Deformation Sensitive Decimation

Alex Mohr * Michael Gleicher *

University of Wisconsin, Madison

Abstract

In computer graphics, many automatic methods for simplifying polygonal meshes have been developed. These techniques work very well for static meshes; however, they do not handle the case of deforming objects. Simplifying any single pose of a deforming object may destroy detail required to represent the shape in a different pose. We present an automatic technique for simplifying deforming character meshes that takes a set of examples as input and produces a set of simplified examples as output. The method preserves detail required for deformation and maintains connectivity information across the examples. This technique is applicable to many current skinning algorithms including example-driven techniques, fitting techniques and Linear Blend Skinning.

Keywords: interactive, skin, deformation, simplification

1 Introduction

Animated characters are often modeled at high resolution to capture fine structure and important detail. While there are situations where the full detail is required, there are also times when the full detail is not only unnecessary but also undesired. For example, interactive systems often gain performance by using lower resolution models when possible and offline systems can benefit from the memory savings lower resolution models provide when the details are unnecessary. Hence, many mesh simplification algorithms have been devised to automatically produce lower resolution versions of meshes. While there is a large range of mesh simplification algorithms, each targeted at different specific applications, none are easily applied to deforming objects.

Naïvely simplifying deforming meshes using a standard algorithm is error-prone. A simple method might simplify the mesh in a neutral pose. Unfortunately, a particular pose of a deforming mesh likely contains geometry that seems unnecessary, but is in fact required to properly represent the shape of the object in other poses. Consider the case of a bar shape that bends like an elbow as shown in Figure 1. If we simplify the bar shape in the neutral pose, the simplification algorithm destroys the mesh detail near the elbow joint that is required for that area to deform correctly.

The general problem is that any simplification of a deforming mesh must preserve the necessary geometry to adequately represent the deformed mesh in all poses. Since current simplification algorithms assume that their inputs are static meshes, they can only produce simplified meshes that adequately represent the input mesh.

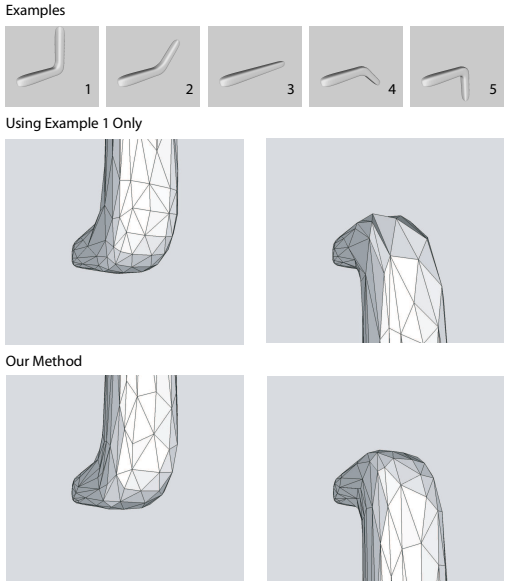

Figure 1: At the top are five examples of a bending bar. The middle row shows the result of using the standard QSlim collapse cost measure to simplify this mesh with Example 1 determining the collapse sequence. Example 1 is well approximated while Example 5 is not. The bottom row shows the results using our collapse cost measure. Notice that both extreme examples are well approximated.

In this paper, we present a mesh simplification technique for deforming objects. The algorithm is a generalization of the QSlim simplification algorithm [Garland and Heckbert 1997] that takes a set of examples of the mesh in different poses and produces a corresponding set of simplified meshes as output. The output maintains the geometric detail required to adequately represent the mesh deformations and preserves connectivity information across the meshes (only vertex positions differ). The output of our algorithm is directly applicable to many skinning mechanisms including mesh animation, Pose Space Deformation [Lewis et al. 2000], Shape By Example [Sloan et al. 2001], EigenSkin [Kry et al. 2002], Multi-Weight Enveloping [Wang and Phillips 2002], and the technique presented by Mohr and Gleicher [2003]. We also demonstrate how our method can be extended to work directly with Linear Blend Skins by including the correct vertex weights and influence sets for the reduced mesh in the output. This extension makes this technique easily applied to automatically building levels-of-detail for game characters.

The exposition is as follows. We begin with a review of related work in mesh simplification. Next we present our simplification algorithm and discuss its use with current skinning methods. Finally we present our results and discuss some applications of our technique.

* {amohr, gleicher} @cs.wisc.edu, http://www.cs.wisc.edu/graphics

2 Related Work

Mesh simplification is an important and well-studied problem in computer graphics. Many techniques for simplifying polygonal models have been presented. The methods vary primarily either because they trade speed for quality differently or because they include features required for some specific application. We present a brief overview.

Most simplification algorithms work by applying local mesh decimation operators. Some early techniques used a *vertex remove* operation. This involves removing a single vertex and its incident faces from a mesh and retriangulating the produced hole. Turk [1992], Schroeder et al. [1992], and Soucy and Laurendeau [1996] present methods that use this technique. While these techniques explicitly preserve the topology of the meshes they simplify, they are limited to operate on manifold surfaces.

Rossignac and Borrel [1993] and Luebke and Erikson [1997] make use of a *vertex cluster* simplification operator. The clustering operator works by placing a grid over the mesh to be simplified and then merging all vertices in a particular grid cell. A merged vertex position is often chosen either arbitrarily or to minimize some error term. This operator is unique in that it does not rely on any of the input mesh's connectivity information—only the vertex positions are considered. Simplification algorithms that use this operator trade off quality for speed—they are often very fast but typically produce poorer approximations than other methods.

Many popular simplification techniques apply a *pair contraction* operator to simplify meshes. Pair contraction works by replacing two vertices in the mesh by a single point and then removing any degenerate geometry. A special case of pair contraction only considers pairs of vertices linked by edges in the mesh. This is referred to as edge contraction. Pair contraction algorithms are quite popular. Some examples include the methods of Garland and Heckbert [1997], Ronfard and Rossignac [1996], Guéziec [1995] and Hoppe [1996; 1993]. The pair contraction operator is attractive since it is the finest grained local mesh decimation operator. Most often it removes just one vertex and two faces from a mesh.

3 Simplification Algorithm

Our simplification algorithm for deforming meshes generalizes the QSlim algorithm [Garland and Heckbert 1997] for simplifying static meshes. We chose this algorithm for several reasons. It is widely used, relatively straightforward to implement, and it makes a good tradeoff between simplification speed and quality.

QSlim works by iteratively applying pair contractions to a mesh. The key elements of this algorithm lie in the order of vertex pair collapses and the placement of vertices following pair collapses. While we present a brief overview of the QSlim algorithm, we leave the full description to [Garland and Heckbert 1997] and focus on our contributions.

The algorithm proceeds in two main stages—an initialization stage followed by the decimation stage. The initialization stage first determines the set of candidate pairs that may potentially be contracted throughout simplification. This set typically includes all the edges and some non-edge pairs that are close together. Then a *collapse cost* is computed for each candidate pair. Each candidate pair is placed in a priority queue keyed on its cost. During the simplification stage, the algorithm selects the candidate pair with lowest cost and removes it from the priority queue. This pair is then collapsed, and the mesh data structures are updated to remove any degenerate geometry. All affected pairs' costs in the local neighborhood are reevaluated, and this simplification stage is iterated until the desired approximation is reached.

3.1 Measuring Contraction Cost

Good contraction cost measurements are fundamental to the performance of our simplification algorithm since it determines the order of pair collapses. To evaluate the cost of contraction, we would like to measure the impact of the contraction on the approximation. QSlim uses a quadric error metric to measure this approximation error. This is done by associating a set of planes with each vertex during the initialization phase. These planes are simply those defined by the planes of the polygons incident on that vertex. From these planes, an *error quadric* matrix $\mathbf{Q}$ is created such that $\mathbf{p}^{\mathrm{T}}\mathbf{Q}\mathbf{p}$ is the sum of squared distances to each of the planes for some point $\mathbf{p}$. See [Garland and Heckbert 1997] for details.

Using these error quadrics, the cost of a pair contraction is computed as follows. First, assume for now that we have already determined the placement of the resulting vertex after pair contraction. Vertex placement is discussed in Section 3.2. The cost of contraction then is simply computed as $\mathbf{v}^{\mathrm{T}}(\mathbf{Q}_1+\mathbf{Q}_2)\mathbf{v}$ where $\mathbf{v}$ is the new vertex position and $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are the error quadrics from each of the collapsed vertices. This sum formulation can count some planes up to three times, potentially giving them more weight than others, but sacrificing this accuracy allows us to simply store a single $\mathbf{Q}$ matrix for each vertex rather than tracking the set of planes explicitly.

While this error metric works well for a static mesh, it alone does not work for a deforming mesh. Consider the arm shown in Figure 3. This object requires substantial mesh detail around the bending joint in order to capture the correct shape in all poses. If we were to simply apply the QSlim contraction cost measure using a single example, we would only be assured of a good approximation of that example. We would like the pair contraction cost measure to account for the deformations of the object over all poses.

Our algorithm generalizes the contraction cost measure by considering all the example meshes simultaneously. We do this by tracking one error quadric per vertex for each example and summing the independent collapse costs for each mesh in the set of examples. That is, the collapse cost for a pair becomes

$$\sum_{i=1}^{k} \tilde{\mathbf{v}}_i^{\mathrm{T}}(\mathbf{Q}_{i,\mathbf{v}_1}+\mathbf{Q}_{i,\mathbf{v}_2})\tilde{\mathbf{v}}_i$$

where k is the number of examples, $\tilde{\mathbf{v}}_i$ is the new vertex position for the collapse of vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ in the ith example, and $\mathbf{Q}_{i,\mathbf{v}_n}$ is the error quadric associated with vertex $\mathbf{v}_n$ in example i.

This formulation correctly weights those pairs that are beneficial to approximating the mesh in all poses. Even so, it relies on a good sampling of the character's deformations as input. One might imagine using an automatic procedure to sample the deformations of a character over all its degrees of freedom. Unfortunately this may not work well since typically only a very small set of character configurations produce valid poses compared to the entire possible pose space. Instead, we rely on the author to provide a set of examples of the character in valid extreme poses, similar to a model sheet or the input required for data-driven skinning algorithms.

This formulation also provides authors a measure of control over the simplification. If there is some pose that must be approximated well, the author can provide many examples of the character in or near that pose, or a weight may be added to each example in the summation.

3.2 Placement Policy

To evaluate the cost of collapsing a vertex pair and to perform the actual collapse operation, a new vertex position must be computed. There are several different possible placement policies, each with a different impact on the resulting simplified mesh.

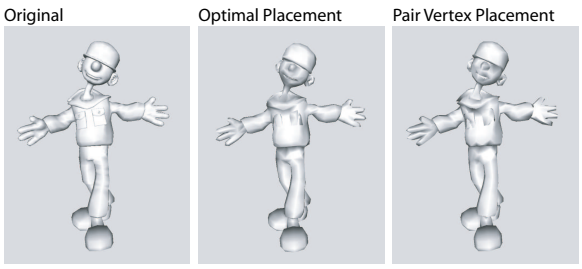

Figure 2: On the left is an example of a Linear Blend skinned character. On the middle and right are simplifications of this mesh using different placement techniques. The middle uses optimal placement while the right chooses a collapsing pair vertex for new positions. While the difference is noticeable in the face, around the arms, and at the bottom of the shirt, the performance advantage of the rightmost skin may outweigh any approximation advantage.

The most desirable placement policy, optimal vertex placement, works by finding the point that minimizes the summed quadric error metric. To do this, the last row of the pair's summed quadric matrix $\mathbf{Q}$ is replaced by $[0,0,0,1]$ to handle homogeneous coordinates. Next, the inverse $\mathbf{Q}^{-1}$ is computed, and its last column is extracted as the optimal vertex position [Garland and Heckbert 1997].

Unfortunately, this inversion is not always possible, and in the case of singular matrices, other policies must be employed. One common case where these matrices are singular is when the geometry is planar. To handle these cases, one may attempt to find the optimal position along the line through the collapsing pair, or to choose the position of one of the two collapsing vertices with smallest error. Another possibility is to arbitrarily choose a point such as the midpoint of the line segment connecting the pair.

QSlim employs optimal vertex placement when possible and falls back on other placement policies when it fails. Like QSlim, our algorithm chooses the vertex placement for each example mesh independently. Since our algorithm handles deforming meshes instead of just static meshes, we cannot use optimal vertex placement in all cases. When the matrix inversion is nearly singular, optimal placement is extremely sensitive to numerical precision. If the input to our algorithm contains several poses nearby in pose space and some quadric error matrices are nearly singular, optimal placement can introduce noise in the simplified meshes from pose to pose. This is not an issue for QSlim since it operates on a single mesh.

Despite the possible singularity issues, our method defaults to optimal vertex placement and falls back on endpoint placement policies when necessary. If a user has difficulty with this policy, we allow them to override it and choose a different policy instead.

4 Application to Skinning Algorithms

In order to ensure that our simplification algorithm is effective for deforming characters, we show how it may be used with several current skinning algorithms including example-driven techniques, fitting techniques, and Linear Blend Skinning.

4.1 Example Driven Techniques

There are many different example driven skinning methods, including both those used in practice and recent research results. The output of our simplification method is almost ideally suited to these methods since the connectivity information is the same for all input and output meshes. For example, mesh animation, which simply stores all possible frames of animation and plays them back at runtime, can make direct use of our algorithm's output. This animation technique is commonly used in games, especially for non-character objects.

Another popular example driven technique is called "blend shapes". This method works by interpolating the vertices of key poses to produce animation. Blend shape animation is often used for facial animation where simple interpolation of vertices works well. Our technique is easily applied to blend shapes. The blend shape meshes themselves are passed to our algorithm as input, and the resulting simplified blend shapes can be used in place of the originals. Again, this works since our algorithm preserves connectivity correspondence across all meshes.

There has been much recent research on example driven techniques as well. Example driven techniques typically use sparse data interpolation of corrections to an approximate skin to compute deformations. Some recent examples include Pose Space Deformation [Lewis et al. 2000], Shape By Example [Sloan et al. 2001] and EigenSkin [Kry et al. 2002]. Since our method produces a simplified set of examples, it is a natural fit for these skinning methods. While these techniques often require some user intervention to build the approximate skins, the difficult step of supplying the correct examples is handled by our algorithm.

4.2 Fitting Techniques

More recently in the research community, fitting techniques for character skinning have been presented. These methods include Multi-Weight Enveloping [Wang and Phillips 2002] and the technique of Mohr and Gleicher [2003]. These methods work by taking a set of example data as input and fitting the parameters of a skinning model using the data.

These techniques are used to produce fast to evaluate and compactly represented approximate skins from a set of examples generated from a complex, high-end rigged character. Our simplification technique may be used for this application by first simplifying all the high resolution examples before they are passed to the skin fitting algorithm.

4.3 Linear Blend Skinning

An important skinning algorithm widely used in interactive systems has many names including Linear Blend Skinning, Skeleton Subspace Deformation [Lewis et al. 2000], Enveloping [Wang and Phillips 2002], and Smooth Skinning. This technique works by assigning each vertex a set of influencing joints and a weight for each of these influences. A deformed vertex is then computed by blending the transformation matrices of influencing joints according to the weights and applying this blended matrix to the vertex.

Since interactive applications benefit most from our simplification technique, it is important that it be applicable to this skinning method. Although recent work presents methods for building these skins by data fitting as described in Section 4.2, they are not usually constructed in this manner. Instead, artists hand-craft these skins using animation systems. Fortunately, our algorithm can still accommodate these skins.

To simplify a Linear Blend Skin using our system without resorting to the fitting techniques described earlier, we must devise some method for determining the proper influence sets and weights for the final simplified skin. One technique might be to simply collapse pairs as normal, but assign the union of the collapsing pair's influence sets to the newly created vertex. Then at the end, a simplified fitting technique could solve for the proper vertex weights.

Unfortunately, the merging of influence sets in such an approach can be undesirable for several reasons. First, the performance of a linear blend skin depends heavily on the size of the influence sets and as vertices are merged, the influence sets grow. The sets could get quite large for low levels of detail. Moreover, current graphics hardware skinning implementations typically only permit a limited number of influences per vertex.

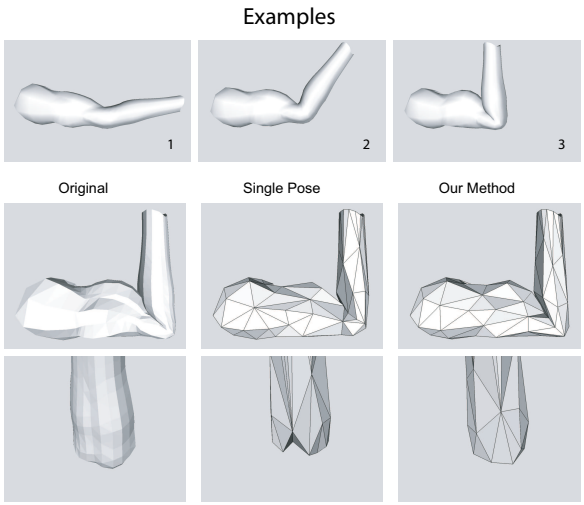

Figure 3: A muscular arm. At the top are three of the input examples. The column labelled "Single Pose" shows the results of using the QSlim cost collapse measure using example 1 to determine the collapse sequence. Notice the loss of volume in the bicep and the artifacts around the elbow. The last column shows the results of using our collapse cost measure. Notice how the overall mesh shape is well approximated.

To handle these problems, we can sacrifice optimal vertex placement during simplification. Instead, we choose one of the collapsing pair vertices' original positions for the collapsed position at every step. This can produce somewhat poorer approximations [Garland and Heckbert 1997]. However, since we are choosing an existing vertex at each collapse step, the influence sets cannot change size and the correct weights are already determined. Such a method provides a simple technique for automatically building levels-of-detail for deforming meshes. While the mesh approximations can be worse using this method, the performance guarantee and implementation simplicity may outweigh the quality difference.

5 Results and Discussion

We have implemented our simplification algorithm to evaluate the results of our technique. We demonstrate our method on several deforming objects, including contrived examples to show particular qualities of our results and objects more typical in a real-world application of this technique.

As shown earlier, Figure 1 demonstrates how our technique can preserve the mesh detail required for accurate deformations in all poses while naïvely applying a simplification algorithm to a single mesh cannot. In this figure, the bending bar requires mesh detail near the bend in order to accurately capture the shape as the bar bends. By using a set of examples, our method accurately preserves this detail, while applying the simplification algorithm to any single example does not.

Figure 3 shows our technique applied to a bending arm with an exaggerated bicep bulge. Simplification using the a single example destroys detail necessary around the elbow while our technique well approximates the shape in all poses.

Our method of vertex placement for producing a Linear Blend Skin is shown in Figure 2. This character has been simplified while neither increasing the influence set sizes nor changing the weights of the remaining vertices as described in Section 4.3. As mentioned earlier, the approximation quality is somewhat lower than it would be with a different placement strategy but the guarantees regarding influence set size and weight preservation may be more valuable.

While our method slows as the number of examples increases, this slowdown is linear in the number of examples and has proved to be acceptable in our tests. Many of the results shown here supplied a set of 50 examples with up to 15,000 faces to our implementation and no simplification took longer than five minutes on a Pentium 4 laptop.

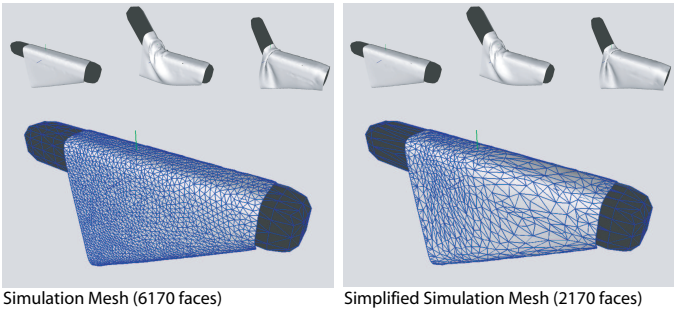

Figure 4: At the left is a mesh used to perform a cloth simulation. The cloth has high resolution because the simulator requires it. However, after simulation is complete, our technique can simplify the resulting meshes for faster runtime playback. Our algorithm maintains detail around the wrinkling parts of the cloth. Notice the varied density of geometry in the figure at the right.

Finally, we present another useful application of our algorithm. Cloth simulation usually requires a large amount of detail in the cloth mesh for accurate results. However, once simulation is complete, the resulting meshes may be simplified by our method as shown in Figure 4. Our algorithm preserves detail where the cloth wrinkles but removes it where the cloth stays flat.

In this paper, we have presented a generalization of the QSlim surface simplification algorithm that works well with deforming meshes. Our method is easy to implement and provides good results. The output of our simplification algorithm may be used with a wide variety of current skinning algorithms including mesh animation, blend shapes, Shape By Example, Pose Space Deformation, Multi-Weight Enveloping, the Mohr and Gleicher fitting technique, and Linear Blend Skinning. We believe that this deformation sensitive decimation will be very useful for automatically generating levels of detail for deforming characters.

References

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*.

GUÉZIEC, A. 1995. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *Proceedings of SIGGRAPH 93*, 19–26.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of SIGGRAPH 96*, 99–108.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: Real time large deformation character skinning in hardware. In *ACM SIGGRAPH Symposium on Computer Animation*, 153–160.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH 2000*.

LUEBKE, D., AND ERIKSON, C. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH 97*, 199–208.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*.

RONFARD, R., AND ROSSIGNAC, J. 1996. Full-range approximation of triangulated polyhedra. In *Computer Graphics Forum*, vol. 15, 67–76.

ROSSIGNAC, J., AND BORREL, P. 1993. Multi-resolution 3d approximations for rendering complex scenes.

SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26.

SLOAN, P.-P. J., CHARLES F. ROSE, I., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, 135–143.

SOUCY, M., AND LAURENDEAU, D. 1996. Multiresolution surface modeling based on hierarchical triangulation. In *Computer Vision and Image Understanding*.

TURK, G. 1992. Re-tiling polygonal surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 55–64.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *ACM SIGGRAPH Symposium on Computer Animation*, 129–138.