3 Skinning

Once motion curves are set for each joint degree of freedom, and for the six degrees of freedom of the root frame with respect to the world, the skeleton has been succesfully animated: that is the essence of Forward Kinematics. However, it's generally not ready for rendering.

Apart from robots which might actually be composed of rigid parts, characters typically have a smoothly deforming skin wrapping around all their internals: rendering the rigid segments of the skeleton as rigid bodies is not going to cut it.

In fact, usually the process starts with a geometric model of the outer skin, sculpted by the artist in a sensible neutral rest pose. The process of *rigging* then adds bones "under" the skin with appropriate joints, initially for the same neutral rest pose (joint angles and translations equal to zero, say). The desire is then that as the bones are animated, they should somehow drag the skin along around them in a natural way. The algorithm that deforms the skin to match the motion of the underlying skeleton is called *skinning*.

3.1 Linear Blend Skinning

The simplest, and still most popular, method is called *linear blend* skinning, or sometimes *Skeleton Subspace Deformation* (SSD). This has a long and unpublished history in the industry side of graphics; Lewis et al. [LCF00] provide a good overview and extend it in useful ways.

Linear blend skinning begins by assigning weights, or partial ownerships, for every vertex on the skin mesh to the underlying bones. For example, a vertex in the middle of the upper leg would be 100% the property of the upper leg bone, that is a weight of 1 for the upper leg bone and a weight of zero for every other bone. A vertex on the very centre of the knee might be 50% upper leg bone and 50% lower leg bone, that is a weight of 0.5 for

the upper leg bone and 0.5 for the lower leg bone but weight zero for every other bone.

More formally, for each vertex i in the skin mesh and for every bone k in the skeleton we want a weight w_{ik}. The weights should represent an average, or blend:

$$0 \leq w_{ik} \leq 1, \qquad \sum_k w_{ik} = 1. \tag{6}$$

You can think of all the vertex weights for a particular bone k as a function over the skin mesh, sampled at the vertices. It should generally be a smooth function, equal to one on the bits of skin that are uneqivocally attached to bone k, equal to zero far enough away from the bone, and smoothly ramping between zero and one over the joint regions for the bone.

Blend weights can be constructed automatically, say based on distance between the skin vertices and line segments representing the bones, but it's hard to reliably create good weights automatically. Usually the artist must "paint" the weights on the mesh directly, using their knowledge of anatomy and rigging experience.

In the "rest space" of the model—that is, the world space where we have placed the skin mesh at rest and the skeleton at rest inside it—every skin mesh vertex has rest space coordinates. Call these coordinates p_{iR} for vertex i. Every bone has a local coordinate system too: say bone k's frame in the rest pose has transform $T^{k\leftarrow R}$ from rest space coordinates to local bone coordinates (that can be worked out by a Forward Kinematics tree traversal). That means any vertex i has local bone k coordinates p_{ik} defined as

$$p_{ik} = T^{k\leftarrow R} p_{iR}. \tag{7}$$

We save this along with the weights[2]. It's important to get an intuition for what these local coordinates mean. Going back to the knee example, the

[2]Depending on how many vertices and how many bones, we maybe wouldn't store all the zero weights and the associated local coordinates to avoid unnecessary memory use. We could also save just the p_{iR} along with the rest space transformations, and construct each p_{ik} on the fly whenever needed.

vertex might have local coordinates $(0, 0.4, 0.05)$ with respect to the upper leg bone and local coordinates $(0, 0, 0.05)$ with respect to the lower leg bone, but in the rest pose these two sets of coordinates represent the same point in space. The upper leg bone sees the point as being somewhere near the end of the bone, and the lower leg sees it as being close to the start of the bone.

We can of course invert the transform, $T^{R\leftarrow k} = (T^{k\leftarrow R})^{-1}$, to convert from local bone coordinates back to rest space coordinates:

$$p_{iR} = T^{R\leftarrow k} p_{ik}, \qquad \text{local to rest} \tag{8}$$

for any bone k. Vertex i has different local coordinates for each bone, but they all map to the same point in space when the bones are in their rest pose, and that point is what the artist designed for the skin at rest.

Since the weights add up to one, we can even write p_{iR} as a weighted sum over the bones:

$$p_{iR} = \sum_k w_{ik} T^{R\leftarrow k} p_{ik}. \tag{9}$$

That is, the skin vertex is at a weighted average of where all the bones think it should be—obviously, since they all think it should be at the same point in space.

However, we can use equation (9) even when the bones are not at rest! The same intuition applies: when the skeleton moves, we still want a skin vertex to be at the weighted average of where the bones think it should be. Remember again that the weights are designed to only allow nearby bones to influence a bit of the skin: vertices at the knee probably will only have nonzero weights from the upper and lower leg bones, so will be placed according to a blend of where the upper and lower legs expect the knee to be.

Putting this in a formal equation, let p_{iW} be the skin vertex coordinates in world space, and let the current transform for bone k to world space be $T^{W\leftarrow k}$ (this transform is presumably a function of time, controlled by the

motion curves on all the joints above bone k in the tree and the placement of the root of the skeleton in world space). The linear blend skinning formula is then:

$$p_{iW} = \sum_k w_{ik} T^{W \leftarrow k} p_{ik}. \tag{10}$$

3.2 Problems with Linear Blend Skinning

Linear blend skinning is remarkably useful for something so simple, but it has its limits. Some artifacts become visible when joints rotate too far. In a rotating joint we expect the skin to rotate around the joint too, maintaining the volume, but the linear blend model instead interpolates skin vertex positions *linearly* between where the bones expect them which "cuts a corner", shrinking the volume of the joint.

Two particular examples are often singled out. The first is at elbows and knees: if the rest pose is the straight limb, an odd contour develops when the limb is bent, a bit like a rag-doll. The other can be even worse: at a swiveling joint (sometimes used to model the wrist or the neck) a big rotation pinches off the skin into a much narrower cross-section a bit like twisting a candy wrapper.

Fixing these requires some intervention. One possibility is to improve the skinning algorithm itself, for example dual quaternion skinning [KCZO08], or even full-fledged volumetric simulation of the physics of the underlying flesh (e.g. McAdams et al. provide a recent example of this approach [MZS+11]). Another possibility is to generalize the notion of "bone" somewhat.

4 Generalized Bones and Joints

We started off this chapter by developing an intuition from real skeletons, with rigid bones, to create a tree of rigid coordinate systems which can

drive skinning. We don't have to stick with this restriction, however: we can still use the formula from equations (7) and (10) even if some of the "bones" don't have an obvious anatomical counterpart or don't have rigid frames. Likewise it's worth pointing out that we can bend the rules on what a joint is to cover generalized bones or other scenarios.

One classic use of generalized bones is at problem joints. For example, at the knee you could actually put in a "knee cap" sort of bone. Its coordinate system would be enslaved to the joint between upper leg and lower leg instead of being free, rotating at half the angle of the joint (so it's midway between the rotation of the lower leg versus the upper leg). You can likewise partially fix a swivel joint the same way.

Another use of generalized bones is other types of not-quite-skeletal deformation. When a character flexes their arm, you might want to see their bicep bulge—which can't happen when skinning a rigid upper arm. Instead you could put in a virtual bicep "bone" which is enslaved to match the rotation of the upper arm but has a scaling (expansion/contraction of space) keyed to either the elbow joint angle or just another animation control (a generalized joint, which doesn't correspond to any physical joint). The chest expanding and contracting with breathing is another common example. Note that scaling is not a rigid transform, but is still an affine transform that can be represented with the same 4×4 matrix structure, for example.

5 Blend Shapes

Generalized non-rigid bones and joints are very useful for tweaking the basic skinning of skeletal motion, but they can't handle more complicated deformations—ones that go beyond simply moving around chunks of the rest mesh in a uniform way. The classic example is the face: an enormous range of expressions are generated by the complex muscle system underlying the skin of the face, even without any actual bone movement. It's

very difficult indeed to imagine what generalized "bone" structure could be put in place to match all of these expressions, let alone how an artist could intuitively control such strange "joints".

An alternative approach to skinning here is much more appropriate: *blend shapes* (also known as "shape interpolation"). This is actually a simpler technique than linear blend skinning, but needs a bit more artist work—though not as much as the artist needing to sculpt every keyframe individually.

The first step is to get the artist (modeler or rigger) to create some basis shapes, meaning deformations of the skin mesh to capture various extremes of expression. For example, a face might have a neutral shape S_0, represented as a long vector containing the positions of the mesh vertices concatenated one after the other—with coordinates in the local head frame (so the face is attached to the head!). To that the user could add S_1, a smiling face, with a different set of positions, and a frowning face S_2, a surprised face S_3, and so on.

The animator can then much more easily set plausible face shapes by taking a weighted average of the basis. For example, a happy and surprised face might have vertices at local coordinates $0.5 \cdot S_1 + 0.5S_3$. The user interface could actually expose the weights for differences between each expression basis shape (S_1, S_2, ...) and the neutral shape S_0. The final positions of the vertices is then

$$S = S_0 + \sum_k w_k (S_k - S_0), \tag{11}$$

where the weights w_k are directly controlled by the animator (set at keyframes, interpolated by motion curves). Note that negative weights, weights that sum to more than one, or individual weights that are greater than one are all possible, generating expressions even more extreme than the basis—but taken too far the results will probably not look good.

Of course, blend shapes can be used with linear blend skinning: the blend shape differences can be added to the rest pose of the mesh before the

bones deform things around; this is a better solution to things like muscles bulging. The jaw bone might well be fully controlled by the blend shapes on the head rather than animated as a separate bone in the Forward Kinematics tree.

There are many further techniques used to easily animate the deformations of meshes, which we will not cover here.