Modeling Animals with Bones, Muscles, and Skin

Jane Wilhelms

USCS-CRL-95-01
January 24, 1994

Baskin Center for Computer Engineering and Information Sciences
University of California, Santa Cruz, CA 95064

Abstract

A new approach to animal modeling and animation using simulated individual bones and muscles, soft tissues, and skin is described. Bones and muscles (made of combinations of ellipsoids) can be generated automatically from a tree structure and joint geometry or designed interactively. Together with soft tissue, these provide the underlying anatomy. A polygonal skin mesh is automatically generated to form a smooth covering. Muscles stretch across joints, and their orientations, sizes, and shapes change during joint motion. The skin mesh automatically adjusts to changes in position due to the effect of neighboring skin points and anchor points associated with the underlying anatomy. Much of the process is automated; parameters may be be adjusted, and components can be added and removed. Manipulation and animation occur at comfortable interactive speeds on graphics workstations.

Keywords: computer graphics, computer animation, computer modeling, animal and skin modeling.

1 Introduction

Modeling animals has been a surprisingly ignored area of research in computer graphics, given how beautiful and varied they are, and how much they are a part of the natural environment. Most animal modeling has concentrated on modeling humans, especially their faces. While this research is also applicable to modeling the human animal, its main interest is in providing a tool for modeling a variety of animals. (Because it relies upon an endoskeleton with muscles and covered by a flexible skin, it really concentrates on modeling vertebrates.) Because of this interest, the method is designed to be semi-automated, to provide for refinement through interactive design, and to create models that can be used flexibly among different animals types.

The basic premise of the model (a natural one though seemingly never used before for whole animal modeling in computer graphics) is that the animal is a structure of bones, individual muscles, and soft tissues covered by a flexible skin. What is particularly unusual here is that the muscles are anchored to different body segments at *origin* (proximal segment) and *insertion* (distal segment) points, and move and change shape according to the relative positions of these points. The flexible skin, which is created automatically once the bones, muscles, and soft tissues ("stuffing") are designed, is anchored to particular of these tissues and moves naturally according to their motion.

The model could accommodate complex surface or volumetric models for bones, muscles, and stuffing, but, in the interest of speed and ease of use, these are all modeled as ellipsoids. Default bones and muscles for the whole model can be generated from a tree structure and basic joint geometry, and these default components can be interactively added to, removed, or changed in shape and position to create the desired animal. Ellipsoids are actually a rather natural shape for muscles (made of three ellipsoids representing two tendons and a muscle body) and bones (the default is made of two larger end ellipsoids and a thin, long shaft between), and make it possible to model quite complex animals interactively. The skin is a polygonal mesh generated by an isosurface program.

Using this method, interesting animal models can be easily created, either conforming to known patterns or with entirely new and fantastic anatomies. Because of the remarkable similarity in the structure of most vertebrates, a similarity that increases with evolutionary proximity, a detailed model of one animal can be used for a range of individuals and for other species with a moderate amount of modification.

2 Background

Most of the research published on modeling animals has been specifically oriented toward modeling humans, and much of that concentrates on modeling faces. Research of most interest here is that which develops models at least semi-automatically (as opposed to brute-force tweaking of parametric surfaces, or digitizing actual objects) and where the models provide a natural deformation during animation. We are not interested, here, in the interesting and difficult issues of how the motion itself is generated.

Some of the best animals have been created for commercial advertising and movies, and details of the methods used are not published. Two excellent recent examples are the polar bears of Rhythm and Hues [RH94, Car94] and the dinosaurs of Jurassic Park [Stu93, Dun93]. Both were initially digitized from material models.

Two major problems with simulating animal (including human) bodies are simulating the effect of muscle bulges dependent on joint positions, and creating a skin near joints that appears natural for any joint position. Several years ago, Chadwick et al [CHP89] presented a method for constructing deformable animated characters. While no attempt was made to be anatomically correct (indeed, characters were meant to be more of the expressive cartoon type than real animals), their multi-layered construction using some physical simulation produced very expressive characters. Squash and stretch was achieved by using free-form deformations controlled by joint positions. Mark Henne used a pseudo-physical model to cause muscle bulges [Hen90]. Gourret et al [GTT89] used a finite element model to model flesh, and demonstrated deformations due to contact with external objects, specifically the hand grasping an object. Badler and Morris also presented a method for dealing with joint modeling [BM82]. Most skin models deform skin due to the spring-like influence of neighboring points, and the influence of underlying structures. None appear to have used simulated muscles that move between fixed end points on different joints, and change shape like real muscles.

Realistic simulation of humans has long been a central interest of Magnenat-Thalmann and Thalmann and their collaborators [MTT83, MTT87, fCGF91, MTT91b]. A recent paper [MTT91a] describes a method of creating natural looking skin over joints, which they emphasize as an extremely difficult problem to deal with simply and well.

Considerable work has been done on the human face [Par82, Wat87, TF88, Wil90, TW90, TW91, LTW93]. The problems of modeling and animating animals are accentuated in facial modeling because of the great flexibility, nuances of expression, and the many meanings associated with facial expressions. Early geometric models soon became parameterized [Par82]. More recently, physical simulation has become involved. Some of the most realistic models have come from Terzopoulos and Waters and their associates [Wat87, TPBF87, TF88]. A recent paper [LTW93] uses a highly automated physics-based approach where a triangulated facial mesh from scanned data is used to create an underlying fatty-tissue/muscle layer, and then a bone layer, which are connected by springs with different stress-strain characteristics. Choice of muscles to simulate facial expression and the ability of skin to slide across the bony surface makes for considerable realism, though they don't discuss the expense of the model. An earlier version [TW90] provided interactive rates on high-end graphics workstations. In [TW91], they describe an adaptive method for creating the mesh over the face. These facial simulations have used individual simulated muscles to create realistic motion; research described in our paper uses simulated muscles over the whole body.

Though a furry surface or realistic skin was not attempted in this present research, noteworthy fur [Kaj89, RH94] and skin [HK93] have been produced. Our initial skin is generated by an isosurface extraction program using the "marching-cubes" approach [LC87, WVG92].

3 Modeling Approach

Modeling begins with a given tree structure describing segments and their connections. A rest geometry for the structure giving the locations of joints can be specified either interactively or through a file. The user can interactively add, delete, and modify bones, muscles, and stuffing to create the underlying structure. All are created from combinations of ellipsoids. (Stuffing is used to model tissues other than bones and muscles that

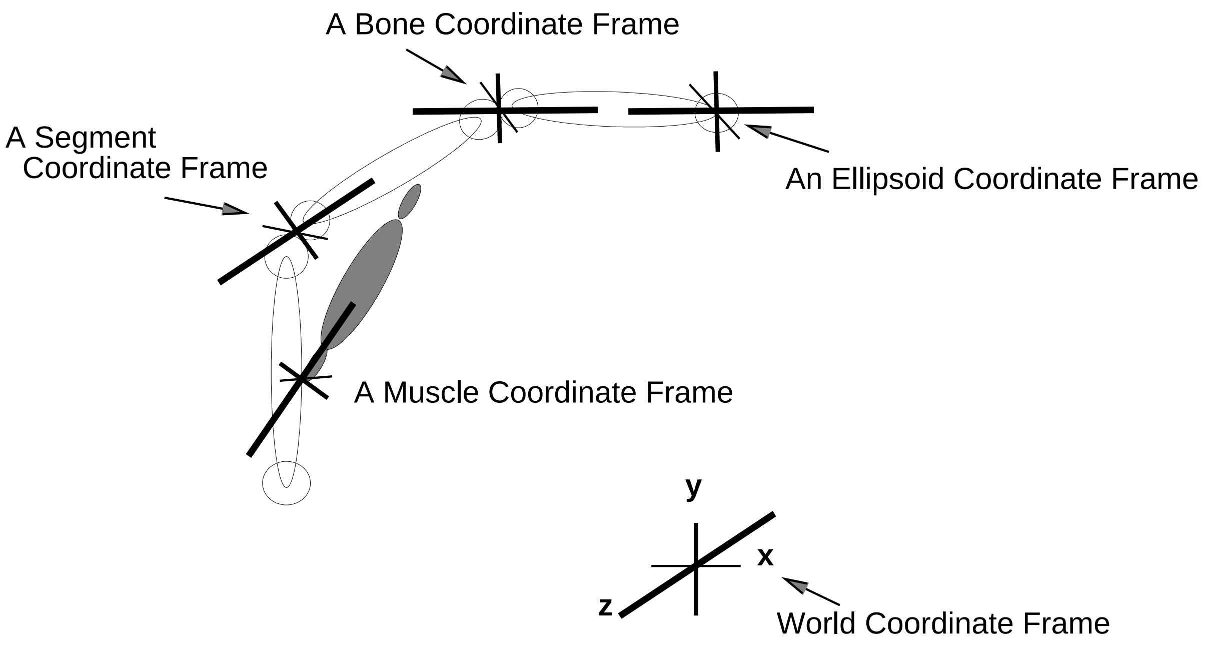
A Bone Coordinate Frame
A Segment Coordinate Frame
An Ellipsoid Coordinate Frame
A Muscle Coordinate Frame
y
x
z
World Coordinate Frame

Figure 1: Coordinate systems used in modeling and animation.

are important to body shape, such as lungs and guts.) Because this can be rather laborious, it is possible to request default bones and muscles to provide a starting structure.

As is typical with hierarchical models, most body parts are defined in local coordinate systems which are transformed to world space coordinates for drawing. Because the use of different coordinate systems is especially important to the method described here, we will point out what the important ones are (see Figure 1). The *world coordinate system* or *frame* is the system into which all parts must be transformed for display. Each segment of the body (e.g., upper arm, lower arm, hand, etc.) is defined in its own local *segment coordinate frame*. Two transformations converting the segment frame to its parent segment coordinate frame are kept with each segment. One transformation defines the default position of the segment to its parent, the other defines the state transformation, due to rotations about that joint from the rest position. Each ellipsoid has its own *ellipsoid coordinate frame* (see Section 3.1). There are also *bone coordinate frames* (see Section 3.2) and *muscle coordinate frames* (see Section 3.3) which define these entities relative to the segment to which they belong.

Once underlying components are created, skin creation is largely automatic. We use an isosurface extraction program to create the skin polygon mesh. A filtering step helps make a smooth skin. Skin points are associated with their nearest ellipsoidal underlying component. The motion of skin during animation is partly due to the influence of this assocation, and partly due to influence of neighbor skin points. This is described in Section 3.4.

Tree, geometry, bones, muscles, stuffing, and skin are all stored in separate files, making it easy to use them for different animals, and to test varying configurations.

3.1 Ellipsoids

The basic primitive is the ellipsoid, specified by the equation $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$. Ellipsoids can be succinctly stored, and solving the equation for a particular 3D point indicates if the point is in, on, or outside the ellipsoid. Ellipsoids are centered at the origin of their own local coordinate frame. Ellipsoid data types specify the major axes (a, b, c), the ellipsoid volume $(\frac{4\pi abc}{3})$, the ratio of the X and Y axis lengths $(\frac{a}{b})$, and a geometric transformation including translation, scale, and rotation that places the ellipsoid in the appropriate coordinate system. (This is the segment coordinate frame for a stuffing ellipsoid, a bone coordinate frame for a bone ellipsoid, or the muscle coordinate frame for a muscle ellipsoid. This is described in more detail below.) The ellipsoid data type also keeps track of the body component and segment to which it belongs, and current matrices transforming points in ellipsoid space to world space and vice versa. These are important because adjusting the skin during motion occurs in both the local and world coordinate systems.

Ellipsoids only need to be drawn if it is desirable to show the body constituents beneath the skin, otherwise their shapes are used to displace the skin vertices. When displayed, non-uniform scaling of primitives from the SGI sphere library is used. Though non-uniform scaling is not recommended with this package, we have found it an efficient and visually acceptable way to image ellipsoids quickly.

3.2 Bones and Stuffing

A bone data type consists of a geometrical transformation which locates the bone in its local bone coordinate system within the segment coordinate system to which the bone belongs, and three ellipsoids. The Z-axis of the bone coordinate system is the longitudinal axis of the bone, and the origin of the bone coordinate system is the *proximal* (nearest the body root) end of the bone. The three ellipsoids are normally (for a simple "dog-bone") lined up in a row along the Z-axis of the bone coordinate frame. Their geometric transformations locate them within the bone coordinate frame.

By default, a newly requested bone lies along the longitudinal axes (the Z-axis) of its segment with a length equal to the longitudinal length of a bounding box around the limb defined by neighboring joints. The center shaft is a long thin ellipsoid whose width is 15% of its length. Two bulbous ends are spherical ellipsoids whose diameter is 30% of the bone length. Certain non-standard bones must be added, and reshaped, such as the pelvis, scapula, and ribs. Bones can be added, repositioned, and the size, shape, and location of the constituent ellipsoids interactively changed. Figure 5 shows the bony skeleton of the test cat ("zuni cat"). Note how the three ellipsoids of the shoulder blade were rearranged to produce a quite un-dog-bone-like shape.

"Stuffing" refers to ellipsoids used to simulate general soft tissue important for shaping the body, such as in the the abdomen and thorax, and is also used to add features such as ears, nose and eyes. The stuffing data type is a single ellipsoid data type. Visually, stuffing appears as purple in the figures. Because stuffing is a simple ellipsoid, the ellipsoid coordinate frame places it in its segment.

3.3 Muscles

Muscles are a more interesting data type. They again consist of three ellipsoids (two tendons and a muscle body between), extending from an origin point on one (the proximal) segment to an insertion point on another *distal* segment (farther from the body root). The local muscle coordinate frame places it relative to the proximal segment. The muscle coordinate frame has its origin at the origin point of the muscle, and the Z-axis of the muscle coordinate frame extends in a direction such that it intersects the insertion point of the muscle. Three ellipsoids are lined up along this Z-axis. When the joint between moves, the orientation of the Z-axis relative to the proximal segment coordinate frame will move also, and the positions and sizes of the three ellipsoids will change so that they just span the distance between origin point and insertion point. (See Figure 2.)

A default muscle stretches from an origin point 10% of the distance from the proximal end of the proximal segment to an insertion point 10% along the longitudinal axis of the insertion segment. When the standard default muscles are requested, four muscles are created for each joint, each with origin and insertion points displaced slightly away from the longitudinal axes of the proximal and distal segments, to simulate an abductor, adductor, flexor, and extensor muscle. The tendons are each 20% of the distance between the origin and insertion points, and the muscle body is 60% of the distance. The width of muscle body and tendons is 40% of their lengths.

To refine a muscle, the origin and insertion points can be interactively repositioned using sliders. A reset function then automatically resizes muscle body and tendon sizes to lie between these two points, using the size relationships stated above. Finally, the user can reshape the ellipsoids defining the muscle body and tendons if the default shapes are not desirable.

The system stores two notions of each muscle: the rest state and the present state. First, muscles are designed and the model stored in the rest state position specified by the rest limb geometry. However, the limbs may be repositioned. The origin and insertion points of the muscle in their local segment frames remain constant, but the muscle between them may need to be reshaped. Also, the new axis of the muscle must be calculated. The adjusted muscle taking into account these changes is necessary for drawing muscles, or for causing changes in the skin.

To find the present muscle configuration, a vector from the origin point to the insertion point, both specified in the origin segment frame, is found. This is easily done by applying the joint transformations between the origin and insertion segment frames to the insertion point, and then subtracting the origin point from the insertion point. This vector originates at the origin point of the muscle, and specifies the Z-axis of the present muscle coordinate frame. It is dependent on changes at the intervening joints, so the length of the new vector (l_{new}) may not be the same length (l_{rest}) as a similarly defined vector in the rest position.

Muscle body and tendon lengths are scaled by the factor relating the new muscle length to the rest length ($\frac{l_{new}}{l_{rest}}$). For the muscle to reshape itself in a fairly natural fashion, the volume of the muscle ellipsoids and their X/Y axis ratios should be constant. New ellipsoid axis lengths are calculated by the following formula (v is ellipsoid volume, and r is $\frac{a}{b}$ from the rest state):

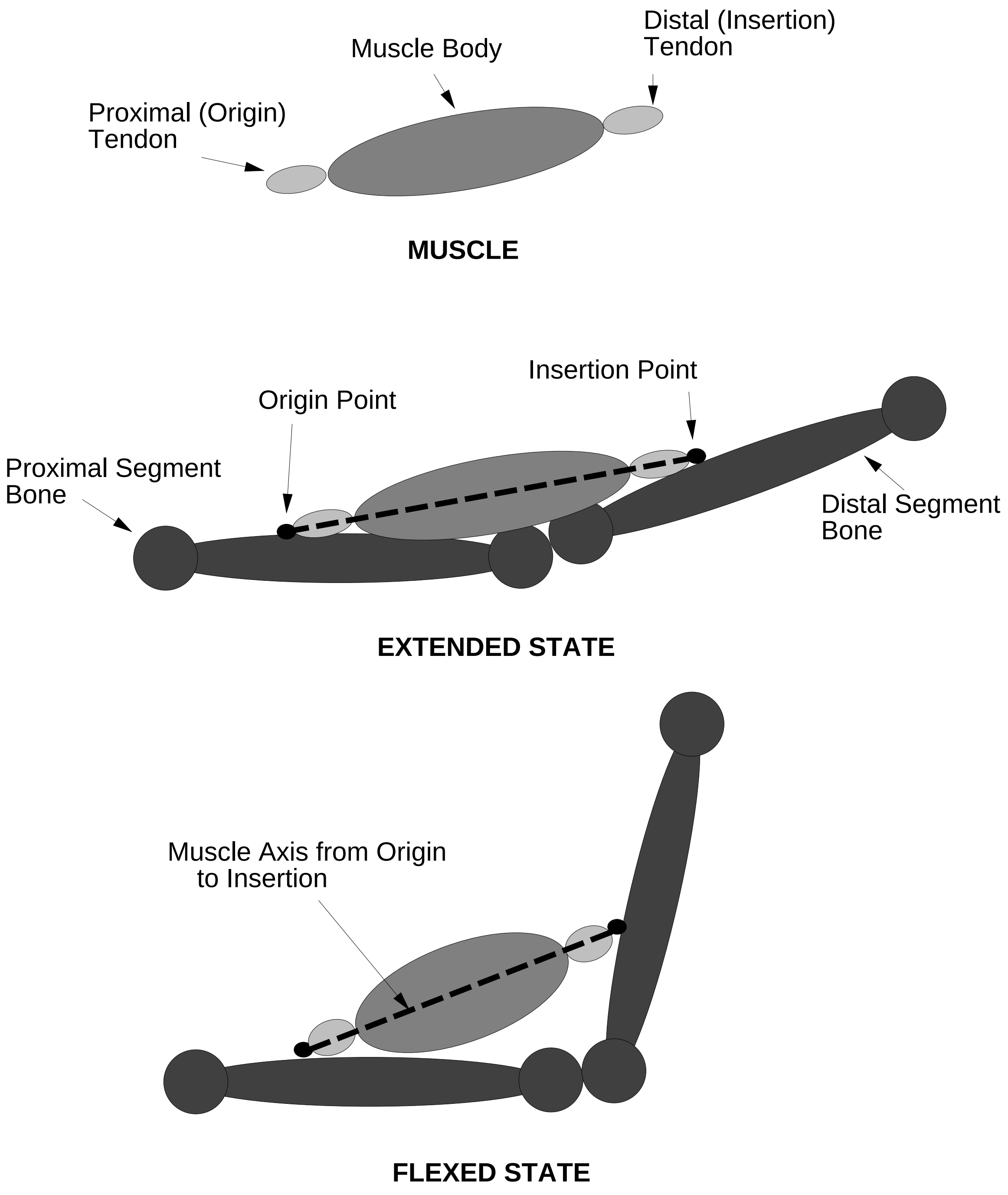

Figure 2: Muscle and bone ellipsoids shown in flexed and extended state with relevant parts labelled.

$$c_{new} = \frac{l_{new}}{l_{rest}} \tag{1}$$

$$b_{new} = \sqrt{\frac{3v}{4c_{new}r\pi}} \tag{2}$$

$$a_{new} = b_{new}r \tag{3}$$

If the new length is shorter than the rest length, this causes the muscle to bulge; and if larger, to become thinner.

Next, a transformation defining the relation of the new muscle axis to the origin segment coordinate frame must found. This can be done using techniques described in standard graphics texts [FDFH90] for rotating a given vector into another arbitrary vector. I.e., find a local coordinate system for which the muscle axis vector is the Z-axis by taking a cross-product of the axis vector with a non-parallel vector, and then taking a cross-product of the axis vector with the new vector. Normalize axes and use them as rows in a rotation matrix (using the SGI vector-matrix convention).

Finally, muscle ellipsoids are placed along this axis at appropriate distances from the muscle origin point to line up the tendons with the muscle body between.

This may sound complex but the calculation need only be done when joints change, and, even then, are quite fast. Muscles can be animated and shown changing shape in close to realtime (see Section 4).

Figure 4 show a structure created almost totally automatically (stuffing was added interactively) from default parameters, showing the muscles (red), tendons and bones (white), stuffing (purple), and a skin mesh covering them.

3.4 Skin

Skin is generated automatically under the influence of user-defined parameters, and skin motion during animation is also automatically generated. As skin behavior is rather more complex than the above constituents, we discuss it in terms of generation, anchoring, modification, and adjustment during motion.

3.5 Skin Generation

A volume of data points on a rectilinear grid of a user-specified resolution is created over the animal model. A world-space bounding box is found for each ellipsoid making up the animal, and used to find which volume data points might possibly be in a particular ellipsoid. The interior data points (lying in the world-space bounding box) are tested by transforming them to the coordinate frame of the particular ellipsoid using the inverse of the matrix that would be used to draw the ellipsoid in world space, and solving the ellipsoid equation using their local position within the ellipsoid frame. If the value is less than or equal to one, that point is in the ellipsoid and is assigned an integer value dependent on the type of ellipsoid. Decreasing values are assigned for bones, muscles, tendons, and stuffing, simulating their relative densities. If a point is not in any ellipsoid, it is given a value of zero, simulating air. This produces a volume not unlike a CT-scan image of the animal.

To produce a smooth skin, this volume is then filtered some number of times (five is often good). The filter used is a Gaussian with a default decay of 2 (which can be changed by the user), affecting 27 points centered at the point being filtered. The decay factors are

$$w_i = \frac{decay^i}{(decay + 2)^3} \tag{4}$$

where the center point is scaled by w_3, 1-adjacent points by w_2, 2-adjacent points by w_1, and 3-adjacent points by w_0. The filtered value actually used is the maximum of the filtered value from this algorithm and the original value, so that the filtering spreads outward from the points included in the animal.

The resulting data volume is sent to an isosurface extraction subroutine using a user-defined threshold to extract a polygonal skin model. Edge lists are created giving neighboring skin points for each skin point, and a rest length for each edge is found. (Skin should always be extracted from the resting position of the animal.)

We used the isosurface approach to generate the skin mesh. One problem with the approach is that polygons are sized by the volume used. It would be better if the generation program produced an adaptive mesh that concentrated skin points in regions of high curvature or great flexibility. On the other hand, the volumetric method allows the skin volume to be filtered before isosurface extraction, which produces a much smoother skin over the sometimes rather bumpy underlying constituents. Perhaps a method of adaptively adjusting the extracted mesh would be a good compromise.

3.6 Anchoring

Once the default skin has been found, it must be anchored to appropriate body parts, and rest lengths of skin points to their anchors on these body parts must be found. This involves converting skin points originally in world space into the frame of the ellipsoids they might be associated with, and selecting the ellipsoid closest to the skin point as the anchor ellipsoid. Figure 3 shows important components in anchoring and moving skin. As time isn't a major consideration with this preliminary step, one could check all skin points against all ellipsoids. In our case, the solution is quite fast because when the volume is originally created, the ellipsoids associated with each volume point are remembered, and this information can be used to limit the search.

While the solution of the ellipsoid equation for a given point indicates if that point is on, in, or outside of the ellipsoid, the value of the equation is the square of the distance from the center, not the distance from the nearest point on the ellipsoid to the point of interest. An iterative Newton-Raphson approach is used to find the near point. Let (x_0, y_0, z_0) be the skin point in the ellipsoid coordinate frame and (a, b, c) be the ellipsoid axis lengths. Given the parameter t, a tolerance tol and initial change of parameter dt are calculated.

$$t = 0 \tag{5}$$
$$tol = abs(0.0000001 * (f(0) - 1)) \tag{6}$$
$$dt = 2 * tol \tag{7}$$

The following iteration occurs until the absolute value of dt is less than the tolerance or ten iterations have occurred.

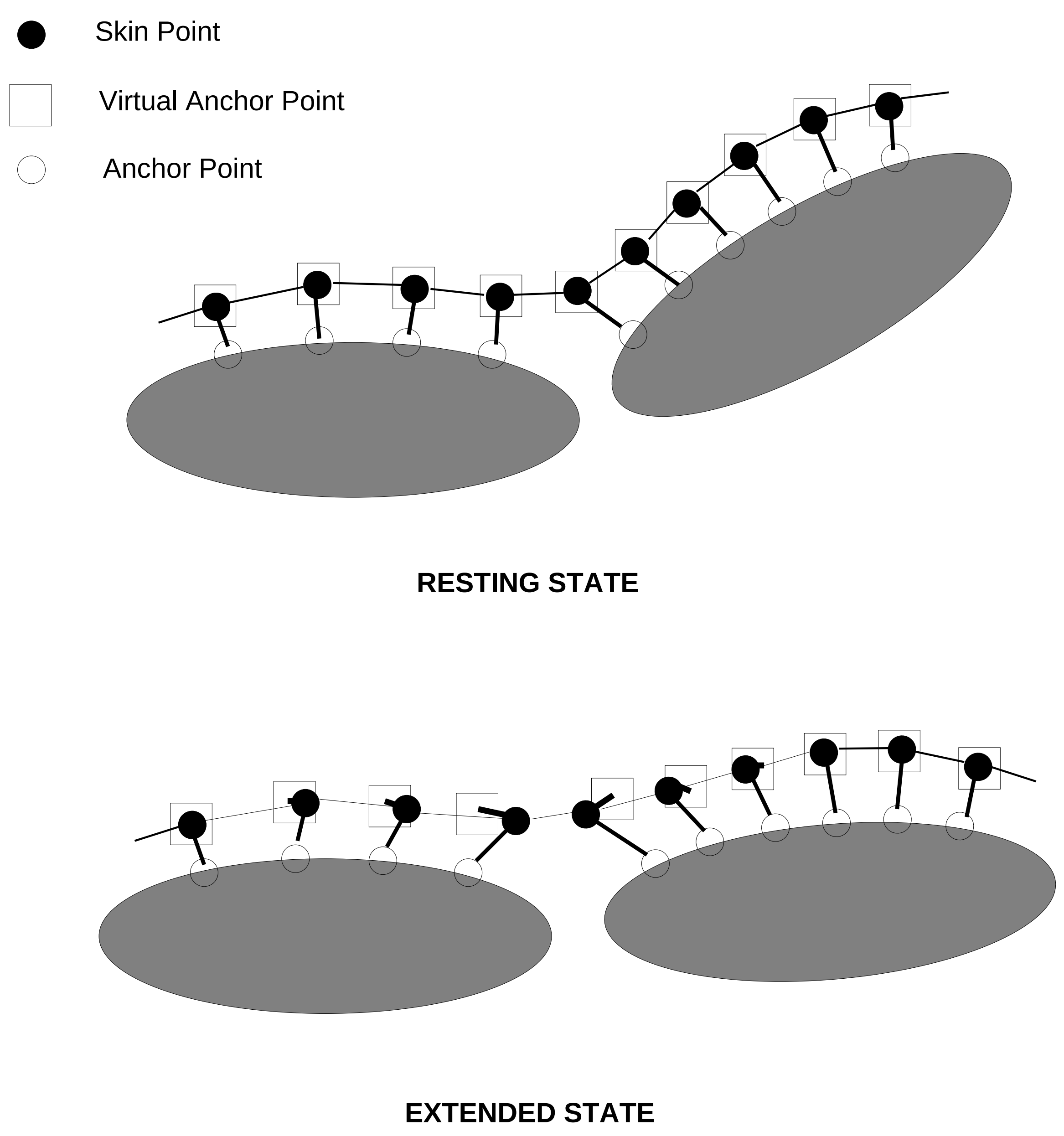

Figure 3: Skin points attached to neighboring skin points, anchor points, and virtual anchor points. Anchor and virtual anchor points stay fixed in their local frames, while skin points move under the influence of virtual anchor points and neighboring skin points.

$$
\begin{aligned}
x &= x_0 * \frac{(a^2)}{(a^2 + 2t)} && (8) \\
y &= y_0 * \frac{(b^2)}{(b^2 + 2t)} && (9)
\end{aligned}
$$

$$\begin{aligned}
z &= z_0 * \frac{(c^2)}{(c^2+2t)} && (10)\\
f(t) &= \frac{a^2x_0^2}{(a^2+2t)^2} + \frac{b^2y_0^2}{(b^2+2t)^2} + \frac{c^2z_0^2}{(c^2+2t)^2} && (11)\\
f'(t) &= \frac{-4a^2x_0^2}{(a^2+2t)^3} + \frac{-4b^2y_0^2}{(b^2+2t)^3} + \frac{-4c^2z_0^2}{(c^2+2t)^3} && (12)\\
dt &= \frac{-1.0*(f(t)-1)}{f'(t)} && (13)\\
t &= t+dt && (14)\\
& && (15)
\end{aligned}$$

The final (x, y, z) is the anchor point and the distance between (x_0, y_0, z_0) and the anchor is the rest length for that skin point for the edge to its anchor point. The anchor position and the skin point are stored in the local ellipsoid frame parameterized by the ellipsoid axis lengths (so as to lie between 0 and 1) by dividing the actual local position by the ellipsoid axis lengths. When the actual location of such a parameterized point must be found in the ellipsoid frame, it is scaled by the ellipsoid axis lengths. As the axis lengths may have changed due to joint changes, these changes are correctly reflected in the new positions of the points in question. This causes skin points and anchor points to move appropriately with changes to muscle shapes.

Occasionally (but unusually), a few skin points are actually inside of an ellipsoid, or closer to the ellipsoid than a minimum allowed distance. When this happens, they are displaced outward along the ellipsoid normal a desirable distance. This can be found using the above iteration and pushing the point outward along the normal to the ellipse at the anchor point by the minimum distance.

Besides the anchor, each skin point also has a *virtual anchor*, which is the default position of the skin point (in the resting position). The virtual anchor is used in skin positioning, and is also parameterized by the ellipsoid axis lengths (see Figure 3.)

3.7 Modification

The skin generated by the above method can be interactively modified by the user. Perhaps the most useful modification is to change the "spring constant" (k) controlling the strength of the pull on a skin point from other points connected to it by edges (see Section 3.8). k can be reset for edges between all skin points or between an interactively picked set of skin points; it can also be set between skin points and their virtual anchors for all skin points, for skin points associated with given segments, or for a selected set of skin points.

Another useful parameter is the length between skin points and their anchors. In some cases, it is desirable for skin to be closer to or farther from their underlying tissues than the default. This length can be reset per segment or for selected points.

Finally, one can reset the actual resting positions of selected skin points. However, in most cases, it is easier and more effective to alter the underlying structure to create the desired effect, rather than tweak points.

3.8 Skin Adjustment During Motion

When joints are moved, skin points should also move taking into account the positions of their virtual anchor points and neighboring points connected to them by edges. In order for this to work well and efficiently, skin points, anchor points, and virtual anchor points are kept both as values in the local ellipsoid frame to which they are related (parameterized by axis lengths), and in world space. Transformation between the two is efficiently done because each ellipsoid stores the matrix transforming its local system to world space, and the inverse of this matrix which transforms points in world space to the local frame.

When a joint is moved, initial positions of each skin point in world space are found by transforming the last position of the skin point in its ellipsoid local coordinate frame to world space using the new relationship of local space to world space. This produces a good approximation of a stable position for the point.

Next, the skin is iteratively adjusted in world space taking into account the influence of neighbor points (which are connected to the point by edges) and the virtual anchor. The number of iterations can be set by the user, and it is possible to iterate continuously in the background when not actively interacting with the program. In practice, however, movement of skin points due to iteration is not visible, even for large motions, after about five iterations, and for small motions, after two or three.

The algorithm for skin adjustment is not using physical simulation, and no integration is used. However, the effect is much the same. Deviations from the rest length of edges connected to each skin point are used for the adjustment. The rest lengths of edges to other skin points was found during extraction of the skin. The rest length of the edge to the virtual anchor is 0. We use the virtual anchor (the rest position of the point) rather than the anchor on the ellipsoid because it gives better visual results. If the ellipsoid anchor is used, the point can rotate around the anchor and find a stable position very close to or even inside the ellipse. While collision detection can take care of this, it is expensive.

The change in position for each skin point is the sum of changes in position caused by each of the edges to which it is attached. Let l_r be the rest length for a particular edge; and let l_p be the present length of that edge. Let P be the 3D vector from the skin point of interest to the point connected to it by this edge, and k be the "spring constant" for that edge. (The default value for k for all edges is 1.) Then, the change in position due to this edge is

$$Dp = k \times \frac{l_p - l_r}{l_p} \tag{16}$$

If the edge in question is between a skin point and its virtual anchor ($l_p = 0$), no displacement is caused, as this is the desired length. In the unusual case where $l_p = 0$ when the edge is between skin points (two points coincide), the displacement is set to be a small amount in a set direction.

For edges between skin points, we find that setting the displacement to zero if the present length is less than the rest length is visually desirable. However, for more finely sampled meshes than we normally use, letting skin points push away from each other could be used to cause wrinkles.

The new position for the point P_{t+1} due to influences of its i edges, each of which has displacement Dp_i is

$$P_{t+1} \quad = \quad P_t + \sum Dp_i \tag{17}$$

Once a new position for a point is determined, the location of that point in the local coordinate frame of its ellipsoid is found and stored.

Collisions between skin points and ellipsoids can be checked for, and points displaced to avoid them when they occur, but they slow the skin adjustment, even when collision detection is only done with ellipsoids on nearby segments. Because virtual anchors tend to keep points from colliding with their own ellipsoids, and the underlying tissues are not normally displayed under an opaque skin, small interpenetrations have little visual effect. We don't do collision detection between widely separated segments or other objects at this time.

4 Experimental Results

The method is demonstrated using a few "animals". The first is a simple armed structure that illustrates what can be achieved using purely default parameters and automatic component generation. The second is the "zuni cat", which is closer to a model that might be used for real animations. The third is a toad, created from the cat description files, to illustrate how the model for one animal can be modified to create a quite different one. (A menagerie is being developed.) Images and animations were done on an SGI Reality Engine with a 150 MHz processor. The FORMS user interface package was used [Ove91]. Animations were created using simple key-framing with a Hermite spline. A facility to save the skin and animate it by simpling redrawing it in a new position (rather than calculating that new position) makes it possible to ensure realtime animation. However, calculating new skin positions and observing the resultant animation is always fast enough to call it "interactive" in that it gives a feel for the motion.

4.1 Two-Armed Structure

The two-armed structure, like all these structures, is defined by a number of (most ascii) files, which are given in a "specs" file used as a command-line argument to the program *fauna*. The tree file indicates that there are seven segments and how they are linked. The geometry file (created interactively) describes how the segments are located relative to each other in their rest configuration. The bones file describes the seven bones created by default. The muscles file describes the twenty-four muscles created by default (four for each joint). The stuffing file describes two ellipsoids added to flesh out the distal appendages. The skin was created from a 60x60x60 resolution volume using seven filter iterations and a filter decay of 2.0. An ascii skin file describes the parameters used in skin generation and gives the name of a binary surface file defining the 2636 vertices and their normals, and the 3175 polygons of the extracted surface.

Skin generation takes about 15 seconds of elapsed time for this low-resolution structure. The skin can be adjusted (taking into account joint changes, reshaping of muscles, and adjustments due to skin edges) and animated at about 8 frames per second (elapsed time) using 1 iteration of skin adjustment, or at 3 frames per second using 5 iterations. Visually, no change is detectable after 5 iterations for rather gross changes in position, or after 2 or 3 iterations for small changes in position, and, actually, differences are subtle after the first iteration. If collision detection with nearby ellipsoids is used, animation with 1 adjustment iteration occurs at about 3 frames per second. If the skin for each frame is precalculated and stored, it can easily be redrawn in realtime. Ellipsoid repositioning and reshaping (for muscles) is quite fast, and the structure can be animated showing bones, stuffing, and muscles (without skin) at 20 frames per second. Adjusting the skin is the major expense during animation.

Figure 4 shows this structure in a non-resting position, with bones, muscles, stuffing, and skin mesh on the left, and the texture-mapped skin polygons on the right. The skin of the figure can be adjusted and animated in close to realtime, as shown in the animation.

4.2 Zuni Cat

The second model is a simulated cat consisting of 27 segments, 39 bones, 123 muscles, and 13 stuffings. The sample volume was of resolution 83x83x83 and it produced a skin mesh of 5518 vertices and 6787 polygons using five filter iterations. The mesh was used as automatically generated, except that two points at the tips of the ears were displaced upwards to produce pointed ears, and the points around the head were drawn closer to the skull ellipsoid by reducing their anchor lengths. Also, points on the trunk of the body and upper limbs were given very loose anchor connections (small values of k to virtual anchors) to simulate the extremely baggy nature of cat skin in those regions (see Figure 8).

Generation of the volume, filtering, and skin extraction took about a minute of elapsed time for this structure. Adjustment and animation of the skin can be done at 3.3 frames per second using 1 adjustment iteration, 1.5 frames per second using 5 adjustment iterations, or about 0.75 frames per second using 1 adjustment iteration with collision detection and response. Actually, though, visual differences between images using these different parameter settings are subtle.

Figure 5 shows the 39 cat bones. Figure 6 shows the bent left leg of the cat. At left are shown bones, muscles, stuffing and skin mesh, with blue vectors showing connections from skin points to anchor points on ellipsoids, and red vectors showing displacements of skin points from virtual anchor points. At right is shown the texture-mapped polygonal skin in this position. Figure 7 shows the cat in a running posture, again with the underlying components at left and the skin at right. Figure 8 shows three cats demonstrating their flexibility. The videotape shows the zuni cat's flexibility during animation.

4.3 Toad

The toad model was created from the cat model files in about an hour. The main constituents are present in both (ears and tail were removed), but their sizes were changed, and the default geometry was reshaped. The skin shown was generated automatically and contains 8441 vertices and 6752 polygons. Figure 9 shows the bones, muscles, and stuffing at left, and the texture-mapped skin at right.

5 Future Work

A more realistic base animal is desirable. A truly flexible animal model should have most of the joints of the vertebrate body. (There are 206 bones in the human body, but many are fused or have little motion, such as the skull, wrist, and foot.) A model that provides a fairly smooth surface for skin application and natural reshaping should have most of the major muscle groups, ignoring, perhaps, the deep ones. Though it will be some hours of work to interactively create such an animal, even with the help of automated default bones and muscles, it should provide a basic model from which other species and individuals can be created fairly easily.

Even better, if one knew the origins and insertions of major muscles, it would be possible to create them automatically, rather than in the rather adhoc fashion now used. Though a more complex model would slow down the program some, work on our present models suggests models with two or three times more components would still be interactively pleasant to work with. Skin is the major bottleneck, not underlying anatomy.

More complex primitives than ellipsoids could be used. An interesting direction would be to extract bones and, as far as possible, muscles, from medical data, and model "real" animals.

At present, the model is kinematic, and joint position changes cause the muscle changes, while, in reality, it is muscle changes that move joints. It would be interesting to explore a physical simulation model based on muscle contraction using this model.

6 Conclusions

Use of approximate anatomical understructures to guide the positioning of skin for animal modeling and animation is a natural, realistic and reasonably efficient method of producing animal models. Ellipsoids are excellent primitives for moderately realistic models, because they are succinct, provide collision detection, and can be made approximately the right shape. Skin can be generated and moved largely automatically in an acceptable period of time, and provides realistic-looking deformations.

Acknowledgments

List processing software by Yumi Tsuji and Allen Van Gelder was used in this software package. Allen Van Gelder contributed many helpful suggestions concerning filtering, skin motion, and texture-mapping. Brad Smith went beyond the call of duty to keep the machine running. This research was supported in part by an NSF grant CCR-8958590.

References

[BM82] Norman I. Badler and M. A. Morris. Modelling flexible articulated objects. In *Computer Graphics 82, Proceedings of the Online Conference*, pages 305–14, Northwood Hills, UK, October 1982. Online Conferences.

[Car94] Phil Carpenter. Commercial spot cola bears. *Cinefex Magazine*, December 1994.

[CHP89] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. *Proceddings of Graphics Interface '89*, 23(3):243–252, 1989 August, 1989.

[Dun93] Jody Duncan. The beauty in the beast. *Cinefex Magazine*, (55), August 1993.

[fCGF91] Creating Realistic Three-Dimensional Human Shape Characters for Computer-Generated Films. Techniques for realistic facial modeling and animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '91*. Springer-Verlag, Tokyo, 1991.

[FDFH90] James D. Foley, Andies Van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Company, Reading, Mass., 2 edition, 1990.

[GTT89] Jean-Paul Gourret, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. *Computer Graphics (ACM Siggraph Proceedings)*, 23(3):21–30, July 1989.

[Hen90] Mark Henne. A constraint-based skin model for human figure animation. Master's thesis, University of California, Santa Cruz, Santa Cruz, CA 95064, June 1990.

[HK93] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 27:165–174, August 1993.

[Kaj89] James T. Kajiya. Rendering fur with three dimensional textures. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 23(3):271–280, July 1989.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (ACM Siggraph Proceedings)*, 21(4):163–169, July 1987.

[LTW93] Y. Lee, D. Terzopoulos, and K. Waters. Constructing physics-based facial models of individuals. In *Graphics Interface '93*, pages 1–8, Toronto, ON, May 1993.

[MTT83] Nadia Magnenat-Thalman and Daniel Thalman. The use of 3d high-level graphical types in the mira animation system. *IEEE Computer Graphics and Applications*, 3(9):9–16, December, 1983.

[MTT87] Nadia Magnenat-Thalmann and Daniel Thalmann. The direction of synthetic actors in the film rendez-vous a montreal. *IEEE Computer Graphics and Applications*, 7(12):9–19, December, 1987.

[MTT91a] Nadia Magnenat-Thalmann and Daniel Thalmann. Human body deformations using joint-dependent local operators and finite element theory. In N. Badler, B. Barsky, and D. Zeltzer, editors, *Making Them Move*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[MTT91b] Nadia Magnenat-Thalmann and Daniel Thalmann. Complex models for animating synthetic actors. *IEEE Computer Graphics and Applications*, September, 1991.

[Ove91] Mark H. Overmars. Forms library: A graphical user interface toolkit for silicon graphics workstations. December 1991.

[Par82] Frederic Parke. Parameterized models for facial animation. *IEEE Computer Graphics and Applications*, 2(9):61–68, November, 1982.

[RH94] Rhythm and Hues. Aurora borealis, skate, and christmas twins, 1993-1994. (Coca Cola Polar Bears).

[Stu93] Universal City Studios. Jurassic park. Movie, 1993.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH '88 Conference Proceedings*, 22(4):269–278, August, 1988.

[TPBF87] Demetri Terzopoulos, John Platt, Alan H. Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH '87 Conference Proceedings*, July, 1987.

[TW90] D. Terzopoulos and K. Waters. Physically-based facial modelling, analysis, and animation. *The Journal of Visualization and Computer Animation*, 1(2):73–80, 1990.

[TW91] Demetri Terzopoulos and Keith Waters. Techniques for realistic facial modeling and animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '91*, pages 59–74. Springer-Verlag, Tokyo, 1991.

[Wat87] Keith Waters. A muscle model for animating three-dimensional facial expression. *SIGGRAPH '87 Conference Proceedings*, 21(4):17–24, July, 1987.

[Wil90] Lance Williams. Performance-driven facial animation. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 24(4):235–242, August 1990.

[WVG92] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992. Extended abstract in ACM Computer Graphics 24(5) 57–62; also UCSC technical report UCSC-CRL-90-28.

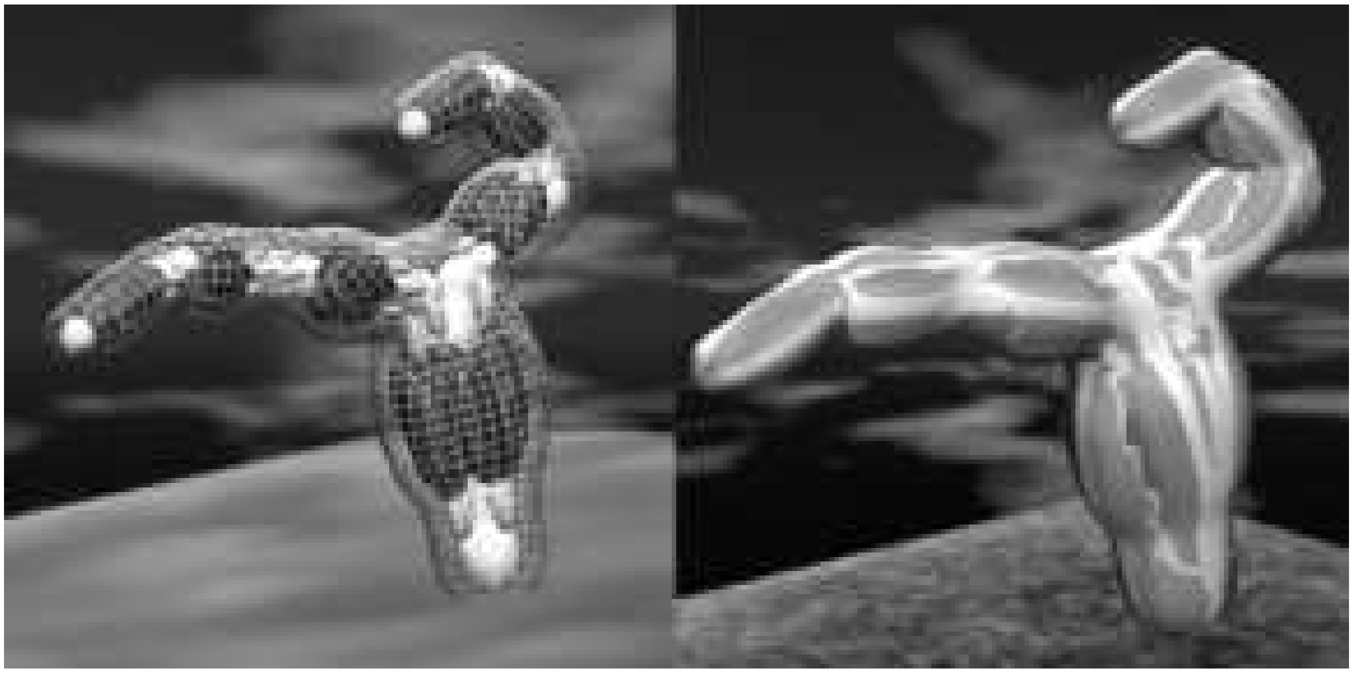

Figure 4: Two-Armed Structure in a non-resting position. At left is shown bones, muscles, stuffing, and skin mesh. At right the texture-mapped skin.

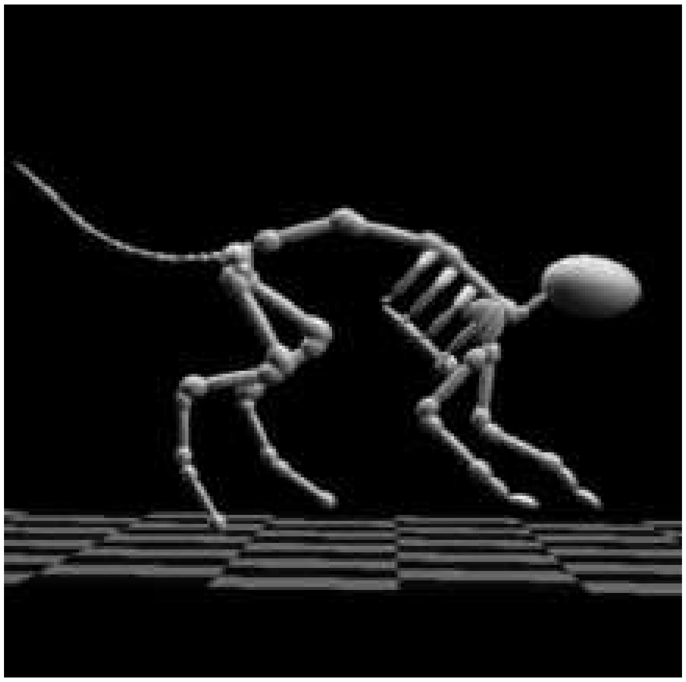

Figure 5: The 39 Bones of the Zuni Cat (non-resting position).

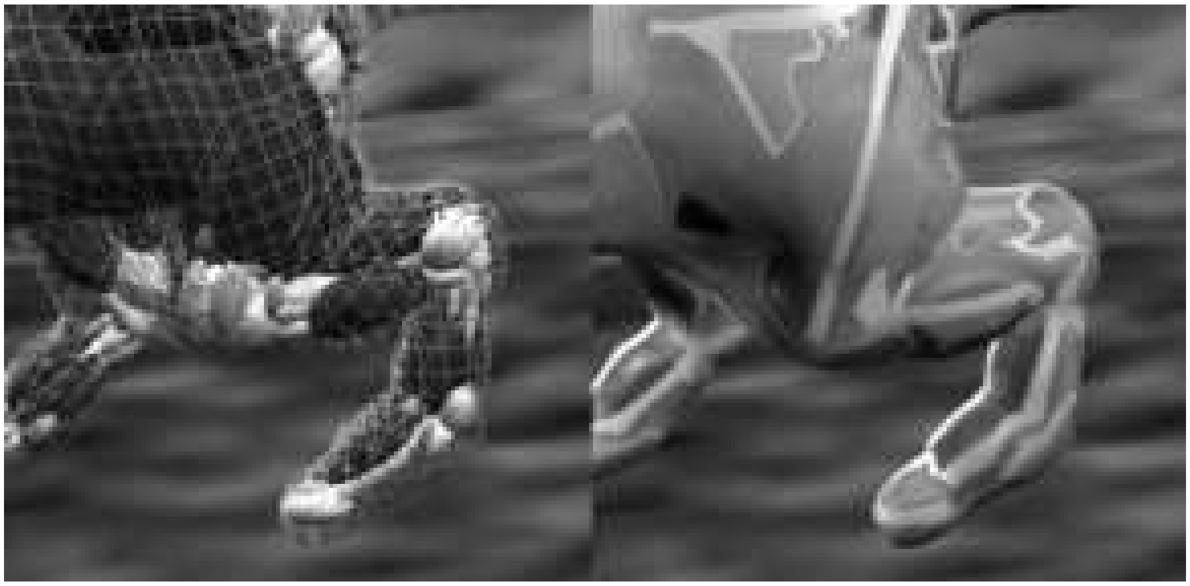

Figure 6: Bent Left Leg showing bones, muscles, stuffing and skin mesh at left, and texture-mapped skin at right. In the image at left, blue vectors show connections from skin points to anchor points on ellipsoids, and red vectors show displacements of skin points from virtual anchor points.

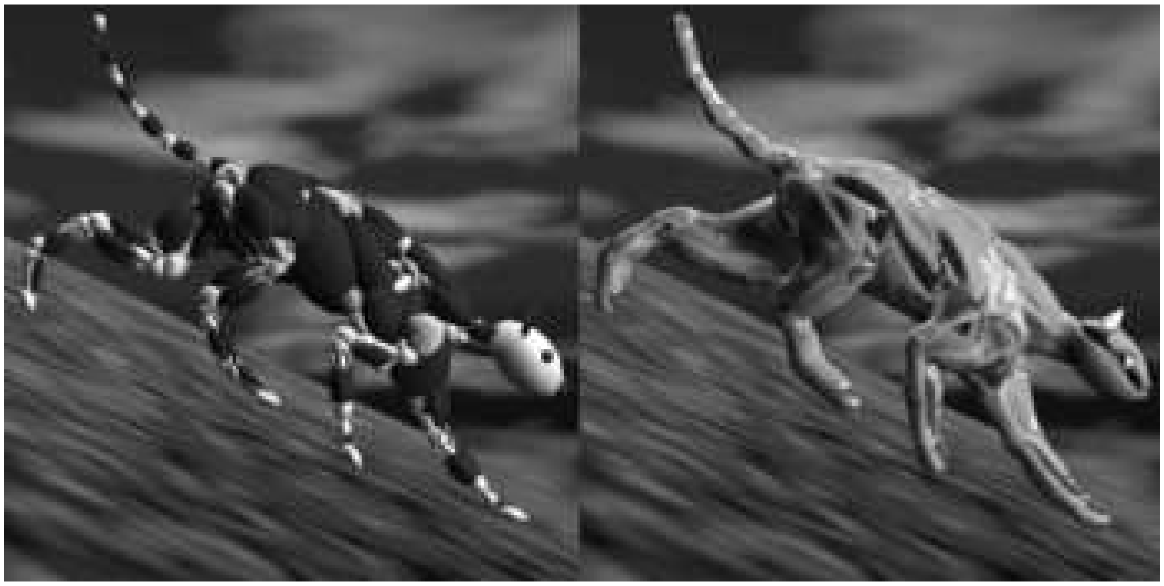

Figure 7: Zuni Cat running, showing bones, muscles, and stuffing at left and texture-mapped polygonal skin at right.

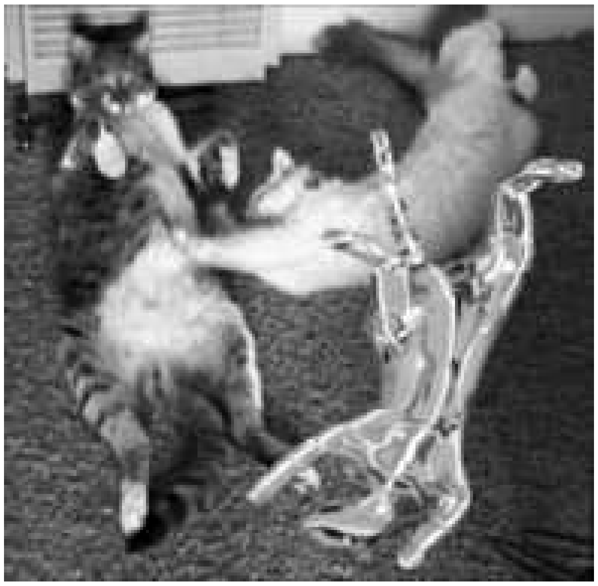

Figure 8: Three Cats demonstrating flexibility.

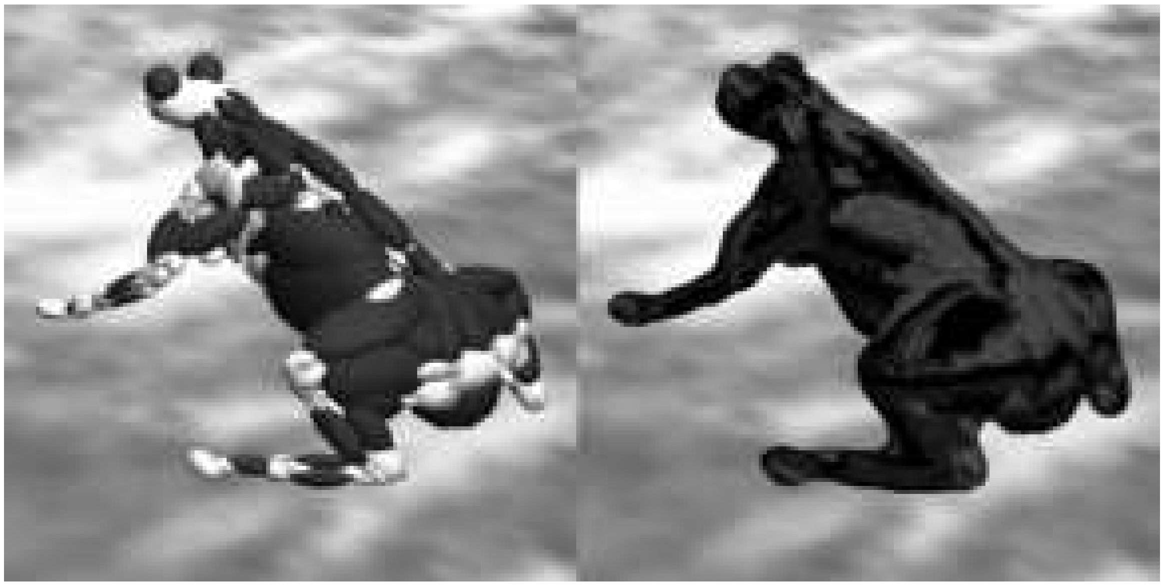

Figure 9: Toad with muscles, bones, and stuffing shown at left and texture-mapped polygonal skin at right.