

Chapter 13

Common Issues and Techniques — DRAFT

Excerpted from Motion Capture and Motion Editing: From Observation to Animation, by Michael Gleicher. This book will be published sometime after its finished by AK Peters. I have no idea when I will have it finished.

This draft is current as of April 28, 2001.

13.5 A Simple Motion Processor

In this section, we consider a simple method for converting marker information into skeletal information. While this method may no longer be state of the art, it is still worth discussing for a number of reasons:

1. it does provide a workable scheme that is considerably easier to implement than succeeding technologies;
2. its simplicity allows for many of the issues in processing to be made clear;
3. the shortcomings of the approach provide motivation for more sophisticated approaches. Some may be created as extensions to this approach, while others will require a completely new approach;
4. while the methods here were (are?) standard practice, they are not widely discussed by the research community, and therefore, have not been written about.

While this section is almost completely devoid of references, I do not want to give the impression that I invented the methods. Taylor Wilson (presently CTO of House of Moves) introduced these methods to me in late 1996 when I was a client of the studio he was working for at the time. Taylor helped me understand what their pipeline and procedures were in order to explain the difficulties they had in our job, and to enable me to recreate my own version of a production pipeline. I believe that these methods were the state of the art of the time, although I did not do extensive research into other available options.

The methods described make some simplifications over the methods used in production practice for pedagogical simplicity. The building blocks are effectively the same, but observations about how to combine steps that expedite the production process are skipped.

Our specific goal in this section is to take a set of labeled marker positions and skeletal parameters and determine the pose of the skeleton that matches these observed marker positions. We make this step distinct from obtaining the two steps from this process:

- we assume that the marker information has been obtained, and any “low-level” processing such as marker identification (labeling), noise removal, and gap filling has been completed by a previous step. While this is often a very practical methodology, especially from a software engineering point of view, it is not necessarily the most sophisticated processing technique. It is possible to make use of the skeletal information to do improved low-level processing, but this comes at a complexity cost.
- we assume that the skeletal parameters, including bone lengths and relationships between the markers and the skeleton, were previously obtained somehow, and that the skeleton is close to the actual performer. Some anecdotal experience shows that the methods work even if the parameter values are not precise. In practice, these could be obtained by measurement, or by estimation.

In Section 13.5.6, we will suggest some methods for interleaving parameter estimation with the steps of marker processing. Some more sophisticated (typically optimization-based) methods compute the skeletal parameters in the process of doing the angle conversion and will be discussed later in this Chapter.

This discussion assumes a right-handed coordinate system with the convention that the positive Y-axis is upward, the positive Z-axis is backwards, and the X-axis points to the right. We will assume that our goal is skeleton whose limbs are aligned with the negative Y-axis (since they point downwards), the shoulders and hips follow the X-axis, and the feet point along the negative Z-axis. While the set of links (the skeletal topology) can be varied, a simple example is shown in Figure 13.1. The rest pose of this skeleton, the pose obtained with all of the joint angles set to 0, is referred to as the “Frankenstein” pose. This pose is not the rest pose of the character we will ultimately animate, nor even a pose that the performer can necessarily stand in.

A pedagogical suggestion: it is much easier to visualize the process with the help of one or two coordinate tripods. Tinkertoys, or even 3 pencils taped together, work well.

13.5.1 Building Blocks

We will refer to coordinate system objects as markers. This implies that for a “marker” we have not only a position, but an orientation. The position of the marker serves as the origin of the coordinate system, so we use the terms origin and position interchangeably. For the “real markers” (e.g. the ones for which positional data was provided by the motion capture system), we do not receive orientation, but we may arbitrarily assign one for consistency (our algorithms will never use the orientation of real markers). If we were using

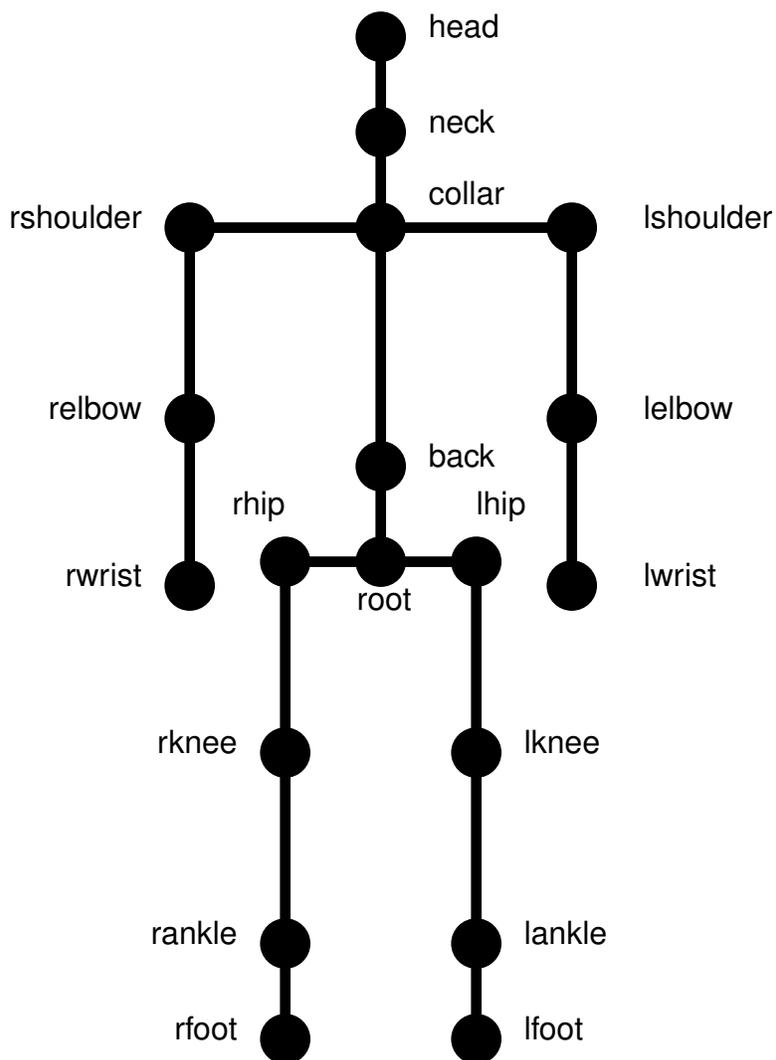


Figure 13.1: Sample skeleton for the simple capture processing discussion. With the joint angles all set to 0, the skeleton is in “Frankenstein” pose.

a capture technology that provided orientation information with marker positions, that could be used in the process.

The processing algorithm consists of two phases. In the first phase, a number of steps are taken in which new “virtual” markers are defined by considering the existing marker set. Two passes over the skeleton are made. In the first, approximate positions for joint centers are determined, and in the second, these positions are adjusted to meet the constraints imposed by the skeleton. In the second phase of the process, the relative angles between the coordinate systems. Our discussion focuses on phase 1, while phase 2 is mentioned in Section 13.5.5.

The exact sequence of steps used in the process depends on the available marker data and the target skeleton. Each new set of marker data and/or skeleton requires a different sequence of steps. Because of this, we prefer to provide the basic building blocks of the method as procedures in a scripting language, and encode each process as a script.

The fundamental operations move and orient markers. These functions are:

`position` – positions a marker relative to other markers;

`move` – displaces a marker by a fixed distance measured in its own coordinate system;

`orient` – defines the orientation of a marker’s coordinate system assuming the origin of the coordinate system is the markers “position,” and using 2 other markers to define directions;

`rotate` – rotate a marker around its origin by an amount specified in its local coordinate system;

`copy` – creates a new marker that copies the position and orientation of an existing marker.

Various versions of these operations are possible. Here, we introduce a simple set. In each case, the first argument to the operation is the name of the marker to be operated on. Argument types in the simple “language” are the names of markers, floating point numbers, and two character strings representing axis names.

`position marker1 marker2 marker3 portion` — Position *marker1* on the line between *marker2* and *marker3*. The parameter specifies the proportion of the distance between the two, e.g. .5 means half-way between the two markers. This command is often used to create new virtual markers. When a new marker is created, it is given the same orientation of *marker2*.

`move marker x y z` — Move *marker* by *x, y, z* measured in its local coordinate system. Note: it only makes sense to move a marker once it has been positioned and oriented.

`orient marker1 axis1 marker2 axis2 marker3` — Re-orient *marker1* such that *axis1* of its coordinate system points towards *marker2*. Then rotate around this axis such that *axis2* points as closely towards *marker3* as possible. The exact operation of this command will become more clear with the examples in the next section.

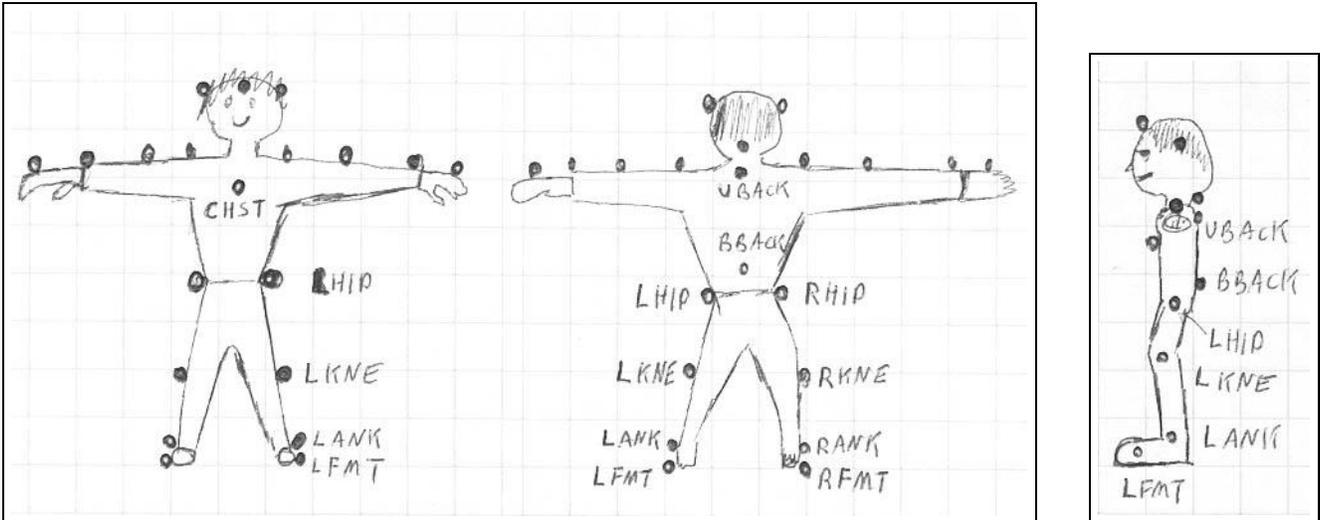


Figure 13.2: Our simple marker set used in the example of this chapter.

`rotate marker axis amount` — Rotatate *marker* by *amount* around *axis*. Note: it only makes sense to rotate a marker after it has already been oriented.

As an implementation detail, it is most likely to be most convenient to express all of the markers in a global coordinate system, and to represent the orientations as matrices. Having ready access to the three unit vectors that define the coordinate system is quite useful in implementing the various operations. In a later phase of the algorithm, the orientations can be converted to a more convenient form.

This list of operations is minimal and sufficient, however, often more complex operations are desirable. The commands can be built from the simpler building blocks, so they mainly serve as a convenience. For example, a 3 marker variant of the `position` command that allows the new marker to be placed relative to 3 other markers can be done with two of the 2 marker commands.

13.5.2 The Processing Process

In order to demonstrate how this simple process works, we work through a simple, but complete sample. While the skeleton and marker set used in this example are realistic, they have been chosen for their simplicity, not because they give particularly good results in real practice. The set of markers is depicted in Figure 13.2, and our simple target skeleton is depicted in Figure 13.1.

For clarity, the “real” motion capture markers have been given 4 character names and will always be written in upper case. Recall that at this point that no real marker exists at a joint or end-effector of the desired skeleton.

The first part of the first phase of the algorithm is to determine approximate positions and orientations for each of the joints. We will then clean these approximations up such that they maintain the proper segment

lengths. In this first phase, we choose rules that compute a joint position given a set of marker positions. The choice of these rules will depend on what markers are available, and what joints are needed.

The First Joint: The root

Our skeleton requires a joint at the “center.” We define this position to be half-way between the hip markers, so that the marker can easily be created by

```
position root RHIP LHIP .5
```

Our definition of the center position is quite simplistic. Chances are, we know either from experience or from measurement that the desired root position is not exactly at this point. Unfortunately, since we have not defined a coordinate system about this point yet, we cannot measure any displacements from this point.

To define a coordinate system for the root, we need to choose another marker that is useful for relating orientations to. In this case, we select the lower back marker. Because we want to define a coordinate system at the root node, with the x-axis facing right, the y-axis facing up, and the character facing down the negative Z-axis. An approximation to this is generated by:

```
orient root RHIP x+ LBAK z+,
```

since the positive Z axis points towards the back of the character. Since the lower back marker is usually placed above the hips, this really points the positive Z axis upwards, in which case we might want to rotate it downwards

```
rotate root x+ 45.
```

A visualization of this process is shown in Figure 13.3.

We have just introduced our first parameter, so we pause to discuss where these numbers come from. The parameters all relate measurements of the real actor to the idealized skeleton. We know that we have the parameters “right” if the process yields a configuration for the skeleton that is what we would expect from the configuration of the actor. A specific test might be that if the actor was standing in the skeleton’s rest pose (defined above to be the Frankenstein position), all of the joint angles would be zero. One possible way to determine the parameters would be to ask the actor to stand in the Frankenstein position, capture that pose, and tweak the parameters such that a correct response is given. Unfortunately, Frankenstein pose typically is not a pose that a real actor can obtain.

Notice that all measurements are made relative to real markers. This is important since as the actor moves around, we need to have the skeleton move accordingly. While this is obvious for positions, it is also essential for orientations. We have used the hips and lower back to define a base coordinate system for which other things can be build from.

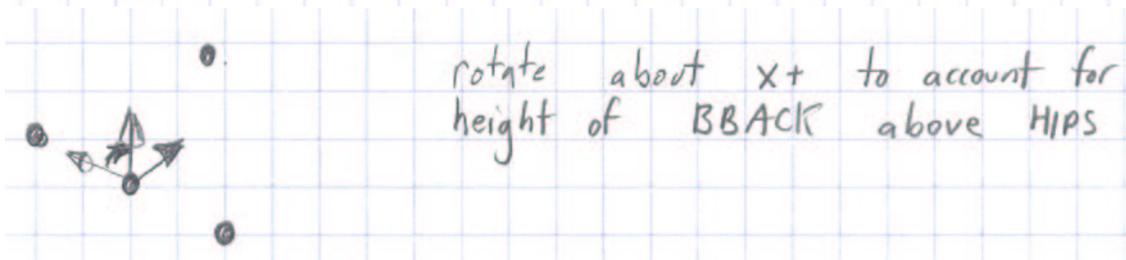
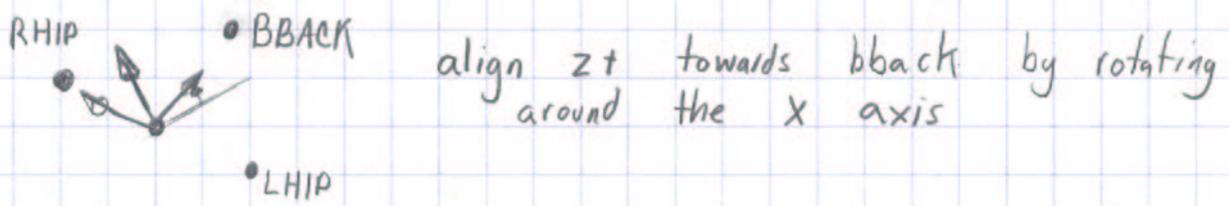
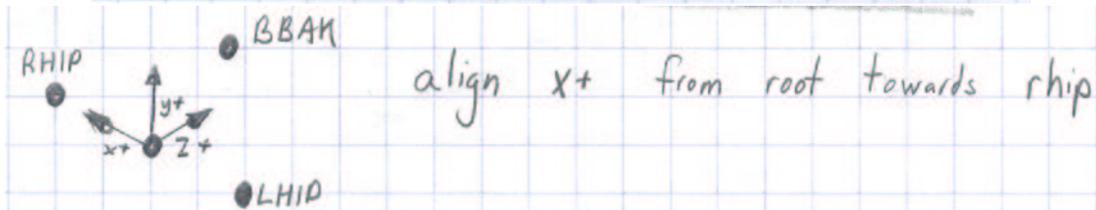
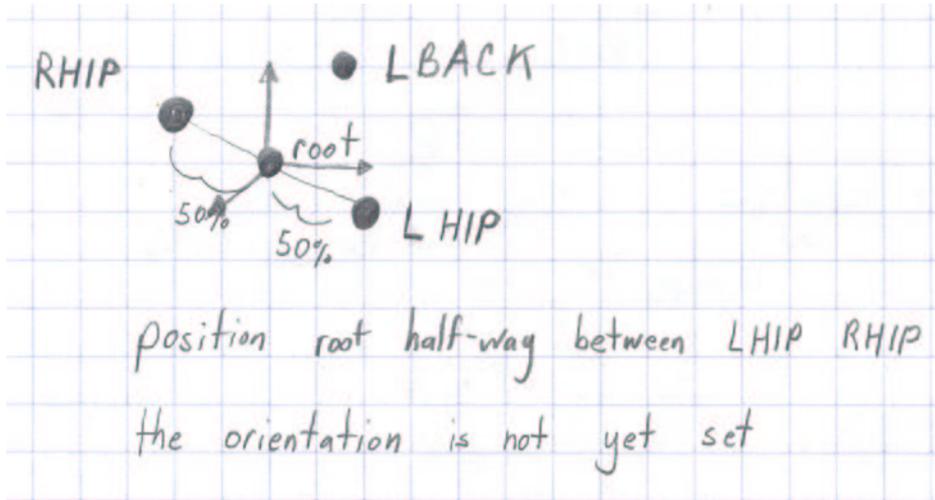


Figure 13.3: The process of finding the root joint's position and orientation visualized.

Subsequent Joints

The process for computing the position and orientation of other joints can be done similarly. The principle difference is that the set of markers that can be referred to in computing these new joints includes not only the original captured markers, but the computed “joint markers” as well. In a sense, the processor creates new “virtual markers” that can be used just as the real markers were. Also, for other joints, we rarely have sufficient information to compute all degrees of freedom, so we rely on simplifications, heuristics, and a priori knowledge of the expected motion. To demonstrate this, we continue the process through the joints of the right leg.

To compute the position of the right hip, we note that this position is in the root coordinate system, and we can use the skeleton’s parameters to determine where.

```
position rhips root RHIP 0
move rhips 6 0 0
```

The parameter value (in this case specifying that the right hip is 6 inches along the X axis from the root) must have been determined somehow beforehand. Because the hips are not truly kinematic rotational joints, there really isn’t a specific position that could be measured, so even with X-Ray imaging, these parameters could not be measured. We will discuss the problem of determining these parameters in Section 13.5.6.

Computing the orientation of the hip is even more difficult. We know the hip should be oriented such that the bone “leaving” the joint points towards the next joint, in this case, the knee. Our skeleton has this thigh bone pointing down the negative Y axis, so if we knew the location of the knee, we could point the Y axis correctly. Also, once we determine the major axis for the thigh bone, we need to determine the rotation about this axis (such that the front of the thigh faces front).

Even within the framework of our simple, procedural processing, there are many potential ways to determine the knee position and thigh orientation. The available information certainly influences our choices. For example, if we had placed real markers on both the inside and outside of the knee, we could compute the knee joint’s position as being between these two locations, and use the vector between them to provide orientation information. With our limited marker set, which was most likely chosen for some other pragmatic concerns, this is not possible.

To simplify our problem, we treat the knee as a planar joint. In this way, the two bones of the leg define a plane, and the axis of rotation for the knee joint is perpendicular to this plane. This is one of several possible simplifications. Some simplification is necessary because the given marker set does not provide enough information to determine all the parameters.

Using the planar leg assumption, we can estimate a position of the knee joint based on the markers. Just as the real joints in the leg define a plane, we can imagine the markers on the leg defining a parallel plane. We first place the new knee joint at the same position as the knee marker, and orient it such that the X axis is perpendicular to this plane. The position of the knee joint can then be found by displacing along the X axis. The -2 represents a parameter that must be determined somehow.

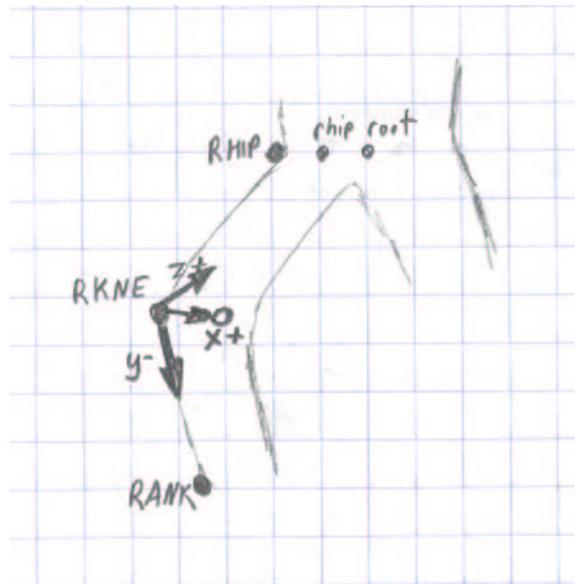


Figure 13.4: A co-ordinate system is defined for the knee. The coordinate system is first placed at the location of the knee marker, the negative Y axis is pointed towards the ankle marker, and the Z axis is pointed as closely towards the hip as possible. This causes the negative X axis to be perpendicular to the plane of the leg such that the knee joint position can be determined.

```
position rknee RKNE RANK 0
orient rknee RANK Y- RHIP Z+
move rknee -2 0 0
```

Notice that while one axis was obvious to define (the Y axis, since we know the bone is oriented toward the ankle), we needed to define a second axis directed perpendicular to the plane of the leg. For this reason, we make the more arbitrary choice for the Z axis as our real concern is defining the leg plane such that the X axis is defined (to be normal to this plane).

Recall, in this phase, we are only finding joint positions that will be used in a later phase to compute skeletal angles. Which is why at this point we can be a little less concerned about getting all of the orientations correct, and why we do not consider the skeletal bone lengths yet.

A similar process can be used to find an estimate of the position of the ankle.

```
position rankle RANK RKNE 0
orient rankle RKNE Y+ RHIP Z+
move rankle -2 0 0.
```

13.5.3 Orienting Co-Ordinate Systems

Having seen the orient operation in action, we can now consider how it is actually implemented, and see some of its pitfalls.

The orient command must compute the directions of the three axes of the coordinate system given the positions of three points, which we will call O , B , and C . The command uses the first point (O) as the center of the new coordinate system, and uses points B and C to define the orientation of the axes. One axis is defined by the vector between O and B . A second axis is defined by the vector between O and C . Because the axes must be perpendicular, the actual second axis is chosen to be as close to the vector OC as possible, yet still being orthogonal to the axis defined by OB . This condition is equivalent to requiring the vector to lie in the plane defined by the three points. Given the two axes, and the requirement that coordinate systems are right-handed, the third axis is uniquely determined.

The first step of the orient command, aligning the first axis towards the second marker, is simple: we merely take the vector between the first two markers, normalize it, and use this as the appropriate axis (e.g. column of the orientation matrix).

The second axis is more problematic. It requires us to maintain the direction of the first axis, but to rotate around the second axis in a way that has this axis point as much towards the third marker as possible, but is perpendicular to the first (so that we can form a coordinate system). In the event that the three markers are co-linear, any vector in the plane perpendicular to the line would qualify, so the orient command is undetermined. Any real implementation of a marker processor must provide a mechanism for handling such singular situations. Generally, more information is needed: for example, we might consider more points, or more than just the current instant in time.

If we do not consider the singularity problem, the coordinate system can be built from the three points. Since we know that the third vector must be perpendicular to the plane that the three points lie in, we can compute its axis by taking the cross product between the vector connecting points 1 and 3, and the one connecting points 1 and 2. The second

we determine an appropriate third axis by finding the cross product of the vector between points 1 and 3 and that first vector. Taking the cross product of these two axes gives the second axis. The trickiest part of implementing this is providing for all of the possible axis combinations, insuring that the cross products are correct to create right-handed coordinate systems. Example code is given in Appendix ??.

13.5.4 Finding Skeleton Points

While the procedures of Section 13.5.2 can find positions for the joints using relationships amongst various markers, they do not consider the constraints of the limb lengths of the skeleton. Another pass over the data can not only orient the joints, but establish the skeletal constraints as well. The trick is to reposition the joints so they are the correct distance apart.

Returning to the hip (which was already positioned to a skeletal length), we can place the thigh by aiming it towards the knee, and positioning the knee so the thigh has the correct length:

```
orient rhips rknee Y- rankle Z+
position rknee rhips rknee 0
move rknee 0 -15 0,
```

where 15 is the parameter that specifies the length of the thigh bone. Notice that this uses our simplification that the leg is planar and that like the original positioning of the knee, fails if the leg is straight.

A similar procedure can be used to compute the position of the ankle, given the length of the shin:

```
orient rknee rankle Y- rhips Z-
position rankle rknee rankle 0
move rankle 0 -14 0.
```

A simple, but complete, example is provided in Appendix ??.

13.5.5 Computing Angles

After these two passes over the skeleton, the first phase of marker processing is complete: we have determined the position and orientation of each bone of the skeleton. What remains is to convert these coordinate systems (most likely represented as transformation matrices, or a set of vectors) into the target representation of the skeleton.

If the skeleton is to be represented in a non-hierarchical fashion, the conversion simply requires each coordinate system to be converted into the corresponding bone's parameters. For example, if we are using Euler angles to represent the configuration of each bone, we would need to factor the orientation matrix into the three Euler angles.

If we are to use a more conventional, hierarchical representation of a skeleton, we first must express each coordinate system relative to its parent in the hierarchy before converting its representation. This is achieved by premultiplying the coordinate system's transform by the inverse of the parent's.

13.5.6 Finding the Parameters

In the course of doing the processing, we employed a number of parameters that specified the underlying skeleton, and its relationship to the markers. These parameters include the lengths of the skeleton's bones, offsets of the joint centers from the actual markers, and rotational offsets between markers and joint coordinate systems. We note that for this process to work properly, the parameters must be chosen correctly such that underlying skeleton corresponds sufficiently to the original performers, and that the relationship between the markers and the skeleton are also realistic. Because we use the actual marker positions, if we choose incorrect parameters, we will determine an incorrect motion.

We should stress that since the performer is not actually a rigid kinematic hierarchy (unless we are capturing a robot), the parameters are an approximation. Ideally, the skeleton is the skeleton that best matches the performer, and the processing result is the motion that has the skeleton come as close as possible

to recreating the observed marker motions. Because the skeleton, and its connection to the markers, is only an approximation to the real performer from whom the markers was measured, the resulting motion cannot be expected to be an exact fit.

The problem is to determine a set of parameters that provide a skeleton and marker linking that reasonably mimic the actual performer. This problem is sometimes called *marker calibration* or *performer calibration*. It should not be confused with the device calibrations typically required by motion capture hardware for determining the relationships between the raw observations and the reference coordinate system.

While measurements can assist in marker calibration, it cannot fully accomplish the task. Even if we were able to use X-ray imaging to see inside of a performer, there are no real joint centers in a human body. Schemes which use external measurements to help determine the performer's parameters are basically ad-hoc.

A correct set of parameters would have the resulting skeleton perform the same movement as the original performer. This definition of success provides a method for assessing the quality of a calibration. If we capture the performance of a known motion, we can see if the resulting skeletal angles match sufficiently closely. This process even provides a mechanism for tuning parameters, and is the core of the more sophisticated optimization-based approaches discussed in Section ?? that automatically adjust the parameters to find the ones that provide for the best match.

If some known motions, or even poses, can be provided, a standard approach to determining parameters is to manually adjust the parameters until the skeletal results match the known performance. Typically, such manual adjustments are done with a single *calibration pose*. This is a simple pose that is easy to make the matching adjustments for.

The use of known poses offers a number of technical and pragmatic challenges. For one, the known poses are most certainly inexact. There is no way to insure that a performer holds their arm bent at exactly 45 degrees, or even perfectly straight. Also, it can be difficult to get a performer to stand in an exact pose, or repeat an exact movement. Worse, such calibrations must be redone periodically as the physical markers on the performer may not be perfectly stable.

An alternative to calibrating from known motions and poses is to look for known details across unknown motions. For example, if we are manually adjusting the parameters we might look for cases where a character's foot goes through the floor, or other impossible situations occur. Corrections are made to avoid these situations. Similarly, by watching the relative motion of a set of markers, we can determine best-fit centers of rotation. By using statistics over a duration of time, good estimates are possible.

Approaches to finding the parameters using the actual motions themselves are called *self-calibration*, and are discussed in later in this chapter.

13.5.7 The Pros and Cons

Understanding the benefits and problems of the simple motion capture processor is the key to gaining insight into more sophisticated methods, as well as determining if they are sufficient and practical for a given application.

Flexibility in the creation of processing procedures: clever practitioners can devise processing scripts and marker placements that achieve good results, however, these scripts must be hand-crafted.

Separation of various tasks: the process assumes that the marker correspondences have already been determined, that missing data is filled in, and that noise is handled either before or after. While this allows for a degree of modularity, it also does not allow for the model information known in later stages to be used in earlier stages.

Simplicity of implementation: because the processor is built from a small set of simple building blocks, the software can be built and tested easily. However, the real complexity comes in devising processing procedures that assemble the building blocks.

Speed and performance predictability: the process performs only very simple computations, and performs them in a pre-defined order. The speed afforded by the process allows an iterative cycle where corrections can be made to marker information and parameters and the process re-applied until the desired results are achieved.

Locality of computations: the position of each joint is primarily computed by markers close to it. While this has advantages in avoiding the propagation of problems, it does not allow information to be shared.

Feed-forward design: the process works from the root outward. Information at the ends cannot be used to address problems closer to the center.

Frame at a time: the process examines only information for the current frame, it does not rely on information from other times. Unfortunately, this means more limited information is available for processing.

Singular and near-singular configurations: the process as described fails completely for singular and near-singular poses (for example, when a limb is straight). The potential upside is that such failures are easy to identify, and can be flagged for possible cleanup later.

No direct handling of constraints: the process does not place additional importance on the position of end-effectors. Often, this is desirable as end-effector positions are more important than angles in order to prevent foot slipping and other problems.