CMSC 425: Lecture 9
Basics of Skeletal Animation and Kinematics

Reading: Chapt 11 of Gregory, *Game Engine Architecture.* The material on kinematics is a simplification of similar concepts developed in the field of robotics, known as the Denavit-Hartenberg parameters.

Game Animation: Most computer games involve *characters* that move around in a fluid and continuous manner. Unlike objects that move according to the basic laws of physics (e.g., balls, projectiles, vehicles, water, smoke), the animation of skeletal structures may be subject to many complex issues, such as physiological constraints (e.g., how to jump onto a platform), athletic technique (e.g., how a football quarterback throws a pass), stylistic choices (e.g., how a dancer moves), or simply the arbitrary conventions of everyday life (e.g., how to hold chopsticks). Producing natural looking animation is the topic of our next few lectures.

Skeletal Model and Tree Structure: The most common form of character animation used in high-end 3-dimensional games is through the use of skeletal animation. A character is modeled as *skin* surface stretched over a *skeletal framework* which consists of moving *joints* connected by segments called *bones* (see Fig. 1(a) and (b)). Note that "skin" refers to the model's surface, which typically includes not only skin but the clothing the model is wearing (see Fig. 1(c)). Animation is performed by modifying the relationships between pairs of adjacent joints, for example, by altering joint angles and deforming the skin accordingly. We will discuss this process extensively later.

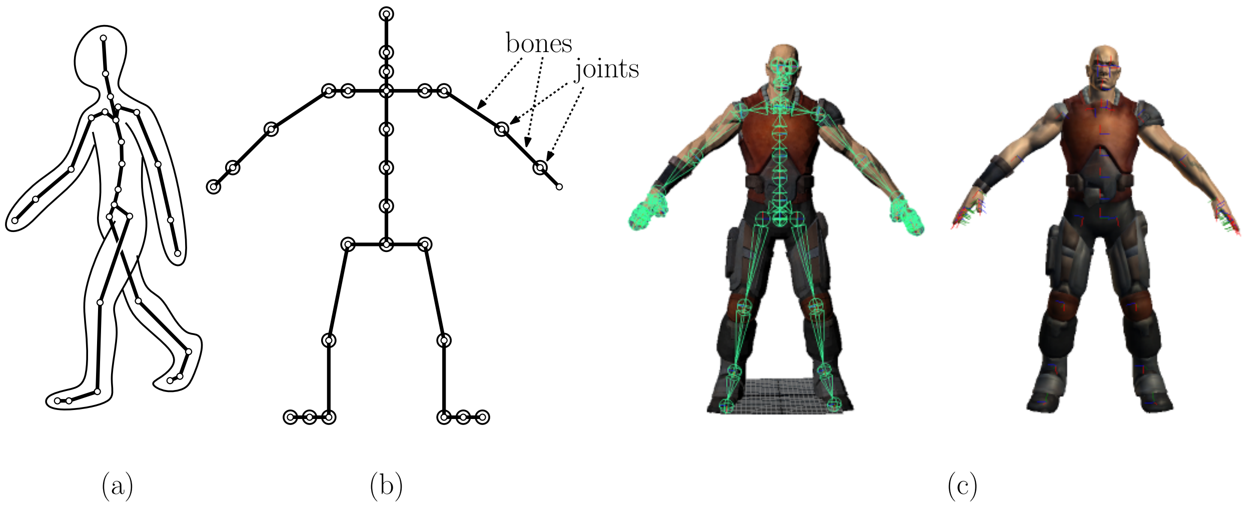


Fig. 1: (a) and (b) skeletal model and (c) the bind (or reference) pose.

A skeletal model is based on a hierarchical representation where peripheral elements (e.g., hands and feet) are linked as children of more central elements (e.g., legs, arms, torso, etc.). Clearly, a skeletal model can be represented internally as a multi-way rooted tree, in which each node represents a single joint. The *bones* of the tree are not explicitly represented, since (as we shall see) they do not play a significant role in the animation or rendering process.

We will discuss later how the skin is represented. For now, let us consider how the joints are represented.

We assume that the tree is represented as any standard (rooted, unordered) multi-way tree. For example, for any node it should be possible to enumerate all the children of this node, to test whether the node is the root, and if it is not the root to determine its parent node. In this lecture, we will denote each joint by an integer index, say j, and we will let $p(j)$ denote j's parent. If we consider just the parent links, the result is an *inverted tree structure*, where all paths lead to the root (see Fig. 2(b)). We can make the assumption that the root is identified by a special index, say $j = 0$. In addition, each node will store some *internal information*, as will be discussed below.

Bind Pose: Before discussing animation on skeletal structures, it is useful to first say a bit about the notion of a *pose*. In humanoid and animal skeletons, joints move by means of rotations[1] (as opposed say to translation, which arises with some robots). Assigning angles to the various joints of a skeleton uniquely specifies the skeleton's exact geometric structure, called its *pose*.

When a designer defines the initial layout of the model's skin, the designer does so relative to a default pose, which is called the *reference pose* or the *bind pose*.[2] For human skeletons, the bind pose is typically one where the character is standing upright with arms extended straight out to the left and right (similar to Fig. 1(b) above).

Joint Internal Information: Each joint can be thought of as defining its own *joint coordinate frame* (see Fig. 2(a)). Recall that in affine geometry, a coordinate frame consists of a point (the origin of the frame) and three mutually orthogonal unit vectors (the x, y, and z axes of the frame). Given the skeleton's inverted tree structure (see Fig. 2(b)), rotating a joint can be achieved by applying a suitable rotation transformation to its associated coordinate frame. Each frame of the hierarchy is understood to be positioned *relative* to its parent's frame. In this way, when the shoulder joint is rotated, the descendants' joints (elbow, hand, fingers, etc.) also move as a result (see Fig. 2(c)).

Change-of-Coordinates Transformation: In order to determine the motion of the various bones that result from some joint rotation, we need to know the relationships between the various joints of the skeleton. There is a very general and elegant way of doing this through the application of affine geometry. Given any two coordinate frames in d-dimensional space, it is possible to convert a point (or free vector) represented in one coordinate frame to its representation in the other frame by multiplying the point (given as a $(d + 1)$-dimensional vector in homogeneous coordinates) times an suitable $(d+1) \times (d+1)$ matrix. The resulting affine transformation is called a *change-of-coordinates transformation*.

Constructing such transformations is an exercise in linear algebra. For the sake of completeness, let us consider the process in a simple 2-dimensional example. Suppose we have two

[1]It is rather interesting to think about how this happens for your own joints. For example, your shoulder joint has two degrees of freedom, since it can point your upper arm in any direction it likes. Your elbow also has two degrees of freedom. One degree comes by flexing and extending your forearm. The other can be seen when you turn your wrist, as in turning a door knob. Your neck has (at least) three degrees of freedom, since, like your shoulder, you can point the top of your head in any direction, and, like your elbow, you can also turn it clockwise and counterclockwise.

[2]I suspect that the name "bind pose" arises because designers attach or "bind" the skin to the model relative to this initial pose.

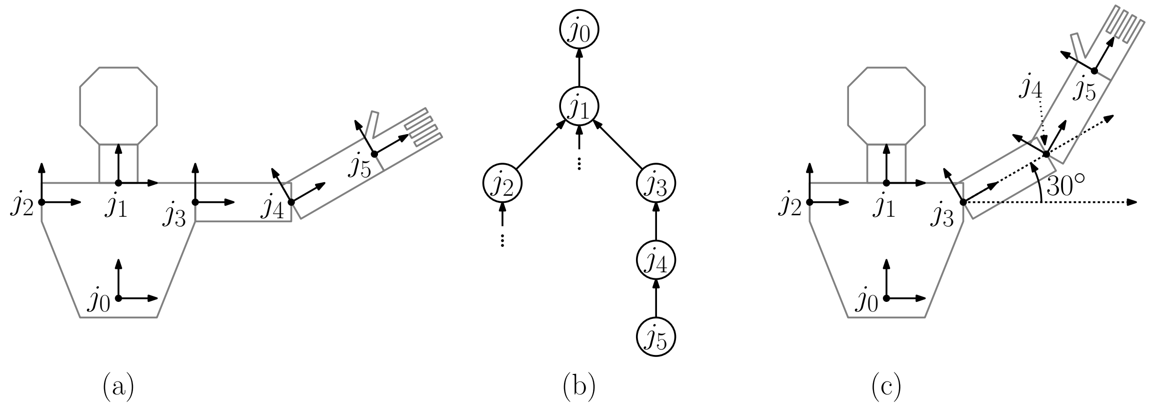


Fig. 2: (a) Skeletal model, (b) inverted tree structure, and (c) rotating a frame propagates to the descendants.

coordinate frames, F and G (see Fig. 3). Let $F.o$, $F.x$, and $F.y$ denote F's origin point, and its two basis vectors. Define $G.o$, $G.x$ and $G.y$ similarly.

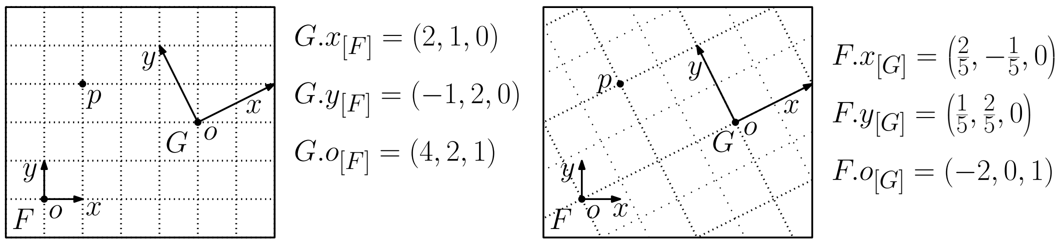

p
G
o
x
y
F
$G.x_{[F]} = (2,1,0)$
$G.y_{[F]} = (-1,2,0)$
$G.o_{[F]} = (4,2,1)$
$F.x_{[G]} = \left(\frac{2}{5}, -\frac{1}{5}, 0\right)$
$F.y_{[G]} = \left(\frac{1}{5}, \frac{2}{5}, 0\right)$
$F.o_{[G]} = (-2,0,1)$

Fig. 3: Change-of-coordinates transformation.

Given any point in space, it can be represented either with respect to F's coordinate system or G's. For any point p, define $p_{[F]}$ to be p's homogeneous coordinates relative to frame F, and define $p_{[G]}$ similarly for frame G. We can do the same for any vector $\vec{v}$.

In order to define the change-of-coordinates transformation, we need to know first what G's basis elements are relative to F. In the above example, it is easy to verify that

$$G.x_{[F]} = (2,1,0), \quad G.y_{[F]} = (-1,2,0), \quad \text{and} \quad G.o_{[F]} = (4,2,1).$$

(Recall that we are using affine homogeneous coordinates, where the last component is 0 to denote a vector or 1 to denote a point.) Also, it is easy to verify that

$$F.x_{[G]} = \left(\frac{2}{5}, -\frac{1}{5}, 0\right), \quad F.y_{[G]} = \left(\frac{1}{5}, \frac{2}{5}, 0\right), \quad \text{and} \quad F.o_{[G]} = (-2,0,1).$$

To obtain the change-of-coordinates transformations from G to F, define $T_{F \leftarrow G}$ to be the transformation that maps a point given in G's coordinate system to its representation in F's coordinate system. This transformation can be represented as a matrix whose columns are

$G.x_{[F]}$, $G.y_{[F]}$, and $G.o_{[F]}$:

$$T_{[F\leftarrow G]} = \begin{pmatrix} 2 & -1 & 4 \\ 1 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}.$$

Conversely, to convert the other direction, define $T_{G\leftarrow F}$ to be the transformation that maps a point given in F's coordinate system to its representation in G's coordinate system. This transformation can be represented as a matrix whose columns are $F.x_{[G]}$, $F.y_{[G]}$, and $F.o_{[G]}$:

$$T_{[G\leftarrow F]} = \begin{pmatrix} 2/5 & 1/5 & -2 \\ -1/5 & 2/5 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

(See any standard reference on linear algebra for a proof.)

Change-of-Coordinates Example: To test this, let's consider the point p and vector $\vec{v}$ in Fig. 4.

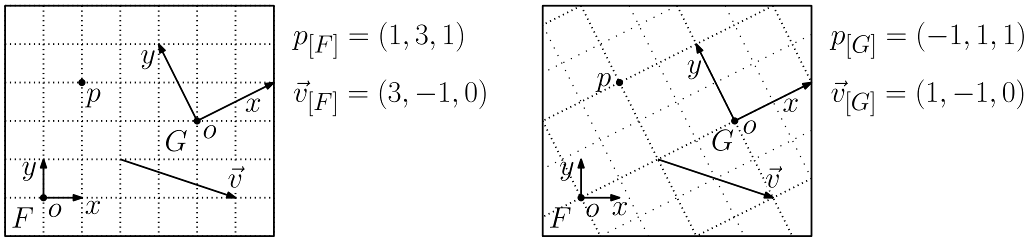


Fig. 4: Example of applying the change-of-coordinates transformation.

Clearly, $p_{[F]} = (1,3,1)$ and $p_{[G]} = (-1,1,1)$. Applying the above transformations, we obtain the expected results

$$T_{[F\leftarrow G]} \cdot p_{[G]} = \begin{pmatrix} 2 & -1 & 4 \\ 1 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} = p_{[F]},$$

and

$$T_{[G\leftarrow F]} \cdot p_{[F]} = \begin{pmatrix} 2/5 & 1/5 & -2 \\ -1/5 & 2/5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} = p_{[G]},$$

Next, consider $\vec{v}$. We have $\vec{v}_{[F]} = (3,-1,0)$ and $\vec{v}_{[G]} = (1,-1,0)$. Again, applying the above transformations, we have

$$T_{[F\leftarrow G]} \cdot \vec{v}_{[G]} = \begin{pmatrix} 2 & -1 & 4 \\ 1 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 0 \end{pmatrix} = \vec{v}_{[F]},$$

and

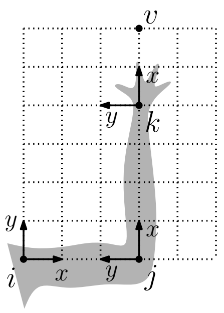
$$T_{[G\leftarrow F]} \cdot \vec{v}_{[F]} = \begin{pmatrix} 2/5 & 1/5 & -2 \\ -1/5 & 2/5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} = \vec{v}_{[G]}.$$

Abstraction Revisited: We have seen the use of homogeneous matrices before for the purpose of performing affine transformations (that is, for moving objects around in space). We are using the same mechanism here, but the meaning is quite different. Here the objects are not being moved, rather we are simply *translating* the names of points and vectors from one coordinate system to another. The geometric objects are themselves not moving.

You might wonder, "What's the difference in how you look at it?" Recall that in affine geometry we defined points and (free) vectors as different abstract objects, that employ the same representation (homogeneous vectors). Here, we are distinguishing two different types of operations (affine transformations versus change-of-coordinates transformations), but using the same representation (homogeneous matrices) for both. Even though the same representation is being used, these should be conceptualized as two very different operations.

Joint Transformations: Returning to the problem of skeletal systems, let us assume that we are working in 3-dimensional space, and consider the skeleton in its bind pose. For any two joints j and k, define $T_{[k \leftarrow j]}$ to be the change-of-coordinates transformation that maps a point in joint j's coordinate system to its representation in k's coordinate system. (At this point we are not considering joint rotations.) That is, if v is a column vector in homogeneous coordinates representing of a point relative to j's coordinate system, then $v' = T_{[k \leftarrow j]} \cdot v$ is exactly the same point in space, but it is expressed in coordinates relative to k's coordinate frame. (In previous lectures I have used p for points and v for free vectors. Since we are using p for "parent", I'll refer to points by the letter v, but don't be confused.)

Given any non-root joint j, define the *local-pose transformation*, denoted $T_{[p(j) \leftarrow j]}$, to be an affine transformation that converts a point in j's coordinate frame to its representation in its parent's $(p(j))$ coordinate frame. Define the *inverse local-pose transformation*, denoted $T_{[j \leftarrow p(j)]}$, to be the inverse of this transformation. That is, it converts a point expressed relative to j's parent's frame back to j's frame. (Recall that these transformations do not change the position of a point. They simply translate the same point from its representation in one frame to another.)



$$v_{[k]} = (2, 0, 1)$$

$$v_{[j]} = (6, 0, 1)$$

$$v_{[i]} = (3, 6, 1)$$

(a) (b)

Fig. 5: Three joints i, j, and k in a (rather nonstandard) bind pose. The point v is represented in homogeneous coordinates relative each frame.

Consider three joints i, j, and k, where $i = p(j)$ and $j = p(k)$ (see Fig. 5(a)). The local-pose transformation for k, $T_{[p(k) \leftarrow k]}$, can be expressed more succinctly as $T_{[j \leftarrow k]}$. Given a point $v_{[k]}$

expressed relative to k's frame, we can express it relative to j's frame as

$$v_{[j]} = T_{[j \leftarrow k]} \cdot v_{[k]}.$$

Similarly, a point $v_{[j]}$ expressed relative to j's frame can be expressed relative to i's frame as

$$v_{[i]} = T_{[i \leftarrow j]} \cdot v_{[j]}.$$

Combining these, we can express a point in k's frame relative to i's frame by taking the product of these two matrices

$$v_{[i]} = T_{[i \leftarrow j]} \cdot T_{[j \leftarrow k]} \cdot v_{[k]} = T_{[i \leftarrow k]} \cdot v_{[k]},$$

where $T_{[i \leftarrow k]} = T_{[i \leftarrow j]} \cdot T_{[j \leftarrow k]}$. Clearly, by multiplying appropriate chains of the local-pose transformations and their inverses, we can walk up and down the paths of the tree allowing us to convert a point relative to any one joint into its representation relative to any other joint.

An Example: To make this a bit more concrete, let us consider an example in 2-dimensional space. Consider the pose shown in Fig. 5(a). (This is not a normal bind pose, since the elbow should not be bent, but it makes for a more interesting case.) Let i denote the shoulder joint, j the elbow joint, and k the hand joint. Consider a point v that lies two units beyond the model's index finger. Its homogeneous coordinates relative to the hand frame are $(2, 0, 1)$. (Since the x-axis points up.) Its coordinates relative to the elbow frame are $(6, 0, 1)$, and its coordinates relative to the shoulder frame are $(3, 6, 1)$ (see Fig. 5(b)). That is,

$$v_{[k]} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} \quad v_{[j]} = \begin{pmatrix} 6 \\ 0 \\ 1 \end{pmatrix} \quad v_{[i]} = \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix}.$$

Because k's coordinate frame lies 4 units along the x-axis relative to j's coordinate frame, the local pose transformation $T_{[j \leftarrow k]}$ (which maps a point in k's coordinate frame to j's coordinate frame) clearly increases the x-coordinate by 4 units. Thus:

$$T_{[j \leftarrow k]} = \begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We can easily check this, since

$$T_{[j \leftarrow k]} \cdot v_{[k]} = \begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ 0 \\ 1 \end{pmatrix} = v_{[j]},$$

just as we expected.

Next, consider how to map a point in j's coordinate frame to its parent's frame i. Observe that the y-coordinate of the transformed point (its vertical distance) is the x-coordinate of the original point. Thus, the middle row of the matrix is $(1, 0, 0)$. The x-coordinate of the new point (its distance to the right of the shoulder) is 3 minus the old y-coordinate. Thus, the first row of the matrix is $(0, -1, 3)$. Therefore, the transformation that achieves this change of coordinates is

$$T_{[i\leftarrow j]} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Again, we can check this, since

$$T_{[i\leftarrow j]} \cdot v_{[j]} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 6 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix} = v_{[i]},$$

as we expected.

Now, to obtain the transformation $T_{[k\leftarrow i]}$, we multiply these two matrices (translating from k to j, then j to i)

$$T_{[i\leftarrow k]} = T_{[i\leftarrow j]} \cdot T_{[j\leftarrow k]} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$

Finally, to check this we have

$$T_{[i\leftarrow k]} \cdot v_{[k]} = \begin{pmatrix} 0 & -1 & 3 \\ 1 & 0 & 4 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix} = v_{[i]}.$$

Again, this is just what we expect to happen.

Of course, applying this in 3-dimensional space will involve handling 4×4 matrices and the associated rotation and translation matrices. While this would be much harder to do by hand, it can be done in by a similar process that is purely mechanical (and hence, easy to program).

Forward Kinematics: Next, suppose that in addition to knowing the local-pose transformations and their inverses, we also know the rotation transformations associated with the individual joints of the system. *Kinematics* (also called *forward kinematics*) is the problem of determining where a point is transformed as a result of these rotations.

We can apply our knowledge of rotation transformations and the local-pose transformations and their inverses to solve this problem. For example, recall the point v in our earlier example (see Fig. 6(a)) Suppose that the elbow is rotated by counterclockwise by 30° (see Fig. 6(b)), and then the shoulder is rotated clockwise by 45°, that is, counterclockwise by −45° (see Fig. 6(c)). The question that we want to consider, where is the point v mapped to as a result of these two rotations? Let v' be its position after the elbow rotation and let v'' be its position after both rotations.

Before getting to the answer, recall from our earlier lecture on affine geometry the rotation transformations in homogeneous coordinates:

$$\mathrm{Rot}(30^\circ) = \begin{pmatrix} \cos 30^\circ & -\sin 30^\circ & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

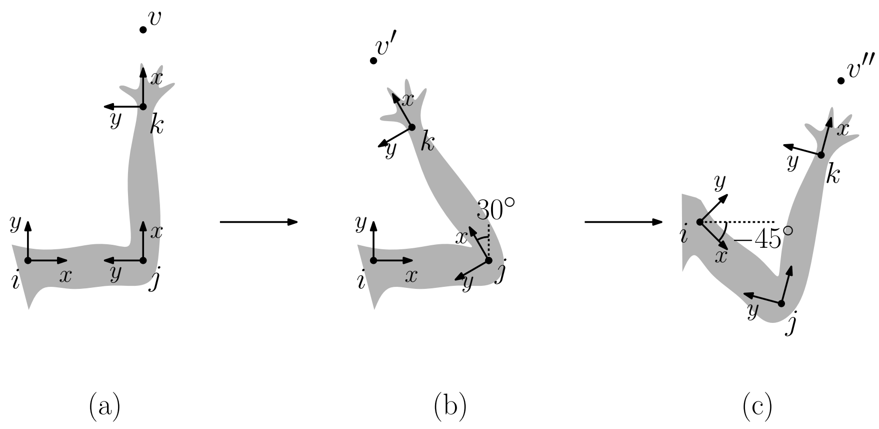


Fig. 6: Forward kinematics example.

and

$$\mathrm{Rot}(-45^\circ) = \begin{pmatrix} \cos 45^\circ & -\sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We need to decide which frame to use as our reference frame. Let's use the shoulder joint i, since it is the most "global". (Generally, we would select the root of our skeleton tree.) We saw already how to compute $v_{[i]}$. Using this as a starting point, let's first consider the effect of the elbow rotation. Because the elbow rotation occurs about the elbow's coordinate frame, we first need to translate v into its representation with respect to j's frame (by multiplying by the inverse local-pose transformation $T_{[j\leftarrow i]}$). We then apply the 30° rotation about the elbow joint. Finally, we convert this representation back to the shoulder frame (by applying the local-pose transformation $T_{[i\leftarrow j]}$). Thus, we have

$$v'_{[i]} = T_{[i\leftarrow j]} \cdot \mathrm{Rot}(30^\circ) \cdot T_{[j\leftarrow i]} \cdot v_{[i]}.$$

This yields a representation of v' relative to the shoulder frame. Since the second rotation is performed about the shoulder frame, we do not need to perform any change-of-coordinate transformation. We can just apply the rotation transformation directly. This yields:

$$v''_{[i]} = \mathrm{Rot}(-45^\circ) \cdot v'_{[i]}.$$

Putting both steps together, we have

$$v''_{[i]} = \mathrm{Rot}(-45^\circ) \cdot T_{[i\leftarrow j]} \cdot \mathrm{Rot}(30^\circ) \cdot T_{[j\leftarrow i]} \cdot v_{[i]}.$$

Top-down or Bottom-up? You might wonder why we did the elbow rotation first followed by the shoulder transformation. Does the order really matter? The issue is that our local-pose transformations have been built under the assumption that the model is in the bind pose, that is, none of the joints are rotated. If we were to have performed the shoulder rotation first, and then attempted to apply the inverse local-pose transformation $T_{[j\leftarrow i]}$ to convert the result from the shoulder's frame to the elbow frame, we would discover that this transformation is no longer correct. The reason is that the entire arm (and the elbow joint in particular) has moved into a new position, but $T_{[j \leftarrow i]}$ was defined based on its original position. To avoid this problem, the transformations should be applied in a *bottom-up manner*, first rotating the descendant nodes (e.g., wrist) and then working up to their ancestors (e.g., elbow and then shoulder).

Take-Away Lesson: I must acknowledge that implementing this by by hand would be a mess (especially in 3-space), but hopefully you get the idea. By using our local-pose transformations (and possibly their inverses), we can change to the coordinate frame where the rotation takes place, then apply the rotation, then translate back. While it would be messy to write down all the transformations, if we have precomputed the local pose transformations and their inverses, this can all be programmed in a straightforward manner by traversing the tree (in postorder) and performing simple matrix multiplications.