<< RETURN

Revisiting the Standard Joint Hierarchy:
Improving Realistic Modeling of Articulated Characters

Victor Ng-Thow-Hing
Email: vng@honda-ri.com, Honda Research Institute USA, Inc.

Figure 1: Human skeleton model

Abstract

The characters in modern video games continue to be updated with improving surface geometry and real-time rendering effects with each incarnation of graphics hardware. In contrast, the standard joint hierarchy used to animate these same characters has changed little since it first started appearing in graphics applications in the 1980's. Although simple to implement, a great deal of realism in joint articulation for human characters is compromised. Several biomechanical enhancements are presented that can capture more realistic behavior of joints in articulated figures. These new joints are capable of handling non orthogonal, non-intersecting axes of rotation and changing joint centers that are often found in the kinematics of real anatomical joints. Coordinated movement and dependencies among several joints are realized. Although the joint behaviour may appear complex, the simplicity of controls for the animator is retained by providing a small set of intuitive handles. An animator is restricted from putting the skeleton into an infeasible pose. We illustrate these concepts with detailed and realistic human spine and shoulder models exhibiting real-time performance and simple controls for the animator.

Motivation

The need to realistically animate human characters is an important element contributing to the appeal and success of interactive games. The 3-D humanoid character performing the action in a game is the direct interface through which players express themselves. Games currently benefit from improvements in graphics hardware by depicting humanoid characters with greater levels of detail in their geometric surface models. However, studies in perception of human motion with different geometric models have suggested that observers may be more sensitive to motion differences if polygonal models are used compared to stick figures

(Hodgins et al., 1998). As the majority of 3-D games continue to use polygonal models with increasing detail, observers may be more sensitive to unrealistic joint motions in animations, such as in the shoulder and torso regions of the body. This is especially relevant to sport simulations where perceived athleticism is tied to the coordination of movement in virtual human players. For animation techniques that deform an outer skin model based on skeleton motion, accurate joint transformations of bone segments can lead to better performance of these methods by improving the association between skin movement and the underlying bones and modelling more accurate muscle deformers in response to joint movement.

Despite the continued work to develop better joint models in both the biomechanics and computer graphics communities (see the History section), their adoption in mainstream computer graphics has been limited. Part of the reason may be the misperception that complex joint models will require more user handles for an animator to control or that these models require large changes in software infrastructure that make their use incompatible with popular 3-D modelling packages, such as *Maya*.

Indeed, the majority of commercial software allows skeleton hierarchies to be built only as a tree of ball-and-socket joints between bone segments. The data structures for bone segments consist of a transformation matrix that is relative to its parent coordinate frame in the tree. The motion of a single degree of freedom (DOF) is usually restricted to the relative motion between two adjacent bone segments. The flexible spine in the torso is often reduced to a few rigid links. The shoulder is often simplified as a three-axis gimbal joint, probably causing many of the difficulties of animating geometry around that region. The joint hierarchy is fundamental to many key processes in game development, from skeleton fitting with motion-captured data to character skinning algorithms whose deformations are weighted by the underlying joint transformations. A great deal can be gained if we can update the hierarchy with more biomechanical realism.

We'll review several biomechanically based principles for creating better realism in joint motion. The implementation of these ideas can be done with existing open architecture software environments like *Maya* or as custom software with any popular 3-D API like *OpenGL*. Each principle is presented in a modular fashion, so that they can be independently implemented or combined as building blocks within a 3-D character. These ideas are implemented and demonstrated on a realistic human skeleton model (Figure 1). First, let's review how the traditional joint transformation tree hierarchy started out in graphics.

History

The origins of articulated joint models for human character representations can be found in the study of kinematics of robotic manipulators. See any standard robotics textbook like *Introduction to Robotics: Mechanics and Control* (Craig 1989). Early animation systems, such as PODA (Girard and Maciejewski 1985) made use of the Denavit-Hartenberg link parameter notation from robotics to represent figures with articulated limbs. Although the notation is a convenient way to relate coordinate frames between adjacent segments with four parameters,

each parameter set only describes a single degree of freedom between two segments. Multiple sets of parameters must be combined to achieve multiple degree of freedom joints.

Although Euler angles are often used to express segment orientations, quaternions (Shoemake, 1985) and exponential maps (Grassia, 1998) have emerged as excellent alternatives that have desirable interpolation properties as well as avoiding singularities inherent with Euler angles. Nevertheless, Euler angles have persisted in popularity probably because their degrees of freedom have natural analogs to motions descriptions such as twist, flexion-extension and abduction-adduction in human movement (see Figure 3).

In the area of biomechanics, physiological joints have been shown to have many complexities that are often neglected in graphical models. For example, biomechanists routinely specify joints with several non-orthogonal, arbitrary axes of rotation (Delp and Loan, 1995) that are better aligned to bone articulation. Many joints have translational components and changing centers of rotation, including the knee which is traditionally simplified as a single DOF hinge joint. In joints like the shoulder, the closed loop consisting of the clavicle, scapula and thoracic surface of the rib cage creates a coupling between the articulations of all these joints. This situation has been modeled by enforcing a constraint on the scapula to stay on the surface of an ellipsoid approximating the rib cage (Garner and Pandy, 1999). Other structures, like the human spine, exhibit high degree of coupling behavior between the vertebrae. Monheit and Badler (Monheit and Badler, 1991) exploited this fact to develop a kinematic model of the human spine that exhibits flexion/extension, lateral bending and axial twist rotation.

There have been several animation systems that focus on modelling accurate, anatomical joints. Maciel et al. (Maciel et al., 2002) develop a model that incorporates joints that can translate and rotate together on a plane and have joint limits that dynamically change with the DOF of any joint. Joint cones (Wilhelms and van Gelder, 2001) have been used to provide a better mechanism of restricting joint angle ranges for ball-and-socket joints. The Peabody system (Badler et al., 1992) collects joints into joint groups that have group angles to configure the joint group's segments. The H-Anim specification (Humanoid Animation Workshop Group, 1999) describes a standardized humanoid joint hierarchy for the purpose of avatar representation. From this great body of work, common elements for joint modeling can be observed and a set of principles for realistic modeling can be created. In some cases, specialized joint models are often needed, as in the shoulder and spine. We'll aim to hide the complexity of these articulations by making them accessible to animators through intuitive controls. In applications where physiological consistency is desired, an animator (or player) should not be allowed to configure a skeleton into a non-natural, impossible pose.

Joint Map Concept

In the traditional joint transformation hierarchy, each segment is positioned and oriented by a set of degrees of freedom that are typically translations and/or joint rotation angles. Since a segment is usually moved relative to its parent, adjusting a degree of freedom moves the segment as well as all its child segments (Figure 2, top). A key observation is that in many natural limb motions, joints behave in a coordinated or dependent way. It would be better to have an animator adjust a set of intuitive parameters and have the joint articulations

automatically behave in a realistic manner with all the coordination dependencies and physiological limitations built right into the model. We can imagine a *joint map* where the input parameters are intuitive degrees of freedom and the outputs are modified joint angles, translations or complete transformation matrices that position and orient the bone segments in our articulated character models (Figure 2, bottom).

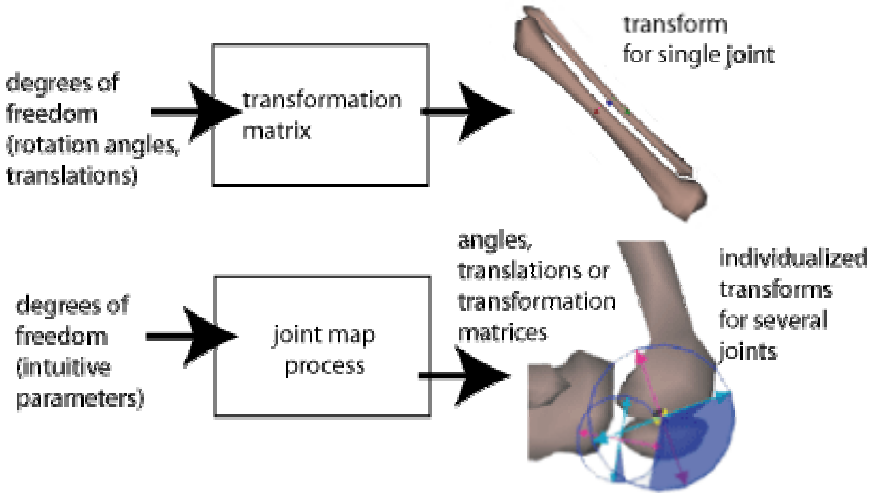

Figure 2: Joint maps (below) convert degrees of freedom to coordinated motion for several joints instead of just a local transformation for a single joint (top).

The joint map is responsible for defining the transformation matrices for one or more joints in its jurisdiction. The outputs of one joint map can be combined with the inputs of another to created increasingly sophisticated behaviours. The process that performs this mapping varies according to the type of joint phenomenon desired. The next few sections identify various principles and explain how these processes can be implemented. In all the examples, we will make references to various motion descriptors that are illustrated in Figure 3.

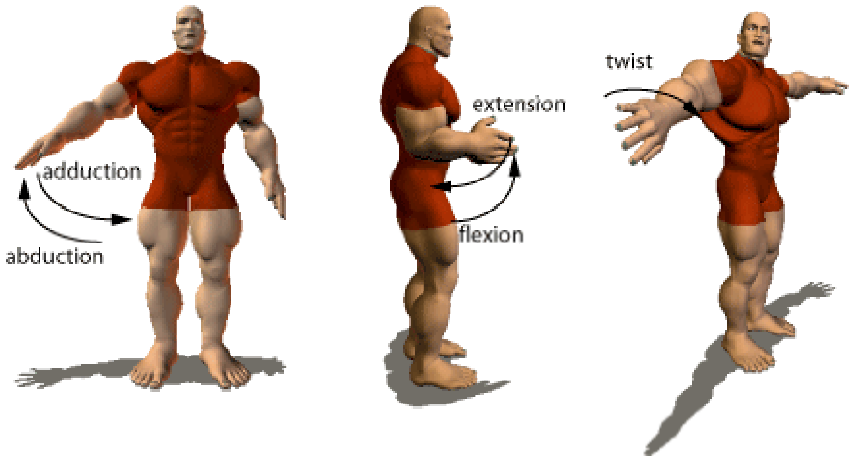

Figure 3: Various terminology used for limb movement. Although the formal term for the arm twisting along its longitudinal axis is pronation/supination, we'll use *twist* to refer to rotation about the longitudinal axis of any body segment.

We will also refer to bones and joint names in the shoulder that are shown in Figure 4.

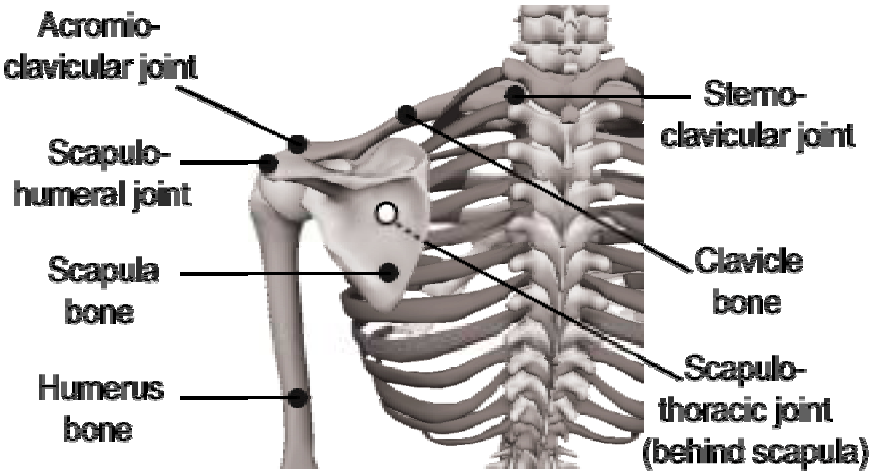

Figure 4: Various bones and joints involved in the shoulder.

One-to-many relationships

A *one-to-many mapping* has a single input and produces a set of one or more transformation matrices for several bones in the skeleton. Usually, these bones are adjacent to each other in the kinematic chain (like the vertebrae in the spine in Figure 5) or a knee where the bones translate and rotate during the bending of the joint. The input description may have a meaning like “flexion/extension” or “arm raise”. Internally, this value then maps to individual joint angles or translations for each joint/bone that participates in the joint map. These in turn can be used to compute the output transformation matrices for each joint. The mapping of the single input to joint angles can be any linear or nonlinear function. This allows different rates of rotation

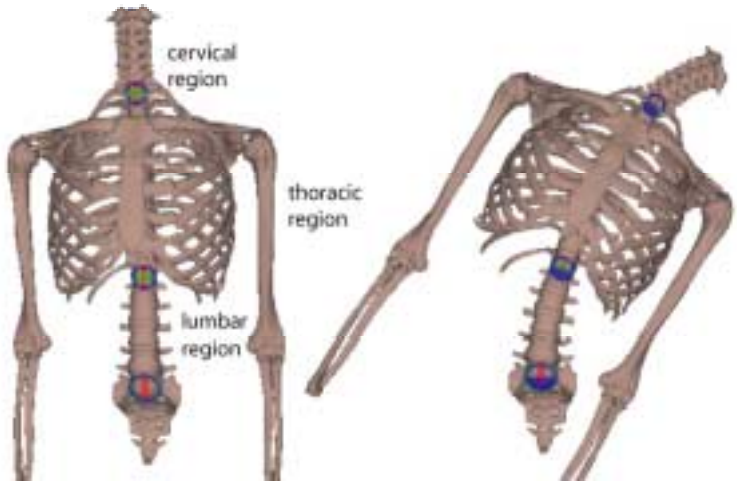

Figure 5: The 24 movable vertebrae are simultaneously rotated in a coordinated manner using only 3 dials, each one located in a different region of the spine. A one-to-many mapping maps the lateral bending DOF to a rotation for each vertebrae within its corresponding region..

for several joints or unit conversions to be made (for example, degrees to radians). The range of the mapping can be bounded to implement joint limits for one or more joints. For example, the domain [0,1] can be mapped to the rotation angles $[\theta_{\min}, \theta_{\max}]$ for a single-axis rotational joint. This technique allows a relatively few set of parameters to control many points of articulation within a skeleton.

Dependencies

Dependency mapping allows the modeling of coupling behavior between different joints within a joint map. For those of you familiar with *Maya*, this can be done using the *set driven key* animation controls the software offers, so that one transformation attribute can drive another. Using dependencies, you can prevent an animator from posing a set of joints into awkward positions or create joints that limit each other's movements depending on their current configuration.
In a dependency, a pair of joints are specified, one as the active joint *a* that drives the other passive joint *p*. Similar to one-to-many mappings, the actual nature of the mapping function used in the dependency component can be any linear or nonlinear function. Interpolating splines are one way to conveniently match dependency relations to specific joint angles. For example, one may want the clavicle to be 30 degrees elevated when the humerus is 60 degrees elevated. Typically, the inputs on dependency mappings will be rotation angles. The dependency component takes two input rotation angles, one from each joint, and contains a mapping function to output a modified angle for the passive joint. Here's a suggestion of several types of mapping relationships:

1. One-to-one mapping: For any given DOF value of *a*, a DOF value for *p* is defined. For instance, the rotation of the scapula (Figure 4) around an axis perpendicular to its outward surface tangent plane is almost linearly dependent on the abduction of upper arm. A linear one-to-one mapping can capture this relationship.

2. One-sided bound mapping: This is a one-to-one map where values of *p* are bounded on one side by a lower or upper limit that is a function of a DOF of *a.* An example of this is the dependency between the abduction of the humerus bone and the elevation of the clavicle bone. The higher the upper arm is raised, the more restricted is the vertical movement of the shoulder's clavicle. The restriction is due to a lower limit placed on clavicle elevation, which can be implemented as a one-sided bound that is dependent on the amount of abduction of the humerus.

3. Two-sided bound mapping: The value of a DOF of *p* is bounded on both sides by limits dependent on a DOF of *a.* Again using the shoulder as an example, when the left upper arm is rotating in the horizontal plane from the left side to the front right of the body, the horizontal movement of the shoulder (at the clavicle) becomes more restricted. A similar phenomenon occurs when the left upper arm is rotating to the back of the body. Since there are both upper and lower limits, a two-sided mapping is appropriate.

Compensating transformations

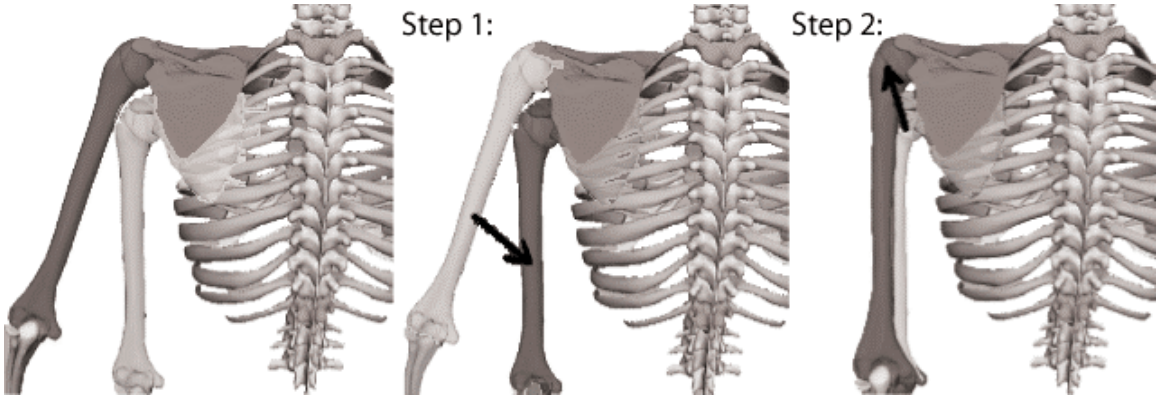

Figure 6: Compensating transformations decouple the propagation of a parent joint's transformation to its children. The humerus rotation is cancelled out so that it does not rotate with the clavicle.

In a standard hierarchical skeleton tree, the transformations of a parent segment are inherited by the child segment. However, this may produce undesirable behavior in some situations. For instance, if we wanted to shrug the shoulders of a human model, the rotation in the clavicle would propagate to the humerus, causing the humerus to rotate away from the body (Figure 6). An animator may want to keep the orientations of the humerus and clavicle independent. The *compensation component* cancels out the orientation transformation (created by any rotation parameterization) of any ancestor segment (not just the direct parent) for a particular segment while maintaining connectivity at the joint with its segment by adjusting the translation of the segment. This allows a segment to have orientation with respect to the world frame.

The compensation component for a particular segment *s* takes as input a single transformation matrix of an ancestor segment and produces a matrix which cancels out the undesired orientation change caused by the rotation of that ancestor. Depending on how far up the skeleton hierarchy tree we wish to cancel out orientations, a segment can have a compensation component for each ancestor. The outputs of each compensation component are then multiplied together to produce a matrix that compensates for all the undesired movements caused by a segment's ancestors up to a certain level in the skeleton tree. In the shoulder, two compensation components are used to create an independent humerus orientation from the scapula and clavicle rotation (Figure 6). The first component cancels the effects of the acromio-clavicular joint (connecting the scapula to the clavicle), and the second component nullifies the sterno-clavicular joint (connecting the clavicle to the sternum of the rib cage).

The computation of the compensation matrix is done in two steps. First, the inverse of the ancestor segment's transformation matrix is computed to cancel out its movement. At this stage, the segment has moved back to its original position and orientation before any of the ancestor transformations have been applied. The segment now has to be reconnected to the same coincident joint location it shared with its parent. In the second step, a corrective

translation is calculated as the displacement of the segment's local origin caused by the ancestor's transformation matrix.

In Figure 6, the scapula, which is the parent of the humerus, is itself a child of the clavicle. On the left, the clavicle rotates from its reference position (light gray), to a new configuration (dark gray). The first step of applying the inverse transformations of the scapula and clavicle to the humerus results in the configuration shown in the center image. The humerus is now disjointed from the scapula. In the second step, we apply the corrective translation (shown by the black arrow in the right image) to the humerus to reunite it with the scapula.

Joint cones

Joint limit cones (Wilhelms and van Gelder, 2001) can provide a better mechanism for limiting the range of ball-and-socket joints than pairs of angle bounds for each joint DOF. In a joint cone, the joint cone is defined using a reference point ***p*** and a space curve ***C*** (Figure 7). The reference point ***p*** is the apex of the cone and is located at the joint center. The curve ***C*** creates a boundary at the bottom of the cone and is defined by a list of user-selected points. An additional vector $\boldsymbol{v}_{rest}$ is defined and positioned at ***p*** so that it lies in the same direction as the bone's longitudinal axis at its rest configuration. The cone provides a way of bounding the movements of two DOFs of a joint simultaneously, such as abduction/adduction and flexion/extension in the humerus at the shoulder. This circular movement of the arm about the shoulder traces the space curve ***C*** and is called *circumduction*. To limit the third twist DOF, an additional pair of angle bounds is associated with each point on the curve ***C*** and the point $\boldsymbol{v}_{rest}$. To get a pair of twist limits for any given configuration within the cone, the twist limits are interpolated from the limit pairs stored at the nearest points.

Joint cone components are used to check for legal joint configurations and project illegal orientations back to the boundary curve. A joint cone component is always attached to one specific joint, say joint *j*. It takes as input the transformation matrix $\mathbf{T}_{input}$ of *j*, which is produced by rotation components. $\mathbf{T}_{input}$ transforms the vector $\boldsymbol{v}_{rest}$ to $\boldsymbol{v}_{input}$ to test if the bone's orientation is within the joint cone. If it is, $\mathbf{T}_{input}$ is passed out of the joint cone component. Otherwise, a new transformation matrix $\mathbf{T}_{adjusted}$ is computed by using $\mathbf{T}_{input}$ so that $\boldsymbol{v}_{input}$ is transformed to $\boldsymbol{v}_{adjusted}$ (refer to Figure 7). The line segment connecting $\boldsymbol{v}_{input}$ and $\boldsymbol{v}_{rest}$ must intersect the boundary of the cone at $\boldsymbol{v}_{adjusted}$. Using $\boldsymbol{v}_{adjusted}$, two rotation angles are newly calculated to produce the adjusted rotation matrix $\mathbf{T}_{adjusted}$

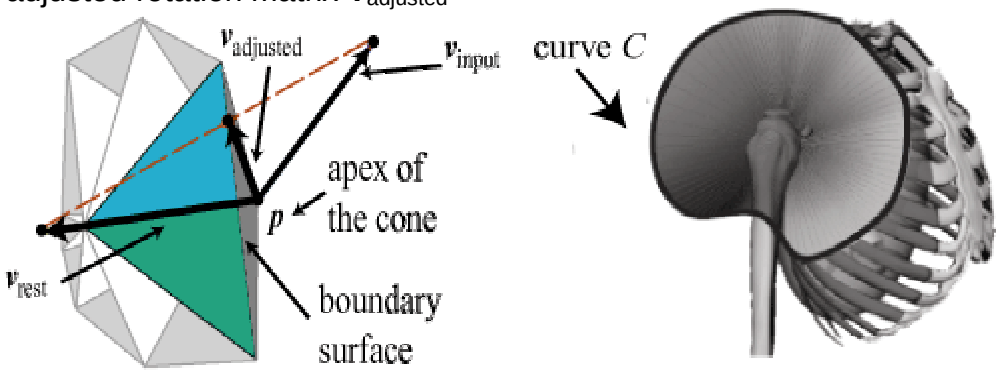

Figure 7: Parts of the joint cone and location at the shoulder to define arm circumduction.

Revisiting Rotations

Most three DOF joint rotations are performed as a series of three consecutive rotations about the orthogonal x, y and z axes. For realistic joints, we would like to accommodate non-orthogonal, arbitrary axes of rotation with changing joint centers. Joints with multiple DOF rotations are created by combining rotation components, each of which produces a rotation matrix for a single axis rotation. An important simplifying assumption being made is that the joint centers for each axis rotation is independent of the rotations about the other axes. While this may not be the case in reality, we can get fairly reasonable-looking articulations. For each rotation component, the following parameters can be provided:

1) A list of *n* consecutive angle intervals: $[a_0, a_1),[a_1, a_2),\ldots,[a_{n-1}, a_n]$
2) For each angle interval $[a_{i-1}, a_i)$, a rotation center point $\mathbf{c}_i=<c_x,c_y,c_z>$
3) A common rotation axis **x**

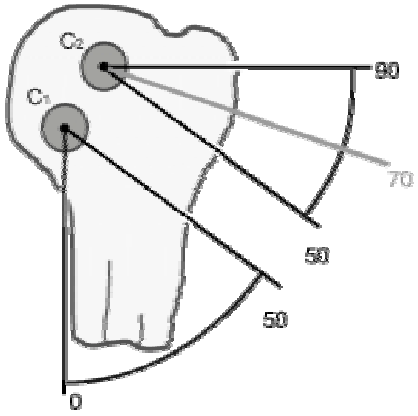

Figure 8: In the humerus, the joint center changes depending on the current rotation angle.

Each angle interval has a different joint rotation center. The ability to model a changing joint rotation center is important to accurately describe rotations in the knee and humerus of the shoulder (Figure 8). This is an efficient way of modeling the effects of surface-surface interactions that would occur in real joint motion without the expense of collision detection. There may be only a single angle interval (and corresponding joint center) defined for a rotation component. With these rotation parameters and an input rotation angle $\alpha \in [a_0, a_n]$, the final output rotation matrix R for the joint component is computed as follows:

$$R = M_n M_{n-1} \ldots M_1,$$

$$M_i = T_i^{-1} Q_i T_i$$

$$T_i = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, T_i^{-1} = \begin{pmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & c_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where R, M_i, T_i, T_i^{-1}, and Q_i are all transformation matrices. Q_i is computed from a quaternion representing the rotation of angle β_i around axis **x**. The angle β_i is determined by

$$\beta_i = \begin{cases} a_i - a_{i-1} & \text{if } \alpha > a_i \\ \alpha - a_{i-1} & \text{if } a_{i-1} < \alpha < a_i \\ 0 & \text{otherwise.} \end{cases}$$

The final rotation of the angle α is the cumulative result of a sequence of quaternion rotations of smaller angles β_i. In practice, these quaternions can be implemented using 3-D API calls that specify a rotation angle about a given axis, like glRotate in *OpenGL*. Each of these quaternion rotations has its own center represented in the T_i and T_i^{-1} translation matrices. This implements a changing joint rotation center of a joint. Notice that for a given α that falls into a certain interval $[a_{j-1}, a_j]$, all M_k (0<k<j) will not depend on α since each of their β_k is a constant equal to $a_k - a_{k-1}$. We can precompute and store all the matrices M_k for the full angle interval rotation to reduce computation. Furthermore, the cumulative matrix product R for the j^{th} interval can have the matrix product $M_{j-1}M_{j-2} \ldots M_1$ precomputed and retrieved to update the articulated figure motion at interactive rates.

Sliding Constraints

Sliding constraints are designed to handle specific joint translational motions such as the scapula sliding over the rib cage (scapulo-thoracic joint). The scapula bone is always gliding on a curved surface defined by ribs, muscles and fatty structures. To represent this in our model, we model the rib cage as an ellipsoid. In order to constrain the scapula bone to be gliding on the surface of an ellipsoid, we define pairs of reference points on the scapula, and make sure that at least one active pair stays on the ellipsoid at all times (Figure 9).

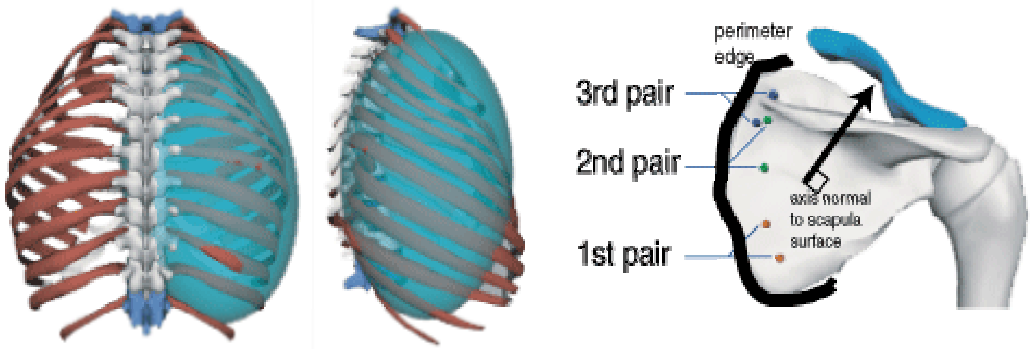

Figure 9: Left) The right side of the rib cage is approximated with an ellipsoid.
Right) Pairs of points on the scapula determine the portion of the scapula's surface that will slide over the rib cage.

Let's see how the DOF are fully determined for the scapula. Think of the scapula as an initially free joint with three DOF for rotation and three DOF for translation. Because it is attached to the parent clavicle bone, the three DOF of translation are determined by its shared attachment point with the clavicle. We further constrain the rotation of the scapula around an axis perpendicular to its outer surface to be dependent on the abduction of the humerus bone using

a dependency mapping, removing another DOF. Keeping the scapula on the ellipsoid determines the remaining two DOFs for rotation. By constraining a pair of reference points on the scapula to the ellipsoid's surface, the configuration of scapula bone is fully determined.

Having several pairs of reference points allows the contact area between the scapula and rib cage to change depending on the configuration of other joints in the shoulder. Referring to Figure 9, right side, the area closest to the 1st pair is more likely to be in contact with the rib cage when the shoulder is lifted. The 2nd pair is more likely to be in contact when the shoulder is lowered. The 3rd pair is active when the scapula is fully rotated clockwise around the axis normal to its surface. Therefore, we can use these three pairs of reference points to find a new contact pair by interpolating over two DOFs corresponding to the amount of shoulder lift and rotation about the scapula.

The scapula will then be rotated twice to constrain these contact points on the surface of the ellipsoid. In the first rotation, a predefined vector going through the joint origin is used as the rotation axis. We rotate about this axis to bring the first reference point onto the ellipsoid surface. In the second rotation, the vector connecting the joint origin and the first reference point is used as the rotation axis to bring the second point onto the ellipsoid. Notice that the second rotation will not change the position of the first contact point since it is on the rotation axis. We can easily determine whether a point <x,y,z> is on or inside the ellipsoid by evaluating the point using the ellipsoid's equation in implicit form and checking the condition:

$$E(x,y,z)=\left(\frac{x}{a}\right)^2+\left(\frac{y}{b}\right)^2+\left(\frac{z}{c}\right)^2-1\le 0,$$

where we assume we have transformed the point to the ellipsoid's reference frame so that it is axis-aligned and has its center at the origin. Using this condition, you can use root-finding techniques (like a simple binary search) to find the amount of rotations needed (when *E(x,y,z)* is close to zero within some tolerance).

Putting them all together

These biomechanical joint models can be implemented in your own 3-D graphics code or as a plugin in an existing animation package. We created plugin implementations in *Maya* as well as in custom OpenGL code. Using *Maya* was helpful because we could take advantage of existing user interface tools for specifying joint parameters such as axes of rotation. For portability of these joint models, you can write your own file format. We used an XML-based format to take advantage of the multitude of XML parsing and management software in existence. The various joint maps can be combined with each other to build complex realistic articulations from a set of simpler behaviours. Using this technique, we built a detailed spine and shoulder model seen in Figure 1. To read more details about our particular implementation, please refer to (Shao and Ng-Thow-Hing, 2003).

Applications

Okay, once we have our human or humanoid model rigged up with these enhanced joint behaviors, how can we use them? The direct method is to simply adjust the new DOFs at key

poses in a traditional key-framing animation system. The bone reference frames can be used directly in character skinning as we simply weight the resulting transformation matrices to compute the blended skin positions.

Inverse kinematics and motion capture fitting is a more challenging problem with these biomechanical models because the traditional kinematic joint hierarchy is often assumed. Typically, fitting a skeleton to a set of markers involves constructing reference frames from a set of markers and finding the angles from the relative differences in orientations of the frames. One way is to map these angles directly to the closest corresponding joint map DOFs. This can sometimes produce reasonable results, but animators often need to adjust the angle trajectories to fix the motion. A more accurate alternative would be to do a search on the DOF angles to find the set that minimizes the distance of points on a skeleton to the marker positions. As the marker positions are usually placed on standard body landmarks, the correspondence is known. This can be done using least-squares optimization techniques. Similarly, inverse kinematics can be done with the same optimization procedure. Optimization typically requires that you know the derivatives of points on the skeleton with respect to the DOFs. As these joint maps are deterministic, you can either compute their derivatives directly or numerically using finite difference methods. Further discussion of the use of optimization for finding optimal joint angles under various constraints can be found in (Zhao and Badler, 1994).

Conclusion

Although we presented these ideas in the context of a human skeleton, these same principles can be applied to any creature, whether existing or fictional to improve the joint motions of your characters. The choice of using a particular principle or all of them will depend on the level of visual realism or subtlety that is desired in the animations of your game application. Although the resulting joint behaviour can be quite complex, the number of handles needed to manipulate these realistic joints is still quite small. This dispels the myth that complex motion always requires complex controls.

Acknowledgements

A great amount of debt is owed to Wei Shao, who worked with me as a Honda student intern to implement many of these ideas as Maya plugins. Finally, my 10 month old son, Sky, keeps me in awe of the fascinating flexibility of the human body.

References

Badler, N. Phillips, C. and Webber, B. 1992. *Virtual humans and simulated agents.* Oxford University Press.

Craig, J. 1989. *Introduction to Robotics: Mechanics and Control.* Addison-Wesley.

Delp, S. and Loan, P., 1995. A graphics-based software system to develop and analyze models of musculoskeletal structures. *Comput. Biol. Med.* 25, 1, 21-34.

Garner, B. and Pandy, M. 1999. A kinematic model of the upper limb based on the visible human project (vhp) image dataset. *Computer Methods in Biomechanics and Biomedical Engineering* 2, 107-124.

Girard, M. and Maciejewski, A. 1985. Computational modeling for the computer animation of legged figures. In *Computer Graphics (Siggraph '85 Proceedings), 19, 263-270.*

Grassia, F. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3, 29-48.*

Hodgins, J., O'Brien, J. and Tumblin, J. 1998. Perception of human motion with different geometric models. *IEEE Transactions on Visualization and Computer Graphics, 4, 307-316.*

Human Animation Workshop Group, 1999. H-anim 1.1 specification for a standard humanoid, www.h-anim.org.

Maciel, A., Nedel, L. and Dal Sasso Freitas, C. Anatomy-based joint models for virtual human skeletons. In *Proceedings of Computer Animation 2002*, 220-224.

Monheit, G. and Badler, N. 1991. A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications* 11, 2, 29-38.

Shao, W. and Ng-Thow-Hing, V. 2003. A general joint component framework for realistic articulation in human characters. In *ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics,* 11-18.

Shoemake, K. 1985. Animating rotations with quaternion curves. In *Computer Graphics (Siggraph '85 Proceedings),* 19, 245-254.

Wilhelms, J. and van Gelder, A. 2001. Fast and easy reach-cone joint limits. *Journal of Graphics Tools* 6, 2, 27-41.

Zhao, J. and Badler, N. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. In *ACM Transactions on Graphics*, 13, 4, 313-336.