Pose-independent Simplification of Articulated Meshes

Christopher DeCoro[*] Szymon Rusinkiewicz[†]

Computer Graphics Laboratory
Princeton University

Abstract

Methods for triangle mesh decimation are common; however, most existing techniques operate only on static geometry. In this paper, we present a view- and pose-independent method for the automatic simplification of skeletally articulated meshes. Such meshes have associated kinematic skeletons that are used to control their deformation, with the position of each vertex influenced by a linear combination of bone transformations. Our method extends the commonly-used quadric error metric by incorporating knowledge of potential poses into a probability function. We minimize the average error of the deforming mesh over all possible configurations, weighted by the probability. This is possible by transforming the quadrics from each configuration into a common coordinate system. Our simplification algorithm runs as a preprocess, and the resulting meshes can be seamlessly integrated into existing systems. We demonstrate the effectiveness of this approach for generating highly-simplified models while preserving necessary detail in deforming regions near joints.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms

Keywords: mesh simplification, level of detail, kinematic skeletons, articulated meshes

1 Introduction

The problem of mesh simplification – reducing the number of polygons in a mesh while preserving visual and geometric detail – has been studied extensively. Most researchers, however, have considered the input mesh to be rigid. In this paper, we present an algorithm that addresses the additional considerations that are required to allow for automatic simplification of meshes deformed using an articulated skeleton, which are commonly referred to as skinned meshes. Most commercial games make extensive use of such models, and they are frequently used in feature animation.

A key observation that lays the foundation for our algorithm is that given the advances in GPU speed, it is not practical to spend extensive time during execution to chose an optimal view-dependent level of detail. In many cases, it is preferable to pass additional, unnecessary polygons to the graphics card, in favor of spending less work on the CPU. In line with this observation, we look to find a view-independent simplification method that shifts as much work as possible into the precomputation stage. However, as we will see, nothing in our method prevents its incorporation into a view-dependent system, in much the same way that other view-independent simplification techniques have been used as the initial simplification phase for view-dependent refinement.

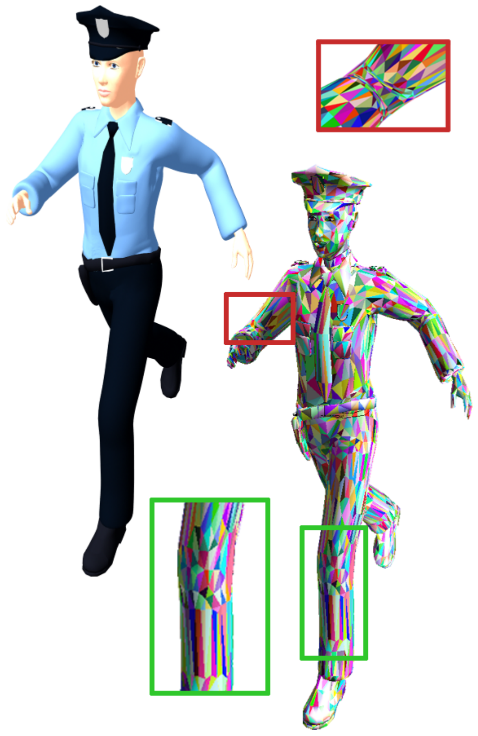
Figure 1: A full resolution model of a police officer is shown on the left, with its simplified counterpart on the right. Faces are randomly colored, in order to clearly show the triangulation. Note that the simplified model maintains high detail around joints, such as the elbows, while detail in the rest of the model is reduced.

A key challenge of simplifying deforming models is that, by definition, they change configurations at runtime. If a mesh can be deformed arbitrarily, it will be impossible to pre-construct any mesh that will be acceptable in all deformations. Our solution is to limit the class of deformable meshes to those that are skeletally articulated - a reasonable assumption given the large number of figures that are animated in this manner, especially in interactive applications. The technique we use is often referred to as linear blend skinning, and we discuss this in further detail in Section 3.1. Further, we associate a probability function with the skeleton that provides the relative likelihood of each possible configuration. This provides enough limitations on deformation to allow our algorithm to find a single mesh approximation that minimizes the geometric error over the corresponding probability distribution. In this paper

[*]e-mail:cdecoro@cs.princeton.edu
[†]e-mail:smr@cs.princeton.edu

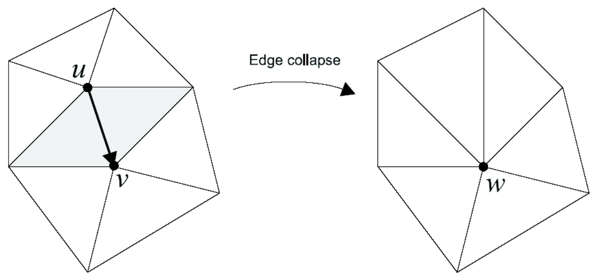

Figure 2: *Edge Collapse. The edge (u, v) is contracted to w. The mesh now has one fewer vertex, and two fewer faces.*

we propose a pose-independent metric for simplification of articulated meshes, and will demonstrate its use in a view-independent simplification framework, although the metric itself is not tied to any particular framework. We define the error to be minimized as the root mean squared error of a model over all configurations of the skeleton. This algorithm extends the QSlim algorithm of [Garland and Heckbert 1997] by adding additional information on possible configurations, which constrains simplification in deforming areas, even when those areas are ideal for simplification in the initial pose. Our main contribution is to show that given a linear deformation model, we can encapsulate knowledge of all poses in a single pose-independent quadric. Like QSlim, we approximate model-to-model error by minimizing point-to-plane distance over all edge contractions. Figure 1 shows an example of a posed police officer model that has been simplified using our method. As can be seen in the simplified version on the right, the deforming regions at the joints have been preserved in higher detail than the surrounding areas, resulting in a smooth bend in those areas.

2 Background and Related Work

Our work is based on the QSlim algorithm, which utilizes the quadric error metric (QEM). This was introduced in [Garland and Heckbert 1997], and analyzed in greater detail in [Garland 1998], and our algorithm will be presented as a modification of QSlim. Many other simplification methods exist, such as [Schroeder et al. 1992] and [Hoppe et al. 1993].

These basic simplification approaches have been applied to a wide range of runtime applications, including both view-independent (see [Hoppe 1996]) and view-dependent (see [Hoppe 1997]) dynamic simplification. This, in turn, has led to such applications as external-memory simplification [Lindstrom 2000]. The quadric metric we use has also been adapted to a wider range of input data, such as meshes with material properties (see [Garland and Heckbert 1998] and [Hoppe 1999]). The user-guided simplification of [Kho and Garland 2003] allows for greater control over detail in user-specified areas. Other methods, such as [Lindstrom and Turk 2000] have looked to visual metrics for determining error, rather than the geometric error used in previous simplification systems.

2.1 Quadric Error Metric

In the QSlim algorithm, simplification is performed by building an error function defined on a vertex and an associated set of planes. The algorithm then iteratively contracts (or collapses) the edges with lowest error by selecting candidates from a priority queue. Each collapse will decrease the number of vertices in the mesh by one, and will usually remove two faces, as in Figure 2.

QSlim minimizes the function $d(v)$, which gives the squared distance of a vertex v from a set of planes adjacent to v, where each plane p through point a with normal n is represented as $[n_x, n_y, n_z, -n \cdot a]'$.

$$\begin{aligned} d(v) &= \sum_{p \in \text{planes}(v)} (p^T v)^2 && (1) \\ &= \sum_{p \in \text{planes}(v)} v^T (pp^T) v && (2) \\ &= v^T \left(\sum_{p \in \text{planes}(v)} pp^T \right) v && (3) \end{aligned}$$

We then define the vertex quadric Q_v as the sum of the outer product of the planes adjacent to v, and use this to represent $d(v)$ for a vertex v and its associated quadric.

$$\begin{aligned} Q_v &= \sum_{p \in \text{planes}(v)} pp^T && (4) \\ d(v) &= v^T Q_v v && (5) \end{aligned}$$

For the initial vertices in the mesh, $d(v)$ will be zero, as is intuitive. However, when two vertices u and v are considered for contraction to a new vertex w, the error quadric follows the additive rule $Q_w = Q_u + Q_v$, and error is computed as $w^T Q_w w$. Note that it is possible under certain conditions to directly solve for the optimal value of w, as discussed in [Garland and Heckbert 1997].

Thus, the overall algorithm is as follows:

1. Compute the quadric corresponding to each vertex
2. Determine the contraction cost for each edge
3. Place the edges in a min priority queue sorted on contraction cost
4. Remove the edge (u, v) with the lowest contraction cost from the priority queue
5. Use the quadric to determine the optimal contraction target of (u, v)
6. Contract u and v, and update the cost of all edges adjacent to u and v.
7. Repeat steps 4 through 6 until the desired mesh resolution is reached.

As will be explained in Section 3, our algorithm makes modifications to the initial quadric computation of Step 1, and makes additional changes in the iterative contraction stage of Step 6.

2.2 Deformation Sensitive Decimation

The previous work closest to our method is the Deformation Sensitive Decimation algorithm proposed in [Mohr and Gleicher 2003]. The system is presented with a model to be simplified in k different example poses, where each pose has the same connectivity, and there exists a one-to-one mapping between vertices of each example.

In the initial quadric computation phase, a vertex quadric $Q_{i,v}$ is generated for each vertex v in each pose i, using the same method as as QSlim, resulting in a total of kv quadrics.

During iterative contraction, the collapse cost of an edge (v_1, v_2) is equal to the sum of contraction errors in all poses:

$$\sum_{i=1}^{k} \tilde{v}_i^T (Q_{i,v_1} + Q_{i,v_2}) \tilde{v}_i \qquad (6)$$

where $\tilde{v}_i$ is the contracted position of the vertices v_1 and v_2. This will have the effect of penalizing contractions that are unfavorable in certain configurations. Contractions that are favorable in all configurations will be performed first.

We analyze the running time of iterative contraction as follows: for each contraction, $O(\log v)$ operations are required to select the minimum contraction candidate, followed by $O(k)$ operations to recompute contraction error in the neighborhood of the contracted pair. As the number of contractions is bounded by the total number of vertices v, the running time of the iterative contraction phase is $O(vk + v \log v)$.

Deformation Sensitive Decimation is shown to be a useful and robust algorithm for simplification of animated models. It is important to distinguish the trade-offs between this method and the method we are about to propose. First, DSD is more general in that is supports arbitrarily deformation methods. Our system is limited to linear-blend skinned models (see Section 3.1); however, given the ubiquitous nature of such models, this is a reasonable constraint for many application areas, especially those of an interactive nature, such as computer games.

Additionally, as we will see in Section 3.2, the iterative contraction phase of our algorithm runs in time independent of the number of sample poses, or only $O(v \log v)$. Also, our method differs from DSD in that we propose a method for automated sampling of example configurations, and only ask the user to provide an approximate probability distribution, rather than requiring the user to explicitly provide examples. Finally, we also propose a method for updating influence weight values after an edge has been collapsed to a new vertex that was not present in the original mesh.

2.3 Skeletally Articulated Simplification

Other researchers have developed user-assisted methods for simplification of skeletally articulated meshes. In contrast to these methods, our approach determines the amount of deformation directly, without requiring manual marking, which not only saves artist time but also allows greater simplification in regions around a joint that are not significantly deformed, such as the sides of a knee.

In [Houle and Poulin 2001], the authors propose simplifying a static pose of the articulated figure. A method is developed for propagating the simplification changes, which are in world space, back into bone space (the local coordinate system of each bone). During iterative contraction, this method assumes that the influence weights (used to control deformation of the model, as explained in Section 3.1) will remain constant throughout simplification, which may or may not be an acceptable approximation.

Another method, which requires the user to manually divide the object into deformable and non-deformable regions, is presented in [Schmalstieg and Fuhrmann 1999]. The deformable regions are preserved to a greater extent than the non-deformable regions. Unlike the previous paper, the authors do not make the assumption that influence weights remain constant throughout simplification. Rather, they provide a method to update the weights based on a shortest-path graph traversal from a deformable node to adjacent non-deformable regions, and assumes that the closer a vertex lies to a bone, the stronger the influence will be. This approach has acceptable results for many models. However, we consider the influence weights to be an essential artistic product of modeling and content creation. Weights determine the shape and size of creases and the overall look of deforming bodies, and automated methods for reassigning weights may not preserve the artist's intent. Thus we have developed an approach that approximates the original weights instead of computing new weights.

Others such as [Shamir et al. 2000] and [Shamir and Pascucci 2001] have proposed simplification of meshes that undergo arbitrary deformation as a function of time. Generally, these are specified as a sequence of frames, and are not subject to constraints such as a skeletal framework. This is similar in principle to our problem; however, those methods have the benefit of being provided all possible frames up front, while our algorithm is capable of handling arbitrary skeleton configurations. Additionally, these methods focus on runtime simplification, with an auxiliary set of data structures such as the T-DAG of [Shamir et al. 2000] precomputed offline. This is required due to the problem of arbitrary deformations, and thus differs significantly from our approach.

Finally, a number of methods have been presented for Skeletal Level of Detail, such as that presented in [Teichmann and Teller 1998]. This is in fact a different problem than the one we are addressing: the skeleton itself is simplified, so as to reduce the amount of skinning computation, but the skin may or may not be simplified. Our system directly addresses the simplification of the skin, not the skeleton. However, an indirect benefit of our system is that as vertices are contracted, certain bones will lose influence over skin triangles. A natural extension would therefore be to perform Skeletal LOD by removing bones, fusing them with their parent once the number of influenced skin triangles has dropped below a certain threshold.

3 Algorithm Specification

Our algorithm can be conceptualized as a set of modifications to the QEM algorithm, and this section will provide the details of those changes. First, in Section 3.1, we will formally specify the input to our algorithm and state relevant assumptions, including the skinning model. Then, we introduce our alternative initial quadric computation step in Section 3.2. In Section 3.3, we discuss the characterization of joints. We then detail our modifications to the iterative contraction step, in Section 3.4.

3.1 Problem Domain

We define our problem as follows. We are given a skeletally articulated mesh $M = (V, F, B)$. V is a set of vertices $v \in \mathbb{R}^3$, each of which has an associated weight vector $[w_{v,1}, \cdots, w_{v,|B|}] \in \mathbb{R}^{|B|}$ that represents the bone influence weights on v. The set of faces $F \subset V \times V \times V$, constitutes the triangle faces that make up the mesh surface of the model.

B is the set of bones in the model skeleton. Each bone $b \in B$ is defined as $b = (M_b, \rho_b(M_b), p_b)$. The matrix $M_b \in R^{3\times 4}$ is the affine transformation induced by the bone, relative to the coordinate system of its parent, $p_b \in B \cup \{\emptyset\}$, where $p_b = \emptyset$ indicates that b is the root bone in the system. This can also be thought of as defining a local coordinate system of b.

The probability of a bone b assuming a particular configuration of M_b is given by the bone probability function $\rho_b(M_b)$, $\rho_b : R^{12} \rightarrow \mathbb{R}^+$, which is defined . The function ρ_b will be nonzero for affine transformation matrices representing valid configurations for b. Frequently, ρ_b can be as simple as a box function indicating minimum and maximum values along each dimension, though greater control can be achieved when a more accurate probability function is used, such as one obtained using motion capture (see Section 3.3).

Although most general, it will rarely be intuitive to consider specification of ρ as a function of R^{12}. For the purpose of user specification of probability functions, we find it useful to perform a change of variables, and represent $\rho_b(M_b)$ instead as $\rho(t_b, \theta_b, s_b)$. Each of t_b, θ_b, and s_b are vectors in $\mathbb{R}^3$, and represent respectively the translation, rotation, and scale of the bone b about each of the three canonical axes in the coordinate system of its parent, p_b. This corresponds closer to the intuitive notion of a bone, in which an artist will conceptualize a bone as rotating a certain number of degrees, or translating to a given position. Using these parameters, we can represent M_b as a product of standard translation, rotation, and scaling matrices.

$$M_b = T(t_b)R(\theta_b)S(s_b)$$

Our method is not limited to this choice of reparameterization, though it has been shown to be useful and intuitive, and could easily be extended to a more general form, such as one that includes shear parameters.

The mesh will deform according to the configurations of the bones and the corresponding influence weights of each vertex. We can define a pose P as representing a particular configuration of the skeleton, including all bone parameters, and denote the initial pose P_0. For any bone, we can perform a preordered traversal of the skeleton tree to generate a matrix $M_{\text{parent}(b)}(P)$ that represents the coordinate system of the parent of b, using pose P. Note that for the root bone, this matrix is the identity.

The vertices in V are specified in the world coordinate system. To perform skinning, a vertex v is transformed into the coordinate system of a bone in the initial pose, by applying the matrix $N_b = (M_{\text{parent}(b)}(P_0)M_b(P_0))^{-1}$, as computed for the initial pose of the model. From this coordinate system, we can then transform the vertex into the transformed coordinate system of a bone by the application of $M_{\text{parent}(b)}(P)M_b(P)$, as evaluated in the current pose P.

We combine, or blend, the influences of all bones in the system by taking a linear combination of transforms, where each bone transform is weighted by its corresponding vertex influence weight. This allows for smooth creases and bends in the surface of the mesh. Thus, the transformation of a vertex from initial world position to the skinned world coordinates of a particular pose P is:

$$M_v(P) = \sum_{b \in B} w_{v,b} M_{\text{parent}(b)}(P)M_b(P)N_b. \qquad (7)$$

3.2 Initial Quadric Computation

In QSlim, the first phase is to compute the initial vertex quadrics, Q_v for each vertex v. In our algorithm, we modify this to define a *pose-independent quadric*, which encapsulates knowledge of deformations over all poses, in addition to the configuration of the static initial mesh.

Consider the "leg" model in Figure 3. When the joint is rotated 90 degrees, the bottom of the knee will be deformed sharply. However, an ideal result would allow for drastic simplification away from the knee, while preserving detail at the knee itself. In addition, if the knee can only bend 90 degrees down but cannot bend upwards, the bottom of the knee will be more significantly deformed than the top, which is rounded rather than creased, and thus we desire greater preservation of detail on the lower side.

To accomplish this, we must incorporate the planes of the deformed configurations into the vertex quadrics. For a vertex v and a pose P, we can define the quadric-valued function $Q_v(P)$, which gives the quadric of v when the skeleton is put into the configuration defined by P. We compute this by recomputing all the M_b matrices accordingly, skinning the mesh, then computing the quadric of v using the deformed coordinates rather than the reference coordinates.

In QSlim, additional constraint planes are introduced by adding quadrics. After joint deformation, however, we must not simply add together the vertex quadrics from each of the resultant deformed meshes. Vertex quadrics are variant under affine transformation,

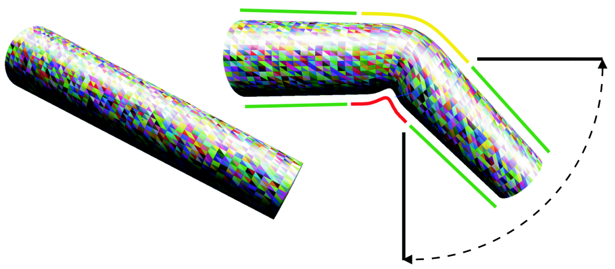

Figure 3: *Leg model, in the reference configuration and a deformed configuration. The black dotted line indicates the range of motion (i.e, the range for which ρ_b is nonzero). We can see intuitively that the sections highlighted in green should have the highest amount of simplification, the yellow section above the knee should retain more detail, and the red section below the knee requires the greatest level of detail.*

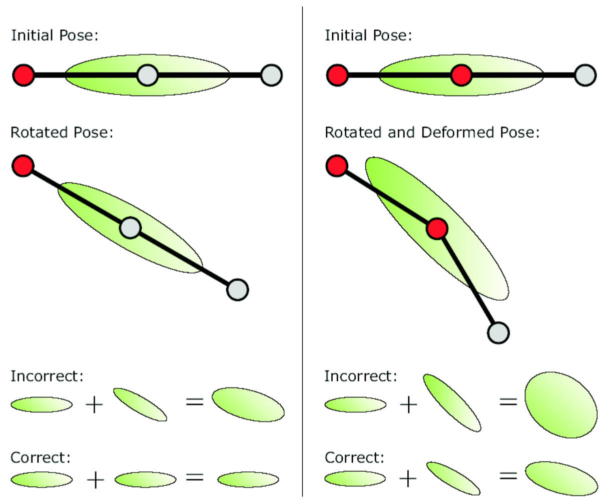

Figure 4: *Combining Quadrics. In the left column, the left joint is rotated 30° from the initial pose, while the area around the middle vertex has not changed. Simply adding the quadrics at the middle joint gives the wrong result, so the quadric from the rotated pose must be rotated back to the initial pose. In the right column, both the left and middle joints have been rotated, causing a deformation. The deformed quadric must be corrected to remove the effect of rotation by the left joint, then combined with that of the initial pose.*

as will occur when the bones are placed into a new configuration, even without deformation of the surrounding neighborhood. As an example, consider the vertices at the right end of the leg in Figure 3. Their quadrics in the deformed pose will be at an angle to the quadrics in the original pose, which when summed together would result in a quadric with a high collapse penalty. However, these triangles are in fact good candidates for simplification, as the local curvature of the surface does not deform and the triangles are coplanar.

Instead, we map the vertices into a common configuration, that of the original pose, which we will call the reference coordinate system. Because a vertex v_i in pose P_i is known to be transformed by $M_v(P_i)$, the inverse $M_v^{-1}(P_i)$ will transform v_i to v in reference coordinates.

We can then write the error of a single vertex v over all poses in our notation, substitute $v_i = M_v(P_i)v$, then factor v out of the summation.

$$
\begin{aligned}
d(v) &= \sum_{i=1}^{k} v_i^T Q_v(P_i) v_i && (8)\\
&= \sum_{i=1}^{k} v^T M_v(P_i)^T Q_v(P_i) M_v(P_i) v && (9)\\
&= v^T \left(\sum_{i=1}^{k} M_v(P_i)^T Q_v(P_i) M_v(P_i) \right) v && (10)
\end{aligned}
$$

The summation term of Equation 10 is a quadric, which we refer to as the pose-independent quadric Q_v. This can be computed at initial quadric computation time, and then used as with quadrics in standard QSlim. This derivation has a natural geometric interpretation as shown in Figure 4. In general, to transform a quadric Q into Q' through a transformation M, we have from [Garland 1998] the rule:

$$Q' \leftarrow (M^{-1})^T Q (M^{-1}) \tag{11}$$

When we transform v by $M_v^{-1}(P_i)$, the corresponding transformation of the quadric $Q_v(P_i)$ becomes

$$Q_v = (M_v(P))^T Q_v(P) M_v(P) \tag{12}$$

as in Equation 10. We can then consider the application of $M_v^{-1}(P_i)$ as removing the effect of bone transformation on the quadric $Q_v(P_i)$, and isolating the effect due to deformation of the surface. This transforms $Q_v(P_i)$ to the reference coordinate system, where the quadrics can be combined. Because the bone transformation is linear, it is easily invertible. This may not be possible with more general, non-linear animation.

Because $Q_v(P)$ is a continuous function, the sum of quadrics for all valid poses is equivalent to integration over the domain.

$$Q_v = \int_{\text{valid } P} \rho(P) M_v(P)^T Q_v(P) M_v(P) dP \tag{13}$$

The function $\rho(P) = \prod_{b \in B} \rho_b(P)$, which is the probability of the current pose, is written as the product of individual bone probabilities. We now have an integral with dimensionality up to $12|B|$. Clearly, direct quadrature methods are not applicable in this situation. Instead, we apply a Monte Carlo integration technique, which consists of replacing the evaluation of the integral with a sum over random configurations. To reduce variance, we use the Recursive Stratified Sampling technique as discussed in [Press et al. 2002]. In this method, we consider the domain to be sampled with N samples as a rectangular parallelepiped $R = (x_a, x_b)$, which is specified by two opposite corners of the region. We then bisect R along the longest dimension i, producing two subregions of equal size.

$$R_{ai} = (x_a, x_b - \frac{1}{2} e_i \cdot (x_b - x_a) e_i)$$

$$R_{bi} = (x_a + \frac{1}{2} e_i \cdot (x_b - x_a) e_i, x_b)$$

The vector e_i is the unit vector in the i-th coordinate direction. Each region is then allocated $N/2$ samples. When the number of samples allocated to a region falls below some threshold N_0, we uniformly sample in the region. We then use the samples to compute a Monte Carlo approximation to the integral for Q_v.

Note that given the reparameterization from R^{12} into translation, rotation and scale, the variables are no longer independent, as the rotational parameters θ_b sample a spherical function. The sampling pattern is required to take this into account, as in [Arvo 1995] so as to avoid an unbiased estimator.

This quadric Q_v now encapsulates the deformation of vertex v, and can be used in the quadric error metric as usual. The overall method for computing the initial quadrics is expressed in the following pseudocode.

```
function ComputeQuadrics()
{
  for 1 to k
  {
    $P$ = generate-configuration()
    reskin-mesh($P$)
    $\rho_{total} = \rho_{total} + \rho(P)$
    foreach($v \in V$)
      $Q_v = Q_v + \rho(P) M_v(P)^T Q_v(P) M_v(P)$
  }

  foreach($v \in V$)
    $Q_v = Q_v / \rho_{total}$
}
```

3.3 Constructing Joint Probability Functions

Box Function: For simplicity, in some cases a simple box function will suffice. This can be defined as a function that takes the value $1/(b-a)$ in the range $[a, b]$, and is zero everywhere else. It defines a joint that has no preferred angle, position or scale, and can move easily anywhere in its allowable range.

Gaussian Distribution: We may also add a "preferred angle" and a "stiffness" factor to each joint, as is common in certain modeling products for inverse kinematics animation. This will give a greater priority to configurations near the preferred angle, and allows the user to have greater control over the final simplified output. As an artist may already set these values, this does not impose an extra burden on content creators. For a preferred angle μ and a stiffness σ, this probability function is represented as a Gaussian distribution with μ and σ as the mean and standard deviation, respectively.

Predefined Animations and Motion Capture: In many cases, the poses of animations will be known in advance. This may be the case either due to manual animation of frames, or the result of motion capture data. In the known examples, some bones may have imperceptible movement, and it may be visually acceptable to simplify their skin polygons as if they did not deform. From the examples given, we can compute the probability distribution of the configurations directly, and use this information to guide our simplification.

3.4 Iterative Contraction

Iterative contraction of edges in our system is mostly identical to that of QSlim. Note that we maintain the desirable property of error quadrics that we may directly solve for the optimal position to minimize error. However an additional consideration must be taken for assigning influence weights to the newly-created vertex.

Each influence weight might be thought of as a continuous vertex attribute, such as color, which would seem to lend the problem to the attribute-preserving simplification of [Garland and Heckbert 1998] and [Hoppe 1999]. However, in our context, an influence weight is not a property of the mesh distinct from geometry, but in fact directly influences the final skinned geometry of the mesh.

We use the following method for updating weights after a vertex collapse. We first compute the distances from the original vertices v_1 and v_2 to the new vertex $v_{collapsed}$, which we refer to as d_1 and d_2 (see Figure 5). We then use the expression $t = d_1/(d_1 + d_2)$ as an interpolant for the linear interpolation between w_1 and w_2, the weight vectors associated with v_1 and v_2, whereby the weight

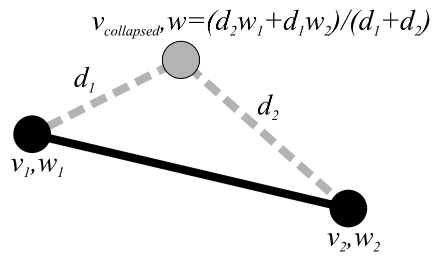

Figure 5: *Weight Update Rule. The vertices v_1 and v_2 with an edge between them (black line) are to be contracted to vertex $v_{collapsed}$. The distances d_1 and d_2 are used to interpolate the weight vectors of the two vertices to compute the influence weights of $v_{collapsed}$.*

vector of the newly created vertex is $w_{collapsed} = w_1(1-t) + w_2t$. Empirical validation for this method is given in Section 4.

During iterative contraction, there is a possibility that geometry may “foldover,” that is, flip the orientation of a triangle, which produces visually unpleasant artifacts. We address this by disallowing collapses where the normal of any triangle in the neighborhood of the collapse would be rotated by more than 90 degrees in the reference pose. This proved to be reasonable, but a more comprehensive solution might compute an average normal over all poses, or find normal cones.

4 Experiments and Results

We tested our algorithm on several models, including a scanned human leg, a police officer, and a horse. Our primary focus in the results was on correctness and resulting mesh quality, rather than speed, as our method is intended for use in a preprocess, yet simplification times are shown to be closely comparable to standard QSlim.

To show the amount of simplification across a model, we color each face of the model based on its size, using a logarithmic scale. The areas with the most simplification are red, and progressively become blue in areas of less simplification. We have found this to be a useful tool in analyzing simplification, though it is most expressive when the original triangles are relatively uniform in size, so that simplified regions stand out clearly.

The leg model, which was taken and modified from a Nvidia skinning demo [Nvidia 2000], is a tapered cylinder that becomes slightly smaller at one end, and bends up to 90° down in the middle. As can be seen in Figure 6, once the mesh is simplified, the regions away from the joint are simplified dramatically, due to their near-coplanarity. In the views of the simplified, undeformed models (bottom row) we see how the region around the joint is preserved in greater detail. Note that the bottom of the knee maintains higher detail over a wider area than the top, due to the greater deformation of the triangles. This is as was expected from Figure 3. Finally, note how the side of knee is simplified more than the top and bottom, as the vertices on the side are limited to positions in the plane of their adjacent triangles in the original mesh. This would not occur in systems that preserve detail based on distance to joints. When the knee deforms, we see that the shape of the crease in the original model is preserved in the simplified versions.

In Section 3.4, we give a method for updating the weight vector after iterative contraction. We compared the results of this method to the results of using a procedural weight function to recompute vertex weights. Most models will not have a procedural specification for influence weights, so the performance of the weight update rule is critical. In our example, we found that for moderate simplification (13 percent of original), the root mean squared difference in weights is insignificant, being less than 1.45×10^{-3}, and the maximum error is 0.021. Even for more drastic simplification (3 percent of original), the root mean squared error is about 5.18×10^{-3}, the maximum error is 0.028, and has an imperceptible effect on the resulting skinned mesh.

Figure 7 shows the performance of our method, as compared to two alternative algorithms. First, we compare to a method that preserves detail near deformable areas by scaling the standard quadrics by a weight that falls off away from the joints. For a vertex v with quadric Q_v, and the position of the nearest bone x_b, we scale the quadric by $Q'_v = [\alpha G(||v - x_b||) + 1]\, Q_v$, where G is a Gaussian, with $\mu = 0$ and $\sigma = 0.1$. In our example, we also set $\alpha = 5$, which has the effect of giving quadrics near joints $\alpha + 1 = 6$ times as much weight as those far away from joints. The choice of μ and α can be tuned as necessary. We also show the results of the standard QSlim algorithm, run on the reference pose. All examples use our method to update influence weights during iterative contraction.

In certain cases, the intuitive logic that the areas around the pivot should be preserved in the highest detail is not always correct. When the joints in Figure 7 are deformed 90 degrees, the area at the pivot itself is actually relatively flat. There are in fact two creased regions adjacent to the pivot area, but the pivot area itself does not form a crease. As can be seen in the example, our algorithm recognizes the creases, and preserves them in higher detail, as well as simplifying the flat regions at the joints. Also, greater simplification is achieved on the outside of the tube, which does not undergo significant deformation. The method of preserving detail at the joint preserves a significant amount of unnecessary detail. Finally, the standard QSlim algorithm results in a relatively uniform level of simplification across the surface of the object, and does not preserve any additional detail at the joint, resulting in the expected loss of quality.

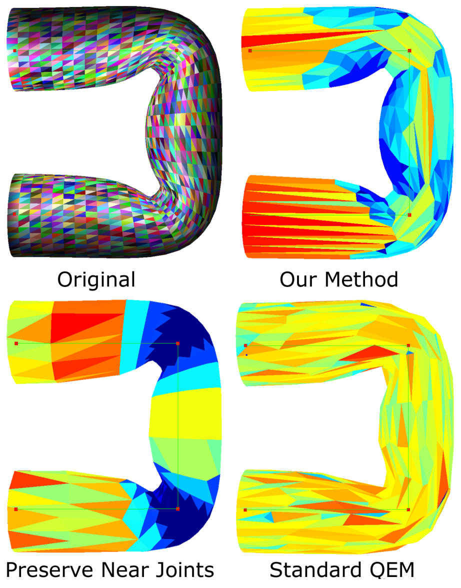

Figure 7: *A full resolution model, one simplified with our method, another that the preserves detail near joints, and the last with standard QSlim. Influence weights have been updated using our method. Note that the area near the pivot is flat, rather than creased. Our algorithm correctly provides more simplification in this area. The method of preserving near joints incorrectly preserves too much detail in this area, and QSlim has a uniform amount of simplification across the model.*

In Figure 8, we show an example of a scanned human leg provided by Cyberware. First, we see the full resolution model, fol-

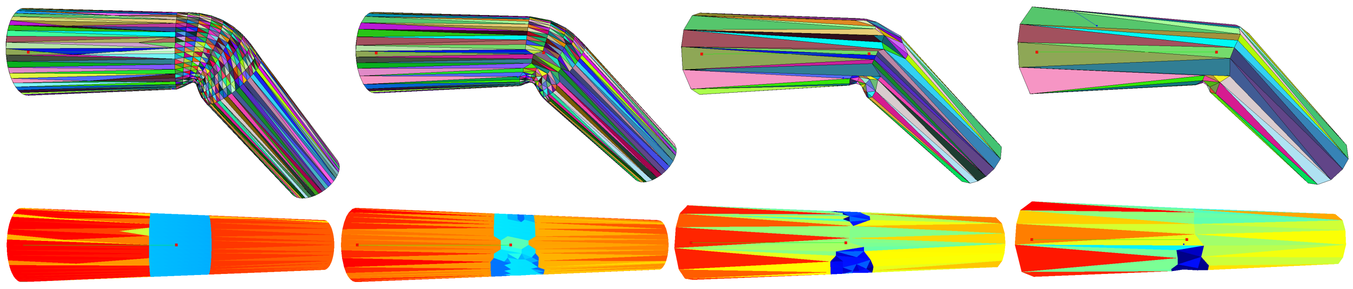

Figure 6: Leg Model, at various levels of simplification, and colored randomly to show the triangulation. The top row shows the leg with the knee deformed at a 45 degree angle. The bottom row shows the leg in the reference pose, but with faces colored based on area. This shows the preservation of detail around the joint, with more detail on the lower side of the joint. From left to right, the meshes are shown after 1700, 2000, 2200, and 2250 collapses, respectively.

lowed by a model simplified with QSlim in the bent position, which we expect would make a good approximation, but loses fidelity in its approximation of the straight pose. The third model has been simplified based on the straight pose, and simplification artifacts are visible at the crease of the knee. Using our method with 4 poses, we achieve a more accurate approximation.

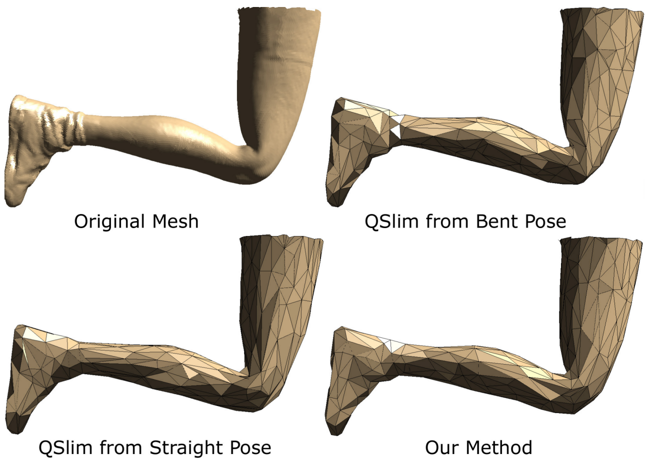

Figure 8: Leg from the scanned female dataset, shown simplified to about 0.5% of the original number of triangles. Our method (using only 4 samples) correctly places additional detail in the deforming regions to better preserve the shape. Note that the crease at the back of the knee in the result produced by our method better approximates the original than the method produced by QSlim from the straight pose.

To compare the effect of varying probability functions, we show a close up of the back of the knees in the Cyberware Scanned Male (Figure 9). The left knee has a 120°range of motion, while the knee on the right has only at 30°range of motion. As a result, the left knee preserves higher detail to allow for better approximation of all possible deformations.

For quantitative evaluation of our approximation, we used the Metro tool [Cignoni et al. 1998], which computes the Hausdorff distance between two surfaces. As shown in Table 1, our method produces a better result than either static method when considering both poses.

For a more complicated example, with multiple bones and joints, we show the police officer model in Figure 10. It can be seen that the deformable regions, especially around the knees and elbows, are preserved in higher detail. We can see from the rear simplified view that the backs of the knees have a large area that is preserved in high detail, as do the heels, elbows and wrists.

Table 2 shows the times required to simplify each of our test models to 10% of the original number of triangles. Although the results prove to be quite fast, due to artifacts of our implementation,

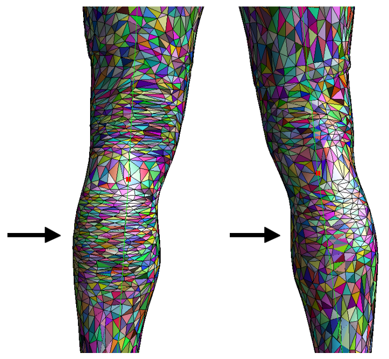

Figure 9: Scanned Male. The left knee has a 120°range of motion, and so preserves higher detail in the indicated region than the knee on the right, which has a 30°range. Note that as in Figure 8, the surface at the crease is flat under deformation and is therefore decimated.

Model	Straight Pose	Bent Pose	Std. Dev.
QSlim/Straight Pose	.01017	.01350	2.4e-3
QSlim/Bent Pose	.01718	.01130	4.1e-3
Our Method	.01122	.01207	6.0e-4

Table 1: Comparison of the results of static methods versus our method. The scanned female leg model was simplified to 0.5% of original triangles, and the Hausdorff distance to the original mesh was computed using Metro. While our method gives a good approximation across poses, the approximation quality of the standard QSlim varies widely.

there is room for greater time efficiency. Our system was implemented in a high-level interpreted language, and we expect that a similar implementation in a more efficient language would provide better performance. Also, increasing the number of samples will increase the time in the initial quadric computation proportionally. In the examples shown, 16 samples were used, but good quality results can be seen for 8 or even 4 samples, due in part to the use of stratification to reduce variance. We can see that the overhead from the initial quadric computation is a small part of the overall time, contributing to around 25% of the total simplification time. As this is the only phase of QSlim changed in any significant way by our method, these results show that our method adds only a small overhead, especially keeping in mind that this is intended for use in a preprocess, and simplification is not performed at run-time. Additionally, as the size of the mesh increases, the running time of the algorithm will consist mainly of the iterative contraction time, reducing the relative time penalty of our method versus QSlim for a single pose.

The majority of the time spent in each function evaluation is used for standard linear algebra operations. An optimized linear algebra package could potentially provide a large performance benefit. However, we wish to stress that all work is spent in a preprocess be-

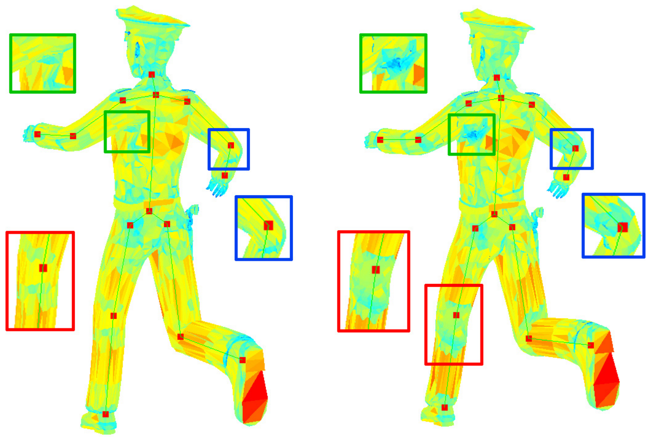

Figure 10: *Rear view of a simplified police officer model with 10% of the original triangles, colored based on triangle size. The model on the left is simplified using QSlim in the original pose, while the model on the right is simplified using our method. Note how our method correctly preserves more detail about the joints, especially the knees, elbows, etc.*

Model	Vertices	Quadric Time	Total Time
Leg	2306	0.25s	0.77s
Human Leg	25158	5.1s	14.0s
Horse	48485	6.5s	26.1s
Police	63881	6.9s	29.3s
Scanned Male	146631	24.2s	95.36

Table 2: *Simplification Times. Each model was simplified to 10% of the original size. Times are given in seconds, using our C# implementation of the algorithm. The Quadric Time is the time to compute the initial quadrics.*

fore actual execution, and no work is performed during rendering time.

5 Conclusions

In this paper, we presented a method for view- and pose-independent simplification of skeletally articulated objects. We have seen how pose independence is possible through the use of the joint probability function ρ, which allows us to have a single quadric that is applicable to many poses. Also, we showed how integration of the vertex quadrics over the domain of possible bone configurations, and transforming quadrics back into the reference coordinate system, allows for quadrics to be correctly combined, then used to simplify the original mesh while taking deformations into account. Finally, we saw how our method accurately preserves creases and deforming regions at joints more effectively than static methods of simplification.

It is important to understand the situations where our algorithm does not provide a significant benefit. In models with significant creases at the joints, the standard QSlim algorithm will preserve additional detail in these regions, which in many cases may be sufficient for animation. Also, in models with widely varying ranges of motion, especially when the most extreme motions are uncommon as defined by the probability function, the single pose-independent model produced by our algorithm may be insufficient. In such cases, it may be practical to consider performing the simplification for each pre-defined animation sequence, rather than for all possible animations at once. For example, a model of a golfer may have legs that remain stationary while making a putt, but naturally move when walking between holes. Such a system might use our method to generate a pose-independent model for each sequence, and switch models as necessary. Additionally, for extreme poses that are short-lived, and in which the focus is drawn to a single model, such as a slam dunk pose, a system might need to assign additional triangles to the model for the duration of the animation. It would then be useful to use our algorithm with a progressive-mesh type system for continuous level-of-detail, so that the exact number of triangles could be specified. While it may be necessary to take some of these considerations into account before integrating our algorithm into a production system, our algorithm provides a useful foundation on which to build.

References

ARVO, J. 1995. Stratified sampling of spherical triangles. *Proceedings of ACM SIGGRAPH 1995*, 437–438.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1998. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2, 167–174.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. *Proceedings of ACM SIGGRAPH 1997*, 209–216.

GARLAND, M., AND HECKBERT, P. S. 1998. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization '98*, D. Ebert, H. Hagen, and H. Rushmeier, Eds., 263–270.

GARLAND, M., 1998. Quadric-based polygonal surface simplification, PhD thesis, Carnegie Mellon University.

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

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

HOPPE, H. 1997. View-dependent refinement of progressive meshes. *Proceedings of ACM SIGGRAPH 1997*, 189–198.

HOPPE, H. H. 1999. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualization '99*, D. Ebert, M. Gross, and B. Hamann, Eds., 59–66.

HOULE, J., AND POULIN, P. 2001. Simplification and real-time smooth transitions of articulated meshes. In *Graphics Interface 2001*, 55–60.

KHO, Y., AND GARLAND, M. 2003. User-guided simplification. In *ACM Symposium on Interactive 3D Graphics*.

LINDSTROM, P., AND TURK, G. 2000. Image-driven simplification. *ACM Transactions on Graphics 19*, 3, 204–241.

LINDSTROM, P. 2000. Out-of-core simplification of large polygonal models. *Proceedings of ACM SIGGRAPH 2000*, 259–262.

MOHR, A., AND GLEICHER, M. 2003. Deformation sensitive decimation. Tech. Rep. 4/7/2003, Univerisity of Wisconsin, Madison.

NVIDIA, 2000. OpenGL vertex weighting demo. http://developer.nvidia.com.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. A., AND FLANNERY, B. P. 2002. *Numerical Recipies in C++*. Cambridge University Press.

SCHMALSTIEG, D., AND FUHRMANN, A. 1999. Coarse viewdependent levels of detail for hierarchical and deformable models. Tech. rep., Vienna University.

SCHROEDER, W. J., ZARGE, J. A., AND LORENSEN, W. E. 1992. Decimation of triangle meshes. *Computer Graphics 26*, 2, 65–70.

SHAMIR, A., AND PASCUCCI, V. 2001. Temporal and spatial level of details for dynamic meshes. In *VRST. 2001*.

SHAMIR, A., BAJAJ, C. L., AND PASCUCCI, V. 2000. Multi-resolution dynamic meshes with arbitrary deformations. In *IEEE Visualization*, 423–430.

TEICHMANN, M., AND TELLER, S. 1998. Assisted articulation of closed polygonal models. *Proceedings of ACM SIGGRAPH 1998*.