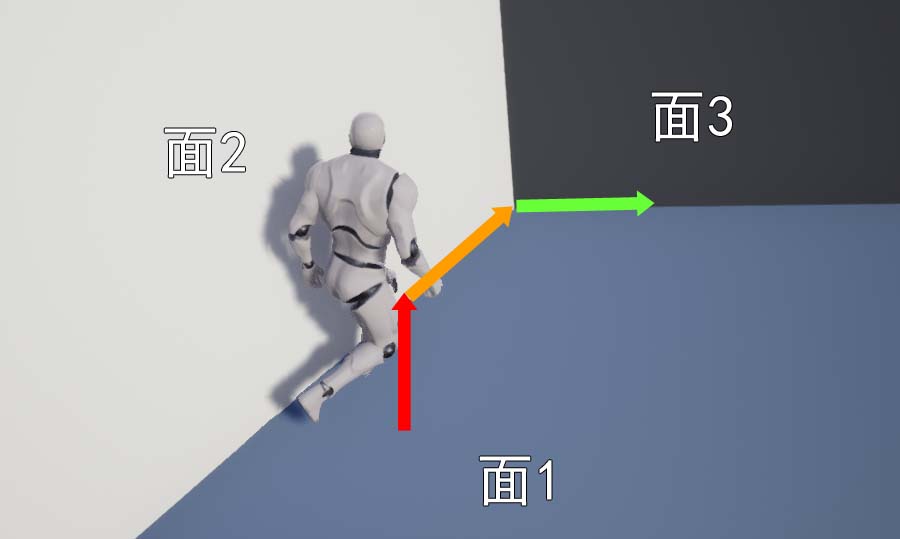
##### ▽▽▽▽▽▽ SlideAlongSurface ▽▽▽▽▽▽



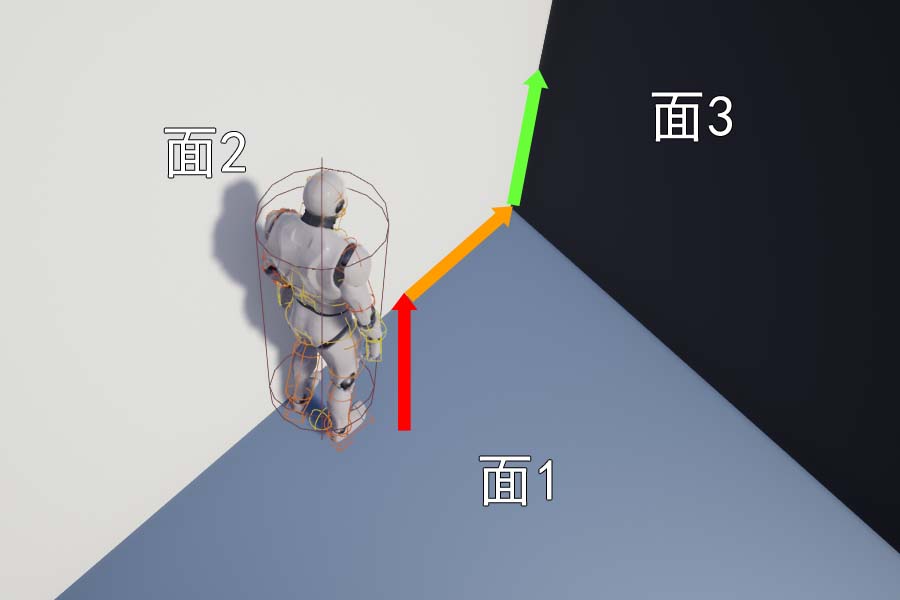
二维的图示并不能很好表示贴墙走的情况，我们看下上面这个截图，红色箭头表示最开始移动方向，撞到面2后，我们调用StepUp失败，尝试SlideAlongSurface，于是移动方向变为贴着面2的黄色箭头， 如果按照黄色箭头的移动过程中很不幸又碰到了一个面，我们需要调用TwoWallAdjust

###### ▽▽▽▽▽▽ TwoWallAdjust ▽▽▽▽▽▽

利用两个面法向量计算面2和面3的夹角，如果夹角大于90度，我们可以将移动方向变为沿着面3的绿色箭头



如果面2和面3的夹角小于90度，我们可以沿着面2和面3的夹缝（如下图的绿色向量）继续移动，这个夹缝向量可以通过面2和面3的法向量的叉乘结果计算出来，当然这个夹缝向量的倾斜角度不能过于陡峭，否则角色也是不能按照这个方向移动的。



###### △△△△△△ TwoWallAdjust △△△△△△

##### △△△△△△ SlideAlongSurface △△△△△△

#### △△△△△△ MoveAlongFloor △△△△△△

到这里MoveAlongFloor就执行完了，然后还需要调用FindFloor，检测地面，调整纵坐标，保证角色始终贴着地表

#### ▽▽▽▽▽▽ FindFloor ▽▽▽▽▽▽

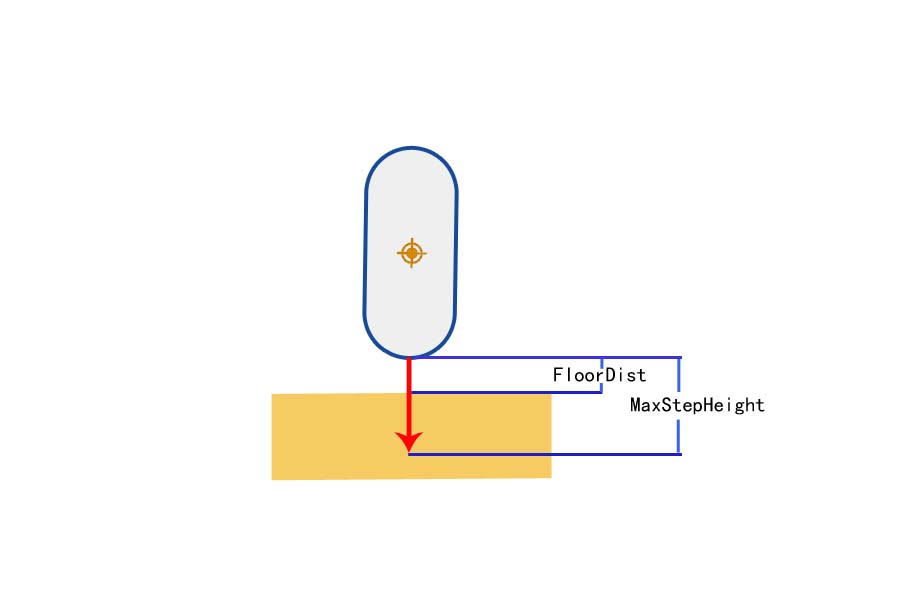
FindFloor返回的结果也是个比较重要的结构，我们看下它的参数

FFindFloorResult

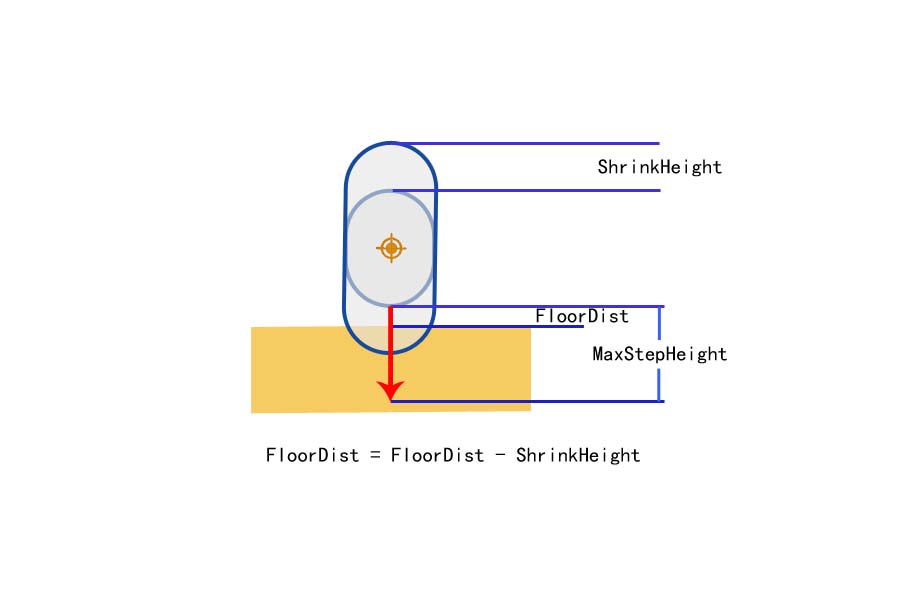
|  |  |
| --- | --- |
| bBlockingHit | 是否跟地面有碰撞 |
| bWalkableFloor | 可以行走的地面 |
| bLineTrace | 是否是通过line trace检测出来的结果 |
| FloorDist | Sweep查询到地面的距离 |
| LineDist | LineTrace查询到地面的距离 |
| HitResult | 跟地面的FHitResult |

##### ▽▽▽▽▽▽ ComputeFloorDist ▽▽▽▽▽▽

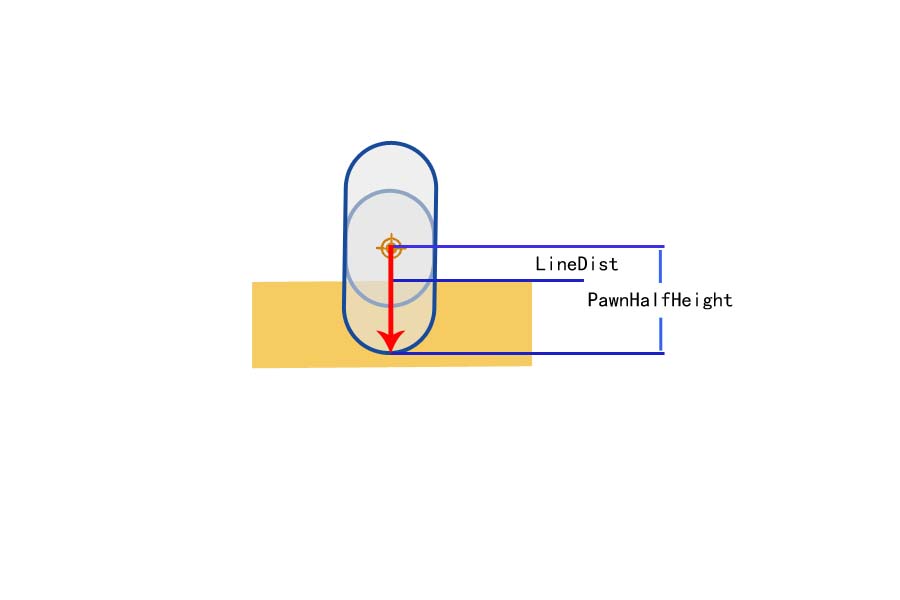
一般情况下，比如下图中的情况，我们只需要一次垂直向下Sweep检测就可以计算出FloorDist，注意检测的距离是之前StepUp向上检测的距离。这时候FloorDist等于返回Hit的Distance



如果返回的的Hit是bStartPenetration是true话则需要用一个缩小的胶囊体来重新向下Sweep，算出来的FloorDist减去缩水的高度就是原胶囊体跟地面的距离



如果用缩小胶囊体Sweep还是有穿透情况，这时候需要改用line trace，从胶囊体的中心向下trace胶囊体的半个身高，如果检测到了hit，则可以计算出陷入到地面以下的高度



注意无论是sweep还是line trace设置胶囊体向上抬的调整高度MaxPenetrationAdjust最大只能是胶囊体的半径，如果陷入地下的深度大于调整高度，一次调整是无法将胶囊体从地面抬出来的，往往需要多帧处理才可以。

##### △△△△△△ ComputeFloorDist △△△△△△

#### △△△△△△ FindFloor △△△△△△

通过调用AdjustFloorHeight根据之前计算的FloorDist来调整角色的垂直坐标

如果FindFloorResult的bWalkableFloor是false，需要调用CheckFall，切换成MOVE\_Falling状态

### △△△△△△ PhysWalking △△△△△△

其他运动状态这里不做具体说明，大体逻辑是基本相似的，区别可能在于计算速度和对返回Hit的特殊处理上。

Physx也提供了[CharacterController移动库](http://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/CharacterControllers.html#character-goes-through-walls-in-rare-cases)，有兴趣的可以参考下。