Assisted Articulation of Closed Polygonal Models

Marek Teichmann Seth Teller

MIT Computer Graphics Group

Abstract

Creating articulated geometric models is a common task in animation systems. In some instances, models are procedurally instanced, and articulated degrees of freedom are designed into the model. In other instances, the model is some geometric assemblage, and an articulated skeleton (sometimes called an "I-K skeleton") is bound to the model by the user, typically by manual indication of a correspondence between elements of each structure. In either case, some binding must be made to couple boundary motions to those of the skeleton; this can be done for example by generating spring networks or spatial deformation fields. Both processes can be tedious in the ordinary case, especially when the model to be articulated is given only as a boundary representation, for example a polygonal mesh representing a character's skin or clothing, or an object's surface.

We present a simple method for assisted articulation of geometric models. Given a 3D polygonal mesh representing an object, an approximation to the mesh's medial axis is efficiently computed using a 3D Voronoi diagram of the mesh vertices, and connectivity information within the mesh. The medial axis is then simplified; the resulting tree structure has chains of edges and nodes. We interpret selected nodes as joints of an I-K skeleton, and the chains connecting them as its links. A spring network is then generated to bind the I-K skeleton to the object boundary, so that skeletal motions will affect the boundary in a reasonable way, as specified by the animator.

We show a user interface that allows interactive editing of the automatically constructed skeleton, and demonstrate the import, and mapping of key-framed motion capture data to, a variety of initially static polygonal objects.

Keywords: Animation, I-K skeleton, Medial axis, modeling, mesh deformation.

1 Introduction

Creating articulated geometric models is a common task in animation systems. In some instances, models are procedurally instanced, and articulated degrees of freedom are designed into the model.

In other instances, the model is some geometric assemblage, and an articulated skeleton (sometimes called an "I-K skeleton") is bound to the model by the user, typically by manual indication of a correspondence between elements of each structure. In either case, some binding must be made to couple boundary motions to those of the skeleton; this can be done for example by generating spring networks or spatial deformation fields. Both processes can be tedious in the ordinary case, especially when the model to be articulated is given only as a boundary representation, for example a polygonal mesh representing a character's skin or clothing, or an object's surface.

The input to our system is a polygonal mesh of genus zero representing an object. We generate a tree-like skeleton with very little user intervention. We proceed in several steps. First, the medial axis of the object is approximated by a Voronoi skeleton. The user then selects a few points on the Voronoi skeleton to identify object features to be animated. The skeleton is then simplified by our algorithm to produce a tree, whose branches reach to the user selected points, and which otherwise lies well inside the object. This tree transformed into an I-K skeleton through user identified skeleton vertices, which become joints. The mesh is then automatically attached to the skeleton via a spring network. Subsequent user or keyframe-based manipulation of the skeleton results automatically in corresponding articulation of the original model.

2 Related Work

Springs and Bones. Koch et. al. [KGC+96] use data from CT and range scans to construct a bone structure, onto which they connect a skin mesh using a set of springs. Wilhelms and Van Gelder [WV97] use existing bone models, and a simple model of the underlying muscles to determine the changes in muscle shape. In both methods the skin is a triangular mesh with springs at edges, is connected via a spring network and is relatively close to the underlying simulated muscles or bone. The springs are generated by locating one (or several in [KGC+96]) points close to the mesh vertices on the underlying structure. In [WV97] a parameterization of the muscles is used to create such attachment points. In [LTW95], a scanned facial model is animated via a spring network which includes an estimated underlying skull, and a layer of springs representing simulated muscles. A method for more precise modeling of muscles connected to an anatomically correct skeleton has been shown in [SPCM97]. In [GTT89], finite element methods are used on a manually constructed structure for computing deformations of an articulated model.

Free Form Deformation. Free form deformation techniques are important techniques for animating meshes. They do however require some manually constructed underlying structure. Such an approach is described in [CHP89]. Recent examples include [MJ96], and [CR94]. For example, in [MJ96], a lattice around the part to be animated is required (see for example Figure 9 in [MJ96]). It is not clear how such a lattice would be constructed automatically, or be made consistent across an entire model. In this setting it appears difficult to assign intuitive animation variables.

Mesh editing. Techniques for multiresolution editing, as in [ZSS97] allow a user to manipulate meshes on a large or small scale. Here too however, it appears difficult to assign simple animation variables to the model in an automatic fashion.

Skeleton construction. The problem of constructing a skeleton that is representative of an object is a well studied problem in pattern recognition, especially in two dimensions [OK95], and has also received some attention in graphics [FCGA97]. However, the notion of skeleton quality is domain dependent at present; there seems to be no algorithm that would always satisfy the subjective needs of an animator. In our method, we solicit a small amount of user input to solve this problem, and identify which parts of an object will be animated.

Implicit surfaces. These have been used mainly for the animation of models constructed by hand. Recently, [SOP95] give a combined method for animating human figures, where rigid body transformations of a scanned polygonal mesh skin on a skeletal structure are combined with deformations based on implicit functions. This skeleton must however be constructed manually. For skeleton based implicit surface models, local deformations can be handled as described for example in [OCG97].

3 Contributions

This paper builds on a number of fundamental ideas from computational geometry and graph theory:

- The Voronoi diagram and medial axis of a set of polygons [Aur91, SPB95].
- Graph algorithms for computing the biconnected components of a graph [CLR90].
- An approximation algorithm for the medial axis of the model polygons, by the Voronoi diagram of a set of points on the surface of the objects [Hub96]. For dense meshes, this can be taken as the mesh vertices. For meshes with larger triangles, these are recursively subsampled.
- The notion of importance of a point on the medial axis: points with larger distance to the surface have greater importance [OK95], as do points selected by the user in our case.

This paper introduces several new techniques:

- Use of the Delaunay triangulation [Ede87] of mesh vertices *and* points inside the object. We use points of the Voronoi diagram to give a structure to the interior.
- An algorithm for reduction of the complex Voronoi skeleton to a simple tree given a small number of user selected Voronoi vertices designating the "tips" of object features to be animated.
- Automatic determination of a good attachment of a spring network to the reduced skeleton.
- A method for identifying which mesh vertices to attach by springs to each I-K skeleton branch, this is an important step for proper spring network behavior.
- A new type of "smart spring", whose constant changes according to local geometry.

3.1 Summary of our Method

We start with the approximation of the medial axis as mentioned above. The user selects a small number of "tips" to indicate which features of the model are to be imbued with articulation freedom. The skeleton is then automatically generated using a new graph reduction algorithm (see Section 4.2). The skeleton forming the medial axis approximation is viewed as a graph. The algorithm proceeds by identifying those vertices of the skeleton which can be removed without disconnecting the graph. Among those, only vertices of low importance (those close to the mesh surface) are removed. This process is repeated until a tree is obtained. Due to the selection of vertices to be removed, the tree lies near the "center" of the model, but is connected to the user selected tips.

The user can interactively add or remove points of articulation, as well as define local coordinate systems at each articulation. The system then connects these articulation points to animation variables, and provide a means to couple external animations (eg. keyframe data) to animate the model via the skeleton.

To connect the skin to the skeleton, we give an algorithm for creating a spring network that allows the skin of the model to follow deformations of the skeleton, without breaks, folds or excessive stretching in the mesh for reasonable motions of the skeleton. We identify which mesh vertices are associated with each skeleton branch using the closest neighbors in a graph of mesh edges and mesh-skeleton Delaunay edges.

At animation time, for each animation frame, the spring network is simulated until it stabilizes, that is until a user specified threshold in the maximum displacement of mesh vertices is attained. The resulting mesh can be output for rendering.

We also allow interactive manipulation of the model by animating the skeleton manually, for example by exercising each joint. This is useful for tuning spring constants and deciding which degrees of freedom to add to or remove from the animation skeleton, and specifying local coordinate systems for easy attachment of external animation generators.

An important advantage of our technique with respect to anatomical modeling methods [WV97], is the low user interaction requirements. Our methods allow model importation and skeleton construction after approximately 30 minutes of user interaction. Finding the I-K skeleton then takes a few minutes of computation for a small model (5000 polygons). For larger models, this computation may take several hours, but need be done only once per model. After this step, the model is ready for animation. We give some running times in Section 6.

Our technique is model independent and requires no specific structure. It is mainly intended for models which have relatively long members, compared to the rest of the body, although we have successfully animated models like the Stanford bunny.

In theory, our algorithm has a significant limitation: it works only for "closed" objects, that is, those that partition space into two components. Moreover, it works best for objects of genus zero (that is, those with the topology of a sphere). However, it often works well for more intricate objects, and we demonstrate that in practice it removes much of the burden of articulating imported objects. Another limitation, which might not be present for example in systems based on implicit surfaces, is that the range of motion resulting in a reasonable surface is limited by the spring network. However real world models tend to also have similar limitations; from our experiments it seems that in general the capability of our system exceeds those real world limitations.

In Section 4 we detail the skeleton construction steps. Some implementation details are given in Section 5, followed by a summary of results in Section 6. We conclude in Section 7.

4 Skeleton Construction

Our technique is based on approximating the medial axis of the mesh. There are a number of techniques for approximating the medial axis of a polyhedral object [SPB95]. The method we use is to sample the surface of the object and compute the Voronoi Diagram of the sample points [Hub96].

4.1 Medial Axis Approximation

The first step in the construction of the skeleton is to ensure the mesh is sufficiently dense, which is required for a good approximation of the medial axis. We add points on its surface recursively until a user specified inter-point distance maximum is met.

Next, we construct the Voronoi Diagram of the mesh vertices. For this purpose we compute the Delaunay triangulation of the mesh vertices. For each resulting tetrahedron, we compute the circumsphere of the tetrahedron, whose center point is a Voronoi vertex [Ede87]. Adjacency of Voronoi vertices is apparent; two Voronoi vertices are adjacent if their corresponding tetrahedra share a face. We thus obtain the skeleton of the Voronoi diagram of the mesh vertices.

This produces a number of points lying on the medial axis, in the interior and exterior of the object. This process takes $O(m^2)$ time in the worst case, where m is the number of mesh vertices, but we use an efficient randomized algorithm [CMS93] and this is not a bottleneck in our system. For example, for the Doll model with 20710 triangles (see Section 6), this process took 32 seconds. The next step is to eliminate those Voronoi vertices on the outside of the object. For each vertex point, this is accomplished by casting a ray from the point in an arbitrary direction and counting the number of intersections.

We use an alternate method in case the model contains more than one surface, but satisfies the property that triangles of each surface are properly oriented (say have outward oriented normals). We consider the triangle closest to the point and make sure the point is in the halfspace delimited by the plane of the triangle, and locally interior to the model, i.e. on the interior side of the triangle. If the model has no holes, this guarantees a correct answer. This technique also works well in practice if the surface has some small holes, as does the Bunny model: we use rays in several different directions to find a ray that intersects the mesh.

We thus obtain the portion of the skeleton of the Voronoi diagram lying on the inside of the object. In addition, with each vertex is associated the radius of the corresponding circumsphere, as constructed by the Delaunay triangulation. This radius approximates the distance from this point to the object surface. Finally, we delete Voronoi edges that traverse of the model boundary. (For models that have several parts, we cannot simply delete edges which intersect any triangle.)

4.2 Skeleton Specification

We next consider this skeleton as a graph and take advantage of metric distance information to simplify the skeleton. Intuitively, the goal is to obtain one branch corresponding to each of the object's members or salient features that the user wishes to animate, or wishes to remain in a fixed position. Since, as mentioned before, the definition of the skeleton is highly model and application dependent, we include a small amount of user input for its construction.

The user selects Voronoi vertices that are close to the "ends" of each branch or member of the model, such as for example at the tip of a finger. For example, in Figure 1, the user has selected the green vertices. These vertices will remain untouched by the algorithm, and will constitute the end points of each skeleton branch. This guarantees that each model member will be represented in the skeleton. Typically, for a humanoid model, the number of such points is less than 25. It is also possible to do simple facial animation by choosing points near a mouth's center and corners, for example.

After this selection step, our skeleton simplification algorithm proceeds to automatically simplify the skeleton, as described below.

4.3 Skeleton Simplification

Our goal is to produce one chain of vertices per branch, in other words a tree, with the selected vertices as leaves. In addition, the skeleton should be as much in the center of the object as possible. We accomplish this as follows.

We consider the skeleton as an abstract graph, where the nodes are the vertices and the edges are the Voronoi edges. This graph is mostly connected. Occasionally, small stray connected components appear, and we take the largest connected component. We refer the reader to [CLR90] for definitions and algorithms related to graphs. We next iterate the following steps until the graph is a tree:

1. We compute the biconnected components of the current graph, initially the largest connected component.
2. For each biconnected component we remove the vertex closest to the mesh surface (as indicated by the associated sphere radius). We ignore the articulation points of the graph, as these might cause the graph to become disconnected, and user specified vertices. We are thus guaranteed by the property of biconnected components that the graph remains connected.

We thus obtain a tree (assuming the user did not select three adjacent vertices; an application of a Spanning Tree algorithm [CLR90] takes care of this problem.) An example of a model with its associated skeleton can be seen in Figure 1.

The running time of one iteration above is $O(v + e)$, where v is the number of vertices, e the number of edges in the graph. The total time is therefore $O(v^2 + ve)$. In our case, v is the number

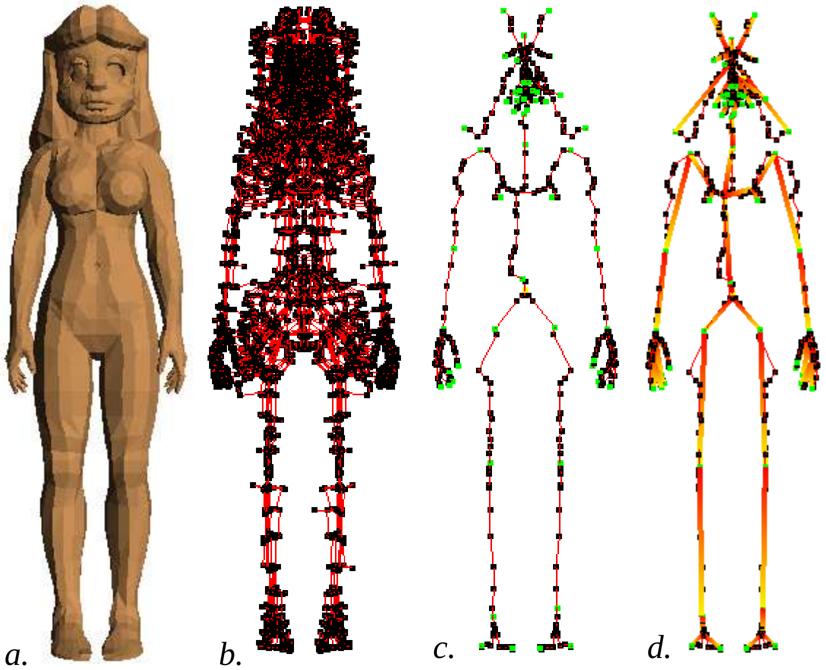

Figure 1: a. A woman model, b. The unsimplified skeleton, c. The simplified skeleton, d. with the I-K skeleton superposed.

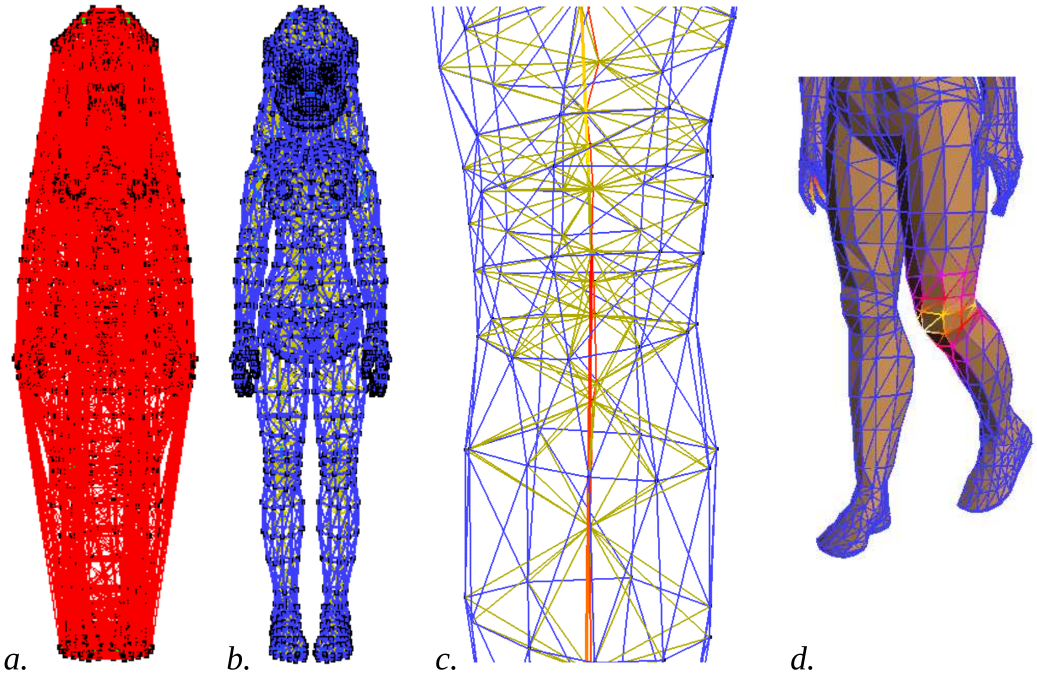

Figure 2: a. The initial Delaunay triangulation. b. After removal of external edges. c. Leg detail. d. The mesh is automatically readjusted when the knee is bent.

of mesh and Voronoi vertices, which can be quite large for a complex mesh. For large models, we use a simplified version of the model generated with the algorithm of [SZL92].

The final step is a simplification of the resulting skeleton. Neighboring vertices that are closer than an interactively set user specified threshold are merged, producing a vertex at their midpoint. This threshold is weighted by the distance to the mesh to take into account mesh geometry.

4.4 Animation Skeleton Construction

At this point, the user is presented with a simple skeleton that is almost ready for manipulation. The user identifies selected skeleton vertices as articulation joints, optionally adjusting their position and splitting edges where desired. Each skeleton branch between user identified vertices becomes an animation skeleton branch. The user can also interactively orient the local coordinate system at each joint, for compatibility with keyframe or motion camera data from external animation systems.

We thus obtain the animation skeleton, where to each branch (shown in red-yellow lines on the right in Figure 1) corresponds a chain of Voronoi vertices. The next step is to associate mesh vertices with each branch.

4.5 Spring Network Construction

We now need to attach the mesh to the skeleton. We do this by constructing a spring network. The network consists of springs at the mesh edges, and springs connecting mesh vertices to the skeleton vertices. The difficulty lies in finding the latter edges. We accomplish this by computing the three dimensional Delaunay triangulation of the points consisting of both the mesh vertices and the skeleton vertices. See Figure 2 for an example. The edges in red have been identified as lying on the outside of the model. The Delaunay triangulation algorithm returns a set of tetrahedra triangulating the convex hull of all the mesh and skeleton points. This step takes $O((m+s)^2)$ time in the worst case, where m is the number of mesh vertices, s the number of skeleton vertices, which is usually small. As in the Voronoi diagram computation, it is fairly efficient in practice, and required 45 seconds for the Doll model.

We take the edges of the tetrahedra and delete all edges that connect a skeleton vertex to a skeleton vertex, and all edges between mesh vertices, except those that correspond to edges in the triangulated mesh. This however occasionally causes small isolated regions of orphan mesh vertices. This occurs when in the original Delaunay triangulation, those vertices were closer to each other than to the skeleton. When the spring network is animated, this would cause that section of the mesh to lose its shape. We simply connect each orphan vertex to the skeleton vertex of closest link distance to it in the graph.

4.6 Spring Network Animation

Finally, each mesh vertex is associated with the closest skeleton vertex, and hence to one animation skeleton branch. This is important for the animation, see below. The skeleton construction is now complete; thereafter, the spring network simulation runs continuously thereafter.

When the user, or an external animation engine, rotates a specific branch of the animation skeleton, we move the mesh vertices associated with the entire skeleton sub-branch being moved, by the same rotation. The skeleton geometry itself however is maintained using standard hierarchical instancing. This method provides a good starting point for the spring network, which achieves stability faster than if we were to rely on the springs alone to find the new mesh vertex positions. In addition, for mesh vertices connected to both a non-rotated and a rotated skeleton section, we use a weighted average of the previous and new positions respectively. The weights are determined according to the number of springs connecting the vertex to each branch.

The springs constants are selected in a way similar to that in [WV97]. Our main contribution here is the introduction of "smart springs", to improve handing of cases where the mesh folds on itself during large motions. In [WV97], the problem is solved by shortening the spring rest length on the springs corresponding to the the mesh edges, and reducing the constant when these springs are compressed. We in addition determine if the mesh has been folded on itself by maintaining the normals of the triangles adjacent to each vertex. If this occurs, the spring constants of the springs attached between the affected vertices are substantially reduced, and the spring lengths are temporarily reduced (until stretched again).

Figure 2d. shows the spring network in action. The colors in the triangle edges indicate the current local "temperature" of the spring network relative to the current maximum. This represents the amount of change each vertex is undergoing due to the network simulation at the moment the image was recorded. Note that, even after a large motion, the majority of points on a branch that was moved remain motionless with respect to the branch, only near the joint do the vertices undergo large changes.

Our system also provides means to do key-frame animation: either via user input positions, or externally input positions. We can save a model at each frame for later rendering after the spring network has stabilized. In figure 3 we show several models in varying poses.

5 Implementation Details

The Delaunay triangulation computations are done using the robust HULL package of Clarkson [Cla], which implements a randomized incremental construction algorithm [CMS93]. This package computes the convex hull of a set of points in any (small) dimension, which by well known transformations can be used for finding the Delaunay triangulation, or as outlined above, the Voronoi diagram of a set of points.

This code returns a list of simplices triangulating the input points. We use the LEDA [MN95] graph data type to store and manipulate the result. For the Delaunay triangulation, we insert the input points as nodes in the graph, along with their position and other bookkeeping information.

Tetrahedral edges become graph edges.

For the Voronoi diagram, the centers of the tetrahedra are used as graph nodes. A graph edge corresponds to each tetrahedral face not on the convex hull of the entire set of points.

6 Results

We have imported and created skeletons for a number of models, from small thousand polygon models, to complex (thirty thousand) polygon models. Although we did not do this for the models in the table below, larger models, are best handled by prior simplification, but our system has no inherent limits on model size. The full resolution model can then be used for animation.

Our models include one (the Doll) which is partially articulated, and contains polygons in their interior. This did not affect the functionality of the spring network. We show the original models and some resulting poses in Figure 3.

In the table below, we give our model sizes in the number of triangles, the number of initial Voronoi vertices before reduction (labeled Voronoi), the number of springs, and skeleton creation times.

Model	Size	Voronoi	Springs	approximate creation time
Doll	5944	9938	36260	13.2 minutes
Stanford bunny	16301	27539	35040	34.5 minutes
Woman	20710	72371	127407	4 hours
Man	29266	46565	181001	6 hours

The computation of the Voronoi diagram and the Delaunay triangulation did not exceed 1 minute for any of the models listed. The number of Voronoi vertices forming the skeleton is, after simplification, on the order of 200 to 500 for all models, and is user adjustable. For smaller models (10000 triangles), the spring network stabilizes in under 3 seconds. For the larger models, depending on the size of the motion, and the value of a "drag" coefficient in our spring simulation, it can take up to 10 seconds for the larger models..

7 Discussion and Concluding Remarks

We have presented a useful tool for animators, which should simplify considerably the process of embedding animation controls into existing boundary models.

A relative advantage of our method over anatomical modeling methods such as [WV97] is the absence of a requirement of a detailed structure below the skin, thus avoiding the need for the manual construction of such a structure. While such a structure allows the modeling of a "muscle bulge" effect, our method is particularly appropriate for models requiring no such effects, although such extensions can be easily imagined.

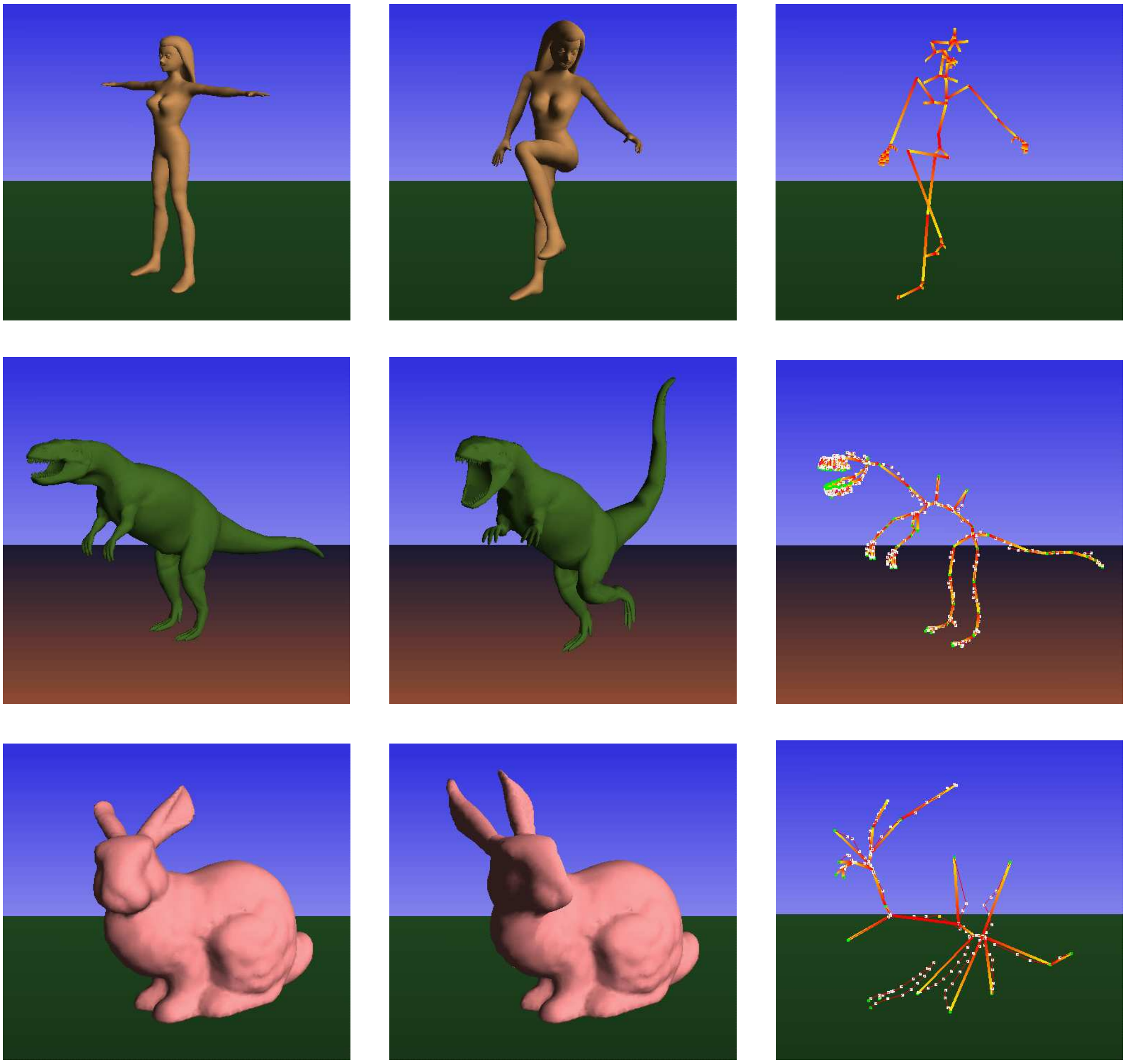

Figure 3: Three models and a pose obtained during animation. The originals are on the left, their skeletons on the right.

Improvements to our methods are possible. The skeleton simplification algorithm, while guaranteed to produce a tree, can take considerable time on large models. We deal with this problem by using an existing simplification algorithm, but it would be interesting to find a faster one. It would seem for example, that it would be desirable to delete more than one vertex per biconnected component; however this is in general not possible, as in, for example, when the component is formed by one cycle. Collision detection with scene objects, and self-collision detection would allow the simulation of interactions of the model with its environment. Finally, an easy extension is to place sampled cylinders around skeleton edges, tapered toward the ends, to improve control of skin when skeleton edges are twisted.

Finally, it would be interesting to combine our methods with multiresolution representations, as in [Hop96].

We would like to thank Viewpoint Data Labs for their generosity in providing many of the models for our work, and the Stanford Data Repository for making their models available.

References

[Aur91] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[CHP89] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 243–252, July 1989.

[Cla] H. Clarkson. *Hull.* Convex hull library, http://cm.bell-labs.com/netlib/voronoi/hull.html.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[CMS93] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3(4):185–212, 1993.

[CR94] Yu–Kuang Chang and Alyn P. Rockwood. A generalized de Casteljau approach to 3D free–Form deformation. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 257–260. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[FCGA97] Eric Ferley, Marie-Paule Cani-Gascuel, and Dominique Attali. Skeletal reconstruction of branching shapes. *Computer Graphics Forum*, 16(5), December 1997.

[GTT89] Jean-Paul Gourret, Nadia Magnenat Thalmann, and Daniel Thalmann. Simulation of object and human skin deformations in a grasping task. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 21–30, July 1989.

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[Hub96] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[KGC+96] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, and Y. Parish. Simulating facial surgery using finite element methods. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 421–428. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[LTW95] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic face modeling for animation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 55–62. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[MJ96] Ron MacCracken and Kenneth I. Joy. Free-Form deformations with lattices of arbitrary topology. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 181–188. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[MN95] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM*, 38:96–102, 1995.

[OCG97] Agata Opalach and Marie-Paule Cani-Gascuel. Local deformations for animation of implicit surfaces. In *SCCG'97*, Bratislava, Slovakia, 1997.

[OK95] R.L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[SOP95] K. Singh, J. Ohya, and R. Parent. Human figures synthesis and animation for virtual space telecon ferencing. *IEEE VRAIS*, pages 118–125, 1995.

[SPB95] E.C. Sherbrooke, N.M. Patrikalakis, and E. Brisson. An algorithm for the medial axis transform of 3d polyhedral solids. In *Proc. ACM Siggraph Sympos. Solid Modeling*, pages 187–199, 1995.

[SPCM97] F. Scheepers, R.E. Parent, W.E. Carlson, and S.F. May. Anatomy-based modeling of the human musculature. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 163–172, July 1997.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[WV97] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 173–180. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 259–268. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.