Eurographics Symposium on Geometry Processing (2007)
Alexander Belyaev, Michael Garland (Editors)

Example-Based Skeleton Extraction

S. Schaefer and C. Yuksel

Texas A&M University

Abstract

We present a method for extracting a hierarchical, rigid skeleton from a set of example poses. We then use this skeleton to not only reproduce the example poses, but create new deformations in the same style as the examples. Since rigid skeletons are used by most 3D modeling software, this skeleton and the corresponding vertex weights can be inserted directly into existing production pipelines. To create the skeleton, we first estimate the rigid transformations of the bones using a fast, face clustering approach. We present an efficient method for clustering by providing a Rigid Error Function that finds the best rigid transformation from a set of points in a robust, space efficient manner and supports fast clustering operations. Next, we solve for the vertex weights and enforce locality in the resulting weight distributions. Finally, we use these weights to determine the connectivity and joint locations of the skeleton.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

In Computer Animation, deformation plays a key role in posing digital characters. In order to manipulate the character, the artist uses some set of deformation handles to deform the character into various poses. In recent years a variety of methods have been developed to assist the artist in this endeavor and allow shapes to be manipulated with grids [SP86], polygons [JSW05] and even points by attempting to preserve the shape's intrinsic characteristics [YZX*04, LSLCO05, BPGK06].

Perhaps the most common and intuitive method for deforming these shapes is skeletal deformation [LCF00]. Skeletons form a compact representation of shape and mimic the way real-world skeletons create deformations of our own bodies. To deform a shape (denoted the *rest pose*) with skeletal deformation, the user specifies a hierarchical set of transformations representing the bones of the skeleton. In a *rigid skeleton*, the transformations consists only of translation and rotation. Therefore, the skeleton can be represented as a hierarchical set of joint locations, each location representing a translation through movement of that joint and orientation through a rotation about that joint's position.

Once a skeleton is specified, we "skin" the vertices of the rest pose by representing each vertex as a weighted combination of each bone. Typically we enforce that $\sum_i \alpha_j = 1$ to make the deformation invariant under rigid transformations where α_j is the weight associated with the j^{th} bone. Now, given these weights, the deformed location $\hat{p}$ of a point p is

$$\hat{p} = \sum_j \alpha_j (R_j p + T_j) \tag{1}$$

where R_j, T_j represent the rotation and translation for the j^{th} bone and p, $\hat{p}$ are column vectors. Today, skeletal deformation enjoys widespread support in Graphics, movies and games. Skeletal deformation is also found in most 3D modeling packages and is even hardware accelerated on today's GPU's.

However, creating the hierarchy of bones and weights for skeletal deformation is a nontrivial task. Ideally, we could automatically construct the hierarchy, bones and weights directly from a set of examples. Specifically, given a set of poses of a character (see Figure 1 left) with the same connectivity, we would like to use these examples to infer how the character moves and estimate the parameters for rigid skeletal deformation to not only reproduce those poses, but create new poses in the same style as the examples. We can then approximate deformations that are the product of possibly unknown deformation methods or expensive, nonlinear computation using this skeletal system in real-time. Further-

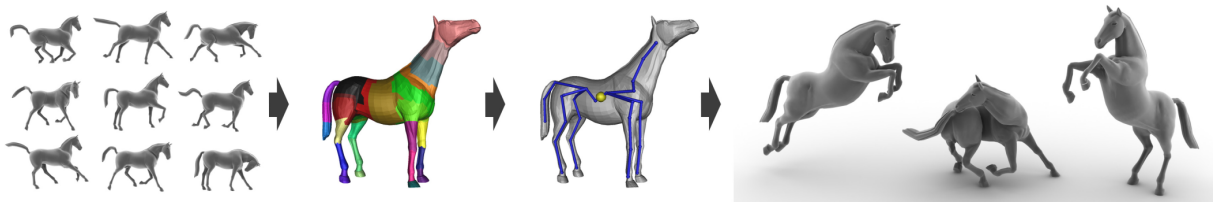

Figure 1: *From a set of example poses we extract clusters representing rigid bones of a skeleton. Next we "skin" the model by estimating bone weights for each vertex that we then use to determine the connectivity of the skeleton and joint locations. With this skeleton, we can create new poses outside of the example set including making the horse jump, chase his tail and rear in the air.*

more, since rigid skeletons are supported by most modeling applications, we can manipulate these shapes using programs such as Maya without the need for special software or plugins and fit these new deformation models directly into existing production pipelines.

2. Previous Work

Several authors have developed methods for extracting skeletons directly from a static pose of a model using segmentation approaches. [KT03] use a fuzzy clustering approach to develop a hierarchical decomposition of a shape, which a skeleton can then be extracted from. [dATM*04] and [TdAM*04] both use voxel descriptions of an object and fit either ellipsoids or superquadrics to estimate a skeleton. Later [KLT05] developed a different segmentation approach based on feature point extraction. [LKA06] also created a fast decomposition method based on approximate convex decompositions of shape for skeleton extraction as well.

Other methods estimate skeletons from a static pose using other data such as feature points [TVD06], various types of distance functions [LWM*03, MWO03], gaussian curvature [MP02] or even probability distributions [FS06]. However, these methods are based on a single pose and rarely discuss skinning because these weights are difficult to obtain from a static pose and have little to do with how the shape actually moves. Furthermore, these techniques may segment or extract regions of a shape that are in fact static, like the tusks of an elephant, because they are based solely on the geometry of the shape.

The complement to skeletal extraction is skinning. Given a set of example poses and skeletons associated with each pose, [WP02] introduce Multi-Weight Enveloping to solve for vertex weights. However, instead of using a single weight associated with a transformation, the authors allow multiple weights, one for each entry in the transformation matrix. The result is a much better fit to the data and smoother deformations though few modeling packages currently support this technique.

[MG03] also take an example-based approach to skinning. Again the input is a set of deformed models, each with a deformed skeleton associated with them. The authors insert various bones to alleviate traditional problems with skeletal animation and determine influence sets using a point clustering method to solve for vertex weights as well. However both of these skinning techniques assume a skeleton and model are provided for all example poses a priori.

Recently, several example-based deformation techniques have been developed. [JT05] developed a technique for compressing the storage space needed for an animation or set of poses. The authors accomplish this task by utilizing mean-shift clustering to robustly determine a set of representative transformations associated with the animation. Then the authors use non-negative least squares to fit the vertex weights to the transformations and avoid over-fitting in the solution. Despite the similarity of this technique to skeletal animation, the method is designed to reproduce and compress an animation rather than create new deformations. In particular, the authors do not build a skeleton to manipulate the shape and utilize affine transformations (uncommon in skeletal deformation) to fit the data better.

[SZGP05] also developed an example-based deformation method using deformation gradients and nonlinear minimization. Their technique can create very realistic looking deformations, but the nonlinear minimization is quite slow and limits interactive applications. [DSP06] later improved the speed of this method by computing a set of representative vertices for use in the minimization, which speeds up the computation and results in interactive posing.

A few researchers have worked on extracting skeletal models from various sources including motion capture markers [KOF05], 3D range data [AKP*04], and even CT scans of the human hand [KM04]. [AKP*04] extract a skeleton from 3D range data and clusters rigid components on the surface using an an algorithm that iterates between assigning vertices to clusters through an integer programming problem and estimation of rigid transformations through an Iterated Closest Point algorithm. [KOF05] attempts to extract a rigid skeleton from the positions of a set of markers from motion capture data. Their algorithm determines rigid parts by clustering vertices based on the standard deviation of the

pair-wise distance between all markers. In our context, these markers correspond to vertices on the surface. While motion capture data inherently uses a small number of markers, our surfaces may have 10 to 100 thousand vertices and computing the pair-wise distance between all vertices for clustering rigid components is not feasible.

[KM04] extract an articulated skeleton from volumetric CT data of a hand in various poses as well as providing a deformation method based on pose space deformations [LCF00] to capture the subtle skin movement as the user manipulates the shape. In contrast to previous techniques, the authors do not need to find bone transformations through clustering because the CT data contains the real-world bones inside of the hand. In practice, the volumetric, skeletal structure is rarely available.

Contributions

We present a method for estimating the complete set of parameters for skeletal animation including the rigid bone transformations, skeletal hierarchy, root node, joint locations and vertex weights from a set of example poses. We can then use this extracted skeleton to create new poses of the character in a similar fashion to the examples. Since, rigid skeletons are supported by most 3D modeling packages, these deformable models fit directly into existing art pipelines. First, we provide what we call "Rigid Error Functions" that allow us to efficiently find the best rigid transformation and the error of that fit for a set of points using a constant amount of space. We then use these error functions to estimate the transformations of the bones in the example poses with a fast, face clustering approach. Given these bones, we "skin" the mesh by solving for vertex weights using a constrained optimization and bone influence maps. With these weights, we determine both the connectivity of the skeleton and the joint locations. Finally, we provide several example models and drastic, new deformations outside of the example set to demonstrate the robustness of our method.

3. Bone Estimation

Given a rest pose as well as a set of example poses, our goal is to estimate a set of bone transformations that govern the motion of the examples. There are several approaches we could take to solve this problem. [JT05] present a robust face clustering method based on mean-shift clustering to extract these transformations. The advantage of this approach is that the clustering can robustly remove outliers in the data.

However, the disadvantage is that the method is based on clustering points in a high dimensional space representing concatenated rotation matrices. If we are given n examples poses, the points lie in a $9n$-dimensional space (or $12n$-dimensional space if translation is used) and nearest neighbor queries become expensive. Furthermore, the clustering does not respect the topology of the mesh and can cluster many disconnected triangles together. For compressing mesh animations, these disconnected triangles are not problematic since data reduction is the only goal. However, in skeletal animation, bones are typically associated with compact sets of triangles on the mesh. Therefore, we use a different approach to determine the bone transformations.

Our bone estimation technique is inspired by work in surface simplification. Algorithms such as QSlim [GH97] use an edge-collapse approach to cluster vertices on the surface. Each vertex has an associated Quadratic Error Function whose minimizer determines the location and error of that vertex. The cost of clustering two vertices connected by an edge is then the error of the function that results from summing the two vertex functions. These vertices are clustered together in a greedy manner until the desired number of vertices (or polygons) is reached.

To find bone transformations, we will find clusters of faces on the surface that transform in a similar manner by modifying the standard surface simplification algorithm. The rigid transformations that best describes the motion of each cluster will then be the transformations associated with each bone in our skeleton.

3.1. Rigid Error Functions

One key piece of the surface simplification algorithm is the Quadratic Error Function [GH97]. Here we construct its equivalent, the *Rigid Error Function* for rigid transformation simplification. We will assume that we have some set of triangles in parametric form $p_i(t)$ from the rest pose and their deformed images $q_i^k(t)$ from the k^{th} example pose where t is a bivariate parameter. In particular, $p_i(t)$ has the form

$$p_i(t) = t_1 p_{i,1} + t_2 p_{i,2} + (1 - t_1 - t_2) p_{i,3}$$

where $p_{i,j}$ is the j^{th} vertex of the triangle $p_i(t)$.

The best rigid transformation R, T that maps the triangles $p_i(t)$ to their image $q_i^k(t)$ is given by

$$E_k(R,T) = \sum_i \int_t \left| R p_i(t) + T - q_i^k(t) \right|^2 dt. \qquad (2)$$

Now, the best rigid transformation in Equation 2 for the k^{th} example pose is the minimizer of that error function.

$$\min_{R^T R = I, T} E_k(R,T)$$

Due to the nonlinear constraint $R^T R = I$, this transformation cannot be found through linear minimization. However, the problem contains enough structure that a simple solution still exists.

If we differentiate Equation 2 with respect to T and integrate, we find that

$$T = \bar{q}^k - R\bar{p}$$

where $\bar{p}$, $\bar{q}^k$ denotes the area weighted average of the centroids

$$\begin{aligned} \bar{p} &= \frac{\sum_i \Delta_i c(p_i)}{\sum_i \Delta_i} \\ \bar{q}^k &= \frac{\sum_i \Delta_i c(q_i^k)}{\sum_i \Delta_i}, \end{aligned} \tag{3}$$

Δ_i is the area of the triangle $p_i(t)$ and $c(p_i)$ is the centroid of the triangle $p_i(t)$. Substituting this definition into Equation 2, we find that the error now becomes

$$\sum_i \int_t \left| R\hat{p}_i(t) - \hat{q}_i^k(t) \right|^2 dt$$

where $\hat{p}_i(t)$, $\hat{q}_i^k(t)$ are the triangles with vertices $p_{i,j} - \bar{p}$ and $q_{i,j}^k - \bar{q}^k$. Therefore, the best rotation R is given by the polar decomposition of the matrix

$$M = \int_t \hat{p}_i(t)\hat{q}_i^k(t)^T dt. \tag{4}$$

The polar decomposition has been used many times in Graphics to find optimal rotations of a set of points [ACOL00], [MHTG05] and factors a matrix $M = RS$ into an orthogonal matrix R and a skew-symmetric matrix S. Unfortunately, orthogonal does not mean a rotation matrix as reflections lie in the space of orthogonal matrices. Also, the polar decomposition is only defined for matrices of full-rank, meaning that the triangles must span 3-dimensional space (impossible for a single triangle or planar region of the surface). Therefore, the polar decomposition can fail to produce the correct solution to this minimization problem.

Fortunately, [Hor87] provides an alternative solution based on the eigen-structure of a 4×4 symmetric matrix that solves these problems. Given the 3×3 matrix M, we can easily show that Horn's 4×4 matrix can be found directly from M [BM92] and is

$$\begin{pmatrix} Tr(M) & v \\ v^T & M^T + M - Tr(M)I \end{pmatrix}$$

where $Tr(M)$ is the trace of the matrix M, I is the 3×3 identity matrix and

$$v = \begin{pmatrix} M_{2,3} - M_{3,2} & M_{3,1} - M_{1,3} & M_{1,2} - M_{2,1} \end{pmatrix}.$$

The unit eigenvector corresponding to the largest positive eigenvalue denotes the unit quaternion representing the best rotation (negative eigenvalues indicate reflections).

Simplification not only requires a method for finding the best rigid transformation from a set of points, but also efficient methods for storing and combining these Rigid Error Functions. We can do so easily by integrating Equation 4 and substituting Equation 3 to form

$$M = \sum_i \Delta_i \left(\frac{3}{4} c(q_i^k) c(p_i)^T + \frac{1}{12} \sum_{j=1}^{3} q_{i,j}^k p_{i,j}^T \right) - \bar{q}^k \bar{p}^T \sum_i \Delta_i.$$

Assuming that R is the optimal rotation (which can be determined solely from M), the error associated with the minimizer of $E_k(R,T)$ can likewise be rewritten as

$$\begin{aligned} &\Sigma_i \tfrac{\Delta_i}{12} \left(9\left|c(p_i)\right|^2 + 9\left|c(q_i^k)\right|^2 + \Sigma_{j=1}^3 \left|p_{i,j}\right|^2 + \left|q_{i,j}^k\right|^2 \right) \\ &\quad -(\Sigma_i \Delta_i) \left(\left|\bar{p}\right|^2 + \left|\bar{q}^k\right|^2 \right) - 2M \oplus R \end{aligned}$$

where $\oplus$ denotes component-wise multiplication of the two matrices followed by summing those products. Notice that, in order to compute M, we only need the scalar $\sum_i \Delta_i$, the vectors $\sum_i \Delta_i c(p_i)$ and $\sum_i \Delta_i c(q_i^k)$ as well as the matrix $\sum_i \left(\frac{3}{4} c(q_i^k) c(p_i)^T + \frac{1}{12} \Sigma_{j=1}^3 q_{i,j}^k p_{i,j}^T \right)$. Furthermore, to compute the error associated with the minimizer we need the scalar $\sum_i \frac{\Delta_i}{12} \left(9\left|c(p_i)\right|^2 + 9\left|c(q_i^k)\right|^2 + \Sigma_{j=1}^3 \left|p_{i,j}\right|^2 + \left|q_{i,j}^k\right|^2 \right)$. These sums use a constant amount of space (17 floats) and combining two error functions amounts to adding these 17 numbers together, which is extremely fast.

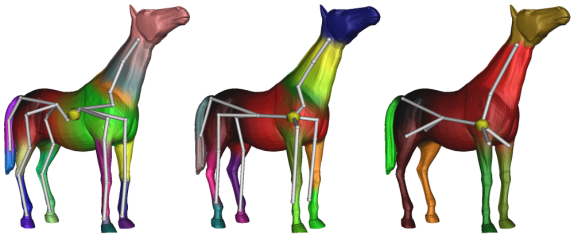

Figure 2: *Horse skeleton with skin weights shown for different numbers of bones. From left to right: 29 bones, 19 bones and 9 bones.*

3.2. Face Clustering

Instead of clustering vertices like surface simplification, we cluster faces. Each non-degenerate face in the rest pose along with its image in an example pose defines a unique, rigid transformation. Therefore, before clustering, we construct a Rigid Error Function for each face in the rest pose and for each example pose. Now each face has n Rigid Error Functions associated with it, one for each of the n example poses. Next, we connect faces together that share an edge on the surface. For each of these adjacent faces, we insert an edge connecting them into a priority queue whose sort key is the error associated with combining those two faces as defined by their Rigid Error Functions summed over all n examples. Finally we remove edges from the queue with the lowest error and cluster faces together until a specific error tolerance or a desired number of clusters is reached. Figure 2 shows the skeletons created by our algorithm for different numbers of bones. In general, fewer bones means less work for the user when creating new deformations, but also greater approximation error.

Notice that this algorithm requires that the mesh is con-

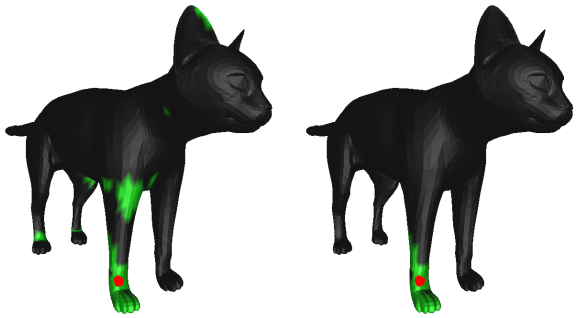

Figure 3: *The influence of the bone associated with the front paw before (left) and after (right) enforcing locality. Without locality the bone affects not only the paw but the rear feet, legs and even the ears all of which will result in undesirable deformations when creating new poses.*

nected. For surfaces made of disconnected pieces, edges between faces from separate pieces may need to be created if clustering across these disconnected pieces is desired. One automatic method for creating these edges is to connect faces along boundaries with other boundary faces based on proximity information.

The implementation of this method is relatively straightforward with one caveat. The Rigid Error Functions in Section 3.1 are somewhat expensive to evaluate as they require n eigenvector computations for the n example poses. Typical surface simplification algorithms perform an edge-collapse, recalculate the error associated with all edges leading to the new cluster and reinsert these edges back into the priority queue after each edge-collapse. However, a single edge-collapse does not just remove one edge, but multiple edges (all adjacent clusters connected to both clusters will have one edge removed to eliminate redundancy after the collapse). Therefore, many of these recalculated edges will never be collapsed but removed by neighboring collapses instead.

By taking advantage of the inherent monotonic property of the Rigid Error Functions (the collapsed error function will have error greater than or equal to the maximum of the two children functions), we can improve the performance of this algorithm greatly. After an edge-collapse, we simply mark the edges connected to this new cluster as dirty instead of recalculating their errors. When we remove the lowest error edge from the priority queue, we check if the edge is dirty. If so, we recalculate the error associated with the edge, mark the edge as clean and reinsert the edge into the priority queue. This procedure avoids many costly evaluations and results in nearly an order of magnitude speed-up in the algorithm.

Figures 1, 4, 6 and 7 show examples of clustering generated using this method. This process is extremely fast (see Table 1) and is over an order of magnitude faster than mean shift clustering though not as robust with respect to outliers. Nevertheless, we have found this technique to be a very good approximation to the rigid movement of the surface. Optionally, we can use a variational technique like Lloyd's method [CSAD04] with our Rigid Error Functions to attempt to improve upon the clusters. In our examples, we have not seen significant improvement with Lloyd's algorithm though it may help with highly deformable objects.

4. Skinning: Weight Estimation

After clustering, we now skin the rest pose and solve for weights α_j in Equation 1 associated with each vertex to reproduce their positions in the example poses. Several example-based skinning methods [WP02, MG03] as well as [JT05] describe methods could be used here. We follow [JT05] and enforce that the weights produce deformations invariant under rigid transformations (weights for each vertex sum to one) and that each vertex may only be influence by a maximum number of bones (4 for all our examples). Limiting the number of bones allows us to compute the deformations in an efficient manner using modern GPUs. Furthermore, we require that each $\alpha_j \geq 0$. While negative weights may be optimal to reproduce the example poses, they can lead to extremely undesirable deformations when the bones are moved to new poses.

These constraints lead to a least squares problem for the weights α_j (one for each bone) of the form

$$\min_{\alpha_j} \sum_{k=1}^{n} \left| \sum_j (\alpha_j R_{j,k} p_i + T_{j,k}) - q_{i,k} \right|^2$$

subject to the constraints

$$\begin{aligned} \textstyle\sum_j \alpha_j &= 1 \\ \alpha_j &\geq 0 \ \forall j \\ \left|\{\alpha_j | \alpha_j > 0\}\right| &\leq 4 \end{aligned}$$

where $R_{j,k}, T_{j,k}$ is the minimizer of the Rigid Error Function for the j^{th} cluster and k^{th} example pose and $q_{i,k}$ is the position of p_i in the k^{th} example pose. This type of least squares problem with equality and inequality constraints can be solved with a general purpose least squares solver with inequality constraints [LH74]. However, our problems are based on skeletal deformation and typically have more structure than a general least squares problem. A simple procedure of repeatedly solving the least squares problem with the partition of unity constraint and forcing the smallest $\alpha_j < 0$ to zero until all $\alpha_j \geq 0$ and 4 or fewer $\alpha_j \geq 0$ suffices.

The result of this minimization is a set of weights α_j for each vertex of the surface. These weights still may not produce good deformations outside of the example poses because the weights may lack locality, a common property enforced in deformation methods. This lack of locality is the product of bone motions that happen to be correlated in the example poses but is extremely undesirable when creating

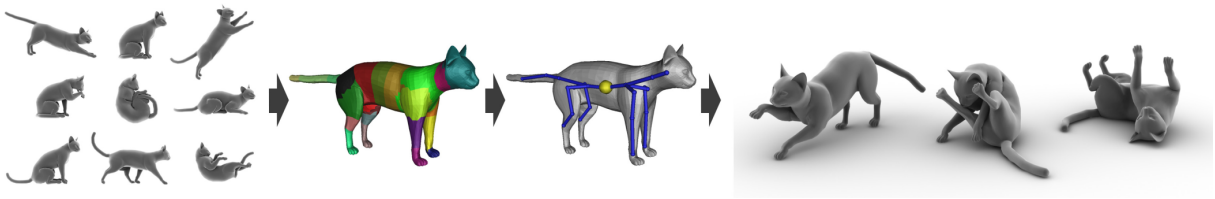

Figure 4: *From left to right: Example poses of a cat, clustering for bone transformations, skeleton found with root shown in yellow and new poses created using this skeleton.*

new poses of the character. Figure 3 (left) shows the influence of the bone associated with the front paw of the cat after minimization. Note how the influence is not connected and even influences the back paw and ear of the cat. Even small weights in these disconnected regions can create undesirable deformations as the user manipulates the paw of the cat due to the lack of locality.

Therefore, we enforce locality by finding an influence map for each bone. For each bone we find the vertex that contains the largest weight and flood outwards until we encounter vertices with weight zero with respect to that bone. We then use these influence maps to restrict the minimization to those bones and re-solve for the weights α_j. Figure 3 (right) shows the restricted influence map using this technique. It is possible, though unlikely, that a vertex may not be covered by any influence map after flooding. In this situation, we extend the influence maps of the neighboring vertices to this vertex and repeat until all vertices are covered. In our examples, our algorithm never had to utilize this step but this case may arise under extreme deformations.

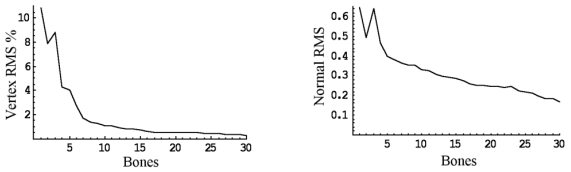

Figure 5: *RMS error for the horse with a variable number of bones both for the vertex positions and the unit normals of the polygons. The error for vertex positions is measured as a percentage of the length of the bounding box diagonal.*

Our goal is to develop a set of weights and a skeleton such that a user can create new deformations outside of the example set and not just reproduce the example poses. However, using these bone transformations and vertex weights, we can compare how closely these transformations/weights match the example poses. Figure 5 shows a plot of the RMS error of the vertex positions as a percentage of the length of the diagonal of the bounding box for the rest surface as the number of bones increases. The RMS of the unit normal is also shown as a function of the number of bones. Table 1 also contains the RMS error for each of our examples. As we increase the number of bones, both measures of error decrease. However, increasing the number of bones is not always desirable and there are more bones that must be manipulated to create new deformations.

5. Skeletal Extraction

So far we have found bone transformations (Section 3) and weights associated with vertices of the mesh (Section 4), but have not created a skeleton to manipulate the shape with. A skeleton consists of two parts: the connectivity between bones and the joint locations between pairs of connected bones.

5.1. Skeleton Connectivity

Several researchers have taken different approaches to determining skeleton connectivity. [AKP*04] bases the connectivity of the skeleton on the adjacency information between clusters representing rigid components, which works well for tube-like regions where the skeleton connectivity is unambiguous. However, this technique does not produce desirable results in regions with more complex adjacency such as the rear of the horse. [KOF05] determine skeleton connectivity using a minimum spanning tree approach similar to our method. However, their edge weights are based on a nonlinear measure of the error of placing a joint in between these two bones. We propose a simpler solution based on the vertex weights from Section 4.

In skeletal animation, vertices of a mesh are typically weighted by multiple, connected bones. These vertex weights actually indicate information about the connectivity of the skeleton. If a vertex is weighted by three bones, then it is likely that these three bones are connected together in some fashion in the skeleton. Therefore, we take advantage of this locality property when determining the connectivity between the bones from Section 3.

Our algorithm constructs a weighted edge graph between all of the bones of the skeleton and is initially a complete graph with all edge weights set to zero. Next, we iterate through all of the vertices in the shape and find their highest weighted bone. Let *max* be the index of that bone. Then,

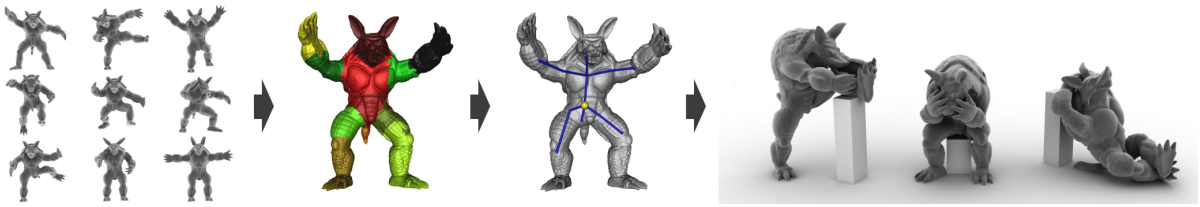

Figure 6: *The example poses of this model were created using Mean Value Coordinates, which we successfully convert to a skeletal deformation model. From left to right: the examples of the armadillo man, clustering for bone transformations, the extracted skeleton (root shown in yellow) and additional poses created with this skeleton including the stretching, sorrowful and relaxing armadillo man.*

for each $j \neq max$, we add the weight α_j to the edge in the connectivity graph between bones j and max. If desired, this weight α_j could be multiplied by the area of the triangles in the one-ring surrounding this vertex to make the result less dependent on the tessellation of the surface, though we found no difference in the result for any of our examples. Finally, we extract a maximal spanning tree to determine the final connectivity of the skeleton. Notice that this algorithm gives strong emphasis to vertices with nearly equal weights, for instance $.5/.5$, as an indication of a joint and less emphasis to vertices with more widely spread weights, like $.95/.05$, that may be related to subtle smoothing effects in the example poses.

While this algorithm determines the connectivity of the skeleton, it does not determine the root. The choice of which bone represents the root of the skeleton is somewhat arbitrary. Artists will typically choose the pelvis or torso of a human skeleton as the root. For shapes such as snakes, the choice of the root is less obvious. For our purposes, we determine the root of the skeleton using the center of mass of the shape. The center of mass for a human happens to be near the pelvis or torso, and, given the importance of this point in physical calculations, there is some physical basis for choosing this point as the root of the skeletal hierarchy. Hence, we choose the bone whose corresponding polygons from the clustering step have a center of mass closest to that of the rest pose as the root of the skeletal hierarchy.

5.2. Joint Determination

Joint determination is a well-studied problem in skeleton extraction [AKP*04, KM04, KOF05]. Assume that bones 1 and 2 are connected. A joint x connecting these two bones has the property that its location is the same with respect to both bone transformations in all positions. Using this description, we define the joint position x as the point that moves the least with respect to both bones, which leads to the quadratic minimization problem

$$\min_x \sum_k \left| \left(R_{1,k}x + T_{1,k}\right) - \left(R_{2,k}x + T_{2,k}\right) \right|^2 \qquad (5)$$

where $R_{1,k}, T_{1,k}$ and $R_{2,k}, T_{2,k}$ are the rigid transformations associated with the two bones for the k^{th} example pose.

Many joints like elbows and knees act as hinges and bend along a single axis. This hinge-like action creates an infinite number of minimizers along the axis of rotation. Thus, there may not be a single minimizer to Equation 5, but an entire subspace. Furthermore, slight noise or variation in the example poses may lead to a minimizer in that subspace far away from the actual surface.

Several solutions to this problem have been proposed involving non-linear minimization [KOF05] and minimizing the distance to the centroid of the boundary [AKP*04]. We use this idea of minimizing the distance of the joint's location to a point that approximates the joint's location $\hat{x}$. The minimization problem then becomes

$$\min_v \sum_k \left| \left(R_{1,k}(v+\hat{x}) + T_{1,k}\right) - \left(R_{2,k}(v+\hat{x}) + T_{2,k}\right) \right|^2. \qquad (6)$$

where v is a vector and we find the solution in the subspace which minimizes the magnitude of v. This solution can easily be found using a pseudoinverse and the final position of the joint is $x = \hat{x} + v$. However, we do not use the centroid of the boundary for $\hat{x}$ because the skeleton connectivity algorithm in Section 5.1 does not require that the connected clusters are adjacent on the surface.

Instead, we estimate the joint's position $\hat{x}$ using a similar procedure to Section 5.1. The intuition is that the weights of the vertices not only indicate information about the connectivity of the skeleton, but also about the positions of the joints. For instance a vertex evenly weighted by two bones $(.5/.5)$ should be very near the joint's location. Therefore, we find all of the vertices $c_i \subseteq p_i$ whose maximum weight is associated with either bone 1 or 2. Let $\alpha_{i,j}$ be the weight for c_i associated with bone j. We build a weighted average over all these c_i to approximate the joint position as

$$\hat{x} = \frac{\sum_i \min(\alpha_{i,1}, \alpha_{i,2}) c_i}{\sum_i \min(\alpha_{i,1}, \alpha_{i,2})}.$$

This position is refined using the minimization of Equation 6 to create the final placement of that joint.

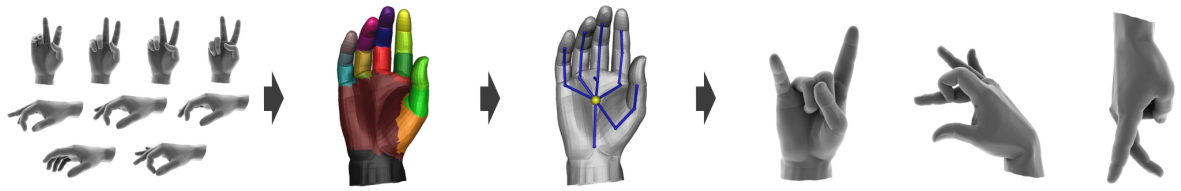

Figure 7: *In the examples (left) the motion of the pinkie and index fingers are nearly identical in all frames. However, our algorithm successfully extracts separate bones for the skeleton. On the right we show several new examples created with this skeleton.*

6. Results

Figures 1, 4, 6 and 7 illustrate our method on several different examples. The number of example poses range from 9 (cat and armadillo man) to 46 (the hand example). We derive the skeletal hierarchy and the joint positions automatically and, in many cases, these skeletons actually resemble the real-world skeletal structures of these shapes.

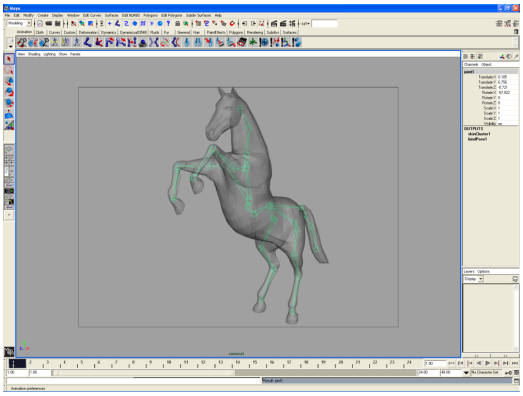

Figure 8: *Our system creates a rigid skeleton, which can be imported directly into standard modeling packages such as Maya (shown here).*

Each of these figures also shows several new poses created with the extracted skeleton and weights. We have also implemented a script that loads the rest pose, skeleton and weights directly into Maya for manipulation purposes (see Figure 8). Since we estimate a standard, rigid skeleton, this data fits readily into existing software and requires no special software beyond standard modeling packages to manipulate these shapes.

Many of these examples are available freely off the web and the methods used to create these deformations (SSD, FFD, etc...) are unknown. While automatically creating skeletons for poses created with skeletal animation is unsurprising (though still difficult) our algorithm can be used to create skeletons for poses that were created using other deformation methods. Unlike the other examples, the source of the armadillo man's deformations is known and were produced using 3D mean value coordinates [JSW05]. The deformations created by mean value coordinates are complex, not local and have no underlying skeletal structure. However, we are still able to extract a skeleton which approximates these examples.

In the example poses for the hand example (Figure 7), the motion of the pinkie and ring fingers are correlated and move in a nearly identical fashion. Transformation clustering approaches [JT05] may not be able to distinguish between the bones of the separate fingers because no topology information is used. Even clustering points for influence maps using rigidity scores [MG03] can fail in this situation and find correlation between the fingers. One solution to this problem is to add more examples poses to demonstrate the disconnected motion of the fingers. However, the topology of the shape provides sufficient information to remove influence between the fingers and our algorithm creates the proper weights for the vertices without the need for further examples.

Several of our examples represent animations such as the hand partially closing and opening again. Our method requires very few example poses (as little as one plus a rest pose) to obtain plausible results. However, we added all of the animation frames into our optimization to illustrate the running times of the algorithm though fewer frames are needed to create good skeletons.

The running times of our algorithm were measured on an Intel Core 2 6700 PC. Table 1 includes timing results for each of the stages of our algorithm on the different examples. Face clustering is very fast and finishes in a matter of a few seconds as opposed to minutes or even close to hours using mean shift clustering. Skinning times encompass both the first minimization, finding the influence map and second minimization. Skeletonization times were trivial in all examples and almost all under 0.01 seconds. The table also displays the RMS error of the vertex positions as a percentage of the length of the diagonal for the bounding box of the rest shape as well as the RMS error of the unit normal. In all cases, our method reproduces the example poses very accurately (less than 1% RMS error) but, more importantly, can be used to create new poses with our hierarchical skeleton.

Model	Faces	Example Poses	Bones	Face Clustering	Skinning	Skeleton Extraction	Vertex RMS	Normal RMS
Hand	15789	46	19	9.96s	6.46s	0.00s	0.14%	0.09
Horse	16843	23	29	5.53s	11.14s	0.00s	0.29%	0.19
Cat	14410	9	24	2.11s	4.01s	0.00s	0.52%	0.30
Armadillo	30000	9	12	5.21s	2.53s	0.00s	0.75%	0.17
Elephant	84638	23	22	32.29s	32.23s	0.02s	0.34%	0.15
Lion	9996	9	19	1.57s	1.85s	0.00s	0.65%	0.31

Table 1: *Running times for various phases of our algorithm with several different models as well as the RMS error for the position and normal with respect to the example poses. All times are measured in seconds.*

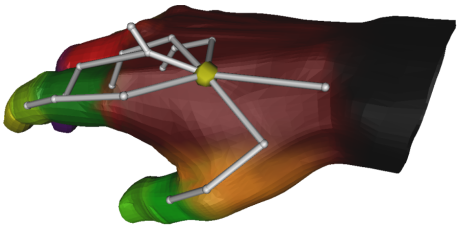

Figure 9: *Sometimes our algorithm extracts bones whose motion is really dependent on that of the surrounding bones as in the case of the knuckle here.*

Figure 10: *New poses created from the hand example spelling out "SGP" in sign language.*

7. Future Work

In the future we would like to investigate methods to derive parameters for other deformations models for use with flexible shapes. Skeletal deformation works well for controlling shapes that have an underlying biological skeleton or behave in a nearly rigid manner. However, for flexible shapes such as cloth, a skeletal model is not intuitive or appropriate. Our algorithm can be used to derive a skeleton for such shapes, but interacting with the skeleton may prove difficult. Other deformation methods such as Free-Form Deformations [SP86] may be more appropriate due to their ability to create provably smooth deformations though we must still develop techniques for estimating the parameters for these methods from a set of examples.

In some cases, the resulting skeleton is not exactly what a human would produce. Sometimes the clustering finds rigid pieces whose movement is not independent of other bones. For example, when the figures of the hand bend, the skin on the underside of the palm near the joint or on the knuckle moves as well. The motion of the skin in these regions is not the same as the motion of the finger so clustering may decide that this region should be a bone. However, in reality, the motion of these parts should be dependent on the only the finger bone and not be posable as separate bones. These extra bones do not affect the quality of the deformation, but may require more work from the user since there are more bones to control. Figure 9 shows an example on the hand where the clustering extracted a bone for the knuckle even though its motion is solely dependent on the finger bone in the examples. One solution would be to detect these dependent movements and remove this degree of freedom from the user so that the skin moves automatically with the finger.

For symmetric shapes, most users would expect that the skeletons as well as the vertex weights should reflect the geometric symmetry of the model. In the armadillo man example (see Figure 6), the model is symmetric even though the skeleton is clearly not. Currently we do not perform any symmetry detection and our current optimization is based solely on how the object actually moves in the examples. For the armadillo man, the two halves of the object behave very differently in the examples and, hence, the skeleton is not symmetric. A more extreme example might be a person that has suffered partial paralysis on one side of their body due to a stroke. Even though the geometry of the object is symmetric, we would expect different skeletons based on its movement. However, if symmetry is desired, we could explore detecting and enforcing such symmetry in the skeleton and vertex weights.

We would also like to develop a system that allows the user to input additional information into the training process for our algorithm. For instance, if the user disagrees with the connectivity of the skeleton that our method develops, they should be able exclude these edges in the connectivity graph from the optimization and have the algorithm recreate a new skeleton that satisfies these constraints. This type of

discussion process with the user should lead to an extremely robust algorithm that requires little user input.

Acknowledgements

We would like to thank Robert Sumner and Jovan Popovic for the models of the horse, cat, elephant and lion and the Utah 3D Animation Repository for the model of the hand.

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of SIGGRAPH 2000* (2000), pp. 157–164.

[AKP*04] ANGUELOV D., KOLLER D., PANG H., SRINIVASAN P., THRUN S.: Recovering articulated object models from 3d range data. In *Proceedings of the Annual Conference on Uncertainty in AI* (2004).

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (1992), 239–256.

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing* (2006), pp. 11–20.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 905–914.

[dATM*04] DE AGUIAR E., THEOBALT C., MAGNOR M., THEISEL H., SEIDEL H.-P.: m^3: Marker-free model reconstruction and motion tracking from 3d voxel data. In *Proceedings of Pacific Graphics 2004* (2004), pp. 101–110.

[DSP06] DER K. G., SUMNER R. W., POPOVIĆ J.: Inverse kinematics for reduced deformable models. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (2006), pp. 1174–1179.

[FS06] FELDMAN J., SINGH M.: Bayesian estimation of the shape skeleton. *J. Vis. 6*, 6 (6 2006), 23–23.

[GH97] GARLAND M., HECKBERT P.: Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 1997* (1997), pp. 209–216.

[Hor87] HORN B.: Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A 4*, 4 (April 1987), 629–642.

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 561–566.

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3 (Aug. 2005).

[KLT05] KATZ S., LEIFMAN G., TAL A.: Mesh segmentation using feature point and core extraction. *The Visual Computer 21*, 8-10 (2005), 649–658.

[KM04] KURIHARA T., MIYATA N.: Modeling deformable human hands from medical images. In *SCA '04: Proceedings of the symposium on Computer animation* (2004), pp. 355–363.

[KOF05] KIRK A. G., O'BRIEN J. F., FORSYTH D. A.: Skeletal parameter estimation from optical motion capture data. In *CVPR 2005* (2005), pp. 782–788.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3 (2003), 954–961.

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of SIGGRAPH 2000* (2000), pp. 165–172.

[LH74] LAWSON C., HANSON R.: *Solving Least Squares Problems*. Prentice Hall, Englewood Cliffs, NJ, 1974.

[LKA06] LIEN J.-M., KEYSER J., AMATO N. M.: Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling* (2006), pp. 219–228.

[LSLCO05] LIPMAN Y., SORKINE O., LEVIN D., COHEN-OR D.: Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3 (2005), 479–487.

[LWM*03] LIU P.-C., WU F.-C., MA W.-C., LIANG R.-H., OUHYOUNG M.: Automatic animation skeleton construction using repulsive force field. In *Proceedings of Pacific Graphics 2003* (2003), p. 409.

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (2003), 562–568.

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478.

[MP02] MORTARA M., PATANÉ G.: Affine-invariant skeleton of 3d shapes. In *SMI '02: Proceedings of Shape Modeling International 2002 (SMI'02)* (2002), p. 245.

[MWO03] MA W.-C., WU F.-C., OUHYOUNG M.: Skeleton extraction of 3d objects with radial basis functions. In *SMI '03: Proceedings of Shape Modeling International 2003* (2003), p. 207.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH 1986* (1986), pp. 151–160.

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 488–495.

[TdAM*04] THEOBALT C., DE AGUIAR E., MAGNOR M. A., THEISEL H., SEIDEL H.-P.: Marker-free kinematic skeleton estimation from sequences of volume data. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology* (2004), pp. 57–64.

[TVD06] TIERNY J., VANDEBORRE J.-P., DAOUDI M.: 3d mesh skeleton extraction using topological and geometrical analyses. In *Proceedings of Pacific Conference 2006* (Taipei, Taiwan, October 11-13 2006).

[WP02] WANG X. C., PHILLIPS C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the Symposium on Computer Animation 2002* (2002), pp. 129–138.

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 644–651.