# 骨骼动画技术美术

## 第一章 综述

### 1.1 动画技术原理

#### 1.1.1 传统动画概述

##### 人类视觉的运动感知原理

从十九世纪开始人们就发现了学名为“独立静态正后像”这一现象，该现象后来被广泛地称为"视觉残留"，即在显示对象消失后人眼仍然能在短时间内保持对这一视觉元素的映像，使人能关联起时域上串行接收的图像信号[1][2]。这种现象一直以来用于指导电影、动画、游戏行业的发展，尤其是在刷新率和刷新方式上的创新。

如果图片刷新间隔长于人眼视觉暂留时间，或者称图片刷新率低于人眼视觉刷新率，人眼就难以生成连续的视觉画面感，结果会呈现为闪烁状态，大脑会将影像解读为离散的静态图片序列。由于不同个体视觉暂留能力不同，单纯选取一个临界刷新率可能会导致部分人群无法接受视频信息，所以通常媒体的刷新率都需要远高于视觉刷新率。但这并不绝对，在特定的室内光照(包括亮度和闪烁频率等)和观察距离下，个体视觉的图像回放能力可以得到加强，这个理论有时被用于在专业电影院中播放电影时适当降低视频素材的刷新率，以降低成本[3]。

我们会使用刷新率(或称回放率)和采样率(或称更新率)来描述一个由图片序列组成的运动片段的规格。其中，刷新率代表的是每秒显示图像的数量，而采样率代表的是每秒显示的不同的图像的数量。刷新率高的视频能避免闪烁现象，而采样率的高低则定义了片段呈现的运动质量。通常来说，决定刷新率的是多媒体播放器软硬件，如显示器；而决定采样率的是录制器软硬件，比如摄像机和游戏程序。

通常来说，24Hz是一般群众可以接受的最低刷新率，观察低于这个刷新率的视频时大部分人就可以感受到闪烁。目前的显示设备通常都能支持60Hz或120Hz的刷新率。在电影录制时，经常会使用采样率仅为24FPS的多媒体源文件，但会使用48Hz或96Hz的刷新率来避免闪烁。采样率并不总是低于刷新率，比如在使用隔行扫描技术时，即使刷新率仅为30FPS，也需要采用60FPS的采样率来保证平滑的运动效果。一般来说，使用逐行扫描时的刷新率要求总是比使用隔行扫描时更高。

视觉残留现象能让人眼识别不同画面之间的关联，但这种关联体现在时域上或空域上却需要另一种生理机制。举例为，一个光源在快速旋转时，由于视觉残留现象，人眼会将其识别成一个光环，但并不会认为该光环正在运动。[4]

##### 经典帧动画原理

使用计算机辅助或者完全主导动画创作是当下动画创作技术的主流，其中，关键帧技术和帧间插值计算成为计算机动画领域的基本技术。包括骨骼动画在内，使用关键帧技术或思想的动画技术统称帧动画，帧动画又可以按存储方式分为序列帧动画和关键帧动画。

帧动画的思想可以追溯到早期动画时期。早期较著名的动画装置如西洋镜(Thaumatrope)便是序列帧动画的代表之一，通过在圆柱体内壁上绘制一系列图案，观察者可以在旋转圆柱体时观察到运动的画面。与此类似的还有手翻书，书中每一页都绘制了一个静态的图像，而观察者可以在快速翻阅纸张时感受到运动画面。手绘动画也是一种帧动画，早期手绘动画就是通过手绘每一帧静态图像，然后通过对图像拍照的方式记录图片，最后播出。除了早期手绘动画外，同一时间也涌现出了各种类型的定格动画。所谓定格动画就是将粘土、折纸或其它材料制成的模型摆成需要的姿态，然后通过拍照记录下来，最后连贯的将照片连续的播放形成的动画。

包括手绘动画在内的经典动画的共同特点是，每一帧的静态图案都是固定而不可复用的，换句话说就是在每一帧，画面场景中的所有物体都必须被重新绘制一次，无论它们有没有发生形态上的实质改变。这种动画被分类为非矢量动画，相对应的，便出现了矢量动画。矢量动画的代表之一是皮影戏，通过操纵皮影人物，将场景和人物等每个部件进行了分离，这样在人物移动时不再需要重复的绘制相同的背景。迪士尼发明的多平面相机录制法是矢量动画的又一突破，它们将场景按照远近分成了若干个动画平面分别绘制，而每个平面(包括相机)又可以按照需求沿任意方向运动，这样便实现了在早期技术不成熟的情况下对三维场景的模拟。这种将人物、物体、场景分离分层，分别运动然后统一拍摄的思路，也指导了后来计算机动画领域的发展。目前计算机动画领域将单个可运动图层称为精灵(Sprite)，正是迪斯尼多平面相机录制法的历史传承。

计算机动画是一种经典的矢量动画，随着Alpha通道的发明[5]，通过计算机辅助组织场景和角色，艺术家可以节省大量的重复工作时间。早期的二维计算机动画属于序列帧和关键帧混合的动画，其中，大部分人物和场景元素的动作都需要使用序列帧。在早期的电子游戏中，很多人物都使用16幅或32幅的动画表记录动画，出于节省空间的考虑，每个动作(如走、跑、跳等)仅能容纳1~3帧的序列帧信息。在运行时，计算机会负责将动画表中指定位置的图案拷贝到场景中。在次世代的二维电子游戏开发中，帧动画仍然承担着大量的作用，随着硬件的进步，目前的一段帧动画已经可以容许包含超过20帧的图片序列。而对于人物在场景中的移动、旋转、缩放，则体现出了关键帧动画的特点。由于人物的移动通常呈直线或曲线，所以不必记录人物每一帧的位置，而只需要记录沿曲线运动的开始时刻和结束时刻，然后在这段时间内让物体沿指定轨迹运动即可。

在主流三维动画领域，传统的序列帧动画依然有一席之地，其主要应用场景是特效和粒子动画。如很多火焰效果就是通过粒子播放序列帧动画实现的。而很多场景的运动，如门的开关等，也是通过三维矢量关键帧动画实现的。

#### 1.1.2 骨骼动画原理

##### 顶点动画与渲染流水线

骨骼动画本质上是一种顶点动画。

所谓顶点动画，是区别于刚体动画的一种三维动画类型。顶点动画的含义是在渲染过程中，输入渲染流水线的网格顶点产生了变化的动画。相对的，刚体动画是一种在渲染过程中，输入渲染流水线的网格顶点不产生变化，转而是相关变换矩阵产生变化的动画。顶点动画的另一个常见应用是流体动画，通过修改顶点位置来产生流体表面的形态变换效果。

渲染流水线大致分为应用阶段、几何阶段、光栅化阶段、片元阶段。其中，光栅化阶段就是通过处理顶点阶段的输入，生成对应的片元的阶段。而片元经过片元阶段生成了我们看到的具有颜色和透明度信息的像素。可见，应用阶段和几何阶段两个阶段主要确定了物体渲染的外形，光栅化阶段和片元阶段主要确定了物体渲染的颜色。由于我们讨论的主要是与物体外形有关的动画，所以我们暂且抛开光栅化阶段和片元阶段。

在应用阶段中，构成物体的网格顶点，以及物体在场景中的变换信息，如平移、旋转和缩放等信息被从CPU传输到GPU中。负责传输顶点数据的数据结构被称为顶点数组，它由顶点坐标数组、索引数组、属性数组和属性指针等成分组成[6]，这些数据的数量级是顶点级，也就是每个顶点都在顶点数组中有一份独特的数据。负责传输变换信息的数据结构被称为常量数组，这些变换信息是以矩阵的格式被存储在常量数组中，并在流水线中的剩余环节中被访问的，常量数组中的数据通常是由整个网格共享的。

在几何阶段中，GPU访问顶点数组中的索引数组和顶点坐标等信息，以及常量数组中的变化矩阵等信息，经过一系列的空间变化，以将顶点的坐标变化到标准设备坐标空间[7]。在这个过程中，三维模型的坐标最终变化为了我们看到的二维屏幕上的坐标，为之后流水线中在屏幕上填充片元和像素提供了几何依据。在几何阶段中，也可能会产生顶点的增加和删除。

顶点动画的生效就是在这两个阶段中。通常来说，在应用阶段即CPU中修改顶点数组，远慢于在几何阶段中由GPU修改顶点位置。类似水面波动一类的动画就是发生在几何阶段的顶点动画。

##### 骨骼动画与顶点数组

骨骼动画本质上是一种压缩算法，是对网格顶点动画的一种压缩。这是由角色动画的特点决定的：

第一，骨骼动画必然会导致顶点位置的变化，这种变化不能在成本允许的前提下由光栅化阶段和片元阶段的渲染策略实现。

第二，骨骼动画所控制的模型不会整个一起运动，而是其身体的各个组成部分分别进行变换，这使得角色动画中不能在整个角色的层次上实施刚体动画。

第三，身体的各个组成部分是互相紧密连接的，在旋转和移动时不能出现断裂，其互相之间的连接处，如膝盖、手肘等位置的顶点需要进行一个平滑的排布，否则就会显得生硬，这使得骨骼动画很难将身体的组成部分拆成若干个不同的网格分别进行刚体动画。

综上所述，顶点动画是骨骼动画的唯一选择。

但是，骨骼动画和流体动画这类其它顶点动画的区别在于，在流体动画中，单凭流体表面的顶点本身、以及其相邻的顶点的几何信息，结合时间和相关函数，就能低成本的计算出其运动轨迹。而对骨骼动画来说，顶点的运动是由更高层的抽象：骨骼节点控制的，在同一根骨骼上的顶点彼此之间需要像刚体那样保持相对位置，在若干骨骼之间的顶点需要主动进行平滑，而顶点的大题形状又必须严格的接受骨骼制约。所以相比其它定点动画，在实现骨骼动画时，还必须而外维护每个骨骼的数据，并将顶点与骨骼联系到一起。

在目前主流的实现方案中，骨骼本身被视为近似刚体，它们的变换矩阵被以一定的顺序安排在了常量数组中，每个骨骼都包含一个特有的编号，用来查找其对应的变换矩阵。接下来，每个顶点需要知道与它相关的骨骼的编号，以保证它能够跟随骨骼运动。因为顶点数量原超过骨骼数量，所以顶点与骨骼的对应关系只能被存在顶点数组中。

综合骨骼结构的特点和性能考虑，多数骨骼动画框架中一个顶点最多可关联四个骨骼，顶点数组中对每个顶点储存了一个四维向量，用于记录与这个顶点关联的四个骨骼的编号。同时，顶点数组中另外存储了一个四维向量，用于记录这个顶点关联的四个骨骼分别对该顶点的权重。在计算顶点位置时，会综合这四个骨骼的变换，结合权重进行插值，以此尽可能地保持骨骼交界处的顶点位置是平滑的。这个权重向量的四个分量之和应当为1，这样才能保证顶点的位置不会相对于骨骼超前或滞后。

在动画运行过程中，应用阶段由CPU负责计算了骨骼的位置和对应的变换矩阵，CPU会将它们输入到常量数组中；而GPU则在几何阶段实时的计算网格上所有顶点的位置，使其大体上围绕骨骼进行运动，最终呈现出角色在运动的效果。

### 1.2 动作技术原理

#### 1.2.1 离线动作开发概述

##### 动作设计

动作设计主要是由艺术人员针对动作应用的场景对动作进行设计的过程。这个过程直接决定了整个动作开发流程的方向和最终的动作演出效果。

##### 动作片段制作

这是离线动作的开发过程，这个步骤通常由艺术人员主导，但有时也需要硬件和软件技术人员的辅助。这个过程产出的动作片段会成为后续开发流程中的主干，用户绝大部分时间接触的都将是离线开发的动作片段。

#### 1.2.2 实时动作开发概述

##### 动作流数据组织

动作流数据组织主要是由技术人员开发的决策机实现的，这个系统将离线开发的动作片段按照指定的顺序、时间、相位进行播放，并按照某个规则对多个片段进行混合。这个过程主要体现的是角色对输入的实时响应，或角色对固定动画流程的演绎。

##### 运动学数据

运动学数据主要是由技术人员通过运动学约束控制器提供的，这个系统将超出离线动作片段的限制实时的生成动作。这个过程主要体现的是角色对虚拟场景的主动响应。

##### 物理学数据

物理学数据主要是由技术人员通过物理学约束提供的，这个系统将实时审视动画播放的物理属性，并对场景中的物理信息进行响应。这个过程主要体现的是角色对虚拟场景的被动响应。

### 参考文献

[1] Roget P. Explanation of an Optical Deception in the Appearance of the Spokes of a Wheel Seen through Vertical Apartures. *Philosophical Transactions of the Royal Society of London*. 1825; 115; 131-140 (presented in 1824).

[2] Burns P. *The Complete History of the Discovery of Cinematography*. 2000; In: *http://www.-precinemahistory.net/introduction.htm*; 2000.

[3] Conrac Corp. *Raster Graphics Handbook*. New York: Van Nostrand Reinhold; 1985.

[4] Ehrenstein WH. Basics of Seeing Motion. *Arq Bras Oftalmal*. 2003; 66(5): 44-53 Sept./Oct.

[5] Smith A R. Alpha and the history of digital compositing[J]. URL: http://www.alvyray.com/Memos/7\_alpha.pdf, zuletzt abgerufen am, 1995, 24: 2010.

[6] Joey D V. *Hello Triangle*[EB/OL]. URL: https://learnopengl.com/Getting-started/Hello-Triangle.

[7] Akenine-Moller T, Haines E, Hoffman N. Real-time rendering[M]. AK Peters/crc Press, 2019: 14-22.

## 第二章 动作定义

### 2.1 空域表达

#### 2.1.1 骨骼层次结构

##### 坐标系

三维软件中的坐标系是一个老生常谈的问题，其中我们最常使用的坐标系是三维笛卡尔坐标系，也叫作三维直角坐标系。

三维直角坐标系由三维空间中的一个原点和三个互相正交的基向量组成，通常我们用来表示这个元点，用、和来表示者三个基向量，一个三位直角坐标系也可以被写成坐标系。

空间中任一点在一个坐标系中具有一个坐标。对于在坐标系中的坐标，它所表示的位置相当于。

空间中任意两点间的差在一个坐标系中可以表示为一个向量。对于在坐标系中的向量，它所表示的方向相当于从原点指向坐标的方向。

三维直角坐标系可以被分为左手坐标系或右手坐标系，一个坐标系属于左手坐标系还是右手坐标系可以称为坐标系的手性。沿着轴正方向看向坐标系，若轴逆时针旋转90°后与轴重合，则坐标系属于左手坐标系；若轴顺时针旋转90°后与轴重合，则坐标系属于右手坐标系。坐标系手性同时定义了旋转的方向。在左手坐标系中，绕某轴旋转的正方向由左手法则定义，即从该轴正方向向负方向看去的顺时针方向；而右手坐标系中，绕某轴旋转的正方向由右手法则定义，即从该轴正方向向负方向看去的拟时针方向。判断叉积结果方向时也会用到左手或右手定则。

在常用的三维软件中，Unity、UE、ZBrush和Cinema4D都是左手坐标系，而Maya、Houdini、Godot、3DsMAX、Blender和AutoCAD都是右手坐标系。

##### 世界空间与局部空间

世界空间坐标系是一个全局通用的坐标系，我们需要用到世界空间中的坐标和向量来定义其它坐标系的原点和基向量。

世界空间的基向量方向本身没有意义，但为了方便开发人员理解，主流的物理引擎通常会人为规定世界空间中的向量或向量作为重力的方向，这便形成了Y-up坐标系和Z-up坐标系。

在常用的三维软件中，Unity、ZBrush、Cinema4D、Maya、Houdini、Godot都是Y-up坐标系，而UE、3DsMAX、Blender和AutoCAD都是Z-up坐标系。

由于在骨骼动画中，我们通常不关心关节在世界坐标系中的状态，而更关心关节相对于另一个关节的位置，如肘关节相对于肩关节、腕关节相对于肘关节等。因此我们在定义物体位置时通常不希望使用世界坐标，而是希望使用物体相对于那个相邻物体的状态。

显然，我们可以以每个物体的中心定义为一个坐标系的原点，以物体的右方向为轴、物体前方向为轴、物体上方向为轴构建坐标系(这里仅以Unity的坐标系类型为例)。这样，忽略物体所代表的其它信息，仅考虑其几何关系，我们将每个物体都抽象为了一个坐标系。这样，在考察B物体相对于A物体的几何关系时，我们可以计算B物体在A物体的抽象坐标系下的坐标。

在这种将物体抽象为坐标系的模式下，为了在A物体运动时能继续保持B物体和A物体的相对关系不变(比如在大臂运动时使其带动小臂)，我们将B物体的抽象坐标系直接定义在A物体的抽象坐标系之下。这样在A物体运动时，由于B物体的坐标系已经定义在了A物体之下，所以B物体相对于A物体的位置并不会改变。同时，定义在B物体的抽象坐标系之下的其它物体也会随着B物体一起被A物体带动。在这种模式下，我们将A物体的抽象坐标系称为B物体的局部空间，将A物体称为B物体的父物体，将B物体称为A物体的子物体，B物体在A物体的抽象坐标系下的坐标被称为B物体的局部空间坐标。

以此类推，根据这种坐标系之间的嵌套关系，我们可以将场景中所有物体的关系定义为一棵坐标嵌套树，这颗树的根节点是世界空间坐标系，那些不会被动跟随任何其他物体的运动而运动的物体(如盆骨骨骼)会以世界空间坐标系为父结点。

显然，顺着这个坐标系的嵌套关系向根节点搜索，每个物体都可以获取到自己的世界空间坐标，且每个物体在世界空间中的最终位置都会受到它所有的父结点和祖先结点的影响，这种影响同样也会顺着这棵树传递到它的所有子结点当中去。不止是位置，旋转和缩放等几何变换也会顺着坐标系树传递到所有子结点中去。

接下来我们定义一棵简单的双向树来储存这个空间中的所有物体的几何信息：

public class CoordinateTreeNode  
{  
 CoordinateTreeNode parent; //树的父结点  
 CoordinateTreeNode[] childs; //树的子结点  
   
 Vector3 Origin; //物体抽象坐标系定义在父坐标系下的原点  
 Vector3 AxisX; //物体抽象坐标系定义在父坐标系下的X轴  
 Vector3 AxisY; //物体抽象坐标系定义在父坐标系下的Y轴  
 Vector3 AxisZ; //物体抽象坐标系定义在父坐标系下的Z轴  
}  
  
public class WorldCoordinate  
{  
 CoordinateTreeNode[] childs;  
}

#### 2.1.2 变换矩阵

##### 齐次坐标

变换是按照某种规则，将顶点或向量进行有针对性的修改的过程。简单来说，将一个坐标从一个坐标系转换到另一个坐标系，已经将一个顶点移动到另一个位置就是最基础的变换。

变换分为线性变换和非线性变换，所谓线性变换，就是可以通过一个变换矩阵来对顶点的坐标进行可逆的修改，也就是满足以下公式的变换：

旋转和缩放都属于线性变换，但平移不属于线性变换。

在线性代数中我们知道，n×n的矩阵是可以用于实现n维线性变换的。由于平移不属于线性变换，所以我们没办法用3×3的矩阵来处理平移。于是我们引入了仿射变换(affine transform)，通过将三维顶点坐标引入四维空间，我们就可以通过4×4的矩阵来解决平移问题了。而为了实现仿射变换而引入的四维空间，就被称为齐次坐标空间(homogeneous space)。

在齐次坐标空间中，坐标的第四个分量w被用于实现平移，设为常量1。而向量是不需要平移的，所以分量w被设为常量0。

在实际工作中，要先保证顶点处于其次坐标空间下，再对其进行变换。由于三维向量左乘四维矩阵时，第四个维度会被默认为0，这导致对三维的顶点进行变换时，会用应用向量变换的效果，而非顶点变换的效果。换而言之，会保留旋转和缩放变换而丢弃平移变换。这可能不符合开发者的意图，而导致一个异常。

##### 平移变换

平移矩阵定义：

其中tx、ty、tz分别表示在x、y、z轴上平移的长度。

平移矩阵作用在顶点(w分量为1)上时：

平移矩阵作用在向量(w分量为0)上时：

##### 旋转变换

绕过点(a,b,c)方向为(x,y,z)的轴旋转角度θ的旋转矩阵定义如下：

这个矩阵很难记忆，但几乎每个图形学相关的类库中都会含有这个矩阵。推导过程略过。

作为旋转矩阵的特例，绕x、y、z轴旋转角度θ的旋转矩阵分别是：

##### 缩放变换

缩放矩阵定义：

其中kx、ky、kz分别表示在x、y、z轴上缩放的比例。

缩放矩阵作用在顶点上时：

##### 复合变换

一般来说，我们会使用一个变换矩阵表达一个物体在场景中的状态，即位置、旋转状态和缩放状态。这个变换矩阵象征着物体从局部坐标中心(在坐标系中与坐标轴重合，保持默认旋转和默认缩放)变换到目标状态(应被渲染的平移、旋转、缩放)的复合变换矩阵。我们必须保证复合变换的结果符合人的直观预期，所以必须保证复合变换的顺序。

复合变换的顺序是：**先缩放、再旋转、最后平移**，这个变换顺序是基于几何变换的应用场景的。

* 若先进行平移再进行缩放，则缩放会计算平移后的顶点坐标，导致平移的距离随缩放的倍率变化，不复合人的直观预期。
* 若先进行平移再进行旋转，则旋转会计算平移后的顶点坐标，导致平移的方向被改变，不符合人的直观预期。
* 若先进行旋转再进行缩放，则缩放时使用的是旋转后的顶点坐标，导致物体不是沿模型空间的XYZ轴进行缩放，而是沿世界坐标轴方向缩放，这在大部分场合下都不符合人的直观预期。

复合变换矩阵等于子变换的连续乘积，由于我们使用的是列矩阵，所以阅读顺序是从右向左，如下：

##### 手性变换

在跨软件的开发过程中，经常遇到两个软件对手性的定义不一致的情况，此时我们需要在拷贝变换数据时对手性取反以保证物体在两个软件中的表现一致。

对于平移矩阵，在手性取反时应将其中一个维度的平移方向取反，例如将本应向平移的矩阵，改为向方向平移。选择具体使用哪一个维度并没有影响，但需要确保场景中所有需要手性取反的物体应用同一个维度。

对于旋转矩阵，在手性取反时需要对旋转矩阵的一整行取反。注意，这一行应当与平移矩阵选择的取反维度相同。比如平移矩阵在手性取反时选择了取反Z轴，那么在使用绕X轴旋转的和绕Y轴旋转的旋转矩阵时，应当取反旋转矩阵第三行。

缩放矩阵不会受到手性的影响。

综上所述，对于任意一个几何变换或空间变换矩阵，如果需要对手性进行取反，只需要选择一个轴，将矩阵的对应一行全部取反。

##### 坐标空间变换

假设有父坐标系和子坐标系，我们已知子坐标系在父坐标系中的坐标，子坐标系在父坐标系P中的旋转，子坐标系在父坐标中的缩放，试求子坐标系下坐标所表示的点，转换到父坐标系中之后的坐标，并反过来求父坐标系下坐标所表示的点，转换到子坐标系中之后的坐标。我们知道，矩阵具有变换的功能，所以我们列出下列的式子：

由于父子坐标空间变换是一对反向变换，所以显然矩阵Mc-p和矩阵Mp-c是一对逆矩阵，这样我们只需要解出其中一个矩阵即可，现在我们来求解由子坐标向父坐标转换的变换矩阵。我们设。

在子坐标系C中，C的原点坐标为O(0,0,0)，C的三条基向量分别为、、，则：

将其中的所有坐标和矢量转换到父坐标P下，设在父坐标系P中，子坐标系C的三条基向量分别为、、，则：

将这个算式写成矩阵的格式，就会变成这样：

将这个算式扩展到齐次坐标空间中，就会变成：

接下来我们考虑、、的求法：我们已经知道子坐标系在父坐标系中的旋转和在中的缩放，对C的基向量在中的表示来说，相当于父坐标系P的x轴基向量先经过缩放再经过旋转的结果，以此类推的分析子坐标系的另外两个基向量，可以得到：

所以：

综上所述，一个从空间B转换到空间A的空间变换矩阵，相当于将空间A的坐标系通过几何变换转移到空间B的几何变换矩阵，相当于空间A的坐标系在空间B的几何变换矩阵的逆矩阵。

#### 2.1.3 旋转表达

##### 双向量法

双向量法是直接基于骨骼层次结构的一种旋转表达方式。

根据我们对坐标系嵌套关系的阐述，旋转本质上代表的是子坐标系的三个基向量在父坐标系中的方向。所以一个旋转显然可以通过三个互相正交的向量来表示。

我们继续对表示旋转所需的数据进行压缩，根据三个基向量的正交性，我们可以在保证旋转数据不丢失的情况下只记录三个基向量中的两个，这便是双向量法的来历。通常来说我们记录的会是表示上方向的基向量和表示前方向的基向量，因为这两个方向更方便开发人员进行理解，相对更加直观。

由于在表示旋转时，基向量的长度没有价值，我们可以使用单位向量来表示双向量。再进一步，由于单位向量已经位于一个球的表面，我们可以使用球坐标系来表示一个单位向量，所以我们可以使用一个四维浮点向量来表示一个旋转。

但双向量法在计算几何中难以被广泛应用，是因为存在两个致命的问题：

* 在修改旋转时，双向量法必须使用额外的合法性判断来确保两个输入向量正交。
* 在对两个旋转进行插值时，双向量法不适用于线性插值，并且很难保证插值结果的合法性。

难以实现插值是动画领域无法容忍的硬伤，因为动画工作流的几乎每个部分都需要用到大量的插值计算。

##### 欧拉角与顺规

与双向量不同，欧拉角表达的是一种过程量而非结果量。

欧拉角是一种直观的利用三维向量表达旋转的方法。它源于刚体动能计算，后来广泛用于航空业，随后被计算机领域引入描述旋转。欧拉角最大的特点在于它表达的是一种旋转的过程而非旋转的最终状态。欧拉角将子坐标系的基向量与父坐标系的基向量方向重合的状态作为初始状态，记录了它沿父坐标系的X轴旋转弧度，沿Y轴旋转弧度，沿Z轴旋转弧度后的结果。

我们引入航空领域的一种定义来更形象的解释欧拉角：在航空领域中，将Y轴作为上方向，X轴作为右方向，Z轴作为前方向。我们将欧拉角中的分量称为俯仰角(Pitch)，因为绕X轴旋转会使物体的面向方向朝上或下旋转；将欧拉角中的分量称为偏航角(Yaw)，因为绕Y轴旋转会使物体的面向方向朝左或右旋转；将欧拉角中的分量称为桶滚角(Roll)，因为绕Z轴旋转不会改变物体面向的方向，而是绕面向的方向滚动。

注意，欧拉角的旋转需要明确旋转轴和旋转顺序，欧拉角的旋转顺序也被称为顺规。在三维空间中，我们把旋转分为内旋(又称局部旋转或动轴旋转)和外旋(又称惯性旋转或静轴旋转)。其中内旋指的是绕自己当前的虚拟坐标轴旋转，每次旋转的旋转轴在上一次旋转中改变，内旋通常在顺规中记为小写字母xyz；外旋指的是绕固定轴，通常是父坐标系的坐标轴旋转，旋转轴不因为上一次旋转改变，通常在顺规中记位大写字母XYZ。航空领域的顺规被记为ZXY，这同时也是Unity的顺规。

欧拉角也存在三个主要的问题：

* 欧拉角没有统一的行业标准。同一个欧拉角向量在使用不同顺规时表达完全不同的旋转。甚至有些平台会使用刚体物理中对欧拉角的定义，这使得在跨平台时不得不重新计算欧拉向量的值，其跨平台性非常差。
* 欧拉角的三个分量在某些顺规下并不总是正交的，非正交性导致动态欧拉变换存在俗称万向节死锁的问题，这使得欧拉变换的结果具有不可预见性，程序员很难准确的预测通过某个欧拉变换后物体的新朝向。
* 欧拉角仍然不适用于插值计算，对两个欧拉角插值得到的并不是两个旋转的中间量。

给定一个欧拉变换(α，β，γ)，其旋转矩阵如下：

内旋在旋转时对应的是旋转矩阵的右乘，而外旋对应的是旋转矩阵的左乘。显然，从旋转矩阵的角度上看，外旋的ZXY和内旋的yxz没有区别。

##### 轴角与四元数

轴角也是一种旋转过程量。轴角表达的是沿轴旋转弧度的结果。由于轴向量可以是单位向量，所以很多时候可以将弧度值乘在单位化的轴向量上，得到被压缩的三维轴角向量，其中。

轴角相比欧拉角更加直观，这使得轴角被更深入的应用在了物理领域。角速度和角动量的计算通常都基于轴角，且在计算角速度和角动量时，可以直接使用三维轴角的向量和或向量差作为角速度和角动量的变化量，就像线速度和线动量的计算方式那样。这是由于，物理系统中的旋转量不是单纯对旋转状态的累积，而是对旋转趋势的累积。

轴角存在插值算法，即对轴向量和旋转弧度分别进行插值。

四元数是一种基于轴角思想但由于轴角的旋转表示法，从数学角度讲四元数是一种超复数，它有一个实部和三个虚部，其所在的数域被称为：

$$q=w+xi+yj+zk=s+\vec{v}\tag1$$

四元数的叉乘，的几何意义是在旋转的基础上应用旋转，四元数符合乘法结合律，而不符合乘法交换律：

$$\begin{split}q\_1\times q\_2
&=s\_1s\_2+s\_1\vec{v\_2}+s\_2\vec{v\_1}+[(x\_1i+y\_1j+z\_1k)\cdot(x\_2i+y\_2j+z\_2k)]\\
&=s\_1s\_2+s\_1\vec{v\_2}+s\_2\vec{v\_1}+[(y\_1z\_2-y\_2z\_1)i+(x\_2z\_1-x\_1z\_2)j+(x\_1y\_2-x\_2y\_1)k-x\_1x\_2-y\_1y\_2-z\_1z\_2]\\
&=s\_1s\_2+s\_1\vec{v\_2}+s\_2\vec{v\_1}+(\vec{v\_1}\times\vec{v\_2}-\vec{v\_1}\cdot\vec{v\_2})\\
&=(s\_1s\_2-\vec{v\_1}\cdot\vec{v\_2})+(s\_1\vec{v\_2}+s\_2\vec{v\_1}+\vec{v\_1}\times\vec{v\_2})
\end{split}
\tag{2}$$

四元数的点积等于两个旋转之间的夹角的余弦值：

$$p\cdot q=s\_1s\_2+\vec v\_1\cdot\vec v\_2=\cos<p,q>\tag 3$$

共轭四元数：

$$若q = s+\vec{v},\, 则\overline{q} = s - \vec{v}\tag4$$

四元数的逆：

$$q^{-1}=\frac{\overline{q}}{q^2}\tag5$$

$$qq^{-1}=q^{-1}q=1\tag6$$

$$(q^{-1})^{-1}=q\tag7$$

使用四元数旋转一个向量可以被理解为一个函数(从R3到R3的映射)，以将向量旋转到。要想表达一个旋转，必须在旋转过程中保证向量长度、向量夹角和手性不变，即：

$$\mid \Phi(\vec{P})\mid = \mid\vec{P}\mid\tag{8}$$

$$\Phi(\vec{P\_1})\cdot\Phi(\vec{P\_2})=\vec{P\_1}\cdot \vec{P\_2}\tag{9}$$

$$\Phi(\vec{P\_1})\times \Phi(\vec{P\_2})=\Phi(\vec{P\_1}\times \vec{P\_2})\tag{10}$$

扩展为一个的映射，要求，于是我们可以重写公式(9)：

$$\Phi(\vec{P\_1})\cdot \Phi(\vec{P\_2})=\Phi(\vec{P\_1}\cdot \vec{P\_2})\tag{11}$$

把和当作标量部分为0的四元数，使，将代入公式(2)得：

$$\Phi(\vec{P\_1})\Phi(\vec{P\_2})=-\Phi(\vec{P\_1})\cdot \Phi(\vec{P\_2})+\Phi(\vec{P\_1})\times \Phi(\vec{P\_2})\\
\vec{P\_1}\vec{P\_2}=-\vec{P\_1}\cdot \vec{P\_2}+\vec{P\_1}\times \vec{P\_2}$$

因此我们可以(10)和(11)合成成新的公式：

$$\begin{split}
\Phi(\vec{P\_1})\Phi(\vec{P\_2})&=-\Phi(\vec{P\_1})\cdot\Phi(\vec{P\_2})+\Phi(\vec{P\_1})\times\Phi(\vec{P\_2})
\\&=-\vec{P\_1}\cdot \vec{P\_2}+\Phi(\vec{P\_1} \times \vec{P\_2})
\\&=\Phi(-\vec{P\_1}\cdot \vec{P\_2}+\vec{P\_1}\times \vec{P\_2})
\\&=\Phi(\vec{P\_1}\vec{P\_2})
\\\end{split}
\tag{12}$$

我们给出一个满足(8)和(12)的函数：

$$\Phi\_\vec{q}(\vec{P})=\vec{q}\vec{P}\vec{q}^{-1},\,\,\vec{q}\in H\tag{13}$$

先证明(13)满足(8)：

再证明(13)满足(12)：

易证明对任意，，所以不妨设q为单位四元数，则：

$$\vec{q}^{-1} =\overline{\vec{q}}=s-\vec{v}\tag{14}$$

在此我们补充一个定理，对于任意，有：

$$\vec{P}\times (\vec{Q} \times \vec{P}) = \vec{P}\times \vec{Q}\times \vec{P}=\vec{P}^2\vec{Q}-(\vec{P}\cdot \vec{Q})\vec{P}\tag {15}$$

于是我们得到：

$$\begin{split}\vec{q}\vec{P}\vec{q}^{-1}&=(s+\vec{v})\vec{P}(s-\vec{v})\\
&=(-\vec{v}\cdot \vec{P}+s\vec{P}+\vec{v}\times \vec{P})(s-\vec v)\\
&=-s\vec v\cdot \vec{P}+s^2\vec{P}+s\vec v\times \vec{P}+(\vec v\cdot \vec{P})\vec v-s\vec{P}\vec v-(\vec v\times \vec{P})\times\vec v\\
&=s^2\vec{P}+2s\vec v\times \vec{P}+(\vec v\cdot \vec{P})\vec v-\vec v\times \vec{P}\times \vec v\\
&=(s^2-\vec v^2)\vec{P}+2s\vec v\times \vec{P}+2(\vec v\cdot \vec{P})\vec v
\end{split}\tag{16}$$

设，其中是非0常数，是旋转轴所在的单位向量，重写(16)为：

$$\vec{q}\vec{P}\vec{q}^{-1}=(s^2-t^2)\vec{P}+2st\vec A\times \vec{P}+2t^2(\vec A\cdot \vec{P})\vec A\tag{17}$$

我们引入向量绕轴旋转度的公式：

$$\vec{P'} =\vec{P}cos\theta+(\vec{A}\times\vec{P})sin\theta+\vec{A}(\vec{A}\cdot\vec{P})(1-cos\theta)\tag{18}$$

比较(11)与(12)可以得到：

$$\left \{ \begin{array}{c}
s^2-t^2=cos\theta\\2st=sin\theta\\2t^2 =1-cos\theta
\end{array}\right.\tag{19}$$

解方程组得：

$$\left \{ \begin{array}{c}
t=sin\frac{\theta}{2}\\s=cos\frac{\theta}{2}
\end{array}\right.\tag{20}$$

于是我们解得：

$$\begin{split} \vec{q}&=s+\vec v\\
&=s+t\vec{A}\\&=cos\frac{\theta}{2}+\vec{A}sin\frac{\theta}{2}
\end{split}\tag{21}$$

在此我们可以形象的理解四元数，即绕着旋转角度的旋转，这一点将四元数和轴角重新联系了起来。通常来说，使用欧拉角向量为的旋转状态为默认状态，则在默认状态上叠加一个旋转，就可以表达一个物体的唯一旋转状态。同时，根据这一特性，我们也可以推导得知，四元数和它的相反数表达同一个旋转。

运用四元数实现旋转比旋转矩阵更便捷，两个四元数相乘只需要16次乘加，而两个4×4的矩阵需要64次乘加。

接下来求解四元数对应的旋转矩阵，首先将(11)用矩阵形式表现：

$$\vec q\vec P\vec {q}^{-1}=\begin{bmatrix}s^2-t^2&0&0\\0&s^2-t^2&0\\0&0&s^2-t^2
\end{bmatrix}\vec P+\begin{bmatrix}0&-2stA\_z&2stA\_y\\2stA\_z&0&-2stA\_x\\-2stA\_y&2stA\_x&0
\end{bmatrix}\vec P+\begin{bmatrix}2t^2A\_x^2&2t^2A\_xA\_y&2t^2A\_xA\_z\\2t^2A\_xA\_y&2t^2A\_y^2&2t^2A\_yA\_z\\2t^2A\_xA\_z&2t^2A\_yA\_z&2t^2A\_z^2
\end{bmatrix}\vec P\tag{22}$$

用来重现上面的矩阵：

$$\vec q\vec P\vec {q}^{-1}=\begin{bmatrix}w^2-x^2-y^2-z^2&0&0\\0&w^2-x^2-y^2-z^2&0\\0&0&w^2-x^2-y^2-z^2
\end{bmatrix}\vec P
+\begin{bmatrix}0&-2wz&2wy\\2wz&0&-2wx\\-2wy&2wx&0
\end{bmatrix}\vec P+\begin{bmatrix}2x^2&2xy&2xz\\2xy&2y^2&2yz\\2xz&2yz&2z^2
\end{bmatrix}\vec P\tag{23}$$

由于是单位四元数，可得：

$$w^2-x^2-y^2-z^2=(1-x^2-y^2-z^2)-x^2-y^2-z^2=1-2x^2-2y^2-2z^2\tag{24}$$

得到对应的旋转矩阵为：

$$R\_q=\begin{bmatrix}1-2y^2-2z^2&2xy-2wz&2xz+2wy\\2xy+2wz&1-2x^2-2z^2&2yz-2wx\\
2xz-2wy&2yz+2wx&1-2x^2-2y^2
\end{bmatrix}\tag{25}$$

由欧拉角转化成四元数的公式如下：

$$q=\begin{bmatrix}w\\x\\y\\z\end{bmatrix}=
\begin{bmatrix}
\cos(\phi/2)\cos(\theta/2)\cos(\psi/2)+\sin(\phi/2)\sin(\theta/2)\sin(\psi/2)\\
\sin(\phi/2)\cos(\theta/2)\cos(\psi/2)-\cos(\phi/2)\sin(\theta/2)\sin(\psi/2)\\
\cos(\phi/2)\sin(\theta/2)\cos(\psi/2)+\sin(\phi/2)\cos(\theta/2)\sin(\psi/2)\\
\cos(\phi/2)\cos(\theta/2)\sin(\psi/2)-\sin(\phi/2)\sin(\theta/2)\cos(\psi/2)\\
\end{bmatrix}\tag{26}$$

由四元数转化成欧拉角的公式如下：

$$\begin{bmatrix}\phi\\\theta\\\psi\end{bmatrix}=\begin{bmatrix}
\arctan\frac{2(wx+yz)}{1-2(x^2+y^2)}\\\arcsin(2(wy-zx))\\\arctan(\frac{2(wz+yz)}{1-2(y^2+z^2)})
\end{bmatrix}\tag{27}$$

#### 2.1.4 隐式表面模型

##### SMPL参数模型

### 2.2 时域表达

#### 2.2.1 插值技术

##### 四元数插值

四元数的插值方法主要有三种，分别记作Lerp(线性插值)、SLerp(球面线性插值)和Squad(球状三次样条插值)。

Lerp是对四元数套用四元矢量的线性插值方法。Lerp方法相比传统的线性插值，需要额外进行一次归一化：

Lerp的问题在于，他没有考虑四元数作为四维超球面(hypershpere)的点的特性。Lerp是沿超球的弦插值的，而不是在球面上插值，这样会导致当β以恒定速度改变时，旋转动画不能以恒定角度进行。旋转在两端看起来较慢，而在动画中间就会较快。

解决此问题的方法是，引入正弦函数以在四维超球面的球上进行插值，即SLerp：

其中是四元数和的夹角，可以通过点乘来获得：

在球面插值的过程中需要注意两个特殊情况：

* 如果，则有。由于和实际上表达同一旋转，所以需要考虑是否有必要对或取反，使其进行插值的差值最小化，避免不必要的自旋转。
* 如果，则可以使用线性差值来进行优化，因为此时。可以在进行球面插值前先运算以判断能否使用线性插值进行优化。

球状三次样条插值可以用于平滑关键帧动画当中的旋转，这个插值算法通过预测四元数序列的切线方向，形成了四个控制点，并对其进行了三次样条插值。这可以避免在动画中在控制点处产生角度的突变：

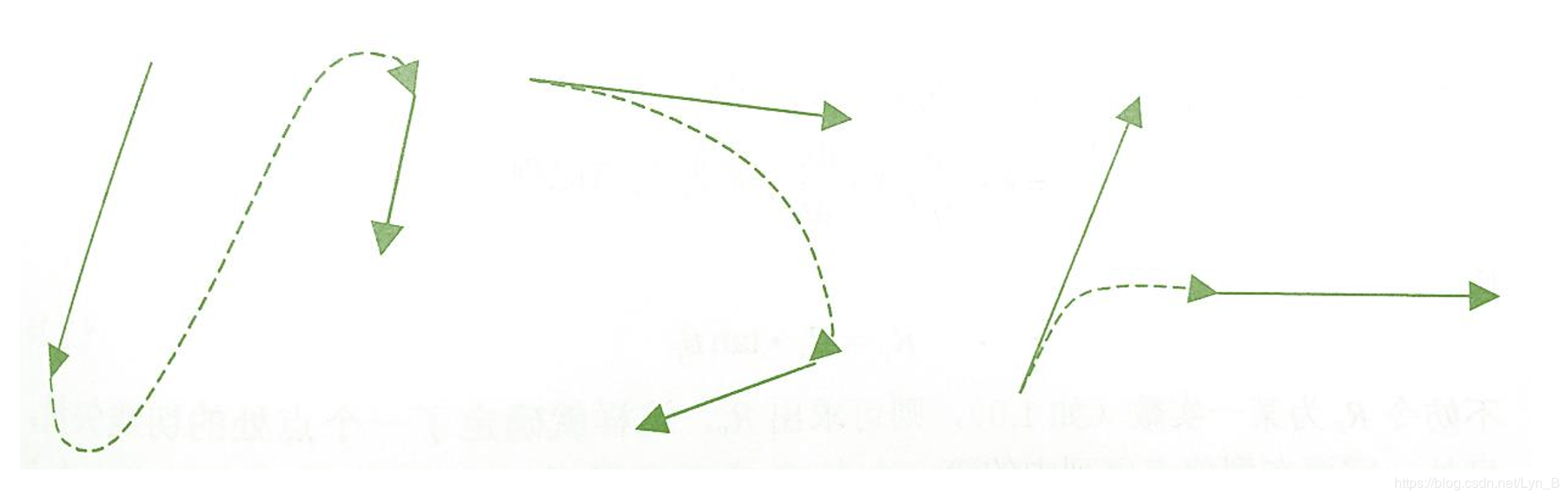
$$f\_{squad}(q\_i,q\_{i+1},t)=f\_{slerp}(f\_{slerp}(q\_i,q\_{i+1},t),f\_{slerp}(a\_i,a\_{i+1},t), 2t(1-t))\\
a\_i=q\_i\exp[-\frac{\log(q\_i^{-1}q\_{i-1})+\log(q\_i^{-1}q\_{i+1})}{4}]$$

##### 动作插值

#### 2.2.2 插值曲线

##### 埃尔米特曲线

Hermit曲线给定曲线两个端点的位置矢量、和切线矢量、，通过插值获得曲线在定义域[0,1]上的三次参数方程：



埃尔米特曲线

我们设目标参数方程为，则有，将0和1代入t易得：

$$\left \{ \begin{array}{c}
\ P(0)&=&d\\\ P(1)&=&a+b+c+d\\\ P'(0)&=&c\\\ P'(1)&=&3a+2b+c
\end{array}\right.\tag{14}$$

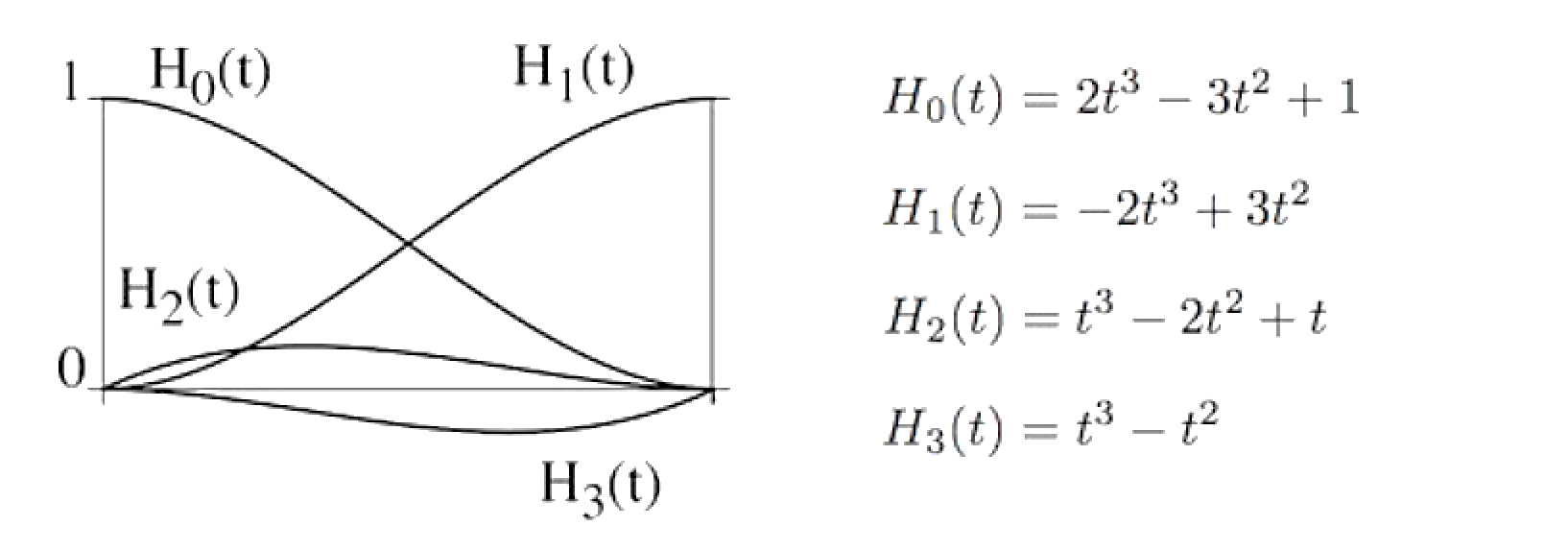
现在我们设，则有：

$$\begin{bmatrix}
h\_0\\h\_1\\h\_2\\h\_3
\end{bmatrix}=\begin{bmatrix}
0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0
\end{bmatrix}\begin{bmatrix}
a\\b\\c\\d
\end{bmatrix}\\\Rightarrow\begin{bmatrix}
a\\b\\c\\d
\end{bmatrix}=\begin{bmatrix}
2&-2&1&1\\-3&3&-2&-1\\0&0&1&0\\1&0&0&0
\end{bmatrix}\begin{bmatrix}
h\_0\\h\_1\\h\_2\\h\_3
\end{bmatrix}$$

我们将参数方程写成矩阵形式：

将列向量abcd的表达式转置，代入上式，得到：

将后两项乘起来可以得到：



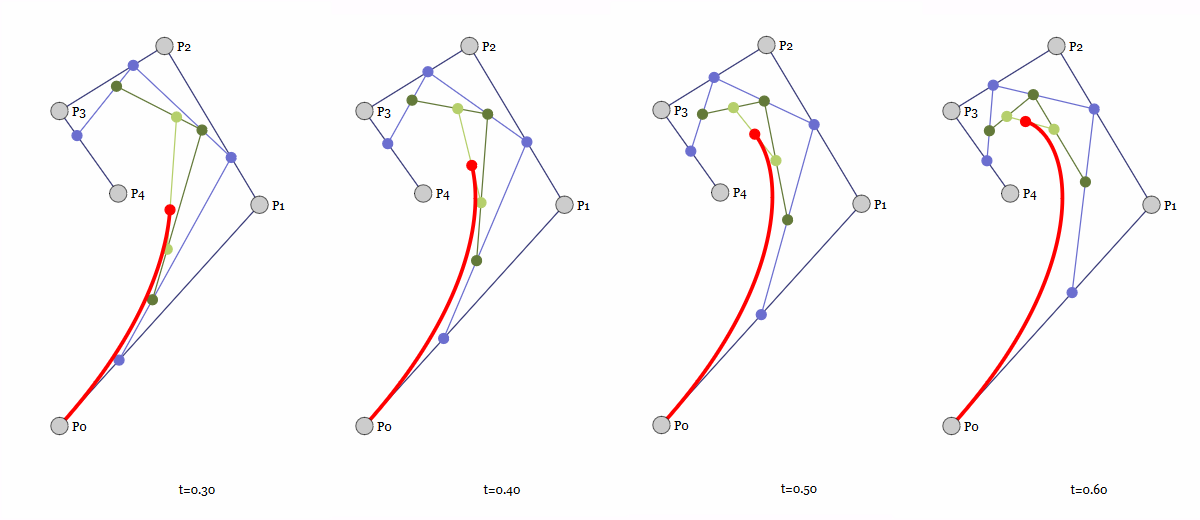
Hermit

$$P(t)=\sum\_{i=0}^3h\_iH\_i(t)\\$$

其中，，，0。

##### 贝塞尔曲线

贝塞尔曲线B(t)也是参数格式的曲线，n阶贝塞尔曲线是n次参数方程，一个四阶贝塞尔曲线的参数t在不同取值下的原理图如下，将t从0到1时B(t)返回的点连接起来，就可以得到贝塞尔曲线的图像：



贝塞尔曲线完全由其控制点决定其形状，n个控制点对应着n-1阶的贝塞尔曲线，并且可以通过递归的方式来绘制。贝塞尔曲线是典型的递归法曲线。

一阶贝塞尔曲线就是对两个控制点进行线性插值，其参数方程为：

二阶贝塞尔曲线由三个控制点组成，假设三个控制点依次为P0、P1、P2，先分别对P0P1和P1P2两段进行线性插值得到两个新控制点P0'、P1'，将二阶贝塞尔曲线降为一阶：

$$P'\_0=(1-t)P\_0+tP\_1\\
P'\_1=(1-t)P\_1+tP\_2$$

然后再对P0'P1'求线性插值：

三阶贝塞尔曲线由四个控制点组成，假设四个控制点依次为P0、P1、P2、P3，先分别对P0P1、P1P2和P2P3三段求线性插值得到三个新控制点P0'、P1'、P2'，将三阶贝塞尔曲线降为二阶：

$$P'\_0=(1-t)P\_0+tP\_1\\
P'\_1=(1-t)P\_1+tP\_2\\
P'\_2=(1-t)P\_2+tP\_3$$

然后再利用新控制点求二阶贝塞尔曲线：

$$P''\_0=(1-t)P'\_0+tP'\_1\\
P''\_1=(1-t)P'\_1+tP'\_2$$

最后对P0''和P1''进行线性插值：

不难发现，n阶贝塞尔曲线展开式系数是n阶二项式展开系数，可以得到n阶贝塞尔曲线的通项公式：

其中，n阶贝塞尔曲线的第i项系数Bi,n(t)又被记作**Bernstein基函数**。

$$B\_{i,n}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-1}\\
B\_n(t)=\sum\_{i=0}^nB\_{i,n}(t)P\_i,\ t\in[0,1]$$

对n阶贝塞尔曲线进行求导可得：

经观察，n阶贝塞尔曲线的导数显然是n-1阶贝塞尔曲线，其控制点为原曲线控制点的组合。

贝塞尔曲线具有**端点性质**：第一个控制点和最后一个控制点恰好是曲线的起点和重点。

贝塞尔曲线具有**凸包性质**：贝塞尔曲线始终位于**包含了所有控制点的最小凸多边形**中。注意，不是按控制点顺序围成的凸多边形。

贝塞尔曲线具有**端点导数性质**：贝塞尔曲线在P0和Pn处的斜率分别等于直线P0P1和直线Pn-1Pn的斜率，即贝塞尔曲线端点的一阶导数仅同其相邻的一条边向量有关，与更远的各边无关。推广的，贝塞尔曲线端点的r阶导数仅同其相邻的r条边向量有关，与更远的各边无关。

贝塞尔曲线最大的问题在于，高阶贝塞尔曲线的路径不够直观，很难让美术人员利用高阶贝塞尔曲线进行设计。但我们注意到，由于贝塞尔曲线具有端点性质和端点导数性质，二阶和三阶贝塞尔曲线的形状相对直观。所以在实际使用中我们会讨论到将多个贝塞尔曲线尤其是二、三阶贝塞尔曲线拼接成直观曲线的算法。

##### 拼接贝塞尔曲线

我们先聊聊什么是**连续性**。

在讨论曲线连续性时，我们经常见到**Cn连续(参数连续性)**和**Gn连续(几何连续性)**的说法。如果不 同曲线都用参数方程来描述，那么在连接点处n阶导数连续，就被称为Cn连续。在图形学中，一般最高只要求C2连续，因为更高阶的连续已经超出了人眼视觉能识别的层次。但Cn连续有一个致命的缺陷，那就是连续性会随参数的选取而变化，这使得Cn连续不具备普适性，不能用于广泛的学术交流。

为了避免这个缺陷，人们引入了几何连续性和自然参数方程，定义曲线的弧长s作为自然参数，这样任何参数方程都可以转换为自然参数方程，在连接点处自然参数方程的n阶导数连续，就称为Gn连续。在图形学中，一般最高也只要求G2连续。

考察G0，G1，G2连续的几何意义。G0就是满足位置连续，即两曲线端点重合。G1满足两曲线在端点上具有相同的切线方向。G2满足曲线在端点上具有相同的自然参数二阶导矢，也就是对一般参数方程具有相同的曲率矢量。

曲率矢量的求法：

设两条贝塞尔曲线P和Q的参数方程P(t)和Q(t)，它们的控制点序列分别是P0、P1……Pn和Q0、Q1……Qm，则：

* 满足C0和G0的连续条件是P(1)=Q(0)，即Pn=Q0。
* 满足G1的连续条件是Pn-1、Pn=Q0、Q1这**三点共线且顺序排列**。
* 满足C1连续的条件是在满足G1连续的基础上，有。

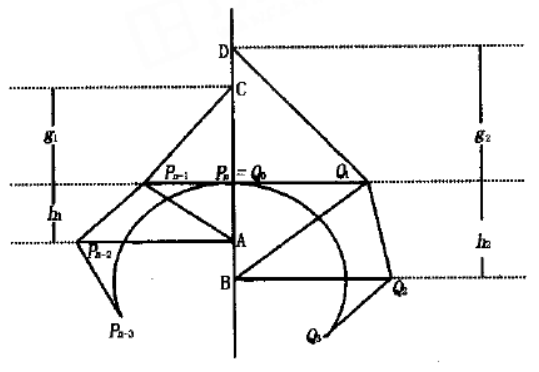
接下来进一步讨论满足C2和G2连续的条件。

根据端点导数性质，C2和G2的达成与Pn-2和Q2有关。记ai=Pi-Pi-1，bi=Qi-Qi-1，则对曲线P和Q的端点求其曲率，也就是曲率矢量的模，得：

$$K\_P(1)=\frac{(n-1)|a\_{n-1}\times a\_n|}{n|a\_n|^3}\\
K\_Q(0)=\frac{(m-1)|b\_1\times b\_2|}{m|b\_1|^3}$$

将Pn-1、Pn=Q0、Q1所在的公切线记为直线l，an-1与an之间的夹角记为θ，b1与b2之间的夹角记为φ，Pn-2到l的距离记为h1，Q2到l的距离记为h2。根据几何关系可知：

$$|a\_{n-1}\times a\_n|= |a\_{n-1}|\times|a\_n|\times\sin\theta
=|a\_{n-1}|\times|a\_n|\times\frac{h\_1}{|a\_{n-1}|}=|a\_n|h\_1\\
|b\_1\times b\_2|= |b\_1|\times|b\_2|\times\sin\theta
=|b\_1|\times|b\_2|\times\frac{h\_2}{|b\_1|}=|b\_1|h\_2\\$$



过端点Pn=Q0作公切线l的垂线v，过端点Pn-2和Q2分别作l的平行线交v于点A、B。连接APn-1、BQ1，过点Pn-1作线端APn-1的垂线CPn-1交v于C，过点Q1作线端BQ1的垂线DQ1交v于D。记C到l的距离为g1，D到l的距离为g2。

在直角三角形ACPn-1和直角三角形BDQ1中，有：

$$a\_n^2=h\_1g\_1\\
b\_1^2=h\_2g\_2$$

于是将曲率变形为：

$$K\_P(1)=\frac{n-1}{n}\frac{|a\_n|h\_1}{|a\_n|^3}=\frac{n-1}{n}\frac1{g\_1}\\
K\_Q(0)=\frac{m-1}{m}\frac{|b\_1|h\_2}{|b\_1|^3}=\frac{m-1}{m}\frac1{g\_2}$$

满足G2的条件是KQ(0)=βKP(1)，此时：

满足C2的条件是β=1且两曲线具有公共的切平面，要满足β=1即：

在特殊条件下，若m=n，则满足β=1的条件是g1=g2。

要满足两曲线具有公共切平面，则点Pn-2、Pn-1、Pn=Q0、Q1、Q2五点必须共面，并且点Pn-2和点Q2应位于另三个顶点所在公切线的同侧。

##### 自然参数贝塞尔曲线

我们已经得到了贝塞尔曲线的参数方程，但显然，如果将参数t作为时间变量，将物体沿曲线运动，我们很难得到一个匀速的运动效果。事实上，在靠近控制点处运动的较慢，而在曲线中段则运行的较快。同时，我们也很难在贝塞尔曲线的拼接处获得良好的平滑速度。

我们可以先求得B(t)相对于t的速度公式，我们先将t看作时间变量，在单个方向上对t求导就可以得到

现在我们以二阶贝塞尔曲线为例，记，将左式展开：

注意到，A为完全平方式，我们设，则。同理我们记，则有，。经整理得：

写成向量形式即：

根据这个公式得出贝塞尔曲线的长度公式：

显然，整条曲线的长度为L(1)，我们求变量t的函数，使得，则。由于对L(t)求逆函数几乎不可能，在这里我们利用L(t)的导数v(t)，使用牛顿切线法求出离散的近似解：

其中是每次前进的长度，如果按来计算前进的长度则：

有了上面的经验，我们再来求三阶贝塞尔曲线的，记，将左式展开得：

观察到，A和E为完全平方式，我们取，则。取，则。注意到，D是β的倍数，我们取，有。类似的，又得到。

C的推导过程比较繁琐，但核心思路如下：设，将C配方得到与a、b、c、a+b、b+c、a+b+c有关得表达式，然后展开得到，易得：

最后整理整个公式：

写成向量形式即：

根据以上公式，我们可以通过一个数据结构Segment来保存一段3D三阶贝塞尔曲线：

public class Segment  
{  
 public float Length => GetLength();  
 public Vector3[] controllers = new Vector3[4];  
  
 public Vector3 this[uint i]  
 {  
 get  
 {  
 return controllers[i];  
 }  
 set  
 {  
 controllers[i] = value;  
 }  
 }  
  
 public Vector3 Value(float t)  
 {  
 return (1 - t) \* (1 - t) \* (1 - t) \* controllers[0] + 3 \* t \* (1 - t) \* (1 - t) \* controllers[1] + 3 \* t \* t \* (1 - t) \* controllers[2] + t \* t \* t \* controllers[3];  
 }  
  
 public float Iterate(ref float t, float delta)  
 {  
 t = t + delta / Derive(t);  
 if (t > 1) return t - 1;  
 else return 0;  
 }  
  
 float GetLength()  
 {  
 float sum = 0;  
 for(float f = 0.01f; f < 1; f++)  
 {  
 sum += 0.01f \* Derive(f);  
 }  
 return sum;  
 }  
  
 float Derive(float t)  
 {  
 Vector3 v1 = 3 \* (controllers[0] - 3 \* controllers[1] + 3 \* controllers[2] - controllers[3]);  
 Vector3 v2 = 3 \* (controllers[0] - 2 \* controllers[1] + controllers[2]);  
 Vector3 v3 = 3 \* (controllers[0] - controllers[1]);  
 float A = Vector3.Dot(v1, v1);  
 float B = -4 \* Vector3.Dot(v1, v2);  
 float C = 4 \* Vector3.Dot(v2, v2) + 2 \* Vector3.Dot(v1, v3);  
 float D = -4 \* Vector3.Dot(v2, v3);  
 float E = Vector3.Dot(v3, v3);  
 return Mathf.Sqrt(A \* t\*t\*t\*t + B \* t\*t\*t + C \* t\*t + D \* t + E);  
 }  
}

通过对多个三阶贝塞尔曲线进行拼接，可以实现一个均匀三阶贝塞尔曲线：

public class Bezier   
{  
 public float length;  
 public List<Transform> points = new List<Transform>();  
 public Segment[] segments;  
  
 public Segment[] GetSegments()  
 {  
 if (points.Count <= 1) return null;  
 Segment[] seg = new Segment[points.Count - 1];  
 for(int i = 0; i < points.Count - 1; i++)  
 {  
 Segment s = new Segment();  
 s.controllers = new Vector3[4];  
 s[0] = points[i].position;  
 s[1] = points[i].position + points[i].forward \* points[i].lossyScale.z;  
 s[2] = points[i + 1].position - points[i + 1].forward \* points[i + 1].lossyScale.z;  
 s[3] = points[i + 1].position;  
 seg[i] = s;  
 }  
 segments = seg;  
 return seg;  
 }  
  
 public Vector3[] GetLine(float freq)  
 {  
 List<Vector3> line = new List<Vector3>();  
 int i = 0;  
 float t = 0;  
 for(; i < segments.Length; )  
 {  
 line.Add(segments[i].Value(t));  
 Iterate(ref i, ref t, freq);  
 }  
 return line.ToArray();  
 }  
  
 public void Iterate(ref int i, ref float t, float delta)  
 {  
 for(; delta != 0; )  
 {  
 if (i >= segments.Length) return;  
 delta = segments[i].Iterate(ref t, delta);  
 if (delta == 0) break;  
 i++;  
 t = 0;  
 }  
 }  
  
 public Vector3 Value(int i, float t)  
 {  
 return segments[i].Value(t);  
 }  
  
}

在此基础上增加Lengths数组还可以进一步增加通过长度查找顶点坐标的功能。

##### 通用动画曲线

#### 2.2.3 关键帧

大多数早期二维计算机动画系统都是采用关键帧实现的。

##### 渐入渐出

### 参考文献

## 第三章 动作片段

### 3.1 手工动画制作

#### 3.1.1 蒙皮绑定

#### 3.1.2 关键帧制作

##### 动画控制器

### 3.2 动作捕捉

#### 3.2.1 光学捕捉

##### 采集方案

动作捕捉的根本目标是重构现实中的物理对象的三维运动方式，并将其添加到虚拟的对象(人物模型)上。根据捕捉运动的硬件不同，可以将动作捕捉大致分为以下四种类型：

* 电磁捕捉，也被称作磁性追踪，或简称磁捕，是将传感器放置在关节处，然后，传感器将位置和方向信息反馈到中央处理器以记录对象的运动。这个方案相对精确，但对使用环境有较高要求，为了避免环境磁场对采集的影响，需要使用线缆布线、或者对传感器自配电池组来实现无线传感，同时又需要对捕捉场地进行一定的电磁屏蔽处理。电磁跟踪的优势是，传感器的捕捉结果可以在很短的时间内(延迟通常不超过一秒)被记录和显示。
* 光学标记阅读法，简称光捕，它不受场地限制，相关人员只需在衣服上佩戴反射标记即可。光学标记阅读法通常无法实时提供反馈，且数据往往包含误差和噪声。这个方法需要运用数字图像处理技术对运动数据进行再生成，所以与电磁跟踪法相比，其需要的带宽和迭代时间都大大增加。光学捕捉的硬件成本相比电磁捕捉更低，但相对的其捕捉效果也更不稳定。光捕的质量与摄像头数量成正比，通常来说光捕需要在房间上下各布置二十余个摄像头。光捕需要的场地相对也更大，根据摄像头的布置角度，房间边缘5~10米的区域会成为无效区，如果动捕演员进入无效区，动作就会发生较大程度的扭曲以使其达不到捕捉目的。尽管当前光捕配套软件也在不断尝试对光捕误差进行修复，目前光捕产生的错误仍然需要由美术人员手动介入修复。
* 惯性捕捉是使用惯性传感器的捕捉方法。其成本比光捕更低，捕捉效果相对来说更接近磁捕，比光捕效果还要更好一些，产生严重误差的可能性也较低，是很适合低成本动作捕捉的方案。相比光捕，惯捕最大的问题是误差积累，在长时间录制中，光捕的误差通常是一帧或若干帧的数据错误，而惯捕的错误会不断累积并最终使角色的虚拟位置与实际位置产生偏差，这会极大的影响需要长时间多角色配合的片段录制。这导致大部分动画电影的动作录制仍然选择了光捕。
* 视觉捕捉是光学捕捉的简化方案，通常仅需要一个摄像头，且基本没有硬件成本。视觉捕捉本身的误差非常高，但通过算法可以避免在视觉捕捉中出现明显的异常。视觉捕捉容易忽略大量的动作细节，这是视觉捕捉还没有大规模用于CG行业的主要原因。

由于光学捕捉是目前最主流的动作捕捉方案，这里主要介绍光学捕捉。当采用光学系统时，需要完成3项主要任务：

* 对二位图像进行处理，以此在多个视频流中定位、识别和关联光学标记。
* 根据光学标记的二维定位，重构其三维位置。
* 三维标记根据处于捕捉状态的物理系统模型(如三角剖分表面）进行约束。

运动捕捉系统中一个较为常见的问题是噪声。噪声多源于物理系统，标记可相对于其初始位置移动，且角色运动的速度越快，标记的摆动幅度和重定位就越明显。另外，采集过程也会产生噪声。典型的误差结果会导致特征点位置数据产生0.5cm左右的定位误差。对于某些动画而言，此类误差令人难以接受。

为了降低噪声影响，用户可以在重构前对数据进行调整。这一步的目标是保持数据的平滑性并保留原有的特征。其中，可去除与典型值差异较大的数据顶点，对其余数据进行过滤。这里还可以采用简单的邻接数据加权平均的方案来对曲线进行平滑，邻接值的个数可以由用户进行权衡。

##### 标记定位和跟踪

光学标记可以使用彩色胶带、反光片、LED灯等简单显眼的物体进行绑定。

标记使用的色彩越多，就会越方便对标记的区分、跟踪和定位。其中，LED灯的标记方案还包括了不是使用不同色彩而是使用不同频闪作为ID进行区别的方案。

通常，标记位于关节附近，因为对应区域是动画对象的敏感部位。因此，标记位置和关节之间在运动时产生的位移一般是运动捕捉系统中噪声问题的根源所在。

如果背景图像是静止的，则可以将其消除以简化处理过程。为了使标记更加明显，很多时候会在颜色单调暗淡的背景场景中，使用更鲜艳的标记进行绑定。

在数据处理的过程中，我们需要在单个视频帧中区别出不同的标记，这一步被称为定位，同时还需要在连续的视频帧中找到相同的标记，这一步被称为跟踪。

但随着标记的数量增多，其被人物或其他标记遮挡的概率就越大。标记在图像中被遮挡的越是频繁，对它们的定位和跟踪难度就越大。

对于定位来说，除了使用不同的颜色或频闪作为ID来辅助定位外，可能还需要人工来进行查漏补缺，手动标注一些靠的过近的标记和被遮挡的标记。

对于跟踪来说，可以通过帧间一致性算法来辅助跟踪标记。即在已知前几帧中的标记位置和速度、加速度的限制条件的情况下，可以预测下一帧的大致位置，以估计跟踪标记。但当遇到标记被遮挡的情况时，跟踪程序可能出现问题。对于跟踪中断的情况，提出了一些启发性跟踪算法来处理被遮挡或速度变化缓慢的标记。但启发性算法解决的问题仍然比较有局限性，当遇到难以被启发性算法处理的问题时，可能会产生路径置换(比如大腿内侧的标记路径在被遮挡后，与大腿外侧的路径产生了错误的连接，本质是系统错误的混淆了两个标记)，或者在标记重现时错误的引入新标记(本质是系统无法在时间上辨明重现的标记是曾经的哪个标记)。

在对标记进行三维重构时，系统可以通过算法一定程度的纠正路径置换等问题，但仍需要用户的干预来解决这类问题。

最简单的标记方案只需要14个标记，即左肩、左肘、左掌、右肩、右肘、右掌、左髋、左膝、左脚、右髋、右膝、右脚、左太阳穴、右太阳穴。对于更精确的行为记录可以向肘、膝、胸、指、踝、脊椎等添加更多标记。

##### 三维位置重构

在对三维标记数据进行三维重构前，需要了解相机在世界空间中的位置和方向数据、以及焦距、图像中心位置和宽高比等属性。

出于校正目的，此处使用了针孔相机模型。针孔相机定义了基本的视锥体用于描述三维空间顶点的图像生成过程。相机的坐标系统定义与远点且位于投影中心位置，沿正z轴方向，距离焦距处设置了近裁剪平面、即投影平面，z轴方向指向了相机拍摄对象的中心。

根据相机的成像逻辑，我们知道图像中点的坐标是世界空间中的点到相机空间原点的连线与相机投影平面的交点。通过在世界空间中定义若干个已知点(通常是在录制场景中预先准备的若干个标注点)，然后在不同相机录制到的图像中查找这些标注点的位置，可以根据线性超定方程和最小二乘法求解得到相机的准确空间参数。

对于镜头、焦距、相机俯仰等非线性特征，还需要人工的辅助输入。

为了对标记的三维坐标实施重构，用户至少需要在两个不同的视点处定义其位置，即C1和C2。在相对简单的示例中，这需要架设两台相机记录其位置。其中，两个视角的正交性越明显，则重构过程的准确性就越大。

典型的系统同时架设8部摄像机进行拍摄。为了保证处理重构时不同摄像机传输的图像位于事实上的同一时刻，我们需要一个时间上的对齐标志。同时，鉴于不同摄像机的开启时间的差异，图像之间仍然可能存在半帧的时域误差。

相对于全局坐标系统，如果各个相机的位置和方向已知，且图像平面位置相对于摄像机已知，则重构的顶点图像坐标I可用于定位三维空间内的顶点P。显然，我们可以得到两条射线方程，即根据摄像机参数和图像中的顶点位置，得到的P在摄像机某视线上的位置。理论上，我们可以根据两直线的交点来获取标记的三维坐标，但由于误差的存在，这两条直线通常不会相交。我们需要计算一点到这两条直线的距离皆最短，即P1位于相机1的射线上，P2位于相机2的射线上，P1P2同时垂直于两条直线，则取P为P1P2的中点。

##### 骨骼匹配

当标记的移动呈现平滑、合理的状态时，后续步骤就是将其与骨骼结构实施绑定。在早期的匹配方案中，骨骼结构的位置直接由标记的位置数据决定，这种方案的问题是在近距离观察下由于噪声、平滑程度和误差项，骨骼不可避免的产生抖动。此外，骨骼关节之间的距离在一段时间内难以维持，这将导致大约10%~20%的长度抖动，多数情况下这种抖动都不能接受，例如这将导致“滑步”问题。

这种误差的根源之一在于，标记是位于人体表面关节外侧的，而非位于真正的关节处(即球棍模型的转腔处)。当演员在膝关节和髋关节保持恒定距离时，膝关节外侧的顶点和髋关节外侧的顶点距离并不能保持恒定。

一类修复方案是逻辑上重定位数字化位置数据，即使用标记信息来计算关节位置。比如，通过在膝关节两侧分别设定标记，然后通过这两个标记估算出膝关节的准确坐标。

此处，还可以使用相关几何性质计算关节的偏移。例如，我们在腕部内外设置了两个标记，目的是处理前臂的旋转，则实际的腕部位置可以通过二者间插值获得。随后，针对由腕部外侧-肘部外侧-肩部外侧组成的平面，在法线方向上从肘部标记处偏移，具体偏移量从当前人物角色处测量。通过每帧重计算法线，用户可以在全部行为过程中方便的维护肘部关节位置。

#### 3.2.2 动捕裁剪

##### 基于循环的动作裁剪

### 3.3 动作片段修饰

#### 3.3.1 基于物理的片段修饰

#### 3.3.2 基于信号处理的片段修饰

##### 频域处理

Bruderlin首次提供了一个对动作片段使用通信领域的频域分析逻辑进行处理的方案[1]。

### 3.4 面部动作捕捉

#### 3.4.1 混合形状

### 参考文献

[1] Bruderlin A, Williams L. Motion signal processing[C]//Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. 1995: 97-104.

## 第四章 动作数据流

### 4.1 动画储存

#### 4.1.1 FBX文件格式

#### 4.1.2 动作数据压缩

### 4.2 动画系统框架

#### 4.2.1 Unity动画系统结构

#### 4.2.2 UE动画系统结构

### 4.3 蒙皮解算

#### 4.3.1 经典蒙皮解算动画

##### 经典CPU蒙皮解算

##### 经典GPU蒙皮解算

#### 4.3.2 预烘焙图的GPU动画

参考Unity插件GPU Instancer - Crowd Animations

#### 4.4.3 预烘焙的网格动画

参考Unity插件 Mesh Animator

### 4.4 动作重采样

#### 4.4.1 动作重定向

##### 基于IK的采样

#### 4.4.2 根运动

## 第五章 动作绑定与约束

### 5.1 运动学约束解算

#### 5.1.1 前向运动学

##### 朝向约束解算

这个约束主要用于控制角色头部，以使其朝向正确的方向，通常会与捻度约束一起工作，其中捻度约束用于控制颈部骨骼。

#### 5.1.2 逆向运动学

##### 双骨骼IK的算法实现

这个约束解算主要用在大臂-小臂约束和大腿-小腿约束中，其描述如下：

输入：

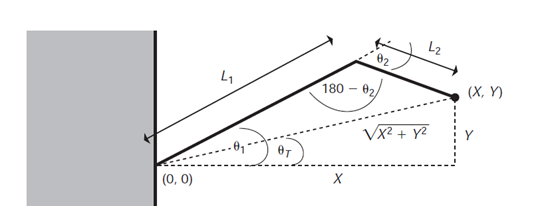
* 当前大臂(大腿)骨骼坐标，小臂(小腿)骨骼坐标，手掌(脚踝)骨骼坐标
* 目标控制点坐标T(新的C坐标)，辅助控制点坐标H(ATH决定了新ABC所在平面)
* 关节距离，关节距离

约束解算：

* 当，限制的位置，使得
* 当，通过修改骨骼的旋转属性限制的位置，使得，

实现流程：

* 第一步，根据、和坐标、，求出的值，**旋转骨骼B**，使，然后计算出新的点C坐标
* 第二步，对比坐标和坐标，**旋转骨骼A**，使与共线，然后计算出新的点、
* 第三步，对比坐标和坐标，**绕轴旋转骨骼A**，使与共面
* 第四步，**旋转骨骼C**，使其与控制点T朝向一致



第一步旋转骨骼B，确定的大小，在这之后骨骼B的旋转不会再受影响。根据、(这两段长度由骨骼本身决定，不会变化)和(AC的目标长度)使用余弦定理确定目标角度，通过当前的(可以用余弦定理求得，也可以记录上一帧的角度保存下来)和目标角度的差求出旋转的度数。旋转轴为垂直于AB和BC的向量，这是为了保证关节方向固定。这一步我们需要避免由于AB和BC完全共线导致旋转轴Axis失效，可以借用AHT平面来确定旋转轴。

接下来连续两次修改骨骼A。对A的第一次旋转能保证C的位置准确。第二次对A的旋转需要作用在轴AC上，这可以保证C的位置不因这次旋转而改变。旋转的目的是让与共面，由于这时T已经和C重合了，所以实际上就是将B旋转到平面。这里通过投影公式获得从AC指向B的垂线与从AC指向H的垂线，通过这两条垂线的夹角就可以获得旋转度数。

最后一步是修改骨骼C，使其与控制点T的朝向一致。

在整个解算过程中插入权重weight，则对T生效的权重posWeight和rotWeight应当影响T的输入。我们将C和T的位置对posWeight进行插值，这个权重将影响将影响第一次和第二次旋转。我们将C和T的旋转对rotWeight进行插值，这个权重将影响第四次旋转。对H生效的权重hintWeight应当影响第三步旋转，这里不是简单的将H和B对hintWeight进行插值，而是在旋转时对旋转度数进行缩放。

算法核心代码如下：

public static void SolveTwoBoneIK(Transform root, Transform mid, Transform tip, Transform target, Transform hint,   
 float posWeight, float rotWeight, float hintWeight)  
{  
 Vector3 PA = root.position;  
 Vector3 PB = mid.position;  
 Vector3 PC = tip.position;  
 Vector3 PT = Vector3.Lerp(PC, target.position, posWeight);  
 Vector3 PH = hint.position;  
 Quaternion RT = Quaternion.Lerp(tip.rotation, target.rotation, rotWeight);  
   
 Vector3 AB = PB - PA;  
 Vector3 BC = PC - PB;  
 Vector3 AC = PC - PA;  
 Vector3 AT = PT - PA;  
 Vecotr3 AH = PH - PA;  
 float oldAngleABC = TriangleAngle(AC, AB, BC);  
 float newAngleABC = TriangleAngle(AT, AB, BC);  
 Vector3 axis = Vector3.Cross(AB, BC);  
 if(axis.sqrMagnitude < 1e-6)  
 {  
 axis = Cross(AH, BC);  
 if(axis.sqrMagnitude < 1e-6)   
 axis = Vector3.Cross(AT, BC);  
 if(axis.sqrMagnitude < 1e-6)  
 axis = Vector3.up;  
 }  
 axis = Vector3.Normalize(axis);  
 Quaternion midR = Quaternion.AxisAngle(axis, oldAngleABC - newAngleABC);  
 mid.rotation = midR \* mid.rotation;  
 PC = tip.position;  
 AC = PC - PA;  
   
 root.rotation = FromToRotation(AC, AT) \* root.rotation;  
 PB = mid.position;  
 PC = tip.position;  
 AB = PB - PA;  
 AC = PC - PA;  
   
 Vector3 ACNorm = AC.normalized;  
 Vector3 ABProj = AB - ACNorm \* Vector3.Dot(AB, ACNorm);  
 Vector3 AHProj = AH - ACNorm \* Vector3.Dot(AH, ACNorm);  
 float maxReach = AB.magnitude + BC.magnitude;  
 if(ABProj.sqrMagnitude > (maxReach \* maxReach \* 0.001f) && AHProj.sqrManitude > 0f)  
 {  
 Quaternion hintR = FromToRotation(ABProj, AHProj);  
 hintR.ToAngleAxis(out float hintAngle, out Vector3 hintAxis);  
 hintR = Quaternion.AngleAxis(hintAngle \* hintWeight, hintAxis);  
 root.rotation = hintR \* root.rotation  
 }  
   
 tip.rotation = RT;  
}  
  
static float TriangleAngle(Vector3 AC, Vector3 AB, Vector3 BC)  
{  
 float lenAC = AC.magnitude;  
 float lenAB = AB.magnitude;  
 float lenBC = BC.magnitude;  
 float c = (lenAB \* lenAB + lenBC \* lenBC - lenAC \* lenAC) / (lenAB \* lenBC) \* 0.5f;  
 c = Mathf.Clamp(c, -1.0f, 1.0f);  
 return Mathf.Acos(c);  
}  
  
static Quaternion FromToRotation(Vector3 from, Vector3 to)  
{  
 float theta = Vector3.Dot(from.normalized, to.normalized);  
 if (theta >= 1f)  
 return Quaternion.identity;  
  
 if (theta <= -1f)  
 {  
 Vector3 axis = Vector3.Cross(from, Vector3.right);  
 if (axis.sqrMagnitude == 0f)  
 axis = Vector3.Cross(from, Vector3.up);  
  
 return Quaternion.AngleAxis(180f, axis);  
 }  
  
 return Quaternion.AngleAxis(Mathf.Acos(theta) \* Mathf.Rad2Deg, Vector3.Cross(from, to).normalized);  
}

##### 双骨骼IK解析解

##### 捻度校正(Twist校正)

Twist Bone约束解算主要用在小臂和大腿的Twist Bone上，有时在大臂和小腿上也会出现。其描述如下：

输入：

* 受约束Twist Bone，简称T
* 参考骨骼S(通常是相对受约束Twist Bone更靠近末端的正常骨骼)的旋转
* Twist Bone允许旋转的旋转轴
* Twist Bone受参考骨骼的影响权重

约束解算：

* 当参考骨骼S绕轴旋转度数时，使Twist Bone沿轴旋转度数

实现流程：

* 在初始化时预存S和T的初始旋转和，然后实时用当前S的旋转左乘初始旋转矩阵的逆旋转，得到骨骼S相对初始情况的旋转程度
* 根据和旋转轴得到参考骨骼S绕轴的旋转分量
* 对受约束骨骼的初始旋转应用旋转，按照权重修改应用的比例，设置T的旋转为

整个算法的主要难点是获取旋转分量这一步，可以将它理解为旋转域的“投影”算法。为了能获取到旋转在任意轴上的投影，可以虚构两个点，通过计算两点对轴线的垂线的夹角来计算出投影值，其描述如下：

输入：

* 归一化的原四元数
* 归一化的投影轴

输出：

* 求出在上的旋转分量

实现流程：

* 虚构一个视点，保证不在投影轴上，这是为了避免后续步骤中求出的垂线长度为0
* 对视点应用旋转，得到新视点
* 从和分别向轴做垂线、，这两条垂线的夹角就是在上的旋转分量

算法核心代码如下：

private Quaternion originRot, invSrcOriginRot;   
private Transform source;   
  
private void Init()  
{  
 originRot = transform.localRotation;  
 srcOriginRot = Quaternion.Inverse(source.localRotation);  
}  
  
public static void SolveTwist(Transform twistBone, float weight, Vector3 axis)  
{  
 Quaternion deltaRot = invSrcOriginRot \* source.localRotation;  
   
 Vector3 axisNorm = axis.normalized;  
  
 Vector3 fromDir = axisNorm == Vector3.forward ? Vector3.right : Vector3.forward;  
 Vector3 toDir = Matrix4x4.Rotate(ori).MultiplyPoint(fromDir).normalized;  
  
 Vector3 fromProj = fromDir - axisNorm \* Vector3.Dot(fromDir, axisNorm);  
 Vector3 toProj = toDir - axisNorm \* Vector3.Dot(toDir, axisNorm);  
 Quaternion twistRot = FromToRotation(fromPorj, toProj);  
   
 twistRot.ToAngleAxis(out float twistAngle, out Vector3 twistAxis);  
 twistRot = Quaternion.AngleAxis(twistAngle \* weight, twistAxis);  
 transform.localRotation = twistRot \* originRot;  
}  
  
static Quaternion FromToRotation(Vector3 from, Vector3 to)  
{  
 float theta = Vector3.Dot(from.normalized, to.normalized);  
 if (theta >= 1f)  
 return Quaternion.identity;  
  
 if (theta <= -1f)  
 {  
 Vector3 axis = Vector3.Cross(from, Vector3.right);  
 if (axis.sqrMagnitude == 0f)  
 axis = Vector3.Cross(from, Vector3.up);  
  
 return Quaternion.AngleAxis(180f, axis);  
 }  
  
 return Quaternion.AngleAxis(Mathf.Acos(theta) \* Mathf.Rad2Deg, Vector3.Cross(from, to).normalized);  
}

##### 循环坐标下降法

### 5.2 物理学约束解算

#### 5.2.1 刚体物理基础

##### 旋转与力矩

#### 5.2.2 刚体关节系统

##### 关节链接

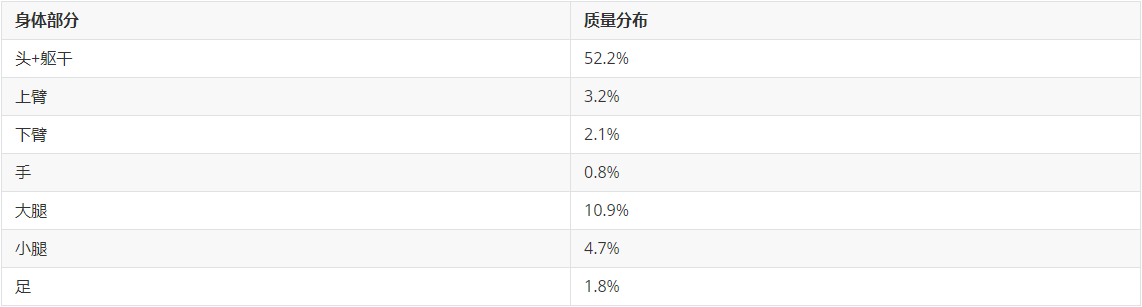
##### 减化坐标

减化坐标是通过Articulation Body引入的一种在骨骼物理动画中非常常用的工具[1][2]。

##### 人体质量分配模型

根据对人体的研究论文[3][4]，我们可以获取质量在人体上的几种基本分布模型：

* 按百分比的质量分布：



* 按典型数值统计的质量分布：



* 更适合刚体关节系统的细分质量分布：



#### 5.2.3 控制理论

##### 弹簧阻尼模型

##### PID控制模型

PID控制算法是应用非常广泛的一种控制算法。PID是比例(Proportion)、积分(integration)、微分(Differentiation)控制器的统合，能相比弹簧阻尼模型更精确的控制受控体接近参考值。其描述如下：

输入：

* 随时间更新的参考值序列，当前控制

约束解算：

* 根据参考值，对控制值施加一阶或二阶影响，即通过修改控制值的速度或加速度，使控制值在空域上接近参考值

实现方法：

* 根据，对参考值施加影响

比例控制器的作用是对系统造成与误差成比例的影响，即，可见比例控制器的效果与弹簧阻尼模型中的弹簧作用类似。单纯使用比例控制器的缺陷有两点：滞后性和稳态误差。

比例控制器的滞后性体现在，虽然控制值与参考值的误差在时域上成比例的缩小，但误差始终存在，并且需要一定次数迭代才能收敛。

比例控制器的稳态误差体现在，当控制值与参考值的误差在时域上放大时(参考值在变化时原理控制值)，误差的放大量与比例控制器提供的误差缩小趋势恰好抵消，使控制值在没有接近参考值的情况下进入收敛。

为了消除比例控制器的两个缺陷，引入了积分控制器。积分控制器对系统造成的影响与误差的积分成比例。即。在发生滞后和稳态误差时，积分控制器会产生作用，激励系统靠近参考值。

由比例控制器和积分控制器组合成的控制器是PI控制器，即。PD控制器的缺陷体现在收敛过慢，即经过一段时间后才能在目标值上产生收敛。

为了加快收敛，减少控制过程中的震荡，引入了微分控制器。微分控制器是在系统上叠加与误差的变化量成比例的影响，也就是在误差增大时增大影响，在误差减小时减小影响，即。这里，如果令，可以转化为，即对系统叠加与参考值变化速率和控制值变化速率的差成比例的影响。在这一点上，微分控制器的效果与弹簧阻尼模型中的阻尼作用类似。

弹簧阻尼模型是一个PD控制器。

总结PID控制器的各分量：

参数加快控制值对参考值的追赶，参数缩小系统误差，参数减小系统振荡。

该公式在离散的时域上起作用时为：

算法核心代码如下：

float proportion = 300, integration = 20, differentiation = 30;  
  
Vector3 currTargetPos;  
Vector3 prevTargetPos;  
Vector3 intedErrorPos;  
  
void UpdatePosition()  
{  
 Vector3 targetPos = currTargetPos;  
 Vector3 targetVel = (targetPos - prevTargetPos) / Time.deltaTime;  
  
 Vector3 currPos = transform.position;  
 Vector3 currVel = physx.velocity;  
  
 Vector3 deltaPos;  
 Vector3 deltaVel;  
  
 deltaPos = targetPos - currPos;  
 deltaVel = targetVel - currVel;  
  
 intedErrorPos += deltaPos;  
  
 Vector3 force = proportion \* deltaPos + integration \* intedErrorPos + differentiation \* deltaVel;   
  
 physx.AddForce(force \* IKIntensity \* Physics.mass);   
}

#### 5.2.4 碰撞解算

### 参考文献

[1] Featherstone R. Rigid body dynamics algorithms[M]. Springer, 2014.

[2] Yin Z, Yin K K. Linear Time Stable PD Controllers for Physics‐based Character Animation[C]//Computer Graphics Forum. 2020, 39(8): 191-200.

[3] Clauser C. E., McConville J. T., Young J. W. August (1969). WEIGHT, VOLUME, AND CENTER OF MASS OF SEGMENTS OF THE HUMAN BODY. Aerospace Medical Research Laboratory, Ohio.

[4] Dempster W. T., Gaughran. G. R. L. American Journal of Anatomy, Vol. 120, No. 1. (1967), pp. 33-54.

## 第六章 动作决策与组织

### 6.1 混合树

#### 6.1.1 动画混合算法

#### 6.1.2 多维动画混合

#### 6.1.3 智能动画混合

##### Momat混合方案

### 6.2 动画状态机

##### 经典有限状态机方案

##### 状态机分级解决方案

##### 状态机分层解决方案

##### 基于权重和优先级的低耦合解决方案

### 6.3 动画层次化组织

### 6.4 运动匹配

## 第七章 程序化行为建模

### 7.1 腿部行为

#### 7.1.1 平衡建模

##### 基于ZMP的平衡检测

ZMP(Zero Moment Point)是零力矩点的简写，目前在机器人领域内被广泛用于确认机器人平衡状态是否稳定[1]。

ZMP根植于结构力学中的一个简单的平衡规则：重心投影位于支撑多边形内，其中支撑多边形指的是支撑系统中所有刚性结构与地面的接触点所构成的凸包多边形区域。这一规则很好理解，由于地面和接触点之间只存在支持力和摩擦力，不存在拉力，所以当重心的投影位于所有接触点的同一侧，也就是位于支撑多边形之外时，就没有任何一个点的力可以平衡重力的力矩，无法阻止系统的侧翻。

重心投影规则的缺点在于，只有系统内部处于基本静止的状态才能准确的预测平衡状态，但人体的运动中存在复杂的力学结构，质量的分布随身体的运动不断变化，而在步行或奔跑过程中，地面也会对脚的运动产生远比静止复杂的摩擦力。在早期的机器人领域中，为了通过重心投影规则来保证机器人的平衡，双足机器人的步行速度不得不被设计的及其缓慢，这明显难以满足动画领域对平衡精确性的需求。

ZMP概念被首次提出于1968年，一直以来被用来指导更高速的双足机器人平衡设计。该概念假设了脚底与地面的接触区域是平面，且能产生足够大的摩擦力来避免脚底滑动。

地面反作用力对人姿态的影响是及其复杂的，但总可以简化为地面对任意一点产生的反作用力()和力矩()。若能找到一点，使其受到惯性力、重力和地面反作用力的和力矩的水平分量为0，则称该点为零力矩点。

以重心为参考系，作用在其上的惯性力和重力的合力如下：

$$F^{gi}=mg-ma\tag1$$

其中为系统总质量，为重力加速度，为质心的加速度。

##### 平衡控制原则

该模型尝试在人形系统受到外来扰动时自动恢复到平衡状态。

对人形系统的扰动基本可以分为五种类型：

* 低误差的静态扰动：来自低速碰撞体的碰撞约束，这些碰撞体对系统的扰动主要体现在其碰撞体积而非碰撞产生的动量。若该扰动对系统的扰动不会影响系统平衡。如和地面、建筑物的轻微接触。在这种情况下只需要通过运动学轻微调整受影响肢体的位置，不需要产生基于平衡的程序化动作。
* 高误差(平衡)的静态扰动：由于静态碰撞的影响使系统失去平衡，通常是检测到角色重心的垂线超出支撑多边形，多发生在前后方向上。在这种情况下需要使用基于平衡的程序化动画来帮助系统恢复平衡。
* 高误差(姿态)的静态扰动：由于静态碰撞的影响使全身的姿势受到较大程度的影响，极有可能产生一个不自然的动作，多发生在左右方向上。在这种情况下也需要使用基于平衡的程序化动画来帮助系统恢复平衡。
* 低动量的动态扰动：来自高速碰撞体的碰撞约束，这些碰撞体对系统的扰动主要体现在其碰撞产生的动量而非碰撞体积。若该扰动对系统的扰动不会影响系统平衡。如轻质量物体对人体的碰撞、或物体对人体末梢的碰撞。在这种情况下只需要通过运动学或物理学调整受影响肢体的位置，并基于物理自动恢复原动作，不需要产生基于平衡的程序化动作。
* 高动量的动态扰动：来自高速碰撞体的影响使系统失去平衡，通常是检测到角色重心的速度突变且有超出支撑多边形的趋势。在这种情况下需要使用基于平衡的程序化动画来帮助系统恢复平衡。

接下来逐一分析检测和应对五种扰动的实现思路。

##### 线性倒立摆模型

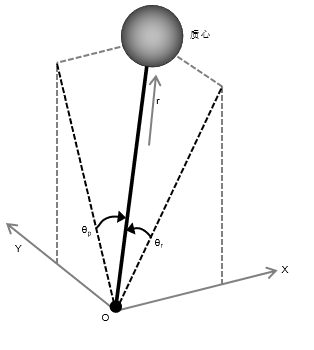
这里介绍Kajita提供的一种基于线性倒立摆物理模型的机器人平衡演算方案[2][3]，这是一种机器人领域内主流的用于落脚点预测地双足系统物理模型。我们可以将这个方案通过运动学约束的方式应用到骨骼动画领域。

在人形生物单足站立时，其上半身的运动学模型可以用一个连接其支撑脚和重心的线性倒立摆来表达。下面简单介绍一下线性倒立摆模型：

以支撑点为原点构建坐标系，质心的位置可以在笛卡尔坐标系下和球坐标系下分别表示为：

$$\vec q=(\theta\_r, \theta\_p,r)\\
\vec p=(x,y,z)$$

其中，,和的定义如下图所示：



根据几何关系可知：

$$x=r\sin\theta\_r\tag1$$

$$y = -r\sin\theta\_p\tag2$$

$$z = r\sqrt{1-\sin^2\theta\_r-\sin^2\theta\_p}\tag3$$

根据三维线性倒立摆的物理模型，我们可以得到质心在笛卡尔坐标系中的受力方程：

$$m\left(\begin{matrix}
\ddot x\\\ddot y\\\ddot z
\end{matrix}\right)
=(J^T)^{-1}
\left(\begin{matrix}
\tau\_r\\\tau\_p\\f
\end{matrix}\right)+
\left(\begin{matrix}
0\\0\\-mg
\end{matrix}\right)\\\tag4$$

其中m是倒立摆的质量，是重力加速度，、和分别是系统的激励力矩和激励力向量，贾可比矩阵的结构如下：

$$J=\frac{\partial p}{\partial q}=\left(\begin{matrix}
0&rC\_p&S\_p\\-rC\_r&0&-S\_r\\-rCrSr/D&-rC\_pSp/D&D
\end{matrix}\right)\\\tag5
C\_r=\cos\theta\_r,\ C\_p=\cos\theta\_p,\ S\_r=\sin\theta\_r,\ S\_p=\sin\_p\\\ D=\sqrt{1-\sin^2\theta\_r-\sin^2\theta\_p}$$

我们将公式(2)、(3)和(5)代入公式(4)，我们可以得到一个描述质心在Y轴上的运动方程：

$$m(-z\ddot y+y\ddot z)=\frac D{C\_r}\tau\_r-mgy\tag6$$

类似的我们得到质心在X轴时的运动方程：

$$m(z\ddot x-x\ddot z)=\frac{D}{C\_p}\tau\_p+mgx\tag7$$

为了让线性倒立摆模型更适应人形生物的步行姿态，我们在线性倒立摆的基础上额外附加了一些约束条件来限制其运动。

约束方程(8)将运动限制在一个平面内：

$$z=k\_xx+k\_yy+z\_c\tag{8}$$

显然，该平面的法线为，该法线法线应当与支撑脚所在地面的法线方向相同。则是重心与地面的平均距离。为了更方便进行约束结算，我们对公式(8)进行二次求导得：

$$\ddot z=k\_x\ddot x+k\_y\ddot y\tag9$$

我们将约束方程(9)代入到倒立摆公式(6)和(7)中，可以得到在约束下的运动方程：

$$\ddot y=\frac{g}{z\_c}y-\frac{k\_1}{z\_c}(x\ddot y-\ddot xy)-\frac1{mz\_c}u\_r\tag{10}$$

$$\ddot x=\frac{g}{z\_c}x+\frac{k\_2}{z\_c}(x\ddot y-\ddot xy)+\frac1{mz\_c}u\_p\tag{11}$$

其中和是新引入的虚拟输入，它们被用于对非线性的力矩和进行补偿：

$$\tau\_t=\frac{C\_r}Du\_r\tag{12}$$

$$\tau\_p=\frac{C\_p}Du\_p\tag{13}$$

如果我们将情况简化为在水平地面上的运动，也就是令、，代入公式(12)、(13)，我们可以将公式(10)、(11)简化为：

$$\ddot y=\frac g{z\_c}y-\frac1{mz\_c}u\_r\tag{14}$$

$$\ddot x=\frac g{z\_c}x+\frac1{mz\_c}u\_p\tag{15}$$

可见，公式(14)和(15)之间是线性独立的，控制重心在水平方向上的运动的参数只有。尽管原始的倒立摆物理模型是非线性的，我们通过额外添加约束获得了它的线性版本。

如果运动进行在斜坡或台阶上，也就是，我们通过得到：

$$x\ddot y-\ddot xy=-\frac{1}{mz}(u\_rx+u\_py)\tag{16}$$

因此，通过引入新的约束，我们可以在倾斜平面下继续应用(14)和(15)提供的动力学：

$$u\_rx+u\_py=0\tag{17}$$

令输入力矩为0，即令，我们可以得到：

$$\ddot y = \frac g{z\_c}y\tag{18}$$

$$\ddot x = \frac g{z\_c}x\tag{19}$$

显然，我们得到了一个力场：

$$F=\frac{mg}{z\_c}r\tag{20}$$

为了对这个力场进行更准确的描述，我们从加速度的角度转换到能量的角度，对公式(18)和(19)积分，我们得到了与时间无关的能量分布函数，我们将这个能量分布的模型称为轨道能量模型。

##### 轨道能量模型

轨道能量模型是对线性倒立摆模型的一种简化[2][4][5]。继续对线性倒立摆模型的推导，我们可以得到轨道能量E在空间上的分布：

$$E\_x=\frac12\dot x^2-\frac{g}{2z\_c}x^2\tag1$$

$$E\_y=\frac12\dot y^2-\frac{g}{2z\_c}y^2\tag2$$

由于我们已知轨道能量的生效模式为一个线性力场：

$$F=\frac{mg}{z\_c}r\tag{3}$$

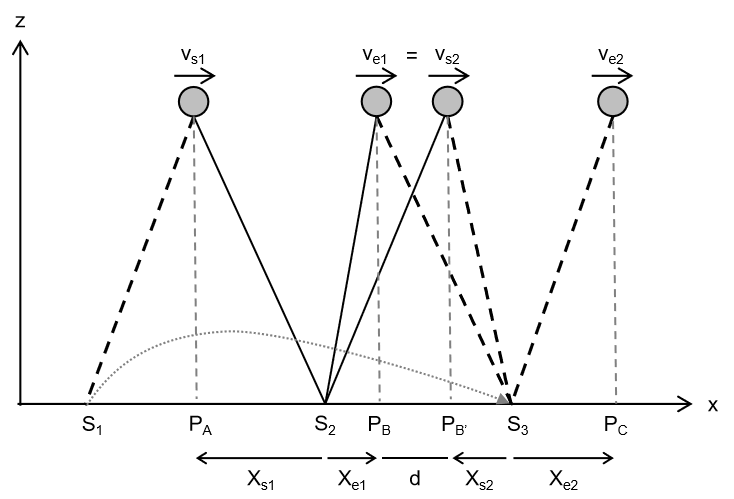
可以容易的得到，轨道能量的值不随坐标的位置改变，即：

$$E\_x+E\_y=C\tag4$$

根据Kajita的文章中对对称轴的推导[2]，证明了在这个力场中重心的运动曲线是双曲线的一支。

在这个模型中，我们认为在步行过程中人体重心的运动轨迹为若干条双曲线和直线的拼接：在单支撑阶段为基于线性倒立摆模型计算出的双曲线，而在双支撑阶段为直线运动。

为了能让人的运动状态接受用户的实时输入控制，我们先尝试在单一轴(X轴)上实现对运动的控制：



轨道能量轨迹控制

一次摆腿的过程中，涉及的重心状态参数包括、、、，它们分别表示摆腿起始和结束时刻重心的坐标和速度(注意方向)。它们具有这样的关系：

$$\left(\begin{matrix}x\_e\\v\_e\end{matrix}\right)=
\left(\begin{matrix}C\_T&T\_cS\_T\\{S\_T}/{T\_c}&C\_T\end{matrix}\right)\tag5
\left(\begin{matrix}x\_s\\v\_s\end{matrix}\right)\\
C\_T=\cosh(T\_s/T\_c),\ S\_T=\sinh(T\_s/T\_c)$$

即：

$$x\_e=C\_Tx\_s+T\_cS\_Tv\_s\\v\_e=\frac{S\_T}{T\_c}x\_s+C\_Tv\_s$$

其中和是双曲正余弦函数，和分别是单支撑的周期和全周期。

为了能控制运动的速度，我们需要修改每次单支撑周期结束时的落脚点位()，以此来生成接下来一段单支撑周期()的初始状态。如果我们给定了希望在点处得到的重心状态，我们可以定义一个误差函数：

$$N=a(x\_d-x\_{e2})^2+b(v\_d-v\_{e2})^2\tag7$$

其中定义了对坐标误差和速度误差的权重。

通过将公式(5)带入到误差函数(6)，我们得到了一个简单的线性规划问题，即求解一个落脚点的位置使得误差函数N最小。据此，我们可以整理得到以下控制策略：

$$x\_{s2}=\frac{aC\_T(x\_d-S\_TT\_cv\_d)+b\frac {S\_T}{T\_c}(v\_d-C\_Tv\_e)}{aC^2\_T+b(\frac {S\_T}{T\_c})^2}\tag8$$

如果我们只对速度控制感兴趣，可以令得到化简过后的：

$$x\_{s2}=\frac{T\_c}{S\_T}(v\_d-C\_Tv\_{e1})\tag9$$

其中，应是在上一轮迈步中就预估到的值。为了能在落地前就获得落脚点的位置，我们还需要提前考虑双支撑阶段的直线运动距离：

$$d=(T\_c-T\_s)v\_{e1}\tag {10}$$

在整个单支撑阶段，摆动腿将在一个预设的时间内从沿某弧线运动到。

总结基于轨道能量模型的平衡演算流程：

* 开始步行，根据和可以在力场中计算出和，作为迈出第一步的结果。
* 根据输入调整和，可以计算出摆动腿的落地位置，实时修正摆动腿位置向目标位置运动。
* 摆动腿落地后的整个双支撑阶段，人物上半身还会继续向前运动一小段距离。
* 进入下一个单支撑状态时更新和，继续执行步骤2。

#### 7.2.2 运动建模

##### 人体运动节奏

Bruderlin等人在论文中系统的描述了人在步行[6]和奔跑[7]两个状态下的运动节奏。

一个步行周期由两个双支撑阶段、一个左支撑阶段和一个右支撑阶段组成：

* 左腿在前，左脚刚开始接触地面，右脚仍未离开地面。右支撑阶段结束，双支撑阶段开始。
* 重心前移，左脚逐渐成为支撑脚，随后右脚离开地面。双支撑阶段结束，左支撑阶段开始。
* 右腿向前摆动，重心前移，右脚刚开始接触地面，左脚仍未离开地面。左支撑阶段结束，另一个双支撑阶段开始。
* 重心前移，右脚逐渐成为支撑脚，随后左脚离开地面。双支撑阶段结束，右支撑阶段开始。

在这个过程中，可以运用数学手段来计算步行的节奏：

定义人体身高(m)，单次迈出的步长(m)、单次迈出的步频(step/s)、人移动的总体速度(m/s)，完成单次迈步所需的时间(s)，它们的关系如下：

$$\\v= f\times l
\\l = f\times 0.24H
\\f^2=v/0.24H
\\T=1/f$$

随着行走频率的提高，双支撑阶段占总行走周期的比例会降低，直到频率达到约3步每秒，即移动速度接近3.888m/s时，行走开始切换到奔跑。

双支撑的时间有以下规律：

这并不是表示速度在3.888m/s以下时一定是行走，通常慢跑速度最低可以达到1.388m/s至2.222m/s。

奔跑周期由两个腾空阶段、一个左支撑阶段和一个右支撑阶段组成：

* 左腿在前，左脚仍在向前摆动，右脚刚离开地面。右支撑阶段结束，腾空阶段开始。
* 向前跃进，左脚刚接触地面，右腿继续向前摆动。腾空阶段结束，左支撑阶段开始。
* 右腿向前摆动，左脚开始离开地面。左支撑阶段结束，另一个腾空阶段开始。
* 向前跃进，右脚刚接触地面，左腿继续向前摆动。腾空阶段结束，右支撑阶段开始。

在这个过程中，也可以运用数学手段来计算奔跑节奏：

定义人体身高(m)，单次迈出的步长(m)、单次迈出的步频(step/s)，人移动的总体速度(m/s)，完成单次迈步所需的时间(s)，它们的关系如下：

$$l=\left\{\begin{array}{c}
0.1394+0.279v\sqrt{\frac H{1.8}}&\qquad v<=6.7m/s\\
2&\qquad v>6.7m/s
\end{array}\right.\\
v=f\times l\\
T=1/f$$

随着速度的增加，步长会增加，在速度达到6.7m/s时，也就是步长达到2m左右时，步长开始趋于稳定，此时随着速度的增加步频开始增长。

腾空的时间与跑步速度的关系不大，随着步频的增加，支撑阶段相对整个周期会变短，进而使得腾空时间的占比增长：

通常来说跑步的频率 (step/s)，低于这个频率的基本可以算作步行。

盆骨相对起跳时的最高腾空高度(m)与腾空时长(s)、重力加速度(m·s-2)、起跳时脚趾离地瞬间盆骨距离地面的高度(m)、落地时脚趾落地瞬间盆骨离地面的高度(m)有关：

##### 落脚点物理约束

该模型试图保证在动画的播放过程中角色的脚底贴合地面，不穿入墙壁和台阶，避免双脚落在悬崖上，并基于双脚的高度动态的调整盆骨高度。

由于该系统的生效时刻应当在所有动作决策和运动学之后，物理学之前，该系统的输入应当为双脚和盆骨的运动学控制点，输出约束后的运动学控制点。虽然名为物理约束，但该约束生效的阶段为运动学阶段。

### 7.2 手部行为

#### 7.2.1 程序化场景交互

### 7.3 环境感知

### 参考文献

[1] Vukobratović, B. Brovac, D. Surla, and D. Stokic, "Biped Locomotion" Springer-Verlag, 1990.

[2] Kajitas S., Kanehiro F., Kaneko K., Yokoi K., Hirukawa H. "The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation." In Proceedings of the IEEE/RSJ International, Conference on Intelligent Robots and Systems, (2001), pp. 239–246

[3] Kenwright B. Real-Time Physics-Based Fight Characters[J]. no. September, 2012.

[4] Raj M, Singh A K. Humanoid Gait Pattern Generation with Orbital Energy[J]. Journal of Circuits, Systems and Computers, 2021, 30(14): 2150259.

[5] Liu F, Chen X. Balance recovery method of walking robot based on orbital energy model[J]. Jiqiren(Robot), 2011, 33(2): 244-250.

[6] Bruderlin A, Calvert T W. Goal-directed, dynamic animation of human walking[C]//Proceedings of the 16th annual conference on Computer graphics and interactive techniques. 1989: 233-242.

[7] Bruderlin A, Calvert T. Knowledge-driven, interactive animation of human running[C]//Graphics interface. 1996, 96: 213-221.

## 第八章 附着物动画

### 8.1 手持道具

##### 持枪动作

### 8.2 服装与布料

##### 可换装角色管理

##### 布料解算

### 8.3 毛发

## 第九章 动作工作流

### 9.1 动作片段验收与评估

#### 9.1.1 动作验收准则

##### 动作验收标准

对于最终的动作效果，可以在以下五个规则的指导下进行验证。这五个规则按照优先级从高到低排列：

* 稳定性：相邻骨骼之间的距离约束不被明显违反，骨骼的生理旋转约束不被明显违反，根骨骼和末梢骨骼不产生肉眼可见的抖动，动作不会完全脱离控制。这个特性被排在第一位，这是因为违反稳定性标准的动画是完全不可接受的，会影响用户对动画甚至整个游戏/电影的正常感知，属于恶性问题。
* 清晰性：动作的幅度足够大，速度足够合理，以使用户注意到动作发生。动作每个姿势的目的明确，能够被用户理解含义，动作的意义不因混合等原因损失。同时，动作需要表达的特点需要被用户感知，比如奔跑动作的速度感，跳跃动作的轻盈感或重量感。这个特性被排在第二位，因为播放动画的根本目的是向用户传达信息，当动作清晰性不足时会使整个动作失去意义。
* 连贯性：在运动发生时，各个动画之间能连贯切换，不因为混合、运动学等原因产生明显突变。人眼对于突变的容忍度极低，但如果连贯性与清晰性产生冲突，可以放弃追求连贯性，比如当某个动作必须立即对用户输入产生响应时可以允许突变的发生。
* 自然性：对于连续发生的动作，其姿势和节奏符合人运动的规律，动画片段不出现明显使人不舒适的姿势，不因为混合产生明显不符合正常生理习惯的姿势，如突然的停顿、无意义的画圈、肢体的摩擦或碰撞等。对于连续发生的动作，每当其动量或角动量变化时，总能找到其变化原因(如重力、肢体施力或与其它物体的交互等)。自然性相比物理性重要性更高，这是因为人眼对于动作自然与否的判断比动作是否符合物理更加敏感。
* 可信性：角色发生的动作能够与场景发生可靠的交互，以使角色可信的出现在场景中。在发生碰撞时肢体不穿模，在运动时应当产生摩擦力的脚不明显滑动。人更难以发现动作中存在的物理问题，并且在有些时候倾向于将其忽略。如果符合物理的动作会导致其清晰性不足，便会放弃其物理的纯粹正确性。

#### 9.1.2 对照组及对照工具

##### 视频对照工具

此处推荐Kinovea作为免费的动画对照工具。

##### 自对照工具

这里设计一组方便对最终动画效果进行验证的自验证工具。

## 第十章 动作导演

### 10.1 动画表现学

#### 10.1.1 迪士尼动画理论

### 10.2 摄像机控制

#### 10.2.1 镜头理论

### 10.3 演出控制

#### 10.3.1 协调动画播放

## 第十一章 机器学习动画

### 11.1 机器学习应用

#### 11.1.1 基于PyTorch的应用

##### 机器学习领域概述

机器学习是人工智能的一个子领域，当下我们听到的有关于人工智能的讨论多数是关于机器学习的。

机器学习的定义，是能从经验中自动改进系统性能的程序[1]。“经验”通常以数据的形式呈现，这些经验可以包括不同的信息形态，其中一个关键的信息是对于模型表现的反馈信息。有的数据集中包含了模型应该输出的值，有的数据集则没有。根据不同的反馈信息，我们需要使用不同的技术进行处理，这将机器学习分为了三个类型：

* 监督学习：数据集中包括模型应该由的正确输出值，这也被称为标记数据。算法的目的是提高输出正确值的正确率。
* 强化学习：数据集中仅包括模型打分值，而没有正确输出值。算法的目的是尽可能地提高输出的打分值。
* 无监督学习：数据集中没有关于模型输出好坏的客观评估。算法的目标通常是认为设定的学习目标。

随着机器学习的发展，根据反馈信息进行的分类开始模糊，出现了大量交叉运用多种反馈的机器学习方式：

* 半监督学习：数据集包含一部分标记和一部分无标记的数据，研究利用无标记数据提升模型性能。
* 模仿学习：在强化学习的过程中利用标记数据来对模型进行指导。

机器学习模型是实现机器学习任务的数学方法论。常见的机器学习模型包括最近邻模型、决策树模型、贝叶斯模型、线性模型和多层神经网络模型。其中，多层神经网络模型因为其高效的性能和端对端的高开发效率在近十年以来被广泛的应用在多种机器学习领域，为了与传统的机器学习模型做出区分，将应用多层神经网络模型的机器学习模型称为深度学习。

机器学习的目标是，获得一个函数，这个函数可以接收若干输入，称为输入向量，然后返回一个给定输出，称为输出向量。机器学习的“学习”过程体现在通过训练来动态的修改目标函数的有关参数，以此使目标函数尽可能在面对任一输入向量时得到符合人类期望的输出。这个训练过程通常是基于统计学的，即通过训练统计输入向量和输出向量的最大可能对应关系，来得到输入向量->输出向量的映射关系。通常来说，训练目标函数的过程可以使用多层神经网络来实现，多层神经网络是一种具有大量可调整的线性参数的训练模型。

##### 深度学习领域概述

深度学习是一种表示型的机器学习模型。

在经典的机器学习模型中，开发者需要手动从复杂的输入中抽取指定的特征值输入映射函数中，而在表示型的模型中，特征向量是由程序自动提取的。提取特征向量的目的是避免“维度诅咒”，这个词汇描述的是在输入向量增加的同时，为了能尽可能均匀的采集到输入向量线性空间中的每个点，我们需要的数据集数量指数级暴增的情况。通过提取特征向量，我们可以大幅度减少提供给深度学习模型的数据集数量，以此降低训练模型的成本。

为了能实现从N维输入向量中提取M维特征向量，表示型学习模型需要通过一个训练器额外生成一个M×N的矩阵来实现对输入向量集的线性映射。由于这些特征的提取是自动的，所以自然不存在标记数据，提取特征的过程是一种无监督学习。

在深度学习模型中，我们提取一些“原始特征向量”，比如动画中的DOF值或IK控制器坐标等。深度学习也需要训练一个用于提取特征向量的模型，但相比其它的表示型模型，深度学习没有将训练特征提取模型和训练输出模型的训练器分离，而是将这两个部分统一了起来。所以在整个训练过程中，开发者只需要管理输入向量和输出向量，也就是所谓的“端到端”的表示学习方法[2][3]。

深度学习起源于数学与工程学，

##### PyTorch框架概述

开发深度学习网络是一种基本技能。主流的深度学习框架有Tensor Flow和PyTorch等，这里我们主要使用PyTorch，此处选择PyTorch，是因为相比于Tensor Flow，PyTorch框架上手更简单，具有更多的网络模板，并且已经实现了自动的梯度计算流程。

### 11.2 可微物理动画

### 参考文献

[1] Blum A, Mitchell T. Combining labeled and unlabeled data with co-training[C]//Proceedings of the eleventh annual conference on Computational learning theory. 1998: 92-100.

[2] LeCun Y, Bengio Y, Hinton G. Deep learning[J]. nature, 2015, 521(7553): 436-444.

[3] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.