第十八章　动画原理与人物动画

在前面的章节中，我们关注的都是如何用计算机表达或显示一个**静态**的场景，而从本章开始我们将带领读者了解图形学的另外一个维度——动画．从现在开始，我们要处理和表达的对象将是一个**动态**的场景序列，这与我们之前关注的问题有很大的不同，因为我们引入了一个时间参数．

本章将带领读者了解图形学如何建模、处理以及生成人体产生的动态效果．虚拟角色是我们在游戏和科幻电影里常见的组成部分，然而我们是否考虑过，动画师们是如何让这些角色自然地运动起来的，以及当我们操控角色时，我们的控制信号又是如何转化为角色的动作的呢？在接下来的章节，我们会先介绍基本的动画原理 (第18.1节至第18.3节)，然后介绍人物动画的相关技术 (第18.4节、第18.5节)．

18.1　旋转的表示

在模拟一个物理世界之前，我们首先需要理解如何表示这一个物理世界，表示物体的所处的状态．

对于一个三维空间内的刚体而言，其拥有六个自由度，三个自由度来源于物体沿着前后，上下，左右三个相互垂直方向进行平移运动，剩余三个自由度来源于物体沿着前后，上下，左右三个相互垂直的坐标轴的进行旋转．

前三个表示位置的自由度我们可以使用物体质心，同三维坐标系原点之间的相对位移的向量 (x, y, z) 来表示．后三个表示物体旋转的自由度的表示方法则较为多样，下文我们会依次描述．

18.1.1　旋转矩阵

二维旋转矩阵

我们先回顾一下二维空间内的旋转．如下图所示，二维空间内的旋转仅有一个自由度．

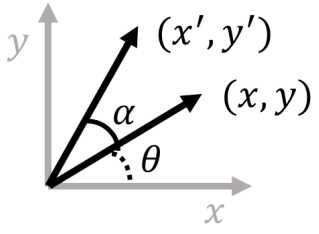


图 18.1: 二维空间内的向量旋转

因而我们不妨假设通过该旋转，平面上的向量 (x,y) 绕着原点逆时针旋转 α 弧度，得到旋转后的向量 (x',y')．假定 (x,y) 是由沿着 x 轴正方向长度为 r 的向量，绕着原点逆时针旋转 θ 弧度得到的，据此可以列出方程：

$$\begin{cases} x &= r\cos\theta \\ y &= r\sin\theta \end{cases} \tag{18.1}$$

$$\begin{cases} x' &= r\cos(\theta+\alpha) \\ y' &= r\sin(\theta+\alpha) \end{cases} \tag{18.2}$$

利用三角函数和差公式，则可以得到:

$$\begin{cases} x' &= r\cos\theta\cos\alpha - r\sin\theta\sin\alpha &= x\cos\alpha - y\sin\alpha \\ y' &= r\sin\theta\cos\alpha + r\cos\theta\sin\alpha &= x\sin\alpha + y\cos\alpha. \end{cases} \tag{18.3}$$

因而我们可以将上面的式子写成矩阵乘法的形式：

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{18.4}$$

绕轴旋转矩阵

在三维空间内，不妨首先考虑绕着 z 轴进行的旋转．假设旋转前向量为 $\mathbf{P}_0=(x,y,z)^\top$，旋转后向量为 $\mathbf{P}_0'=(x',y',z')^\top$．

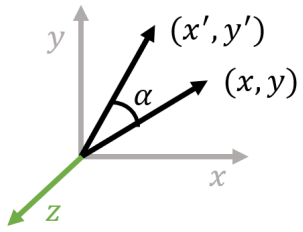


图 18.2: 三维空间内的向量绕 z 轴旋转

对于 z 分量，由于 z 分量在绕 z 轴旋转时保持不变，因此 $z'=z$．对于 xy 分量，由于 z 分量保持不变，因此向量旋转方式等同于在 xy 平面上进行的二维旋转．据此我们可以得到矩阵形式：

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{18.5}$$

类比于绕 z 轴旋转的情况，我们可以分别得到绕 x 轴，y 轴旋转的矩阵表示:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{18.6}$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{18.7}$$

将绕 z, x, y 轴进行旋转的矩阵分别记为 $R_z(\alpha), R_x(\alpha), R_y(\alpha)$, 则公式 18.5,18.6,18.7可以分别简写为:

$$\begin{aligned} \mathbf{P}_0 R_z(\alpha) &= \mathbf{P}'_0 \\ \mathbf{P}_0 R_x(\alpha) &= \mathbf{P}'_0 \\ \mathbf{P}_0 R_y(\alpha) &= \mathbf{P}'_0 \end{aligned}$$

三维正交变换

我们已经推导完毕了三维空间内绕坐标轴的旋转矩阵，但是对于不是绕着坐标轴进行的旋转仍然无能为力．因此我们考虑对空间进行正交变换，使得变换后的空间内旋转轴恰好为三维旋转中的某一个．

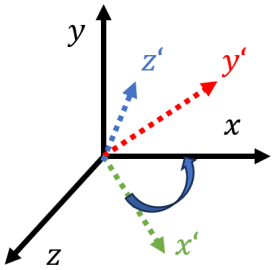


图 18.3: 三维空间内的正交变换

我们不妨考虑从正交坐标系 xyz 变换到另一个正交坐标系 $x'y'z'$，如图 18.3 所示．我们希望能够构建矩阵 R, 使得其满足:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = R \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{18.8}$$

我们考虑怎么求解矩阵 R.

为了将沿 x 轴正方向的单位向量 $\mathbf{e}_x = (1,0,0)^\top$ 变换到 x' 下，我们不难发现，旋转矩阵 R 第一列的取值，恰好为 xyz 坐标系下沿着 x' 正方向单位向量的取值 $\mathbf{e}'_x$．类似的，我们可以说明旋转矩阵 R 的第二，三列的取值，分别为 xyz 坐标系下沿着 y', z' 正方向单位向量的取值 $\mathbf{e}'_y, \mathbf{e}'_z$．据此我们有:

$$R = \begin{pmatrix} \mathbf{e}'_x & \mathbf{e}'_y & \mathbf{e}'_z \end{pmatrix} \tag{18.9}$$

如果我们从变换后的正交坐标系 $x'y'z'$ 变回正交坐标系 xyz，则我们需要求解 R^{-1}．由于矩阵 R 的每一列模长均为 1，且根据正交坐标系的性质知道 $\mathbf{e}'_x, \mathbf{e}'_y, \mathbf{e}'_z$ 三者相互垂直，因此我们不难证明 $R^\top R = I$．因此 R 必然为正交矩阵，$R^{-1} = R^\top$．

正交矩阵有如下几种性质：

- 每一列代表变换后对应的坐标轴.
- 所有列向量均为单位向量，列向量之间两两正交.
- 假设 A 为正交矩阵，则 $A^{-1}=A^{\top}$, $AA^{\top}=I$, A 的行列式值为 ± 1.
- 假设 A,B 为正交矩阵, 则 $A^{\top},AB$ 也是正交矩阵.

经过检查，我们发现式18.5,18.6,18.7中的矩阵均为正交矩阵.

三维旋转矩阵

现在去我们需要面对绕任意轴的问题. 假定我们需要绕旋转轴 u 旋转 α 弧度，利用已有的知识，我们可以尝试给出如下的解法：

- 通过旋转轴构造正交坐标系 $x'y'z'$, 使得 x' 恰好为 u.
- 将正交坐标系 $x'y'z'$ 变换为正交坐标系 xyz, 此时旋转轴 u 经过变换后恰好为 x 轴.
- 绕着 x 轴旋转 α 弧度.
- 将正交坐标系 xyz 变换回 $x'y'z'$.

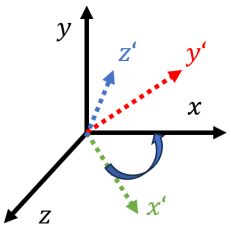


图 18.4: 正交坐标系 $x'y'z'$ 变换为正交坐标系 xyz

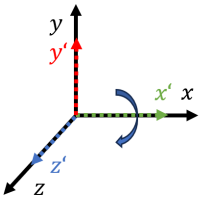


图 18.5: 绕 x 轴进行旋转

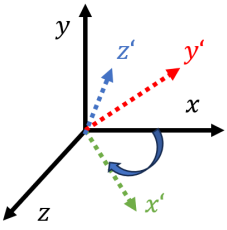


图 18.6: 正交坐标系 xyz 变换回正交坐标系 $x'y'z'$.

此时我们需要考虑的便仅有：如何构造满足条件的正交坐标系 $x'y'z'$?

- 取任一单位向量 t, 其与 u 不共线.
- 取 $w=t\times u$, 此时有 t,w 垂直.
- 取 $v=u\times w$, 此时有 u,w,v 三者两两垂直.

此时坐标系 uvw 正交，因此可以取正交坐标系 $x'y'z'$ 为正交坐标系 uvw，经验证其满足所有约束条件.

假定从正交坐标系 xyz 变换到正交坐标系 $x'y'z'$ 的正交变换矩阵为 $R=(uvw)$. 设旋转前向量坐标为 $\mathbf{P}_0$, 旋转后向量坐标为 $\mathbf{P}_0'$, 则有恒等式:

$$\mathbf{P}_0'=RR_x(\alpha)R^{\top}\mathbf{P}_0 \tag{18.10}$$

据此，绕旋转轴 u 旋转 α 弧度的旋转矩阵为 $RR_x(\alpha)R^{\top}$.

事实上，根据 Rodrigues 公式，存在一个更为直接的旋转矩阵求解方法. 假定我们需要绕旋转轴 u 旋转 α 弧度，则旋转矩阵 R 满足:

$$R=I+\sin\alpha[u]_{\times}+(1-\cos\theta)[u]_{\times}^2 \tag{18.11}$$

其中 $[u]_{\times}=\begin{pmatrix}0 & -u_z & u_y\\ u_z & 0 & -u_x\\ -u_y & u_x & 0\end{pmatrix}$.

18.1.2　四元数

二维平面复数

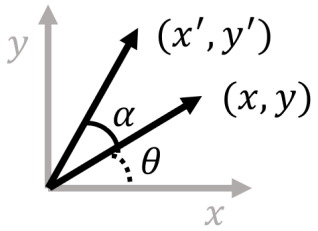


图 18.7: 二维空间内的复数乘法与向量旋转

对于一个复数，其可以被记为 $z = x + y\mathrm{i} \in \mathbb{C}$，其中 $x, y \in \mathbb{R}$，$\mathrm{i}^2 = -1$．我们将该复数与平面上的向量 (x, y) 做一一对应．

根据欧拉公式，则有 $z = r\mathrm{e}^{\mathrm{i}\theta} = r(\cos\theta + \mathrm{i}\sin\theta)$．因而我们可以得到，逆时针将向量 (x, y) 旋转 a 弧度后得到的向量，以复数形式表示为 $r\mathrm{e}^{\mathrm{i}(\theta+\alpha)}$．同时利用 $r\mathrm{e}^{\mathrm{i}(\theta+\alpha)} = \mathrm{e}^{\mathrm{i}\alpha} r\mathrm{e}^{\mathrm{i}\theta}$，因此我们可以声明：单位复数 $\mathrm{e}^{\mathrm{i}\alpha}$ 可以表示绕着原点逆时针旋转 α 弧度的旋转．

四元数定义与运算

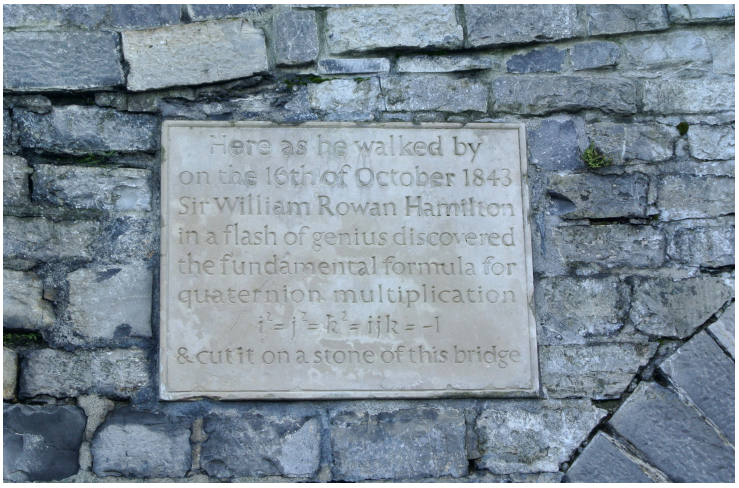


图 18.8: Hamilton 发明了四元数

定义四元数 $q = a + b\mathrm{i} + c\mathrm{j} + d\mathrm{k}$，其中 $a, b, c, d \in \mathbb{R}$．类比于二维复数运算，四元数虚部同样满足 $\mathrm{i}^2 = \mathrm{j}^2 = \mathrm{k}^2 = -1$.

四元数虚部 i, j, k 相互的运算法则可以类比于三维空间上的叉积运算，i, j, k 分别对应 x, y, z 轴上的单位向量．据此我们可以得到如下运算法则：

$$\begin{aligned} \mathrm{ij} &= \mathrm{k} \quad & \mathrm{jk} &= \mathrm{i} \quad & \mathrm{ki} &= \mathrm{j} \\ \mathrm{ji} &= -\mathrm{k} \quad & \mathrm{kj} &= -\mathrm{i} \quad & \mathrm{ik} &= -\mathrm{j} \end{aligned} \tag{18.12}$$

由于四元数虚部运算规则与三维向量叉积几乎相同，因此四元数也可以表示成实部 $w = a$ 和虚部向量 $\mathbf{v} = (b, c, d)^\top$ 拼接而成的形式，记作 $[w, \mathbf{v}]$.

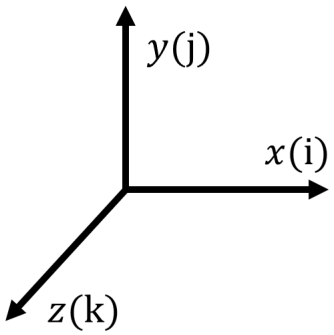


图 18.9: 四元数虚部和右手坐标系 xyz 的对应

四元数的加减法，模长，数乘，点乘运算法则与传统的四维向量运算规则类似．假定 $q_1 = a_1 + b_1\mathrm{i} + c_1\mathrm{j} + d_1\mathrm{k} = [w_1, \mathbf{v}_1]$,$q_2 = a_2 + b_2\mathrm{i} + c_2\mathrm{j} + d_2\mathrm{k} = [w_2, \mathbf{v}_2]$, 则运算规则如下：

$$\begin{array}{lcl} q_1 \pm q_2 & = & (a_1 \pm a_2) + (b_1 \pm b_2)\mathrm{i} + (c_1 \pm c_2)\mathrm{j} + (d_1 \pm d_2)\mathrm{k} \\ tq_1 = ta_1 + tb_1\mathrm{i} + tc_1\mathrm{j} + td_1\mathrm{k} q_1 \cdot q_2 & = & a_1a_2 + b_1b_2 + c_1c_2 + d_1d_2 \\ \|q_1\|_2 & = & \sqrt{q_1 \cdot q_1} \end{array} \tag{18.13}$$

针对四元数的乘法运算，由于我们已经知道了虚部两两相乘的结果，因此我们可以对其作暴力展开，展开结果如下，共 16 项．

$$\begin{array}{rcl} q_1q_2 & = & (a_1 + b_1\mathrm{i} + c_1\mathrm{j} + d_1\mathrm{k})(a_2 + b_2\mathrm{i} + c_2\mathrm{j} + d_2\mathrm{k}) \\ & = & (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) \\ & + & (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\mathrm{i} \\ & + & (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\mathrm{j} \\ & + & (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\mathrm{k} \end{array} \tag{18.14}$$

为了简便起见，我们一般使用实部与虚部向量的方式表示四元数乘法．使用实部与虚部之后，四元数乘法具体规则如下：

$$q_1q_2 = [w_1w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2] \tag{18.15}$$

四元数的共轭定义与复数共轭定义类似，均为将虚部取反的结果．例如四元数 $q = [w, \mathbf{v}]$ 的共轭则为 $q^\star = [w, -\mathbf{v}]$．

利用四元数乘法与共轭运算的定义，我们不难得出 $qq^\star = [w^2 + \mathbf{v} \cdot \mathbf{v}] = \|q\|^2$．因此我们可以构造任意非零四元数 q 的逆元为 $q^{-1} = \frac{q^\star}{\|q\|^2}$．与复数的逆元类似，四元数的逆元同样满足 $qq^{-1} = q^{-1}q = [1, 0]$．

定义单位四元数为所有模长为 1 的四元数，也就是所有位于四维超球面上的四元数．根据定义不难得出单位 q 满足 $q^{-1} = q^\star$．类似于单位复数的分解 $z = \cos\theta + \sin\theta\mathrm{i}$，我们对于单位四元数也可以做同样的虚部实部分解 $q = [\cos\theta, \mathbf{u}\sin\theta]$，其中 $\mathbf{u}$ 是单位三维向量．

四元数表示旋转

通过引入四元数，我们尝试证明如下结论，来说明其在进行三维旋转操作之中的便利性:

四元数与三维旋转等价性　假定 $q = [\cos\theta, \mathbf{u}\sin\theta]$ 是一个单位四元数，其中 $\mathbf{v}$ 是三维空间内的单位向量，$v = [0, \mathbf{v}]$ 是一个实部为 0 的四元数．则有:

- $qvq^\star$ 实部为 0．
- $qvq^\star$ 虚部为向量 $\mathbf{v}$ 绕旋转轴 $\mathbf{u}$ 旋转弧度 2θ 后的结果．

证明　我们可以将向量 $\mathbf{v}$ 分解为平行于旋转轴 $\mathbf{u}$ 以及与旋转轴正交 (垂直) 的两个分量 $\mathbf{v}_\parallel$ 和 $\mathbf{v}_\perp$．接下来我们只需要说明，平行分量在进行上述变换后保持不变，垂直分量在进行上述变换后等价于在与垂直于旋转轴 $\mathbf{u}$ 的平面内逆时针旋转了 2θ 弧度即可．

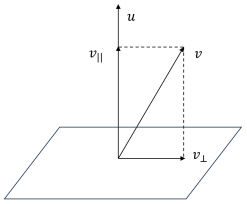


图 18.10: 向量 $\mathbf{v}$ 的分解

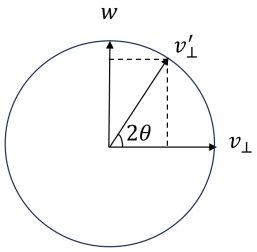


图 18.11: 向量 $\mathbf{v}$ 旋转的顶视图

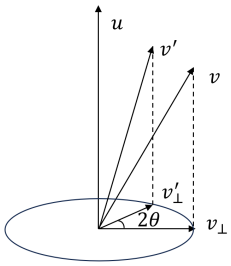


图 18.12: 向量 $\mathbf{v}$ 旋转的前视图

首先我们证明平行分量在进行上述变换后保持不变，且实部为 0 ．不妨假设 $\mathbf{v} = \mathbf{u}$, 则根据四元数乘法规则有：

$$\begin{aligned} qvq^\star &= (\cos\theta + \mathbf{u}\sin\theta)(0 + \mathbf{u})(\cos\theta - \mathbf{u}\sin\theta) \\ &= (-\mathbf{u}\cdot\mathbf{u}\sin\theta + \mathbf{u}\cos\theta)(\cos\theta - \mathbf{u}\sin\theta) \\ &= (-\sin\theta + \mathbf{u}\cos\theta)(\cos\theta - \mathbf{u}\sin\theta) \\ &= -\sin\theta\cos\theta + \mathbf{u}\cos\theta\cdot\mathbf{u}\sin\theta + ((-\sin\theta)^2 + (\cos\theta)^2)\mathbf{u} \\ &= 0 + \mathbf{u} \end{aligned} \tag{18.16}$$

这说明平行分量在进行上述变换后保持不变，且实部为 0.

之后我们证明垂直分量在进行上述变换后等价于在与垂直于旋转轴 $\mathbf{u}$ 的平面内逆时针旋转了 2θ 弧度．根据假设我们有 $\mathbf{v}\cdot\mathbf{u} = 0$．取单位向量 $\mathbf{w} = \mathbf{u}\times\mathbf{v}$, 则 $\mathbf{w}\cdot\mathbf{v} = \mathbf{w}\cdot\mathbf{u} = 0$. 据此有：

$$\begin{aligned} qvq^\star &= (\cos\theta + \mathbf{u}\sin\theta)(0 + \mathbf{v})(\cos\theta - \mathbf{u}\sin\theta) \\ &= (\mathbf{v}\cos\theta + \mathbf{w}\sin\theta)(\cos\theta - \mathbf{u}\sin\theta) \\ &= \mathbf{v}\cos^2\theta + \mathbf{w}\sin\theta\cos\theta - \mathbf{v}\cos\theta\times\mathbf{u}\sin\theta - \mathbf{w}\sin\theta\times\mathbf{u}\sin\theta \\ &= \mathbf{v}(\cos^2\theta - \sin^2\theta) + \mathbf{w}2\sin\theta\cos\theta \\ &= \mathbf{v}\cos 2\theta + \mathbf{w}\sin 2\theta \end{aligned} \tag{18.17}$$

这说明垂直分量在进行上述变换后等价于在与垂直于旋转轴 $\mathbf{u}$ 的平面内逆时针旋转了 2θ 弧度，且实部为 0.

由于任意向量 $\mathbf{v}$ 均可以分解成关于旋转轴 $\mathbf{u}$ 的垂直分量和平行分量线性叠加的结果，因此 $qvq^\star$ 实部必然为 0，虚部必然为向量 $\mathbf{v}$ 绕旋转轴 $\mathbf{u}$ 旋转弧度 2θ 后的结果，原命题得证.

根据18.1.2的结论，对于将向量 $\mathbf{v}$ 沿着单位长度旋转轴 $\mathbf{u}$ 旋转 θ 弧度的问题，可以按如下方式处理:

- 构造四元数 $q = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2}$，$v = [0, \mathbf{v}]$
- $qvq^\star$ 的虚部即为旋转后得到的结果.

同时，由于对于任意四元数 q, v, $qvq^\star = (-q)v(-(q^\star)) = (-q)v(-q)^\star$ 恒成立，因此单位四元数 q 和 $-q$ 表示的是一样的旋转.

假定我们要对向量 v 顺次进行单位四元数 q_1 和 q_2 所表示的旋转操作，则其结果为 $q_2(q_1 v q_1^\star)q_2^\star$. 类比于复数共轭运算的性质，我们不难得出 $(q_2q_1)^\star = q_1^\star q_2^\star$，因此上式可以进一步简化为 $(q_2q_1)v(q_2q_1)^\star$. 因此，先进行单位四元数 q_1 表示的旋转，再进行单位四元数 q_2 表示的旋转，复合之后旋转的四元数表示即为 q_2q_1.

四元数插值

给定两个单位四元数 p, q，和一个参数 $t \in [0, 1]$, 我们希望能够找到这么一个中间四元数 $r(p, q, t)$ 作为插值结果，使得当 t 取值从 0 变为 1 的时候，$r(p, q, t)$ 取值能够平滑的从 p 过渡到 q. 对于二元插值问题，常用的解法是取 $r(t, p, q) = a(t)p + b(t)q$ ，使用 p, q 的加权求和作为插值结果.

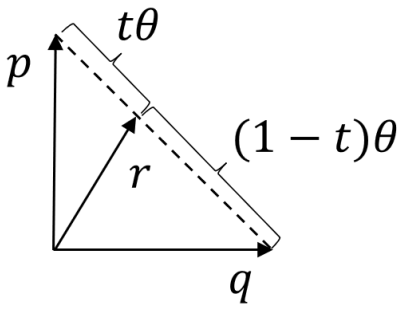


图 18.13: 四元数的线性插值 (Lerp)

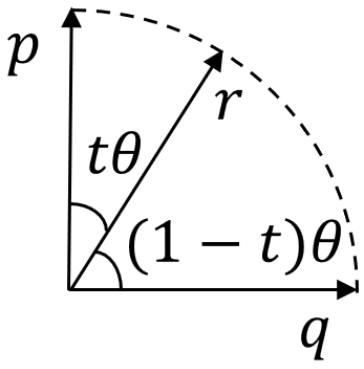


图 18.14: 四元数的球面线性插值 (Slerp)

最简单的构造方法是进行线性插值，也就是取 $a(t) = 1 - t, b(t) = t$. 此时 $r(p, q, t) = (1 - t)p + tq$. 但是线性插值效果并不理想. 一方面，线性插值的结果都落在四维超球面的二维弦上，因此插值结果不一定是单位四元数. 即使通过归一化将结果变为单一四元数，但是其在球面上的运动速度并不均匀. 不难得出当 t 在 $[0, 1]$ 内匀速运动时，归一化之前四元数模长越小，归一化后其在球面上的运动速度越快.

另外一种想法是在四维超球面上进行线性插值. 假定 p, q 之间的夹角为 θ，也即 $\cos\theta = p \cdot q$, 我们找出球面上位于 p, q 连线上的点 r, 使得 $p \cdot r = \cos t\theta$, $q \cdot r = \cos[(1 - t)\theta]$，此时球面上的线性插值就可以克服普通线性插值的所有问题. 仍然假设 $r = a(t)p + b(t)q$, 我们考虑如何求解 $a(t)$ 和 $b(t)$ 的取值.

对插值公式两侧点乘 p，得到：

$$\begin{aligned} r \cdot p &= a(t)p \cdot p + b(t)q \cdot p \\ \cos t\theta &= a(t) + b(t)\cos\theta \end{aligned} \tag{18.18}$$

对插值公式两侧点乘 q，得到：

$$\begin{aligned} r \cdot q &= a(t)p \cdot q + b(t)q \cdot q \\ \cos[(1-t)\theta] &= a(t)\cos\theta + b(t) \end{aligned} \tag{18.19}$$

联立方程18.18,18.19, 解得:

$$\begin{aligned} a(t) &= \frac{\cos t\theta - \cos[(1-t)\theta]\cos\theta}{1-\cos^2\theta} = \frac{\sin[(1-t)\theta]}{\sin\theta} \\ b(t) &= \frac{\cos[(1-t)\theta] - \cos t\theta\cos\theta}{1-\cos^2\theta} = \frac{\sin t\theta}{\sin\theta} \end{aligned} \tag{18.20}$$

上述插值方法被称为球面线性插值 (spherical linear interpolation, Slerp), 其公式为:

$$\mathrm{Slerp}(p,q,t) = \frac{\sin[(1-t)\theta]p + \sin t\theta q}{\sin\theta} \tag{18.21}$$

18.1.3　欧拉角

任何三维旋转实际上都可以分解成三个基本旋转的叠加. 这三种基本旋转就是上文提到的绕 x, y, z 轴的旋转，由公式18.5,18.6,18.7表示. 绕 x, y, z 轴的旋转也被分别称为 Pitch(俯仰)，Yaw(偏航), Roll(翻滚).

欧拉角没有规定旋转叠加的顺序，因此所有 $3! = 6$ 种叠加顺序都是可行的. 例如，如果先绕 x 轴旋转 α 弧度，再绕 y 轴旋转 β 弧度，最后绕 z 轴旋转 γ 弧度，则最后得到的旋转矩阵为 $R = R_z(\gamma)R_y(\beta)R_x(\alpha)$. 另外，包括 $x-y-x$ 在内的其余 6 种叠加方式也可以表达出所有的三维旋转.

欧拉角最大的问题是会遇到万向锁. 欧拉角的表示方式可以视为一个被固定在三维陀螺仪内的物体，旋转按照由内到外的方式依次叠加. 如果陀螺仪的三个轴，至少有两个落在了同一直线上，那么旋转着两个轴所带来的效果是一模一样的，因此在该状态下欧拉角的自由度最多为 2. 但是旋转空间的包含 3 个自由度，因而出现了自由度丢失的现象.

18.1.4　轴角表示法

在18.1.1节中我们提到，旋转都可以表示为绕单位长度旋转轴 u 旋转 α 弧度的形式. 而轴角表示法使用向量 αu 来表示旋转. 该表示法中，归一化的向量方向表示旋转轴，向量长度表示旋转大小. 根据轴角表示法的性质，我们不难发现: 对于任意 $k \in \mathbb{N}$, $(\alpha + 2k\pi)\mu$ 表示的都是一样的旋转.

利用上文提到的公式18.11, 我们可以快速地实现从轴角表示法到旋转矩阵的转换.

18.2　运动学

18.2.1　前向运动学

通过上面的讲述，我们已经学会如何表示一个刚体所处的状态. 但是现实世界中，绝大多数物体不满足刚体的性质. 以人体为例，由于存在可以自由活动的关节，人体不应被简化

为一个刚体，而应该简化成通过关节相互连接的类似于树形结构的铰接刚体．由于刚体不可形变的性质，我们可以进一步地将刚体简化为连接关节之间的骨骼，并通过关节的旋转和相对位移唯一确定刚体的 6 个自由度的信息．如图18.16所示，我们将足部，大腿，小腿等在运动中结构几乎不动的部位视为刚体．

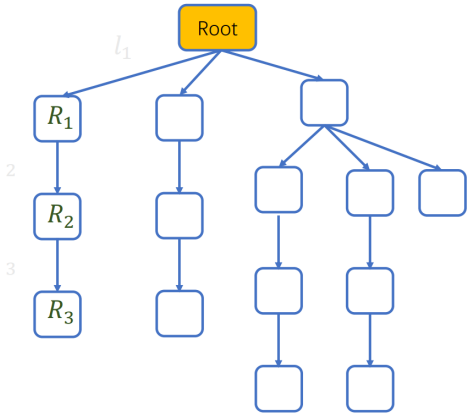


图 18.15: 关节与骨骼形成的树结构

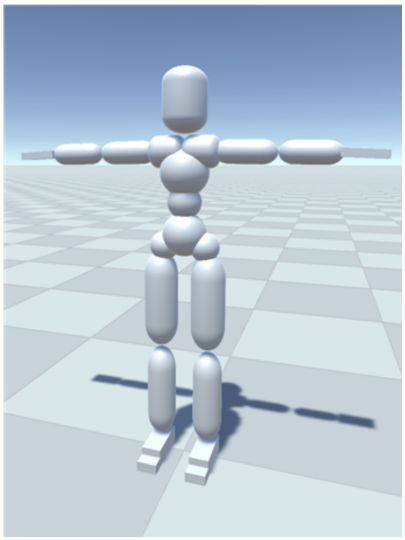


图 18.16: T-pose

对于人体而言，我们可以将进一步整个铰接刚体系统抽象为一棵树，如图18.15所示．通常我们选用腰部的关节作为整棵树的根关节，其余关节通过骨骼直接或者间接的同根关节连接．除根关节外，其余所有关节可以类比于树，定义其唯一的父关节，因此我们可以使用当前关节同父关节的相对旋转，来唯一确定其所处的状态．但是我们希望，从相对旋转中额外解算出关节所处的位置，方便进行更深度的探索．这个问题便是前向运动学 (Forward Kinematics)．

我们称关节相对于 xyz 正交坐标系的旋转为全局旋转，相对于父关节表示的局部正交坐标系的旋转为局部旋转．对于根节点，其局部旋转即为其全局旋转．当所有关节的旋转为 0 时，整个系统有一个初始的姿态，在该姿态下所有关节的全局旋转均为 0．对于人体，常见的初始姿态下，人体如图18.16所示，身体直立并且手臂平举，类似于英文字母 T, 因而被称为 T-pose．以图18.17为例，我们假设有一个仅由三根骨骼，四个关节组成的铰接刚体系统，且 P_0 是系统的根关节．由于此时所有关节全局旋转均为 0, 此时我们可以将父关节的位置 P_f，简单加上骨骼的长度 $l_{f\sim i}$，得到子关节的位置 P_i．据此我们有 $P_1 = P_0 + l_{0\sim 1}, P_2 = P_1 + l_{1\sim 2}, P_3 = P_2 + l_{2\sim 3}$．

但是当系统不处于初始姿态下时，我们需要首先求出当前关节的全局旋转 q_i．如果关节 i 就是系统的根关节，则此时当前关节相对于 xyz 正交坐标系的旋转，即为当前关节的局部旋转．假定 $q_i^{\texttt{local}}$ 代表当前关节的局部旋转，则有 $q_i = q_i^{\texttt{local}}$．否则我们假定父节点 q_f 的全局旋转已知，则关节 i 的全局旋转等于关节 i 相对于父节点 f 正交坐标系的局部旋转 $q_i^{\texttt{local}}$，复合上父节点 f 正交坐标系相对于 xyz 正交坐标系的全局旋转 q_f．也就是说，此时 $q_i = q_f q_i^{\texttt{local}}$．获得了所有关节的全局变换后，我们可以获知经过旋转后，子节点 i 相对父节点 s 的位移为 $l'_{f\sim i} = q_f l_{f\sim i} q_f^\star$．最后类比于初始姿态的做法, 我们将位移顺次相加后，便得到了当前节点的坐标．

仍然以图18.17为例,首先我们计算得到所有关节的全局旋转 $q_0 = q_0^{\texttt{local}}, q_1 = q_0 q_1^{\texttt{local}}, q_2 =$

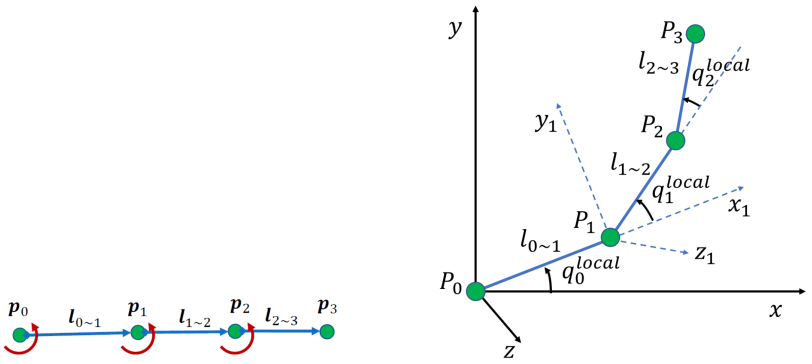


图 18.17: 简化铰接刚体系统

$q_1 q_2^{\texttt{local}}$. 之后我们计算通过应用全局旋转后，关节相对于其父关节的位移 $l'_{0\sim1} = q_0 l_{0\sim1} q_0^\star, l'_{1\sim2} = q_1 l_{1\sim2} q_1^\star, l'_{2\sim3} = q_2 l_{2\sim3} q_2^\star$. 最后我们将位移顺次相加，得到 $P_1 = P_0 + l'_{0\sim1}, P_2 = P_1 + l'_{1\sim2}, P_3 = P_2 + l'_{2\sim3}$.

基于前向运动学的原理，BioVision 公司开发了用于表示人体动捕数据的文件格式 bvh. 其包含了旋转为 0 时候关节信息的树形结构，也包含了每一帧根关节的相对位移，每一帧关节的相对旋转信息. 根据这些信息便能够方便的还原出人体动作. 限于篇幅 bvh 文件格式的细节不再展开.

18.2.2　关键帧动画

基于前向运动学，我们可以使用关节旋转和根节点位移的序列，来表示一段连续的动作序列. 但是如果动作序列的所有信息都需要我们一个一个的输入，其无论是时间成本还是制作成本，都会变得非常高昂.

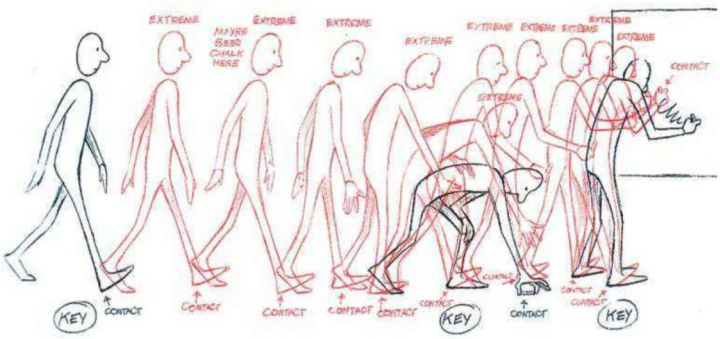


图 18.18: 关键帧动画

在手工绘制动画的时代，动画画师也会遇到相同的问题，针对该问题动画界给出的答案是中间画 (in between). 动画师只绘制包含关键转折在内的关键帧，而在关键帧之间的起到衔接过渡的较为次要的动画，会由另一批画师绘制. 这样子便可以让多个画师同时参与动画的绘制，同时能够较好地解决画师之间不同风格带来的问题.

类似于中间画的做法，运动学上也可以只编辑关键帧，通过关键帧之间进行插值的方式来降低生成动画的复杂性，但是此时我们可以使用计算机进行自动的中间动画生成. 针对根节点的位置，我们可以采用 Bezier 曲线的方式对于根节点所处的位置进行插值. 针对

关节的旋转，我们则可以使用 Slerp 公式18.20进行更为平滑的旋转插值，也可以进一步的将 Bezier 曲线的思想应用到旋转插值上．由于关键帧之间不存在较为显著的位置，旋转的突变，因此使用上述算法进行动作的平滑过渡即可实现很好的插值结果．

18.2.3　逆向运动学

前向运动学 (Forward Kinematics) 所解决的问题是，根据根节点的位置以及关节的局部旋转，还原出所有关节的位置．而逆向运动学 (Inverse Kinematics) 的任务恰恰相反，其需要根据给定的部分关节的位置 (包括但是不仅限于末关节)，反向求解出一组关节局部旋转的解．逆向运动学可以被用于机械臂或者是虚拟角色的控制，也可以帮助修复一些质量较差，没有满足部分接触要求的动作捕捉数据．

相比于前向运动学，逆向运动学面临最大的挑战便是解空间的复杂性．一方面，由于铰接刚体系统自身的限制，并不是对于任意的关节位置要求，其都存在一种合法的关节旋转，如图18.19所示．另一方面，对于有解的关节位置要求，其很可能存在多种不同的解，如图18.20所示．而此时我们希望能够选取出更为自然的解．一般来说，我们会更倾向于关节旋转更为平滑，相邻帧解更为接近的方案．

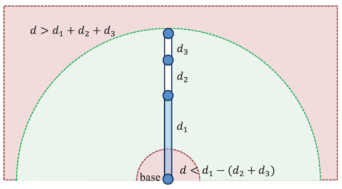


图 18.19: 末端点在红色区域内时系统无解

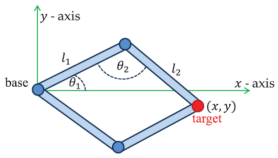


图 18.20: 系统存在两个不同的解

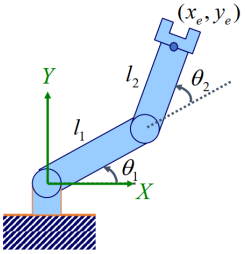


图 18.21: 双骨骼系统

首先我们从较为简单的双骨骼逆向运动学介绍．如图18.21所示，双骨骼系统由三个关节与两个骨骼组成，利用前向运动学不难得到其末端关节位置为:

$$\begin{aligned} x_e &= l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ y_e &= l_1 \sin\theta_1 + l_2 \arcsin(\theta_1 + \theta_2) \end{aligned} \tag{18.22}$$

在给定末端关节位置后，三个关节两两之间的距离都已经确定，因此我们可以唯一确定三个关节所确定的三角形并求解对应的角度．由于具体推导过程较为复杂，这里略去其推导细节．

对于恰好只有两个个骨骼的逆向运动学任务，其存在一个较为复杂的数值解，但是但是对于骨骼数量大于两个的逆向运动学任务，其并不存在关节角度的数值解．因此我们只能通过迭代的方式从数值上不断逼近这个解．一种较为通用的逆向运动学算法是循环坐标推演 (Cyclic Coordinate descent, CCD)．

循环坐标推演的思想是，每一次调整一个关节，使得当前关节旋转后，末端关节距离目标点尽可能地接近．更具体的，假设当前铰接刚体系统总共有 $n+1$ 个关节，与 n 个连接关节的骨骼，我们希望末端的 n 号关节距离目标 t 尽可能地接近．我们会按照 $n-1, n-2, \cdots, 0$ 的顺序依次调整旋转．假定目前我们想要通过调整第 i 号关节的旋转，使得 n 号关节距离 t 尽可能地接近，则当且仅当 p_i, p_n, t 三点共线的时候 p_n, t 的距离最近，据此我们可以计算

得到需要对 i 号关节施加的旋转．我们可以通过重复上述过程若干轮，来实现对于目标更好地逼近．

例如，我们目前有一个三个骨骼的系统，并希望将该系统末端节移动到位于红色圈处的目标，如图18.22所示．首先我们旋转从根节点开始的第三个关节，使得最后一根骨骼对准目标点，也就是第三，四个关节和目标点三点共线，因为此时末端关节距离目标点的距离是最近的．第一次旋转如图18.23所示．之后我们旋转从根节点开始的第二个关节，固定第三个关节，使得第二，四个关节和目标点三点共线．第二次旋转后如图18.24所示．最后我们旋转根节点，使得第一，四个关节和目标点三点共线．第三次旋转后如图18.25所示．经过三次旋转，我们成功地将末端点移动到了一个距离目标更近的位置，但是仍然不够接近，因此我们可以继续重复上述过程，最终达到一个足够接近的解．

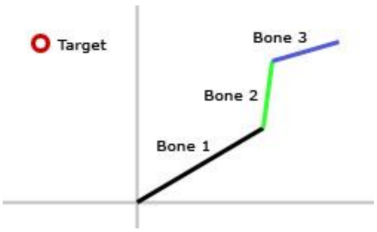


图 18.22: 初始状态

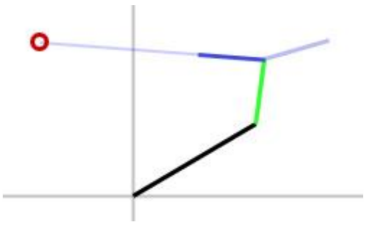


图 18.23: 第一次旋转后

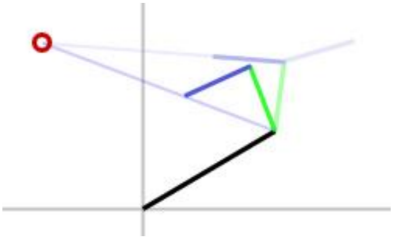


图 18.24: 第二次旋转后

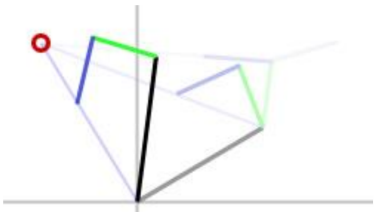


图 18.25: 第三次旋转后

循环坐标推演的优点与缺点都非常明显．从优点而言，循环坐标推演的实现非常简单，因此进行单次迭代复杂度也较低．同时，循环坐标推演可以很容易地扩展到包括局部约束，如允许的角度变换在每一步都有上下限．循环坐标推演也可以通过分割子链的方式，来处理更为复杂的树形关节结构．从缺点而言，循环坐标推演的运动分布较为不自然，其可能会产生大角度的旋转，导致对连续动作进行逆向运动学求解时，解的连续性较差．同时，对于精度要求较高的求解场景，其迭代效率也不够理想．针对这些缺点，后来提出了包括 Jacobian 矩阵法，FABRIK 法等，限于篇幅此处不再描述．

18.3　骨骼蒙皮

利用前向运动学，我们已经可以复原角色的关节位置了，但是此时的角色依然只具有一具由骨骼连接关节形成的骨架，因此我们需要使用第八章的几何模型给角色蒙上一层表皮，使得其看上去更俩类似于真实的角色．

对于静止的角色，其蒙皮固定不变，因此较容易进行蒙皮上顶点 (vertex)，面片 (mesh) 的生成与编辑．但是由于我们可以利用18.2.1节讲述前向运动学驱动骨架，因此我们需要给出根据骨架计算的顶点坐标，在进行后续的渲染工作．

为了带动皮肤，我们首先需要执行骨架与皮肤的对应关系．对应关系通常针对标准姿势进行．常见的标准姿势有18.2.1节提到的 T-pose．但是由于 T-pose 在肩膀处形变较大，绑定效果一般，因而也有部分采用了将手臂 45° 下垂的 A-pose．

产生标准姿势下的骨架结构与蒙皮定点后，我们便需要找到皮肤定点与骨架的对应关系．最简单的想法下，我们可以将顶点绑定到与其距离最近的骨骼上，并使用骨骼的旋转和位置驱动该顶点．对于绝大多是的顶点，其都能够找到唯一控制的骨骼．但是在关节分界的位置会出现顶点控制骨骼的突变，这导致关节处的蒙皮会有明显的割裂感．为了克服这个问题，工业界提出了线性混合蒙皮 (linear blend skinning) 的技术．

对于以骨骼 i 表示的坐标系，我们在给定时刻 t，通过前向运动学过程后，从标准姿势变成时刻 t 所处的状态时，进行的变换为 $M_i^{(t)}$．假设蒙皮上的顶点 i 在标准姿势下的坐标为 v_i，则通过线性混合蒙皮，最终其位置在 xyz 正交坐标系下的坐标为:

$$v_i' = \sum_{j=1}^{m} w_{i,j} M_j^{(t)} v_i = (\sum_{j=1}^{m} w_{i,j} M_j^{(t)}) v_i \tag{18.23}$$

这里 $w_{i,j}$ 表示第 j 根连杆对第 i 个顶点在施加变换之后的权重，且满足 $\sum_j w_{i,j} = 1$．通常情况下，至多只有 4 个关节的 $w_{i,j}$ 非零．不难发现，线性混合蒙皮只是将绑定到不同连杆的结果进行了加权求和，而加权的系数通常需要通过人力进行手工调整．

但是直接使用线性蒙皮的话，缺点依然存在．在部分关节局部旋转较大的时候，关节处可能会出现不必要的重叠与自交现象，导致出现类似于“糖纸”(candy wrapper) 的扭曲现象．为了避免这种现象，后续工业界提出了对偶四元数蒙皮，旋转中心蒙皮等多种方法来解决这种问题，限于篇幅这里不再展开．

在获得顶点坐标后，我们便可以根据第十四章的内容，对生成的蒙皮进行贴图，渲染等一系列操作．此时角色便不再是一具仅有骨骼的火柴人，更加像一个有血有肉的真实人类了．

18.4　动作捕捉

当我们有一个虚拟角色和它的骨骼框架以后，一个很直接的让其动起来的方法，就是手动指定每个关节的弯曲程度．整个过程就像操控一个玩具小人一样，我们将其摆成我们想要的姿势．然而这样只能得到一个静态的姿势，如何产生整个的运动呢？如果能结合之前讲过的插值内容，可以发现做法也是很简单的：我们把几个关键的姿势摆出来，通过插值产生姿势之间的切换，如图18.26．

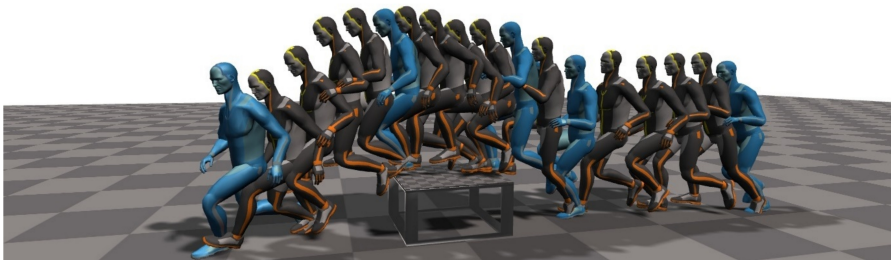


图 18.26: 通过蓝色关键帧和插值算法得到的角色动画 ©Robust Motion In-betweening

然而这样的方法只适用于大范围的，非精细的运动．由插值产生的动作往往较为平滑，缺少细节和风格．在一些快速的多变的运动中，我们需要非常多的关键帧才能确保动画的

质量，而这会导致我们的工作量大幅度上升.

这时我们就要依赖另一个产生动作数据的重要方法：动作捕捉 (Motion Capture)，一般读为 Mocap. 动作捕捉是通过设备记录人和物体运动的方法. 就角色来说，我们需要一些专业的动捕演员，让其穿戴特定的设备进行表演. 在每一时刻，演员的肢体运动甚至面部表情会都会被设备记录下来，转化成骨骼的旋转数据，记录完毕后将其交给后续的处理流程.

图 18.27: 动捕演员 ©Squeeze

相比于插值动画，动捕数据由于是对演员的动作进行了密集的捕捉，可以较为简单地获得非常风格化和流程的动作. 动作捕捉在如今有非常多的应用，下面列出了几个主要的方面：

- 休闲娱乐：如游戏、影视、虚拟偶像、元宇宙.
- 体育运动：如运专业体育训练、演出动作的优化.
- 医疗：用于帮助医生诊断、治疗一些与运动相关的疾病，以及受伤复建等.
- 机器人：例如对无人机、机械臂的跟踪和定位.

在进行动作捕捉时，我们需要一些设备来完成动作的提取和转化，根据不同的原理，我们可以将设备进行分类，我们将在下面的章节对这些设备进行简单的介绍.

18.4.1　外骨骼

基于外骨骼 (exoskeleton) 的动捕设备是一种早期较为常用的动捕设备，如图18.28所示. 外骨骼设备的原理十分简单，由于动捕演员穿戴了一套外骨骼，演员的肢体动作会带动外骨骼运动，通过测量外骨骼各关节的角度来近似人体上对应关节的角度；在拥有各关节角度之后，结合外骨骼设备的内参 (如每个部件的长度等)，我们就可以利用18.2.1节中的方法还原人体的动作了.

基于外骨骼的动捕技术最大的问题在于穿戴的设备特别妨碍动捕演员做动作，当我们需要做一些比较难或者比较危险的动作时 (比如躺在地上) 穿着外骨骼就很难实现. 但另一方面，外骨骼技术非常适用于手部动作的采集，因为手相对身体比较小，采用外骨骼技术能够获得更加准确的数据.

18.4.2　惯性传感器

基于惯性传感器的动捕技术 (如图18.29) 主要依赖于惯性测量单元 (Inertial Measurement Unit, IMU)，它一般是一个六轴传感器，包括 3 自由度的加速度计 (Accelerometers) 和 3 自由度的角速度计 (Gyroscope). 通过这些传感器的信息，我们可以得到人体各部位的相对运动，进而使用逆向运动学 (第18.2.3) 或者优化的方法来求解每个关节的旋转.

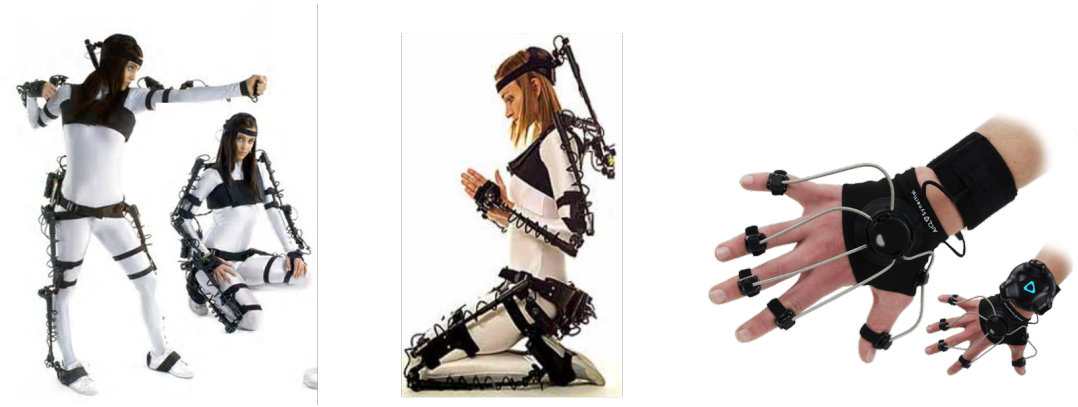


图 18.28: 基于外骨骼的动捕设备

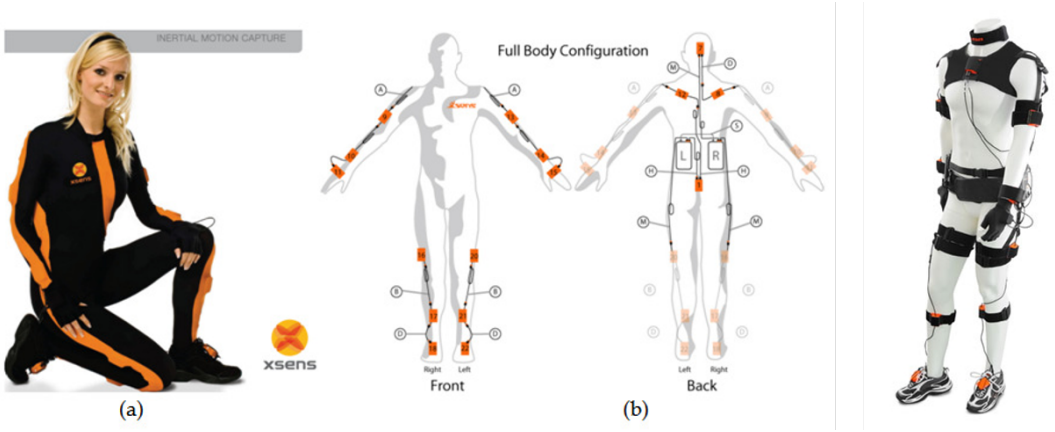


图 18.29: 基于惯性传感器的动捕技术

基于惯性传感器的动捕技术存在的一个不可避免的问题就是传感器的漂移．由于惯性传感器测量的是加速度或者速度，为了得到位置，需要对加速度或速度进行积分，而随着时间的推移，这个积分的误差会累积，导致得到的位置关系越来越不准确．在实践中，常常会因此看到人物会飘起来或者陷进地面．我们往往会借助其他技术来减少漂移带来的影响，比如一些启发性的辅助措施，例如我们可以借助“人的脚总是接触地面”这个先验知识来对整个人体的位姿加一个修正项；另外一方面我们也可以使用更精确的或者添加更多的传感器来减小积分时产生的误差，例如可以借助重力传感器来避免在竖直方向产生过大的漂移，类似还可以借助磁场传感器、光流传感器等．

18.4.3　光学动捕

光学动捕 (Optical Mocap) 是目前最为流形的一种动捕技术，其主要原理是通过相机来进行三维定位．动捕演员需要穿戴一些反光或发光的标记点 (如图18.30a)，此外场景中会放置多个 (一般是 6-8 个，如图18.30b) 摄像机，每个相机都能够拍摄到这些标记点，通过多个视角的标记点位置来重建每个标记点的三维坐标．于是，我们只需要在动捕演员身体上的每个关节处都放置若干个标记点，通过这些点的三维坐标就可以反推出每个关节的位置和朝向．

多视角重建

这里介绍一种利用两个相机重建标记点三维位置的方法．如图18.31所示，我们已知两个相机的焦距、位置、朝向信息，就可以在为每个相机作一个平面表示相机拍摄到的图片所在的屏幕，一个标记点会在两个屏幕上分别投射出一个点，我们从两个相机的焦点出发向各自屏幕上对应点引出一条射线，那么两条射线应当能够相交，这个交点就是还原出的三维位置．

这个方法虽然简单，但并不常用，因为实际情况中两个相机会带来较大的测量误差．一般来说我们至少需要 3 个相机来定位一个点，且相机越多误差越低，尤其是当场景中出现遮挡关系时更加需要多个相机来准确地重建．

标记点的丢失

光学动捕的一个常见问题在于标记点在动作过程中很可能会丢失，比如当动捕演员趴在地上时，没有任何摄像机能够看到胸口的标记点；此外，在进行复杂动作的过程中可能会使标记点错位．这些问题对光学动捕是十分不利的，所以人们往往需要耗费大量精力为采集到的数据进行补点操作，补齐丢失的标记点或是纠正对应错误的标记点 (例如有些手上的标记点可能会被算法误认为是腿上的标记点)．这种后处理工作繁琐且昂贵，在业界这种工作往往是按照动捕序列的时长以秒为单位计费的．近期也有一些研究借助机器学习来简化这个补点的过程，感兴趣的读者可以深入调研．

无标记点的光学动捕

由于标记点占据一定的体积，动捕演员携带标记点做动作往往会受到影响，尤其是在采集动物的动作数据时．针对这个问题，目前最常见的解决方案是采用基于多视角相机的无标记点的动捕技术 (如图18.32)．其原理与普通的光学动捕技术类似，已知多个相机的内参

(a) 标记点

(b) 相机组

图 18.30: 光学动捕

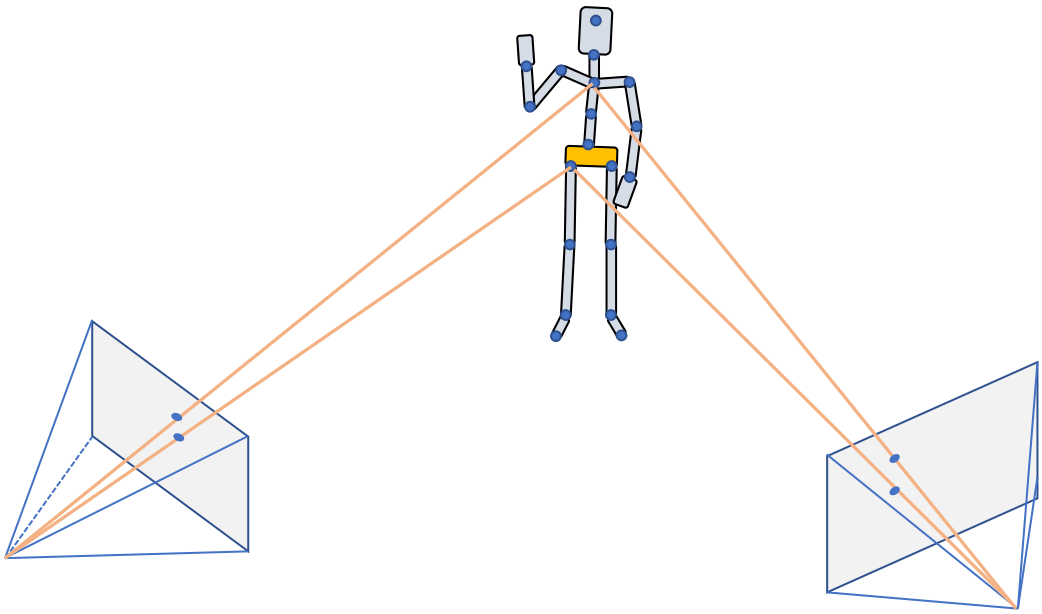


图 18.31: 双视角还原标记点位置

和位姿，通过识别每个相机拍摄出图像中的关节等特征来重建人体姿态．但这类方法的准确性会受光线等因素影响．

基于深度相机的无标记点光学动捕　深度摄像机拍摄出来的图片中每个像素点除了颜色信息外还会携带深度信息，通过额外的深度信息我们可以重建出人物的表面网格，进而重建出人物的姿态．图18.33a展示了一个深度相机，图18.33b和18.33c分别展示了两种使用深度相机重建出的人物动画．当然，在不同身体部位之间出现遮挡时，深度相机动捕就会遇到问题；另外，单视角深度相机也难以区分拍摄到的是身体的 (前、后、左、右) 哪一侧．

总体来说，无标记点的光学动捕精确度不如传统的光学动捕技术高，当然这也说明这方面技术还有很大的进步空间，它们也是近期研究的热点．

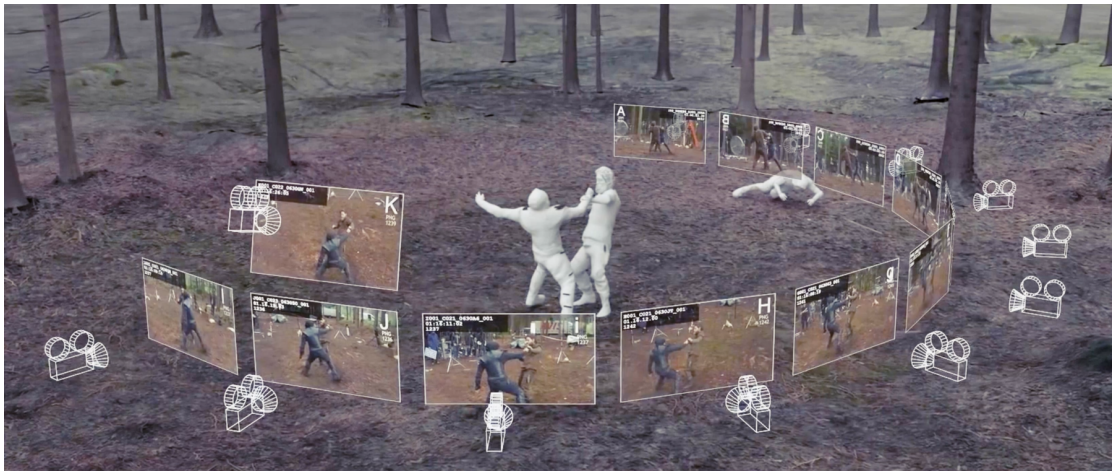
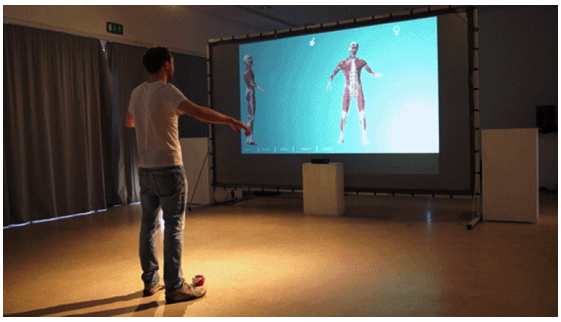


图 18.32: 无标记点的光学动捕

(a) 一个深度相机

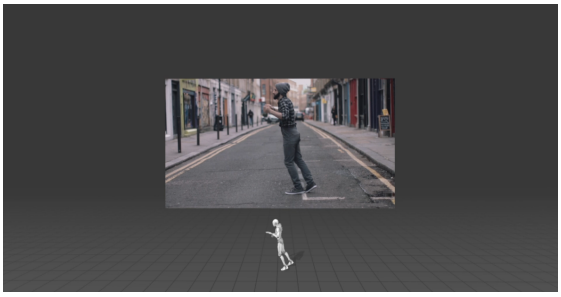
(b) 使用深度相机重建的骨架



(c) 使用深度相机重建的人物动作

图 18.33: 基于深度相机的动作捕捉

(a) [OpenPose, 2D Pose estimation]



(b) [3D Video-based Pose estimation, source: DeepMotion Inc.]

图 18.34: 基于单视角视频的运动估计

18.4.4　其他运动估计方法

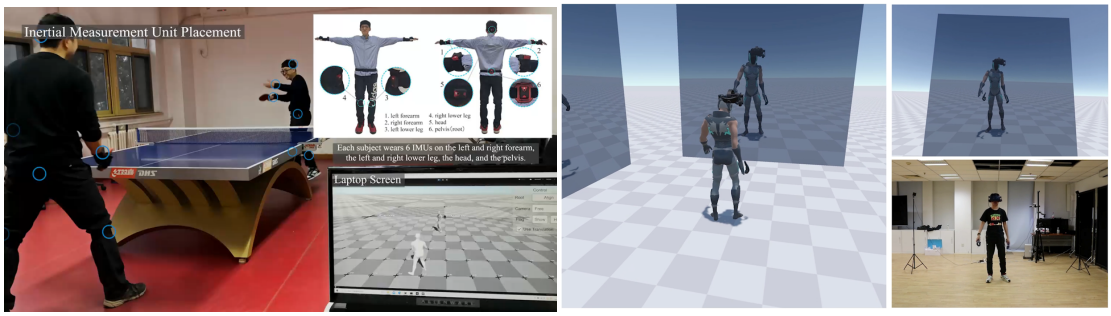
另外还有一些运动估计 (motion estimation) 的方法，它们可以借助低成本的设备还原出一个较为合理的动作，但由于这些方法的输入往往是不够的，不能准确地还原出人体动作，所以被称为运动估计而非动作捕捉.

基于单视角视频的运动估计

图18.34展示了两个基于单视角视频运动估计的研究工作. 由于单视角相机看到的是三维动作到二维的投影，输入具有一定的歧义，我们只能结合大量的训练数据来估计一个最大可能性的动作.

使用少量传感器进行运动估计

这类方法往往只需要借助很少量的 (5-6 个) 标记点或传感器进行运动估计 (如图18.35a)，这类方法可以借助逆向运动学技术还原出人物动作. 在虚拟现实场景中，我们甚至只有 3 个已知的标记点 (VR 头盔、两个手柄，如图18.35b，有时甚至只有头盔这一个标记点的信息). 对于这种信息极度缺失的病态问题，目前的技术能够达到的效果也很有限，但这同样



(a) 借助稀疏的惯性测量单元进行运动估计　　(b) 虚拟现实中的运动估计

图 18.35: 使用少量传感器进行的运动估计

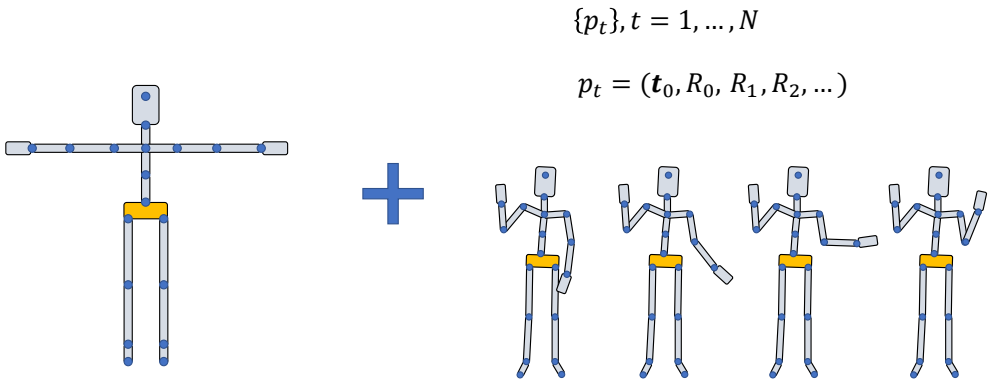


图 18.36: 动作数据

意味着我们还有很大的研究空间.

18.5　动作合成

利用上一节的动捕技术，我们可以得到一系列动作数据，接下来我们会介绍利用这些数据能够完成哪些任务.

18.5.1　动作数据

在开始之前，我们首先需要了解动作数据在计算机中是以什么样的格式存储的. 动捕数据一般以 BVH (Biovision Hierarchy) 文件的形式存储，这类文件一般包含两部分 (如图18.36)：

1. 动捕角色在 T-pose 下的大小和姿态.
2. 每一帧的角色姿态描述. 这部分信息被表示成一个集合 $\{p_t\}$，每一帧的信息 p_t 形如 $p_t = (\mathbf{t}_0, R_0, R_1, R_2, \cdots)$，其中 $\mathbf{t}_0$ 表示角色根节点 (还记得18.2.1节中我们将人体拓扑结构抽象成一棵树吗？) 的三维坐标，$R_0, R_1, R_2, \cdots$ 分别表示每个关节的旋转.

显然，我们可以根据文件中的数据还原出一个角色的动作序列.

利用这些数据通常分为几个环节 (如图18.37)，首先需要把采集到的数据重定向 (retar-

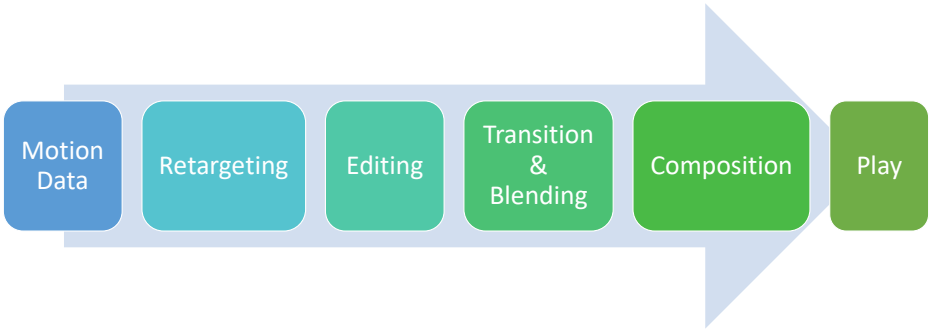


图 18.37: 动作数据的使用

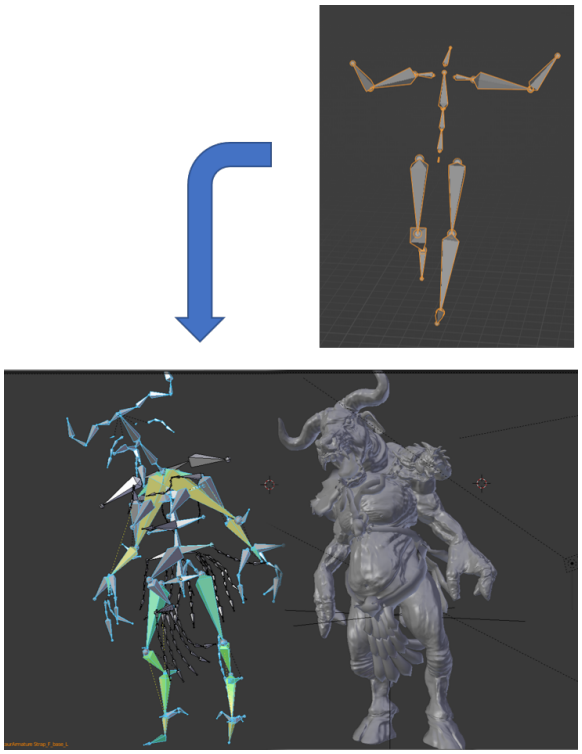


图 18.38: 动作重定向

geting) 到虚拟角色上，在此基础上我们需要对动作进行编辑 (editing)，然后进行动作的连接和混合 (transition & blending)，最后我们会把动作放到动作图 (motion graph) 中，实现可交互的动作组成 (composition) 和生成.

18.5.2　动作重定向

一般来讲，动作捕捉的采集对象是人，而虚拟角色往往多种多样，其身体结构可能也会比人复杂得多，这个时候我们就需要进行动作重定向 (如图18.38). 和人相比，虚拟角色可能具有不同数量的骨骼、不同的骨骼名称、不同的静止姿态、不同的骨骼比例、不同的骨架结构等等. 正是这种复杂性，使得重定向任务会遇到各种各样的问题，例如一些不够鲁棒的重定向会导致角色的脚底离开地面 (如图18.39a)，或者出现穿模现象 (如图18.39b).

动作重定向的流程可以分为以下几步：

1. **关节映射**. 在动捕数据中的关节和虚拟角色的关节之间建立一个映射.

(a) 角色脚底离开地面　(b) 穿模

图 18.39: 动作重定向时会遇到的问题

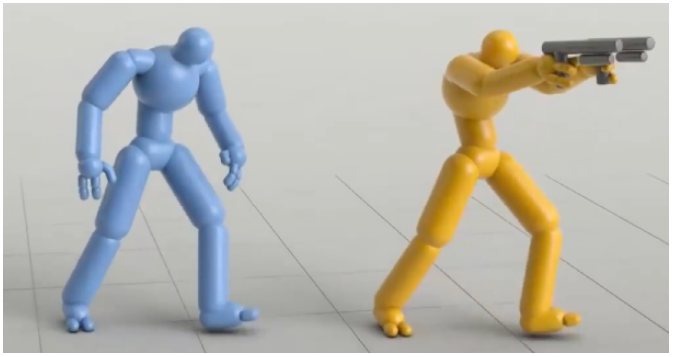


图 18.40: nucl.ai Conference: Ubisoft Toronto "IK Rig" Prototype[1]

2. **调整大小**. 有时动捕数据中的长度单位与虚拟角色的不一致，动捕数据中的单位大多为米或英尺，而虚拟角色用的则可能是任何单位. 这个步骤要处理的一个典型场景是走路的动作，对于这个场景我们通常会把根关节的位移缩放到高度与虚拟角色腿长相近，从而保证走路过程中脚不离地.
3. **复制或重定向关节旋转以修正初始姿态**. 为虚拟角色计算出每个关节的旋转以使它的初始姿态 (T-pose 或 A-pose) 能够匹配动捕数据的初始姿态. 这个步骤在虚拟角色和动捕数据中的身体长度不一致时会比较繁琐，此时我们可以借助一些启发性的规则来确定关节旋转，例如对于动捕数据中没有出现的关节我们可以将其旋转设为零或设置为相邻关节旋转的平均值.
4. **使用逆向运动学进行后处理，以解决诸如脚底打滑、穿模等问题**. 这些问题往往需要调整关节位置来解决，仅仅修改动捕数据中的关节旋转是不够的，所以逆向运动学就不可避免.

为了更方便地进行重定向，业界提出了一些新的动捕数据的表示方式 (其重定向效果如图18.40)，称为 IK Rig. 在这种表示方式中，我们不再记录每个关节的旋转，而是记录一些关键关节的位置. 于是我们只需要重定向相应关节的位置，再在虚拟角色上做逆向运动学，即可完成高质量的动作重定向. 目前很热门的虚幻引擎 Unreal 也支持 IK Rig 格式 (如图18.41).

[1] https://www.youtube.com/watch?v=V4TQSeUpH3Q

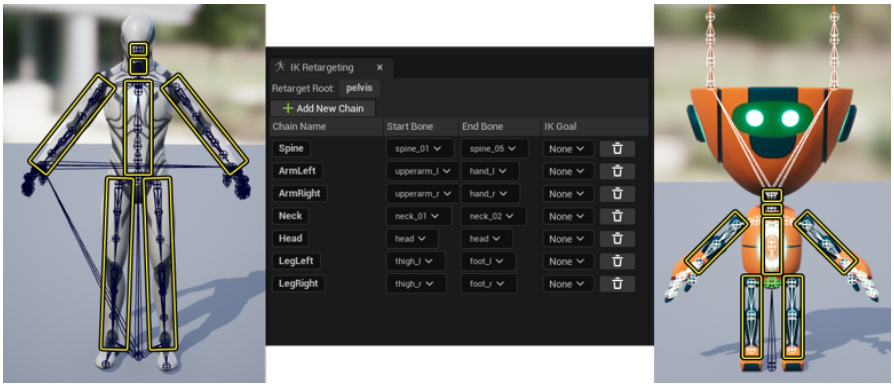
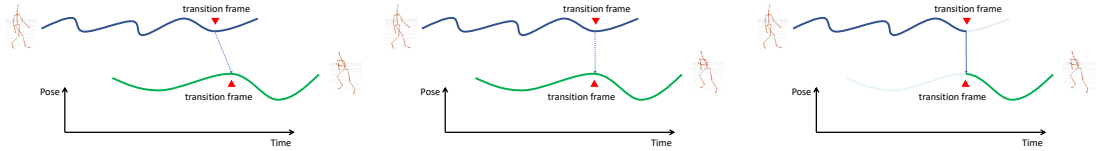


图 18.41: Unreal 中的 IK Rig



(a)　(b)　(c)

图 18.42: 动作连接的步骤

18.5.3　动作连接

在完成动作重定向之后，我们还需要进行动作连接，这是因为我们采集到的动捕数据往往只是一个动作片段，为生成完整、流畅的动作还需要将多段动捕数据中的动作串联起来. 例如我们现在已经重定向好了一段走路的动作和一段跑步的动作，我们想让虚拟角色先走路两秒钟，再跑步两秒钟，那么就需要借助动作连接技术从走路尽可能连贯地过渡到跑步.

基础的动作连接方法

我们假设需要连接的两个动作分别为动作 A 和动作 B，一个最简单的动作连接可以分为以下三步 (如图18.42)：

1. 在 A 的动作序列和 B 的动作序列中分别挑选一帧 (图18.42a中的 transition frame)，这两帧越相近越好.
2. 将 A 与 B 的动作序列进行时间对齐，使得挑选出的 transition frame 处于同一时刻.
3. 保留 A 序列 transition frame 前的部分和 B 序列 transition frame 后的部分，并直接拼接形成新的动作序列.

这个方法的缺点很明显：除非保证 A, B 动作在 transition frame 完全相同，拼接处会出现一个不连续的跳变. 但是这个方法在游戏中很常见，因为游戏中经常会有一些循环动作，角色在做完一个动作后总要恢复到动作开始前的姿态，比如我们经常看到一个角色做完“挥拳”或者“开枪”的动作之后总会恢复站立的姿态.

平滑的动作连接

在一般情况下，我们在动作过渡的时候需要做一个平滑操作. 这时我们不应再只考虑一帧的切换，而是要考虑两个动作在 transition frame 前后一段时间的序列. 如图18.43所示，

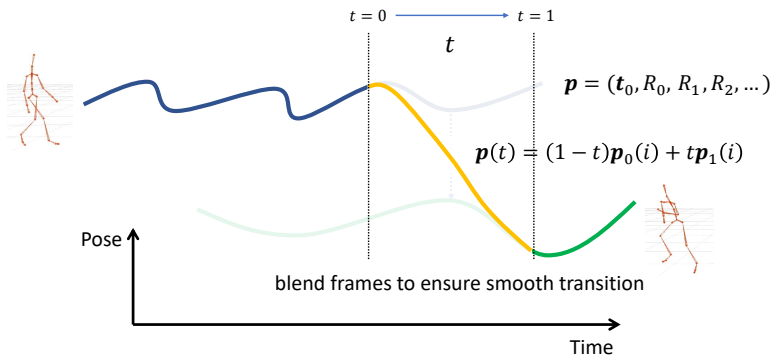


图 18.43: 平滑的动作连接

我们设这个过渡片段的开始时间为 0，结束时间为 1，再设动作序列 A 为 $\boldsymbol{p}_0$，B 为 $\boldsymbol{p}_1$，那么对于时刻 $t \in [0,1]$，我们的过渡动作可以取动作序列 A 和 B 的线性插值：

$$\boldsymbol{p}(t) = (1-t)\boldsymbol{p}_0(i) + t\boldsymbol{p}_1(i) \tag{18.24}$$

其中 i 表示时刻 t 对应的帧数，使用这样的插值我们能做到在时间从 0 到 1 推进的过程中，角色动作会从 A 逐渐过渡到 B. 为了达到一些特殊的平滑效果，我们也可以不使用线性插值，将式18.24中的 t 换成一个关于 t 的单调增函数 $\phi(t) \in [0,1]$，变成如下形式：

$$\boldsymbol{p}(t) = (1-\phi(t))\boldsymbol{p}_0(i) + \phi(t)\boldsymbol{p}_1(i) \tag{18.25}$$

其中的 $\phi(t)$ 可以取二次函数、指数函数等.

这里需要注意的一点是，在对 $\boldsymbol{p}_0$ 和 $\boldsymbol{p}_1$ 进行插值的时候，会涉及到对旋转的插值，此时不能像对位置插值一样简单的用线性组合计算插值后的结果，对旋转的插值读者可以回顾18.1.2节中介绍的方法.

动作对齐

至此我们已经能够较为平滑地拼接两段动作了，但是这样还会存在一个问题：例如动作 A 是一段向右的行走，动作 B 是一段向左的跑步，那么我们拼接之后会发现虚拟角色在从走切换到跑时突然调转了一个方向，并且在此过程中脚底还会出现明显的打滑现象. 我们希望避免这种 180 度大转弯的现象，所以需要将 A 和 B 两段动作进行对齐 (注意之前我们已经做过一次时间上的对齐了，现在我们要对齐的是 A 和 B 中人物的运动方向，如图18.44).

在进行运动方向对齐之前，我们首先需要定义角色的朝向坐标系 (facing frame)，它是一个固定在角色身上的局部坐标系，一个局部坐标系可以由两个参数 $(R, \mathbf{t})$ 来表示，R 表示局部坐标系在世界坐标系下的旋转 (即世界坐标系的 3 个坐标轴在旋转 R 变换下会变成局部坐标系的 3 个坐标轴在世界坐标系下的坐标)，$\mathbf{t}$ 表示局部坐标系的原点在世界坐标系下的位置. 朝向坐标系的 $\mathbf{t}$ 一般取根关节在世界坐标系下的坐标，R 一般有如下两种选取方式：

- R 是一个绕 y 轴的旋转 (参考18.1.3节中介绍的欧拉角)，满足朝向坐标系的 z 轴指向角色的面朝方向.
- R 是一个绕 y 轴的旋转，满足朝向坐标系的 x 轴是世界坐标系中肩膀方向与臀部方向的平均方向.

我们采取前一种选取方式，图18.45展示了某时刻角色的朝向坐标系，这里我们采用了 y 轴向上的习惯定义坐标轴 (图中没有显示与纸面垂直的 y 轴).

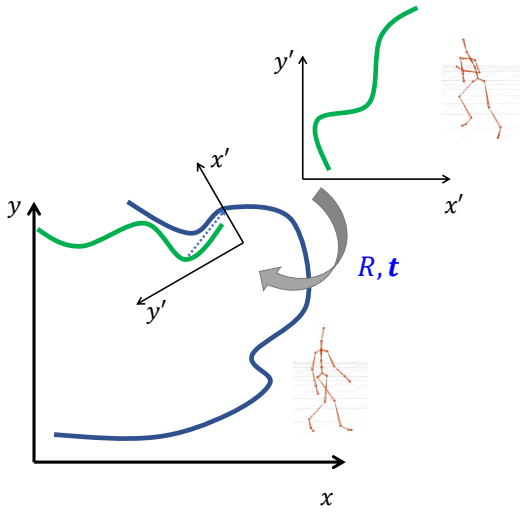


图 18.44: 运动方向对齐

$$R = \theta \boldsymbol{e}_y$$

$$\boldsymbol{t} = (t_x, 0, t_z)$$

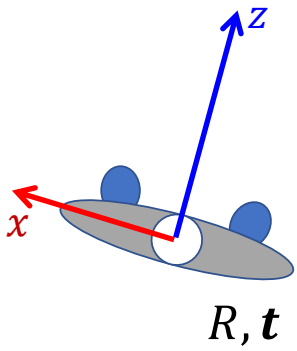


图 18.45: 朝向坐标系．图为俯视角色视角，R 采用轴角法表示，z 轴指向角色面朝方向，y 轴垂直纸面向外．

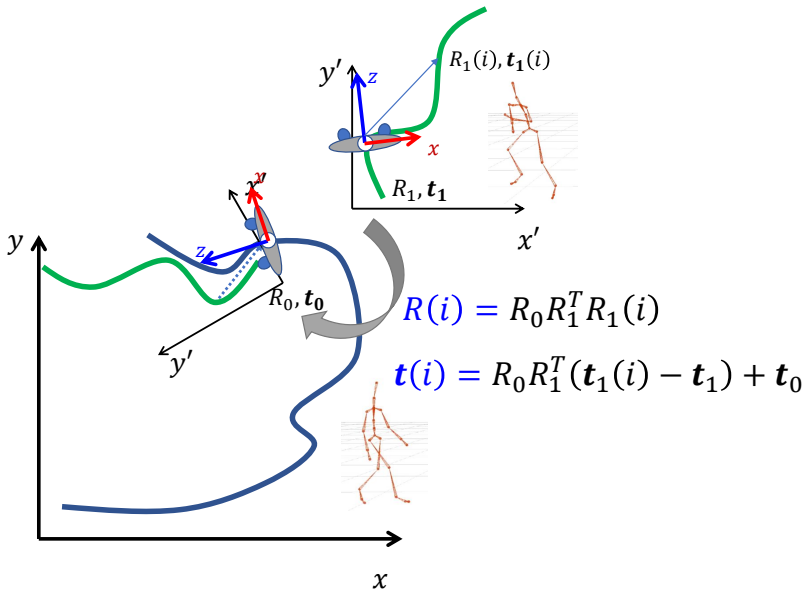


图 18.46: 将两个序列进行运动方向对齐

接下来我们假设每一帧中的坐标都是在该帧的朝向坐标系下定义的. 那么接下来我们可以将动作 B 在 transition frame 时刻的朝向坐标系进行旋转、平移，使其变为与动作 A 在 transition frame 时刻的朝向坐标系相同，并按照同样的旋转、平移去变换 B 序列中每一时刻的朝向坐标系 (如图18.46)，记变换后的序列为 B'，那么我们对 A 和 B' 进行平滑操作即可.

接下来的问题就在于如何计算这个“对齐变换”. 参考图18.46，我们考虑 transition frame 后的某一帧 i，在 B 序列中的朝向坐标系为 $R_1(i), \mathbf{t}_1(i)$，在 A 序列中的朝向坐标系为 $R(i), \mathbf{t}(i)$，另外设 transition frame 时刻 A, B 序列中的朝向坐标系分别为 $R_0, \mathbf{t}_0$ 和 $R_1, \mathbf{t}_1$，那么这其中只有 $R(i), \mathbf{t}(i)$ 使我们想求的变量，其余均为已知量. 考虑在帧 i 时刻任取一个 $R_1(i), \mathbf{t}_1(i)$ 坐标系下的三维坐标 $\mathbf{x}$，设其转换为 $R_1, \mathbf{t}_1$ 坐标系下的坐标为 $\mathbf{x}_1$；然后在 $R(i), \mathbf{t}(i)$ 坐标系下去同样的坐标 $\mathbf{x}$，设其转换为 $R_0, \mathbf{t}_0$ 坐标系下的坐标为 $\mathbf{x}_0$；那么我们应该有如下关系 (这也是“对齐”的含义)：

$$\mathbf{x}_0 = \mathbf{x}_1 \tag{18.26}$$

首先我们考虑如何将 $R_1(i), \mathbf{t}_1(i)$ 坐标系下的 $\mathbf{x}$ 转换为 $\mathbf{x}_1$. 这可以拆成两步来做：将 $\mathbf{x}$ 转换为世界坐标系下的坐标 $\mathbf{x}_B$，然后将 $\mathbf{x}_B$ 转换为 $\mathbf{x}_1$. 第一步转换只需要将局部坐标系的旋转和平移依次作用在 $\mathbf{x}$ 上：

$$\mathbf{x}_B = R_1(i)\mathbf{x} + \mathbf{t}_1(i) \tag{18.27}$$

第二步转换由于是世界坐标变为局部坐标，所以是一个逆变换，也就是先做平移的逆变换，再做旋转的逆变换：

$$\mathbf{x}_1 = R_1^\top(\mathbf{x}_B - \mathbf{t}_1) \tag{18.28}$$

由式18.27,18.28可得

$$\mathbf{x}_1 = R_1^\top[R_1(i)\mathbf{x} + \mathbf{t}_1(i) - \mathbf{t}_1] \tag{18.29}$$

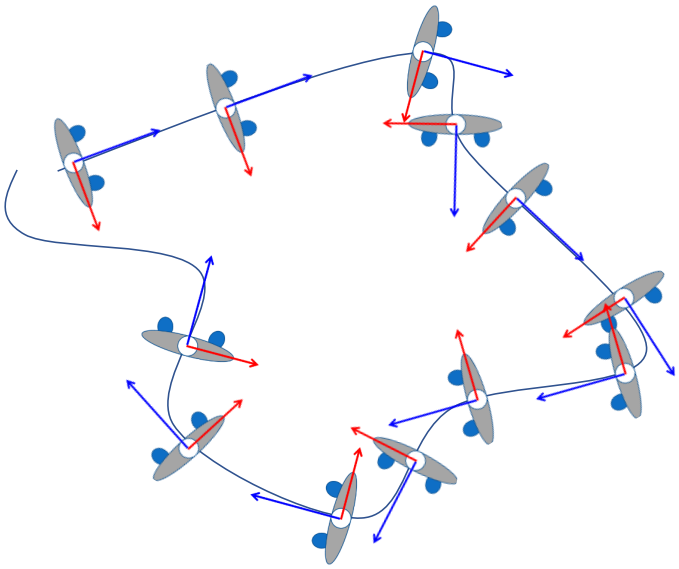


图 18.47: 曲线跟踪

同理我们也可以得到 $\mathbf{x}_0$ 与 $\mathbf{x}$ 的关系：

$$\mathbf{x}_0 = R_0^\top [R(i)\mathbf{x} + \mathbf{t}(i) - \mathbf{t}_0] \tag{18.30}$$

那么由式18.26,18.29,18.30可得

$$[R_1^\top R_1(i) - R_0^\top R(i)]\mathbf{x} = R_0^\top \mathbf{t}(i) - R_0^\top \mathbf{t}_0 - R_1^\top \mathbf{t}_1(i) + R_1^\top \mathbf{t}_1 \tag{18.31}$$

由于 $\mathbf{x}$ 的任意性，有

$$\begin{aligned} R_1^\top R_1(i) - R_0^\top R(i) &= 0 \\ R_0^\top \mathbf{t}(i) - R_0^\top \mathbf{t}_0 - R_1^\top \mathbf{t}_1(i) + R_1^\top \mathbf{t}_1 &= 0 \end{aligned} \tag{18.32}$$

解得

$$\begin{aligned} R(i) &= R_0 R_1^\top R_1(i) \\ \mathbf{t}(i) &= R_0 R_1^\top [\mathbf{t}_1(i) - \mathbf{t}_1] + \mathbf{t}_0 \end{aligned} \tag{18.33}$$

曲线跟踪

当然我们有时候并不需要进行对齐，比如你就想要实现一个走路过程中突然掉头跑步的效果，直接做插值就可以了．另外，很多时候我们拿到的动作数据已经去掉了全局的位置信息，即根关节的位置 (有时也有朝向) 不随时间变化，此时就不涉及到两段动作序列之间对齐的问题了，我们需要自己逐帧确定根关节的位置和朝向．这种去掉全局位置信息的动作数据在游戏中很常见，根关节的位置是游戏程序实时确定的，我们在网络卡顿的时候经常看到一个人在原地跑步就是由于在播放动作数据，但游戏程序的位置更新没有跟上．

使用没有全局位置信息的动作数据需要进行曲线跟踪 (path fitting)，我们需要手动指定角色根关节在一段时间内移动的路径 (一般假定这个路径是在一个平面内)，那么对于路径上任意一点，其切向即为角色的朝向，位置即为角色根关节的位置，由此可以确定出相应时刻下角色的朝向坐标系 (如图18.47)．

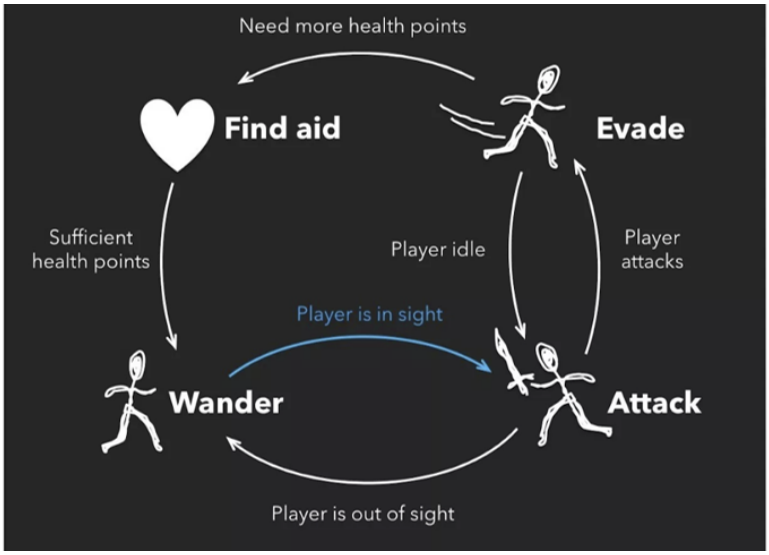


图 18.48: 一个动作图

18.5.4　动作图

至此位置，我们掌握的技术仅仅能够将两个动作平滑地链接在一起．但这还不够，我们还希望实现更加丰富的动作合成 (motion composition)，即希望让计算机能够自动化地计算出角色动画，使其能够支持：

- 实时的用户控制．
- 角色自动避开场景里的障碍物．
- 与场景中其他角色进行交互．
- 其他需要的功能或特性．

动作图 (motion graph) 就是用来描述动作合成规则的数据结构．

动作图的本质是一个有限状态自动机 (finite state automata)，每个状态表示一个动作，状态之间如果满足一定条件就会发生转移，在动作图中．用一个节点表示一个状态，用一条有向边表示一个可能的转移，每一条边会对应一个转移条件，图18.48展示了一个简单的动作图．在生成一段动作时，我们会根据动作图，从某一个状态开始，当遇到用户输入打断或者当前动作播放完毕时，选择一个符合条件的转移并执行．在从一个动作状态转移到另一个动作状态时，我们可以使用前面介绍的插值方法做一个平滑的过渡．而对于如何选择下一个动作，取决于我们任务的复杂性，有时我们可以直接挑选一个合适的动作进行转移，有时则需要向后多考虑几步转移以进行更长远的规划 (例如当我们让虚拟角色走上一个台阶时，需要指定每跨出一步后脚的位置，就可以借助深度优先搜索以做出后几步的规划)．

那么给定一段动捕数据，如何生成一个动作图呢？假设动捕数据包含 n 帧，我们可以构造一个 $n \times n$ 的矩阵 (如图18.49)，第 i 行第 j 列元素表示第 i 帧动作与第 j 帧动作的距离．这里我们可以根据需要去定义一个距离，例如采用每个关节的旋转的距离，或是利用前向运动学计算每个关节的位置差，还可以考虑速度等信息．接下来我们找出矩阵元素中所有的极小值点，假设第 a 行第 b 列是一个极小值点，那么说明我们可以在动作的第 a 帧和第 b 帧进行一个分割，并且这两帧可以互相转移．这样根据所有的极小值点，我们可以将动捕数据分割成多个小段，每个小段可以作为动作图的一个节点，再通过潜在的转移关系在这些节点之间连边，从而构造出动作图．当然，直接这样构造出的动作图可能质量很差 (比如它可能包含过多的状态，每个状态的动作太短)，实践中还需要用户手动筛选一些极小值点，

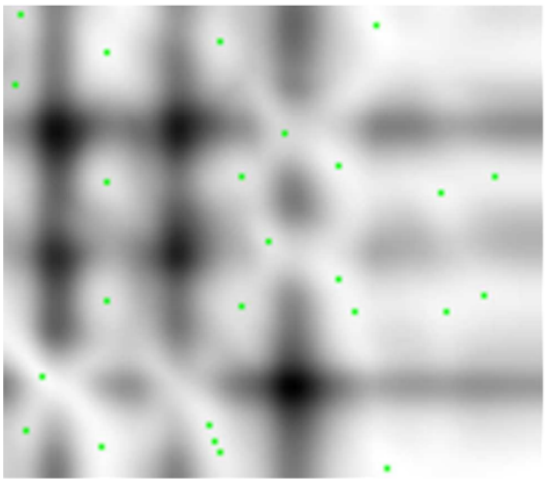


图 18.49: 由一段动捕数据构建出的一个矩阵．矩阵的元素值越大颜色越深，绿色的点为极小值点．

并做一些其他的处理．

总体而言，在动作图的基础上，我们实现动作合成是在以下步骤中不断循环的过程 (每一帧执行一次这样的循环)：

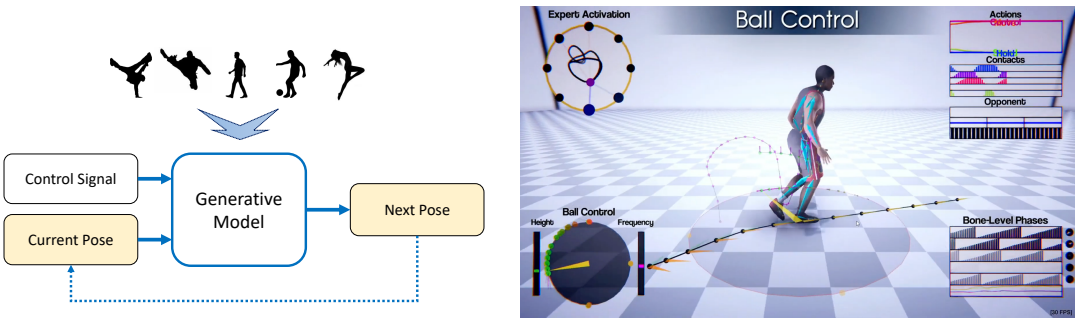
1. 检查用户输入．
2. 检查当前场景的环境信息．
3. 检查当前场景中是否存在需要互动的其他角色．
4. 根据所有信息决定是否要进行状态转移．
5. 在状态转移之后，要去获取下一个动作的姿态．
6. 进行后处理．
7. 更新角色姿态．
8. 更新环境信息．

18.5.5　其他动作合成方法

动作图的应用虽然广泛，但是在实际应用的时候还需要很多的技巧和方法才能够生成高质量的动作合成．因此，人们提出了一些新的方法用于动作合成，包括动作匹配 (motion matching) 以及一些基于学习的方法．

动作匹配比动作图实现了更细粒度的动作控制，它会在每一帧都去判断下一帧要切换到哪个动作，切换的粒度不再是动作片段，而是姿态．这种细粒度的切换能够带来更快的用户操作响应，并且由于每一帧都会自动在动捕数据中搜索一帧最合适的姿态作为下一帧，它不再需要构建动作图时对数据的切割、建立转移等复杂的操作了．

随着机器学习的兴起以及生成模型的流行，这些方法也被引入到动作生成的任务当中来．相比传统的基于数据的方法，基于学习的方法不再是对数据的简单重组，而是会从数据中提取一些模型并用于生成．图18.50展示了两种基于学习的动作合成工作．



(a)　(b)

图 18.50: 基于学习的方法