Real-Time Avatar Construction Using Shape Tape

Technical Report Number 02-036
5-Sep-02

Samir Naik

Effective Virtual Environments Research Group

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175

Real-time Avatar Construction Using Shape Tape

Samir Naik

Tech Report 02-036 - University of North Carolina - Chapel Hill

naiks@cs.unc.edu

ABSTRACT

Motion capture is used in video games and computer animated films to create realistic animations of moving virtual characters. In games and films, an actor's motion is captured offline, saved, and finally replayed to animate a virtual character. The same principles can be applied to Virtual Reality (VR), but the motion capture must be done in real-time so when a user moves, her corresponding virtual character – her avatar – mimics her movement. Additionally, the user must identify with her avatar, so the shape of the avatar's limbs should closely match the shape of the human's in length, radius, and curvature. Finally, the avatar should also be able to interact with other virtual objects in the environment, thus we need a representation that is suitable for use with collision detection algorithms. In this paper, we analyze the problem of tracking a user's limbs with Shape Tape™ and constructing a virtual representation of the user in real-time. We will compare and contrast this problem to problems in sampling/reconstruction, geometric modeling and computational geometry and finally introduce possible solutions.

Keywords

Avatar, tracking, collision detection, sampling, reconstruction, medial axis.

1. INTRODUCTION

An important test of the effectiveness of a Virtual Reality (VR) simulation is to ask the user "did you feel like you were really there?" or "did you feel present in the virtual environment (VE)?" The answer to these questions often depends on whether the user felt she was an active participant in some "virtual activity." A VE in which a user can use her hands, arms, and legs to participate in a virtual activity is much more engaging than a VE in which the user is merely a spectator.

To create such an experience, we have to make the virtual environment more interactive. Our current system limits interaction with the VE to walking around and looking at virtual objects. The user sees the computer generated virtual world by viewing it through a head mounted display (HMD). As she moves her head and walks around the lab, a wide-area tracker captures the position and orientation of her head and the image of the VE is redrawn in the HMD to match her new viewpoint.

If we want the user to see her avatar in the VE and to affect the objects in the VE, we can estimate her torso position and orientation from how her head is situated, but we must also need a method of knowing where her arms and legs are. This can be done by placing trackers on her hands and feet and using inverse kinematics to determine the position of the other parts of her arms and legs. However, inverse kinematics does not always accurately depict how the user's limbs are moving. It also cannot tell us any information about the size of the user's limbs, i.e. is the user a man or a woman, adult or child? Any large differences between user's shape and motion and the avatar's shape and motion will cause the user to disassociate with her avatar. Additionally, if the inverse kinematics solution does not estimate the position of the limbs correctly, our collision detection algorithm may falsely detect collisions between the avatar and the VE.

We can avoid the problems of inverse kinematics by using a full body tracker that tracks several positions along the user's limbs. Specifically, we want to experiment with a new tracking device called Shape Tape. Shape Tape consists of 4 long tapes that are each spiraled around a different limb. Each tape returns position and orientation at regularly spaced intervals along its length. Having tracker data at many places on the tape helps us in two ways. First, it gives us a sampling of the shape and radius of the user's limbs so we can make a better estimate of the size of the user. And, second, it gives us a better sampling of how the main bones on the user's arms and legs are moving, not just the hands and feet. The resulting avatar will more accurately depict the shape and motion of the user, and collision detection between the avatar and the VE will be more correct.

Thus, our problem is how to reconstruct an "accurate" avatar. We separate the problem into two pieces: an accurate reconstruction of the user's motion and an accurate reconstruction of the user's shape. In this paper, we will briefly discuss the former, but mainly focus on the latter. These reconstruction steps also are grouped into categories of offline computation, and run-time computation, which shape computation usually taking place offline, and motion computation handled at run-time. These

Figure 1. A user wearing Shape Tape (image: Measurand, Inc.) and an example of what the resulting avatar might look like.

stages are discussed in more detail later.

To help us analyze our problem, we introduce three reference fields: sampling and reconstruction theory, geometric modeling and computational geometry. We will compare and contrast our problem with problems presented in papers from these fields. In Section 2, we will discuss why we chose these fields, and how they will help us think about our problem. In Section 3, we discuss how these fields apply to our input data, and in Section 4 we discuss how these fields apply to our desired output. Finally, using the analysis of our inputs and outputs, we will formulate possible solutions to our problem in Section 5 and decide which is the most feasible to implement.

2. BACKGROUND

Our reference fields of sampling/reconstruction, geometric modeling and computational geometry contain ideas that are used throughout the remainder of this paper.

1. **Sampling/Reconstruction Theory**. Shape tape returns discrete samples of some continuous function describing the motion and shape of the user's limbs. From this data, we wish to separate the motion information and the shape information and reconstruct separate continuous functions for both. These functions can then be applied separately to the motion and shape of the avatar.

   Machiraju, et al. discuss the problem of sampling a continuous function, and reconstructing the function from the sampled data. They focus on giving the user the ability to set a point-wise error bound on the reconstruction. We use this paper for the background it gives in reconstruction and sampling theory, but not for it's discussion of error bounds.

2. **Geometric Modeling**. We must choose a geometric representation for our avatar in the VE. Representing the avatar with a polygonal mesh makes it easier to render and allows us to use a common polygon-based collision detection package to detect collisions between the avatar and the VE.

   Hoppe, et al. discuss the problem of reconstructing a mesh from partial data of an unknown surface. Their data is in the form of unorganized 3D point samples, and their solution is general, making only the assumption that the sample data is uniformly distributed. We compare and contrast the unorganized point data to our Shape Tape data, and discuss the how the extra information Shape Tape returns changes the surface reconstruction problem.

3. **Computational Geometry.** An alternative to creating a mesh directly from the sampled data is to find the *skeleton* of the 3D data and use this skeleton to control a stock mesh. In Computation Geometry the medial-axis transform is commonly used to find a skeleton of a 3D data set.

   Amenta, et al. also discuss the problem of reconstructing a mesh from sample data, but make more assumptions about their data than Hoppe, et al. Mainly, they assume they have a "good" sample of the unknown object. They use the *medial axis* of a surface in their definition of a good sample.

   Since the medial axis is used throughout this paper, we will define it here. The medial axis of a 3-dimensional surface in $\mathbf{R}^3$ is the closure of the set of points with more than one closest point on the surface [2]. Another definition is the medial axis of a 2D (3D) region is defined as the locus of the center of all the maximal inscribed circle (sphere) of the object. An example is shown in Figure 2. Notice that in 2D the medial axis is made of 2D curves. In 3D the medial axis is a surface.

   We also use this paper to discuss the problem of reconstructing an avatar from our Shape Tape data. Since the paper makes assumptions about their data that also apply to our data, we discuss Amenta's algorithms and how we might make use of them to solve our problem.

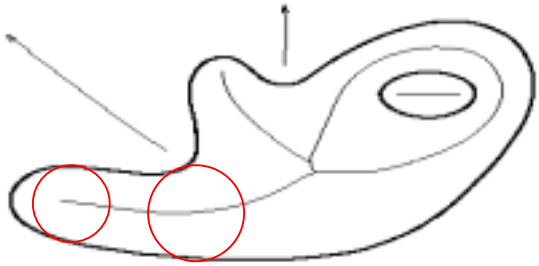

Figure 2. The medial axis of a simple curve in 2D with the maximal circles overlayed (figure from [2]).

3. INPUTS

The input to the system is given by Shape Tape. Shape Tape consists of lightweight, wearable, flexible ribbons. Each ribbon contains 81 fiber optic sensors, each of which returns its position and orientation relative to the base of the tape at a synchronous update rate of 110Hz. The data returned by Shape Tape is detailed in Table 1. Our system is set up with 4 tapes, one to be spiraled around each leg and each arm. Figure 3 shows a single tape and a graph of the data returned by the tape. We analyze the data returned by the Shape Tape from the background of sampling/reconstruction theory and geometric modeling.

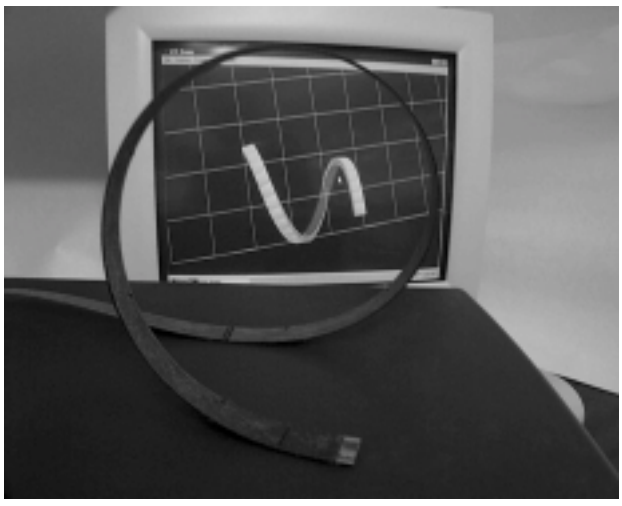

Figure 3. A single shape tape ribbon, with the resulting data set rendered on the monitor. Notice that the position and orientation of the tape's spiral shape are correctly captured (image from [7]) .

	Point 2	Point 3	Point 4	Point 5
Rx,mm:	17.9	34.8	50.7	66.1
Ry,mm:	8.9	19.5	31.8	44.5
Rz,mm:	0	-0.1	-0.2	-0.3
Ux:	0.99516	0.98068	0.9567	0.94721
Uy:	0.09829	0.19562	0.29104	0.32056
Uz:	-0.00089	-0.00265	-0.00528	-0.0051
Nx:	-0.0983	-0.19564	-0.29107	-0.32053
Ny:	0.99512	0.98051	0.95633	0.94722
Nz:	-0.00897	-0.01782	-0.02644	0.00582
Bx:	0	-0.00089	-0.00265	0.0067
By:	0.00902	0.01799	0.02683	-0.00388
Bz:	0.99996	0.99984	0.99964	0.99997
roll,deg	-0.7	-1.5	-2.2	0.3
pitch:	5.6	11.3	16.9	18.7
yaw,deg:	-0.1	-0.2	-0.3	-0.3

Table 1. Example of Shape Tape data set. Data of first 4 of 81 sensors, at some time T, is shown (Point 1 is the base of the tape and does not change). R is vector from point 1 to data point. U, N and B are orthogonal unit orientation vectors. 'Yaw', 'pitch' and 'roll' are rotation about y, z and x respectively (data from [7]).

3.1 Sampling/Reconstruction

Take, for example, the case of a single tape wrapped around a user's arm. The tape is sampling two attributes of the user's arm: it's shape and it's movement. Shape is sampled in the spatial domain, that is, at any instant in time, we take the sample data from all 81 sensors and attempt to reconstruct a surface from it. Shape reconstruction doesn't depend on data from the past or future. Conversely, motion is sampled in the temporal domain, that is, we track the movement of a few sample points over a period of time. Reconstruction of the motion of these few points does not depend on the position of the rest of the data points.

For the following discussion, we treat the problems of spatial sampling (shape) and temporal sampling (motion) separately.

3.1.1 Spatial Sampling

We can think of the surface of the user's arm as a continuous function, f(**x**), where **x** is a position in 3D, and that the Shape Tape is sampling this function at regularly spaced intervals along the tape. Machiraju, et al, assume that f(**x**), is band limited. A function is band limited if there exists a *cut-off frequency* ω_c, such that the strength of any frequency component of f(**x**) greater than ω_c is zero. Additionally, he assumes that f(**x**) is sampled at or above the ***Nyquist frequency*** [3]. The Nyquist frequency of a signal is defined as twice the maximum frequency of the continuous signal. If a signal has been sampled above the Nyquist frequency, a perfect reconstruction of that signal is always possible.

Low-frequency components of f(**x**) contain the general shape of the arm, while high-frequency components contain details such as folds and creases in the skin. Machiraju assumes that all high frequency data of the arm can be reconstructed, because f(**x**) was sampled above the Nyquist frequency. By observation, however, we know that our sampling rate is probably below the Nyquist rate, and that it is not high enough to reconstruct all of the details of the human arm. But, it is high enough to capture the general shape of the arm. Therefore, we make the additional assumption that the user's arms (and legs) are not very complex, and that we only need to reconstruct low frequencies to capture the general shape of the arm.

This brings up a possible experiment to test how hi-fi an avatar needs to be for a user to identify with it. We are assuming that a lower frequency approximation of the shape of the arm is satisfactory for the avatar. It may be possible to test this quantitatively by varying the detail of geometry of the avatar, and testing how this changes the level to which a user identifies with it.

3.1.2 Temporal Sampling

We also assume that we have a continuous function over time, f(t), describing the motion of the arms and legs. We can apply sampling theory as we did above, and assume that f(t) is band limited. That is, if the Shape Tape generates its tracker data at $2\omega_c$ Hz, then we assume that in the motion of the arm there are no frequencies greater than ω_c.

It may be worthwhile to examine this assumption further. High-frequency components of f(t) contain the detail of motion, while low-frequency components contain general, gross motion patterns [4]. We can add to our earlier experiment by varying the amount of detail in the avatar's motion, and testing how the user's identification with the avatar changes. If we find that our assumptions about the complexity of the user's limbs and motion are not valid, we may have to choose a motion tracker that would give a higher sampling rate than Shape Tape. For now, however, we will proceed using these assumptions.

3.2 Geometric Modeling

The data set returned by Shape Tape is a subset of the unorganized point set assumed by Hoppe, et al [1]. Hoppe presents a general algorithm for creating a computer model from 3D data and thus, they assume the input set consists of sample points in $\mathbf{R}^3$ without any additional structure or organization. We do not need a general solution, but rather a solution to the specific problem of constructing a surface from Shape Tape data. We can use the additional information that Shape Tape gives us to create a more efficient algorithm. Specifically the differences between the Shape Tape data set and the unorganized point set are listed in Table 2 and discussed below.

Shape Tape returns data points in order along the length of each ribbon, therefore our data set is not unorganized. For each sample $\mathbf{x_i}$, we know its nearest neighbors along the length of the tape. Because of the spiral arrangement of the tape, these may not necessarily be the nearest neighbors in world-space. But, if we take a few adjacent loops in the tape, containing *m* sensors, and group samples according to distance and orientation, we can find the *k* points in our data set which are nearest to $\mathbf{x_i}$. These points make up the k-neighborhood of $\mathbf{x_i}$.

Shape Tape	Unorganized points
Ordered points	Unordered points
Position and orientation	Position
Explicit control of sampling density	Uniform sampling density
Known domain	Unknown domain

Table 2. Comparison of shape tape data with unorganized point data.

If the points are unorganized, finding the k-neighborhood for a given point takes O(n + k log n) time. Hoppe, et al assume a uniform sampling density, and therefore use a simple spatial cubic partitioning scheme. Points are entered into bins according to which cube they lie in, and are accessed through a hash table. This reduces the time to find the k-neighborhood to O(k). With the ordered points that Shape Tape returns, we have to find the k-neighborhood from the set of *m* points in neighboring rings of the spiral. This takes O(m + k log m) time. We cannot use a hash table to speed up this computation, because we cannot assume that Shape Tape provides a uniform sampling density.

In addition to returning the position of each sensor along the ribbon, Shape Tape returns the orientation of each sensor. This is different from the unorganized point problem because samples are assumed to be infinitesimal points, and thus have no orientation. But since Shape Tape is a flat ribbon-like device, each sensor can be thought of as a small disc that samples the position and normal of the surface. In fact, this information would be useful to Hoppe, et al. In order to calculate a signed distance function to the surface, they calculate the tangent plane to each sample point by computing the covariance matrix of the k-neighborhood. We can skip this computation, since we already have the orientation of the sample disc.

When constructing a surface from unorganized points, it is difficult to distinguish holes in the data due to insufficient sampling from an actual hole in the surface. Thus, Hoppe, et al. assume a minimum density, *p*-dense, sample. A *p*-dense sample means that sphere of radius *p* can be centered on any sample point, and at least one other sample point is guaranteed to lie within this sphere. Assuming a uniform sampling density allows Hoppe, et al., to be certain that holes in the sample correspond to holes in the surface. With Shape Tape, we don't have to assume a uniform sampling density, since we can explicitly control it. Additionally, we know that the user's arms and legs have no holes in them, so we know that holes in the sample data do not correspond to holes in the surface. This is important since the spiraling nature of Shape Tape leaves many holes in the data.

Another difficult problem is with surface reconstruction techniques is reconstructing sharp edges and creases. Amenta, et al., state that a good sample is one in which the sampling data is at least inversely proportional to the distance to the medial axis [2]. Since the medial axis is very near the surface where sharp edges and creases appear, we need a higher sampling rate there. While arms and legs have creases near the joints, we likely don't need to precisely reconstruct them – it may be sufficient to generate them procedurally (again, this is subject to experimentation). However, we can still apply Amenta's sampling criterion to tracking arms and legs since we can arrange the tape to sample regions of high curvature, such as wrists and hands, better than regions of lower curvature, such as thighs. Amenta also mentions that if surface normal data is available, reconstruction is possible from much sparser samples. In general, we have a sparser sample set than Amenta considers "good", so we should use normals to our advantage.

Other than the four differences listed in Table 2, our input data is the same as in the unorganized points problem. Our problem is distinctly different, however, because we know our desired output surface is an arm or a leg, and not an arbitrarily complex surface. We will discuss how we can fit our input data to the surface of an arm or leg in the following section.

4. OUTPUTS

Our goal is to create avatars of human arms and legs that look good enough for the user to identify them as her arms and legs. We look at our output requirements from the point of view of the fields, computational geometry and geometric modeling

4.1 Comp. Geometry & Geometric Modeling

What does it means to for our avatar to look good geometrically? Amenta, et al. state that given a good sample from a smooth surface, the output of the reconstruction algorithm should be topologically equivalent to the surface. Additionally, as sampling density increases, the output converges to the surface, both pointwise and in surface normal. Again, this assumes a high enough sampling rate, which we likely don't have. Thus, we only aim to reconstruct a surface as well as our sampling rate will allow us to.

We must decide how we want to represent our avatar in the system. The two main parameters for our representation are 1) how easy it is to render and 2) how suitable it is for collision detection. For instance, a natural way to render point samples is by splatting. But splatting doesn't lend well to collision detection. Instead, we might choose to represent our avatar as a set of implicit surfaces. Collision detection is easy with implicit surfaces, since the math is very simple. However, implicit surfaces must be sampled, e.g. using marching cubes, and thus are not ideal for rendering. Conversely, if we used a parametric representation, collision detection would be difficult, but rendering easy. We can choose different representations for rendering and collision detection, but since our system aims to run in real-time, we would like to minimize computation by choosing the best representation.

In our system, we will represent our avatar's skin as a polygonal mesh. There are two possible ways to generate a mesh representation: reconstruction from sample data and use of a deformable stock mesh with an underlying skeleton. The details of how to compute these will be discussed further in Section 5; here we discuss the output requirements.

Reconstructing a mesh from sample points. We must compute connectivity information from our sample points, which splatting or implicit representations do not require. Also, collision detection is not as easy with a polygonal mesh as it is with implicit surfaces. However, it is not as easy to find an implicit form that fits our data points as it is to find the polygonal mesh. Also, rendering a polygonal mesh is

straightforward, much more so than rendering an implicit or parametric representation, and it is well optimized in most graphics hardware.

Using a stock mesh with underlying skeleton. We can obtain the mesh from a character-modeling program. For the mesh to deform correctly with the movement of the bones, we need to capture the actual bone data from the user. To do this, we model the human skeleton as rigid segments between articulating joints. The assumption of rigidity is realistic, since bones undergo negligible deformation during normal human activity [5].

5. POSSIBLE SOLUTIONS

Having analyzed our input data and output requirements, we suggest possible solutions. The following suggestions are made without having done any actual experimentation or mathematics to ensure that the algorithms will produce the correct result. It is merely a discussion of algorithm ideas that have come to mind after thinking about our three themes in regards to our problem.

5.1 Skin and Bones

One approach to our creating a realistic avatar is to use a technique called Skin and Bones. This technique is commonly used in computer games and describes a character built out of a single deformable mesh rather than one made up of separate objects. The mesh is procedurally deformed by an underlying bone structure that is driven by the tracker data. The steps of the skin and bones algorithm are:

1. **Getting the bones.** To find the bone structure, we use the medial axis algorithm discussed by Amenta, et al. The medial axis transform of 3D data points generally results in medial surfaces in 3D. We would like our bones to be straight line segments, so we have to apply the medial axis transform again to each medial surface previously computed to obtain 3D curves. Only some of these curves will actually be useful for representing bone structure, the rest are an artifact of the medial axis transform, and do not apply. Since, this algorithm will produce a similar set of 3D curves for each user, we can likely determine useful curves by experimentation.

   We can't use these curves directly as our bones however, since they may contain influence from muscle and fat. Our themes suggest two ways of handling this problem. From computational geometry and/or geometric modeling, we may apply a curve decimation approach to simplify the curve into straight line segments (Figure 4). From sampling/reconstruction, we may run a low-pass filter on the curve to eliminate high-frequency data caused by the muscles and fat, and leave only the lowest frequency data, which should give a better estimate of the bone structure. We need to quantitatively analyze these two options to determine which yields the best result. Specifically, we want to determine which of our 3D medial curves map to bones, and which of the above methods (or combination of them) is best for finding bones and joints.

   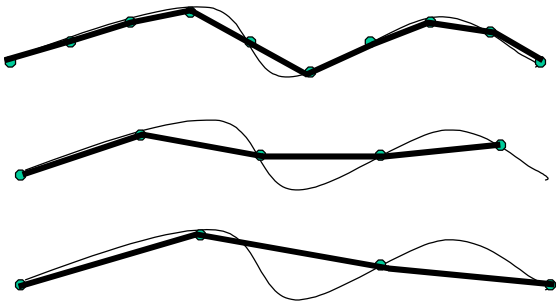

   Figure 4. Progressive curve decimation is shown. First we connect all sample points, then every other sample point, finally every third sample point.

   The process of finding the skeleton is costly but can be done offline, since the user's skeleton doesn't change. Therefore the worst case $O(n^2 \log n)$ running time for finding the skeleton of a polygon with n vertices doesn't affect our system's runtime performance.

2. **Creating the mesh.** To create a mesh for our avatar, we can use a standard character modeling package. This mesh needs to have an underlying bone structure, so the package should be able to take as input the bone structure we calculated from the tracker data.

   Alternatively, we may try to reconstruct a mesh from the tracker data and treat it as a stock mesh. This would give us a mesh that looks like our user's arms and legs, but is procedurally deformed by the computed bone structure as the user moves his arms. However, this approach requires that for each user of the system, we reconstruct a mesh, and associate bone weights for each vertex (discussed next). This is very costly in terms of man-hours, thus we can probably rule this method out now, unless an automatic method of associating these weights is available. The algorithm for surface reconstruction is given in Section 5.2.

   These methods are both done offline and do not contribute to our system's run-time performance.

3. **Assigning weights to mesh vertices.** Each vertex of the mesh must be associated with one or more bones to define how the vertex moves as the arm moves. This is an offline computation and is typically done within the character modeling package. For example, skin near the elbow is affected by the movement of the upper arm and the lower arm. A particular vertex on the lower arm side of the elbow might have been affected 20% by the orientation of the upper arm and 80% by the lower arm. How these weights are assigned will affect the final position of the vertex and we need to define these weights for every vertex. The next section discusses the use of these weights to transform mesh vertices.

4. **Finding world space transform of the mesh vertices.** To render the deformable mesh, we need to transform each vertex separately into world space. The transformation of the vertex is given by:

   $$\text{FinalPosition} = \text{position}[1] * \text{weight}[1] + \text{position}[2] * \text{weight}[2] + \ldots.$$

   Where each position[N] is the initial position of that vertex multiplied by the transformation matrix of bone N [6].

These multiplications are done at run-time for each frame, but are very cheap in graphics hardware, and hardly affect our performance.

The skin and bones approach has the nice property that it is widely used in the game industry, meaning it has proven real-time performance. The disadvantage is that while the bones are modeled after our user, the skin has no relation to the surface of the users arms and legs. This raises another possible experiment on whether it is better for the avatar to have bones which are in proportion with the user's, or for the avatar to have a surface which resembles the user's in size and shape.

5.2 Real-time Meshing

We may find from experimentation that it is important for the avatar's surface to closely match the surface of the user's arms and legs. In this case, it might be better to dynamically reconstruct the surface from our tracker points in real-time in order to capture the details of the deformation of the user's arms and legs.

Both Hoppe and Amenta give algorithms for reconstruction. Hoppe, et al. define a signed distance function in $\mathbf{R}^3$, and then polygonalize its zero-set to create the output mesh [2]. Zero-set algorithms produce approximating, rather than interpolating, meshes. Essentially, approximating algorithms are applying a low-pass filter to the sample data before reconstructing. Since the purpose of our surface reconstruction is to gain a more precise avatar than a stock mesh would give us, and since Shape Tape inherently does low-pass filtering also, we should not use this method, since it will probably overly smooth our data.

Amenta, et al. use a Voronoi based approach to surface reconstruction, using a *crust algorithm* to compute the surface. The crust algorithm basically uses the Voronoi diagram to compute a medial axis, and then uses the medial axis to compute a *crust*. A crust is a closed set of edges connecting only sample points in adjacent Voronoi cells. Specifically the algorithm is [2]:

1. Compute the Voronoi diagram of a set of samples S. Let V be the set of Voronoi vertices.
2. Compute Delaunay triangulation of $S \cup V$.
3. The crust consists of the Delaunay edges between points of S.

Notice that the resulting crust is a subset of the Delaunay triangulation of the input points (Figure 5).

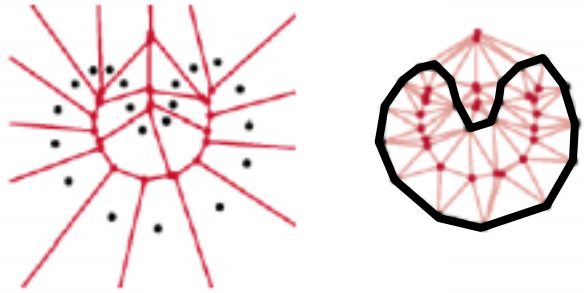

Figure 5. The two-dimensional crust algorithm. On the left, the Voronoi diagram of a point set S sampled from a curve. On the right, the Delaunay triangulation of $S \cup V$, with the crust edges in drawn in bold (figure from [2]).

The crust algorithm does not extend smoothly to 3D, but by estimating the normal of the sample points, it can be made to work. This gives Shape Tape data an advantage, since we are already given normal information.

While this algorithm may give us a good representation of the user's arms and legs, it may be too slow to use. The crust algorithm runs in $O(n^2)$ in the worst case, and all other steps are $O(n)$. Since Shape Tape doesn't produce many sample points, the $O(n^2)$ running time may not hurt our performance too badly. However, Amenta reports using an SGI Onyx and taking 2 minutes to compute the reconstruction of a 939 point sample. While this is obviously not currently suited for real time, we might analyze how much computing power we would need to gain real-time performance or if graphics hardware acceleration is possible.

5.3 Deformation by Virtual Exoskeleton

Hopefully, we can find a combination of the two ideas presented above, taking the best from both: accurate skin and real-time performance. A possible way to do this would be to use a system in a way similar to the skin and bones system, but instead of using an underlying bone structure to deform a stock mesh, we would use an invisible *exoskeleton* to deform a reconstructed mesh. This involves some offline computation:

1. Reconstruct a mesh of the user's arms and legs from the Shape Tape data.
2. Find Bezier control points for the mesh.
3. Associate certain Shape Tape sensors with the Bezier control points. The movement of these sensors will be responsible for deforming the mesh.

After this offline computation is done, we can use the system similar to the skin and bones system. As the user moves his arms and legs, the Shape Tape sensors associated with Bezier control points also move. In turn, we recalculate the mesh of our Bezier patch at each frame, render it and test collisions against it.

The advantage of this method is that we are using a reconstructed mesh computed offline, so our avatar looks like our user, but we are not reconstructing at every frame. We are not calculating a bone structure, so we eliminate the process of assigning bone weights to each vertex. We are using our tracker data to move the skin of the avatar directly, instead of using it to move the bones and deforming the skin procedurally. This means the skin of our avatar is deforming similar to how our user's skin is deforming, because the data actually comes from tracker readings on the surface of the user's skin. Additionally, since most computation is done offline, as in the skin and bones method, this technique is likely to run in real-time.

The difficult part of this algorithm, however, seems to be the computation of Step 2, automatically computing Bezier control points from a mesh. Usually in geometric modeling, we are given the control points and want to calculate a mesh. It is not clear if it always possible to perform the computation in reverse. Still, this method is promising since it may neatly combine our previous two solutions and show good performance.

6. CONCLUSION

We have discussed the problem of reconstructing, in real-time, a realistic avatar, in both motion and shape. We analyzed our input

data and desired output from the point of view of the fields of sampling/reconstruction theory, geometric modeling and computational geometry. We compared and contrasted our problem to problems from these fields, and analyzed how we can use our input data with algorithms from these fields to solve our problem. We presented three possible solutions based on our analysis. Two of these solutions, the skin and bones technique and the real-time meshing techniques are combined into a method, deformation by virtual exoskeleton, which shows promise.

7. REFERENCES

[1] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J**,** Stuetzle, W. "Surface Reconstruction from Unorganized Points", Proceedings of SIGGRAPH '92, pp. 71-78**.**

[2] Amenta, N., Bern, M. "Surface Reconstruction by Voronoi Filtering", *14th Annual ACM Symposium on Computational Geometry*, 1998, pp. 39-48.

[3] Machiraju, R, Yagel, R. "Reconstruction Error Characterization and Control: A Sampling Theory Approach", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, Num. 4, December 1996, pp. 364-378.

[4] Bruderlin, A., Williams, L. "Motion Signal Processing", Proceedings of Siggraph '95, pp. 97-104.

[5] Bharatkumar, A., Daigle, K., Cai, Q., Aggarwal, J. "Lower Limb Kinematics of Human Walking with the Medial Axis Transformation" In Workshop on Motion of Non-Rigid and Articulated Objects, Austin, Texas, USA, 1994

[6] Lander, J. "Skin Them Bones: Game Programming for the Web Generation". Appeared in Game Developer Magazine, May 1998.

[7] http://146.9.11.240/~casl/Projects/NASA/Yearly/2001/OldVersions/Shape%20Tape.doc