Volume 28 (2009), Number 2 pp. 1–10 COMPUTER GRAPHICS *forum*

Poisson-based Weight Reduction of Animated Meshes

Eric Landreneau and Scott Schaefer

Texas A&M University

Abstract

While animation using barycentric coordinates or other automatic weight assignment methods has become a popular method for shape deformation, the global nature of the weights limits their use for real-time applications. We present a method that reduces the number of control points influencing a vertex to a user-specified number such that the deformations created by the reduced weight set resemble that of the original deformation. To do so we show how to set up a Poisson minimization problem to solve for a reduced weight set and illustrate its advantages over other weight reduction methods. Not only does weight reduction lower the amount of storage space necessary to deform these models but also allows GPU acceleration of the resulting deformations. Our experiments show that we can achieve a factor of 100 increase in speed over CPU deformations using the full weight set, which makes real-time deformations of large models possible.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms

1. Introduction

Surface deformation is a popular tool in animation and mesh editing. While there are many different methods for performing such deformations, barycentric coordinates have recently been shown to be an effective tool for both image [HF06] and mesh deformation [JSW05, JMD*07]. Part of the popularity of this deformation method is its simplicity. Given a high-resolution (or target) surface, we can deform the shape by embedding it in a low-resolution control mesh that approximates the target shape. The user then deforms the surface by modifying the control mesh (see Figure 1).

Furthermore, these deformations are easy to compute. For each vertex in the target surface v, we represent that vertex as a weighted combination of the vertices c_i of the control mesh

$$v = \sum_k \alpha_i c_i$$

where α_i are the barycentric coordinates of the vertex v with respect to the control mesh. As the user modifies the vertices of the control mesh, we can compute the deformed location of v using the same weighted combination of vertices

$$\hat{v} = \sum_i \alpha_i \hat{c}_i \tag{1}$$

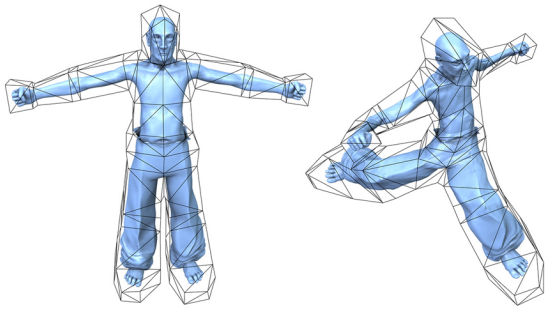

Figure 1: *The user deforms the rest pose (left) by modifying the control cage (right) using barycentric coordinates.*

where $\hat{v}$ is the deformed position of the vertex v and $\hat{c}_i$ are the deformed location of the control vertices. Notice that these coordinates are also translationally invariant; that is

$$\sum_i \alpha_i = 1 \tag{2}$$

and the basis functions associated with a vertex c_i are smooth, which guarantees the deformations are smooth as well (typically C^∞). Since the weights α_i are independent of the deformed location of the control vertices, these coor-

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

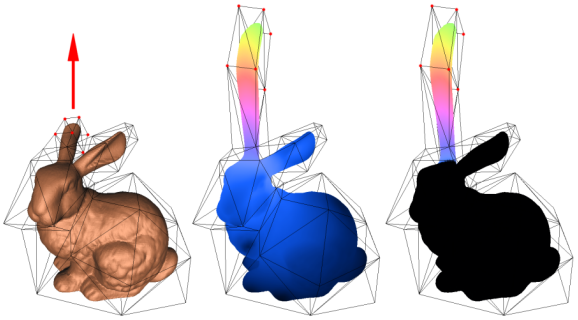

Figure 2: *A bunny's ear in the rest pose (left) is stretched using globally defined mean-value coordinates (middle) and using reduced weights (right). Color corresponds to vertex movement. Every vertex in the global model exhibits some movement, but all movement on the reduced model is locally restricted to the ear.*

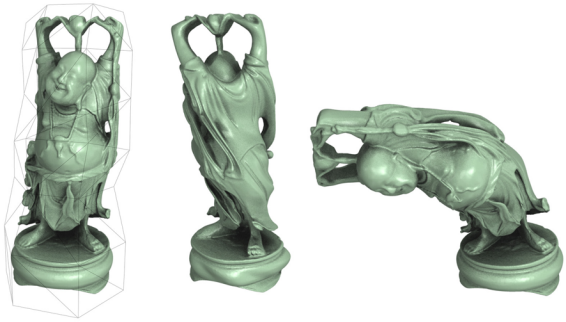

Figure 3: *An example of our weight reduction applied to a model of Buddha in the rest pose (left) and deformed poses (right). The model was reduced from 45 weights down to 12 weights using our Poisson reduction and is indistinguishable from the original model.*

dinates may be precomputed and stored yielding relatively fast deformations.

Unfortunately, most barycentric coordinate techniques are global in nature. Each vertex in the target mesh is weighted by a combination of every vertex in the control mesh. As the complexity of the control mesh increases, the storage space and time required to deform the target mesh's vertices linearly increases as well. Even for sparse control meshes consisting of 50 to 100 vertices, the deformations may be too slow for real-time animation. In addition, any deformation using these globally supported weights will influence the entire target mesh. For example, if the user moves a single ear of the rabbit model in Figure 2, even vertices on the other side of the model will move slightly. Such movement is often counterintuitive, especially with articulated figures where the user may expect the motion of a body part to be isolated to that portion of the model.

Given a control vertex c_i, its basis function defined by the weights α_i typically decreases with respect to Euclidean distance [JSW05] or geodesic distance in the control mesh [JMD*07]. Therefore, many of the weights α_i for a single vertex will be very small and only a few control vertices will have significant influence on the vertex. Since the magnitude of the weights decrease with distance and the deformation model may contain extraneous degrees of freedom, limiting the number of weights influencing each vertex of the target mesh is not only possible but desirable as well. By reducing the number of weights we can expose/increase locality in the resulting deformations. Since the number of weights is constant with respect to the control mesh, we can also increase the complexity of the control mesh without increasing the storage or deformation time of the target mesh. Finally, if the number of weights are small enough, we can accelerate these deformations by implementing the deformation equation on the GPU, which requires a small, fixed number of weights to maximize computational efficiency. Figure 3 demonstrates the type of reduction possible with our method. In this example each vertex of the Buddha model is weighted by all 45 vertices of the control mesh. After weight reduction, each vertex has a maximum of 12 weights and the model is virtually identical to the original under deformation.

1.1. Contributions

We present a Poisson-based optimization technique that reduces the number of control point influences of a deformable mesh to a specified count. To expose the limited degrees of freedom in the model we require the user to specify a number of example poses demonstrating the set of plausible deformations of the character. From these examples we show that we can maintain the appearance of the surface under deformation and provide an iterative optimization technique capable of handling even large meshes consisting of millions of polygons. Using this reduced set of weights, we show how we can implement these barycentric coordinate deformations efficiently on the GPU and achieve a factor of 50 speed-up over a CPU implementation. Moreover, we show that our method can be applied not just to barycentric coordinates but to other deformation methods as well and give an example of weight reduction for skeletal animation.

2. Previous Work

Though barycentric coordinates have existed since Möbius in 1827, only in recent years has significant work been done in this field. Given the relatively new interest in this field, most previous work has focused on developing new forms of barycentric coordinates or proving properties about their deformations [FK08] instead of manipulating or reducing the number of weights. However, all of these methods have

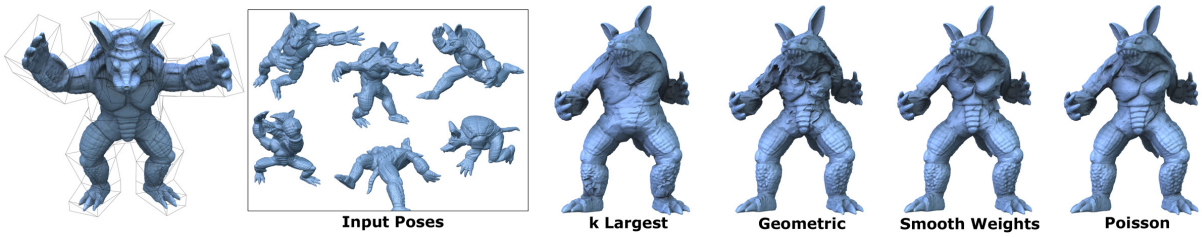

Figure 4: *From left to right: A deformed armadillo man containing 110 weights per vertex computed using mean value coordinates, the example poses used for optimization, reduced to 8 weights per vertex using k-largest weights, geometric reduction, smooth weight reduction, and Poisson reduction.*

globally supported basis functions and are applicable to our technique.

Wachspress developed one of the first extensions of barycentric coordinates to convex polygons for finite element analysis [Wac75]. Later, [War96] extended this method to convex polytopes in higher dimensions. [FHK06] also showed that entire families of coordinates could be constructed for convex shapes. However, since all of these methods relied on the convexity of the control polygon/mesh, they found limited use in Computer Graphics.

[Flo03] addressed this problem by creating mean value coordinates, which were well-defined for arbitrary polygons in 2D. Later [JSW05] and [FKR05] extended these coordinates to higher dimensions. [HF06] and [JSW05] also demonstrated how these barycentric coordinates could be used for image and surface deformation.

The disadvantage of mean value coordinates is that they can contain negative weights, which had the potential to create undesirable deformations. [LKCOL07] modified mean value coordinates to be positive; however, this modification made the coordinates (and the resulting deformations) only C^0. [JMD*07] introduced harmonic coordinates, which utilized a discrete solution to the harmonic equation to create barycentric coordinates and guarantee that the weight functions are always positive and smooth. More recently [HS08] demonstrated a connection between barycentric coordinates and statistics to create a non-linear optimization for positive barycentric coordinates as well.

While this paper focuses on barycentric coordinates, many techniques represent deformations as a weighted combination of control vertices. For example, free-form deformations [SP86] use a lattice based control mesh and the coordinates of vertices inside of the lattice are given by the value of the Bernstein basis functions. Radial basis functions [Boo89] may also be used for image/surface deformation where the weights of each control vertex are found through a linear system of equations.

3. Weight Reduction

We will assume that for each vertex of the target surface, we are given the weights α_i for $i = 1 \ldots n$ where n are the number of control vertices. We are also given a user-specified constant $k < n$, which is the maximum number of non-zero weights we will reduce to. Notice that, for barycentric coordinates in 3D, k must be greater than or equal to 4. We will consider 4 possible methods for weight reduction and show that Poisson-based minimization provides the best results.

There are two components to weight reduction. First we must choose the set of control vertices of size k that will influence a particular vertex. Second, once that set is chosen, we must determine the values of the weights for the k control vertices. We concentrate mainly on the later problem though we must still address the choice of influence set.

There are many possibilities for choosing the influence set of a vertex. However, the combinatorics of choosing the best k control vertices from n, the choice of which may be affected by the influence set of neighboring vertices on the target surface as well, is computationally intractable for even small meshes. Therefore, we use a simple heuristic and choose the influence set for a vertex based on the largest k weights. This choice is reasonable since, as mentioned previously, the weight functions are proportional to distance meaning that many control vertices will have small influence on a given vertex. There may be other heuristics for choosing influence sets that work better, but we find that the k largest weights does well in practice.

If we simply select the k largest weights for a vertex v and renormalize the weights such that they satisfy Equation 2, we obtain a new, reduced set of weights for v. Given that the weights we removed had small magnitude, a commonly held belief is that these normalized weights will perform well as a reduced set. However, this is not the case as is illustrated by Figures 4 and 5. Though the influence sets exhibit a large degree of spatial locality as expected, large tears and normal discontinuities such as those illustrated by Figure 6 are visible in the model where adjacent influence sets are different. Even when the weights we remove have very small magnitude, these normal discontinuities are present. Therefore, we

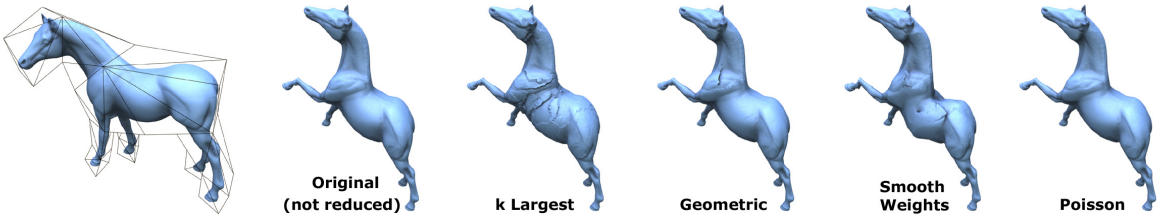

Figure 5: *A deformed horse containing 51 weights per vertex computed using harmonic coordinates. From left to right: rest pose with control cage, original deformed surface, reduced to 8 per vertex using k-largest weights, geometric reduction, smooth weight reduction, and Poisson reduction.*

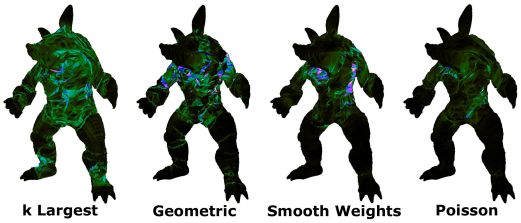

Figure 6: *Normal deviation for several reduction techniques. Color intensity denotes amount of deviation.*

must consider more sophisticated techniques for choosing the value of these weights.

3.1. Smooth Weights

Choosing the k largest weights generates a poor solution because sharp normal discontinuities are visible in the deformed meshes. One solution is then to find a set of smooth weights that satisfy the property that the weights are zero outside of the influence set for a vertex.

Let $\bar{A}_i$ be the original barycentric coordinate function of the i^{th} control point before optimization such that $\bar{A}_i(v) = \alpha_i$ for vertex v. For each barycentric weight function, we maximize the smoothness of this influence restricted function by minimizing

$$\sum_i (\nabla^2 A_i - \nabla^2 \bar{A}_i)^2$$

subject to the constraint that, at each vertex, $\sum_i A_i(v) = 1$ where $A_i(v)$ is zero at a vertex if i is not part of v's influence set. We choose to optimize the criteria that the laplacian of the reduced weights match the laplacian of the original weight set to preserve the property that, if number of reduced weights k is set to the number of control vertices, then $A_i = \bar{A}_i$. As this minimization does not contain any geometric constraints, we add the constraint that A_i have linear precision:

$$\sum_i A_i(v) c_i = v.$$

For polygon meshes, ∇^2 has a familiar form as a summation of cotan weights [PP93]. Performing this constrained minimization results in a sparse, linear system of equations of size $mk+4$ where m is the total number of vertices in the surface.

Many barycentric coordinate techniques such as mean value coordinates may produce negative weights and, hence, negative weights are even desirable in weight reduction. However, for some methods such as harmonic coordinates or skeletal animations (see Section 4), the weights are always guaranteed to be positive. Positive weights not only guarantee the convex hull property for bounding deformations, but can also avoid overfitting in the optimization [JT05]. For these methods, we add an additional constraint that $A_i(v) > 0$. This constrained quadratic minimization with inequality constraints can be minimized using non-negative least squares [LH74], though for such large systems of equations, non-negative least squares may not be practical as these solvers require repeated solutions of the quadratic.

Instead we found that, by iterating a local solver, we were able to minimize the entire system of equation effectively. To do so, we minimize the weights of each vertex separately by holding the weights of the vertices of its one-ring constant. The size of the quadratic is then reduced to k variables and is easy to solve even with non-negative constraints. Each minimization of a vertex reduces the global error over the surface and this process converges to the minimum. In practice the number of iterations necessary to converge to the solution depends on the size of the mesh but we found that iterating this process a number of times equal to 0.1% of the number of vertices produced good results. We use this technique, including the non-negative constraints where appropriate, in Sections 3.2, 3.3 and 4 as well.

While this minimization is far more costly than simply choosing the k largest weights, the result is a much smoother deformation. Figure 4 shows an example of such a minimization applied to a barycentric coordinate deformation. For models with small discontinuities in the smoothness of the weight function, this minimization may be adequate. However, this method still performs poorly for more complex animations as seen in Figure 4.

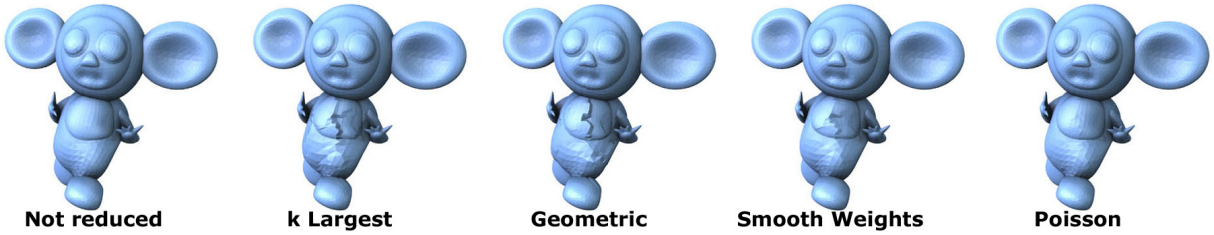

Figure 7: *A model with a skeletal animation created with Pinocchio contains up to 17 weights per vertex (far left). We reduce the model to* 5 *skin weights per vertex using k-largest weights (middle left), geometric reduction (middle), smooth weight reduction (middle right), and Poisson reduction (right).*

One of the reasons that this method, as well as simply choosing the k largest weights, does not approximate the deformations well is that it does not use any information about the types of plausible deformations that may be generated by the user. The control mesh of these free-form deformations does not have any limitations placed on its shape, and any configuration of the control vertices will generate a valid deformation. While each control vertex is independent from one another and may be moved in isolation, in practice many control vertices move in concert to create plausible deformations of an object. For example, in Figure 1 the control vertices at the tip of the hand move in almost a rigid fashion to maintain the shape and volume of the hand under deformation.

Since these restrictions are not encoded in the control mesh, we must have some method of sampling the plausible space of deformations. A popular technique for sampling a model's plausible deformation space is through a set of example poses provided by the user [MG03, SZGP05, HXS08]. Solving our weight reduction problem using plausible deformations simplifies the problem a great deal as we do not need to approximate the original shape for every possible control mesh configuration. Furthermore, this technique allows an artist guide the optimization towards the model's most common deformations.

Therefore, we require that the user specify a number of example poses exercising the degrees of freedom of the deformation. The number of example poses required to achieve a good result highly depends on the example poses and the complexity of the model. Good results may be obtained with only one example pose though we find in practice several are needed to express the full range of motion of a character (we typically use 4-6 examples).

3.2. Geometric Weight Reduction

One possible method for utilizing these examples poses is a modification of the method of [JT05]. This method minimizes the distance to the deformed example poses in the L_2 norm, which results in a minimization of the form

$$\min_{\alpha_i} \sum_{j}^{poses} \left| v^j - \sum_{i \in inf(v)} \alpha_i c_i^j \right|^2 \tag{3}$$

where v^j is the deformed position of v in the j^{th} pose, c_i^j is the position of the i^{th} control vertex in the j^{th} pose and $inf(v)$ denotes the influence set of v. Combining this equation with the constraints from Equation 2 creates a constrained quadratic minimization with a unique minimum. Note that, like [JT05], we utilize non-negative constraints when appropriate.

Figures 4 and 5 shows an example of the result of applying this minimization to the weights. While the silhouette of the model is well approximated, the model suffers from similar problems as the k-largest weights and has banding artifacts caused by normal discontinuities. Since we perceive shape not only through the geometric shape of the silhouette but also through the way in which light interacts with the surface, we can improve the perceived quality of the deformations by optimizing the difference in the normals of the surface.

3.3. Poisson-based Weight Reduction

Direct optimization of the weights to make the normals of the reduced deformation match the example poses produces a non-linear optimization. An alternative is to optimize the tangents of the surface [GZ08]. The approach we take is to minimize a Poisson equation similar to [TLHD03, YZX*04]. Given a function ϕ defined over the target mesh, the Poisson equation is the solution to

$$\min_{\phi} \int_S |\nabla \phi - w|^2 \, dA$$

where S is the surface and w is a guidance field over the surface. Equivalently, this equation may be written as

$$\min_{\phi} \int_S (\nabla^2 \phi - \nabla \cdot w)^2 dA$$

where ∇^2 is the laplacian. If the guidance field w is also given as a divergence of a scalar field ψ over the target mesh,

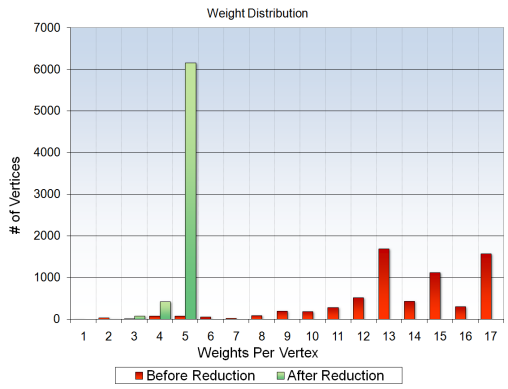

Figure 8: *Weight distribution of the model from Figure 7 before and after weight reduction with a maximum of 5 weights.*

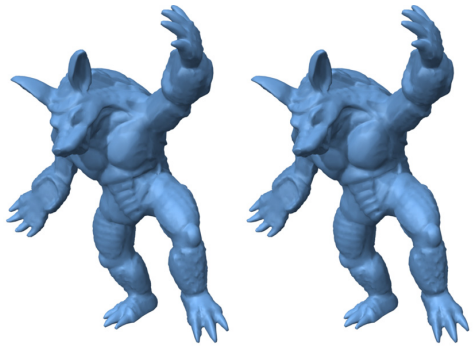

Figure 9: *Vertex shader calculation of normals. The left shows the result of exact normals computed from the positions of the deformed vertices. The right shows normals calculated by GPU transformation of rest-pose normals using approximate affine transformations.*

then we can write the equation as

$$\min_{\phi} \int_S (\nabla^2\phi - \nabla^2\psi)^2 dA. \tag{4}$$

In other words, the equation tries to match the laplacian of the unknown function ϕ to that of the given function ψ.

Using Equation 4, we can write our minimization problem as

$$\min_{\alpha} \sum_{\gamma=0}^{2} \sum_{j}^{poses} \sum_{\ell}^{vertices} (\nabla^2(\sum_{i \in inf(v_\ell)} \alpha_{\ell,i} c_i^j[\gamma]) - \nabla^2 v_\ell^j[\gamma])^2 \tag{5}$$

subject to the constraints in Equation 2 where $c_i^j[\gamma]$ represents extracting component γ from the vertex c_i^j ($x = 0, y = 1, z = 2$).

Similar to the geometric minimization in Section 3.2, this minimization is a constrained, sparse quadratic in the unknown weights α, which can be solved using Lagrange multipliers and the repeated, local minimization technique described in Section 3.1.

4. Weight Reduction for Skeletal Animation

While we have focused on barycentric coordinates and other control mesh weighted deformation schemes, our technique may be used on many different types of deformations. Here we give an example of skeletal animation based on its popularity in computer graphics and real-time applications. Instead of blending control vertices, this method blends bone transformations to deform vertices.

Given a set of bone transformations as matrices M_i for the i^{th} bone, the deformed position $\hat{v}$ of a vertex v from the rest-pose is

$$\hat{v} = \sum_i \alpha_i M_i v.$$

If we substitute this definition into Equation 5, we obtain

$$\min_{\alpha} \sum_{j}^{poses} \sum_{\ell}^{vertices} (\nabla^2(\sum_{i \in inf(v_\ell)} \alpha_{\ell,i} (M_i^j v_\ell)[\gamma]) - \nabla^2 v_\ell^j[\gamma])^2.$$

For skeletal animation, we typically require that $\alpha_{\ell,i} \geq 0$ so we can use a non-negative least squares solver to minimize this equation as in Section 3.3.

Figure 7 shows an example of this reduction applied to skeletal animation. In this example the skeletal weights were computed automatically using Pinocchio [BP07]. Before reduction each vertex had a variable number of weights many of which were influenced by the entire skeleton (17 bones). After minimization the maximum number of bone weights was 5 and Figure 8 shows the weight distributions before and after minimization. The result with the reduced weights is very similar to the original deformation, takes less storage space and can be animated more efficiently.

5. GPU Implementation

In recent years, graphics processors on modern video cards have become useful tools for tasks other than graphics rendering, such as linear blend skinning. A GPU works best with small, fixed-size data, so a model in which each vertex has tens or hundreds of weight influences may be unacceptably slow when animated on a GPU. Therefore, our technique not only optimizes a model so it consumes less space and performs better under CPU deformation, but the reduction also makes the model viable for GPU accelerated deformation.

To perform this acceleration, we implemented a vertex shader to calculate the model's deformation on the GPU. Vertices of the deformed control mesh are sent to the card as constants in the shader. Weight indices and weight values are sent as per-vertex texture coordinates and the shader

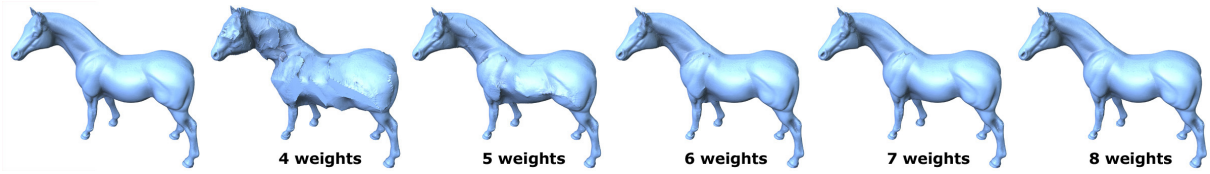

Figure 10: *Poisson weight reduction varying the maximum number of weights. The far left is the original shape. Then, from left to right, the maximum weights varies from 4-8.*

performs a simple summation to find a vertex's deformed position according to Equation 1.

A vertex shader does not contain adjacency information, so finding the normal of the vertex is nontrivial. With skeletal animations, the vertex normals can be transformed using the inverse transpose of the weighted bone matrix. However, for barycentric coordinate animations, this is not the case. While we could calculate the deformed mesh and read-back its vertices to recompute vertex normals on the CPU, this method is unnecessarily slow. Instead, we provide a method for approximating the barycentric coordinate deformation locally as an affine transformation and use the inverse transpose of this matrix to transform the normal.

Given a set of control vertices, $\hat{c}_i$ for some deformed control cage (not necessarily an example pose), the positions of the control vertices in the rest pose c_i^0 and weights α_i, we approximate the barycentric coordinate transformation using a weighted least squares formulation. In particular, we find the affine transformation that maps each control vertex c_i^0 to its deformed location c_i weighted by the value α_i.

$$\min_M \sum_i |\alpha_i (M c_i^0 - \hat{c}_i)|^2$$

Solving for the inverse transpose of the matrix, we obtain

$$M^{-T} = \frac{1}{\beta} adj \left(\sum_i {\alpha_i}^2 c_i^0 c_i^T \right) \left(\sum_i {\alpha_i}^2 c_i^0 {c_i^0}^T \right)$$

where $adj(X)$ represents the adjugate of the matrix X and β is the determinant of the matrix we compute the adjugate of. Since the normal will be renormalized after multiplication, β never needs to be calculated.

Multiplying the normal by this matrix yields a good approximation of the transformed normal. When $\hat{c}_i = c_i^0$, this method returns the identity transformation and the result is exact. Furthermore, when the size of the influence set of the vertex is four, this minimization yields the precise affine transformation that the control points represent. When the influence set is greater than four vertices, this method approximates the deformation with an affine transformation. Figure 9 shows an example of lighting a model with normals produced by this method. The left figure shows the result of computing area weighted normals from the positions of the deformed vertices and is exact. On the right, the figure's normals use this approximate affine transformation method. The result is nearly indistinguishable.

We tested our GPU implementation on an Intel Core i7 920 CPU with an Nvidia 280 GTX video card. To test the deformation, we used an animation of the model in Figure 1, which contains 702,538 polygons and its control mesh contains 137 vertices. Our CPU implementation achieves about 1.2 fps using the full weight set (137 weights/vertex) and 13.9 fps using the reduced weight set (8 weights/vertex). However, our GPU implementation was able to achieve 142.9 fps using the reduced weight set. Hence, barycentric coordinate animation can be practical even for large meshes using GPU acceleration.

6. Results

We computed weight reductions of several different models using different types of barycentric coordinates or deformation methods. Furthermore, we compared the *k*-largest, geometric, smooth and Poisson reductions. Figure 4 shows an example where the weights were computed using mean value coordinates. In this example the chest expands noticeably using the *k*-largest weights. Minimizing the difference to the geometry reduces the inflation but contains severe artifacts in the normals. However, the Poisson minimization is able to generate a faithful reproduction of the original shape.

Figure 5 depicts a similar example using harmonic coordinates where we constrained the reduced set of weights to be positive under minimization. The geometric minimization does a much better job than the *k*-largest but there is a discontinuities in the normals of the shoulder that the Poisson minimization removes.

Figure 7 shows an example of our method applied to skeletal animation where the weights were generated by Pinocchio, which solves an equation based on heat dissipation to create automatically create a smooth, plausible deformation. Unfortunately, this smoothness comes at the cost of high numbers of weights as nearly a quarter of the vertices are weighted by the entire skeleton (17 bones). After reducing to only five bones per vertex (constrained such that the weights were positive), the Poisson reduction is able to maintain the appearance of the original model and is now suitable for GPU acceleration. Figure 8 shows the weight distribution before and after reduction.

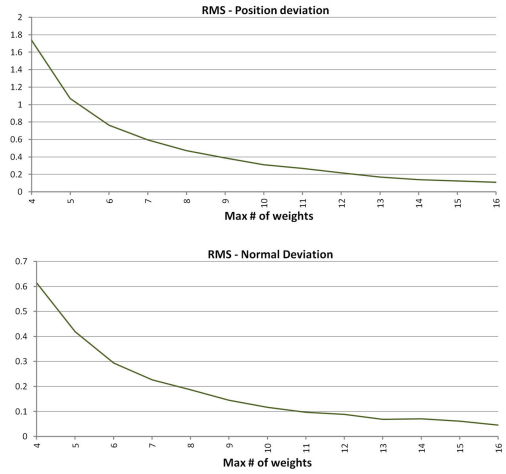

Figure 11: *RMS error of the model in Figure 10 averaged over several poses while varying the maximum of number of weights in the optimization. The normal deviation is measure in radians and the position error is measured as a percentage of the bounding box diagonal.*

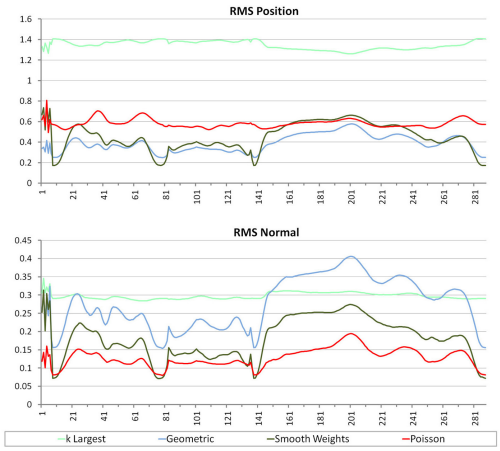

Figure 12: *RMS error of the armadillo man reduced to 8 weights per vertex during each frame of an animation. The normal deviation is measure in radians and the position error is measured as a percentage of the bounding box diagonal.*

Figure 12 quantifies the differences between the different reduction techniques. The figure shows the armadillo man reduced from 110 weights per vertex down to 8 using six example poses. We reduced this model using the four different methods and performed an animation where none of the example poses were part of the animation. The figure shows a graph of the deviation of the normals and the geometry for each frame of the animation. The *k*-largest weights performs the worst of the three methods in both the normal and geometry metrics. The geometric reduction does well when measuring the deviation of the surface from the original animation; however, the deviation of the normals was significant. The smooth weight optimization fits the geometry relatively well and improves upon the normal error too. However, the Poisson reduction fits the normals of the deformation the best but does not match the geometry as well. Nevertheless, this deviation in the geometry is rarely if ever noticeable in the animation. In contrast, it is easy to see the normal approximation error in the model produced by the other methods.

We also compared the effect of changing the maximum number of weights the optimization is allowed to use. Figure 10 shows the horse reduced using the Poisson minimization with the maximum number of weights varying from 4-8. For barycentric coordinate animations, we cannot use less than four weights per vertex since simplices in 3D contain four vertices. Furthermore, only allowing four weights per vertex does not leave many degrees of freedom to optimize the smoothness of the deformation and the result is poor as expected. However, as the number of weights per vertex increases, the quality of the deformation improves dramatically. Figure 11 numerically shows the effect of increasing the maximum weights for this example.

In Table 1, we show the time it takes to perform each weight reduction technique on different models. These timings were computed on a AMD Athlon 64 X2 4200+ with 2GB of RAM. As expected, the k-largest and geometric methods are simple and easy to calculate. The Poisson and Smooth Weight methods are also slower than the geometric technique. Furthermore, imposing non-negative constraints can significantly increase the running time of the optimization. Although these optimizations cannot be performed in real-time for large meshes, the optimizations only need to be performed once. Using the optimized weights, we can perform deformation in realtime with the deformation speed being linearly proportional to the number of non-zero weights per vertex. As Section 5 shows, the reduced weight set allows us to use the GPU to accelerate these deformations and we can achieve over a factor of 100 increase in performance over a CPU implementation with the full weight set.

7. Conclusions and Future Work

Our weight reduction technique reduces the number of control points influencing a vertex to a user-specified number such that the deformations created by the reduced weight set resemble that of the original deformation. We accomplished this reduction by solving a Poisson equation for the reduced weights such that the gradients, and hence the normals, match a set of example poses provided by the user and provided an efficient method for optimizing these weights. We also showed how to implement these deformations efficiently on the GPU and obtain over a factor 10 increase in

Model	Verts	Deformation Method	Max Weights	Optimized Weights	Poses	Timings(in seconds)			
						kL	G	S	P
Buddha	427,616	MVC	45	12	3	5.48	72.3	464.9	360.3
Bunny	35,947	MVC	47	8	4	.5	2.5	18.0	25.3
Armadillo	15,002	MVC	110	8	7	.4	1.6	9.6	16.9
Horse	48,485	MVC	51	8	3	.7	3.3	22.7	24.4
Armadillo	15,002	HC	110	8	7	.4	2.3	124.3	133.6
Horse	48,485	HC	51	8	3	.7	6.1	379.2	420.8
Cheb	6,669	SD	17	5	7	.2	.3	6.4	7.9
Human	120,226	SD	17	5	8	2.6	5.3	129.5	163.2

Table 1: *Reduction times for different techniques measured in seconds. kL = k-largest, G = Geometric, S = Smooth Weights, P = Poisson. The results are shown for different deformation methods including Mean Value Coordinates (MVC), Harmonic Coordinates (HC) and Skeletal Deformation (SD). HC/SD utilize non-negative constraints during all of the optimizations while MVC does not.*

performance over a CPU implementation using the reduced weight set. Furthermore, our weight reduction method is applicable to other deformation techniques and we provided an example application to skeletal animation.

In the future we would like to address the problem of determining influence sets in more detail. One area that our weight reduction method may perform suboptimally is when the control mesh contains planar regions. In these areas all of the vertices in the planar region may have large weights associated with them. However, if these vertices all deform in a planar affine fashion, clearly these vertices contain redundant information. We believe that it is possible to detect these subsets and remove them from the influence list. However, if the example set indicates that this removal is possible and the user produces a movement in which the vertices are not planar, then poor deformations will result. Hence this method is highly dependent on the example set provided by the user to describe the full range of motion of the character.

8. Acknowledgments

We'd like to thank Ilya Baran and Jovan Popović for their Pinocchio software, which was used for the Cheb model and animation in this paper. We would also like to thank Nathan Bajandas for contributing the old man model. This work was funded by NSF grant CCF-07024099.

References

[Boo89] BOOKSTEIN F.: Principal warps: thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 6 (1989), 567–585. 3

[BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3 (2007), 72. 6

[FHK06] FLOATER M., HORMANN K., KÓS G.: A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics 24*, 1–4 (2006), 311–331. 3

[FK08] FLOATER M., KOSINKA J.: On the injectivity of wachspress and mean value mappings between convex polygons. *Advances in Computational Mathematics* (2008). To appear. 2

[FKR05] FLOATER M., KÓS G., REIMERS M.: Mean value coordinates in 3d. *Computer Aided Geometric Design 22*, 7 (2005), 623–631. 3

[Flo03] FLOATER M. S.: Mean value coordinates. *Computer Aided Geometric Design 20*, 1 (2003), 19–27. 3

[GZ08] GINGOLD Y., ZORIN D.: Shading-based surface editing. *ACM Transactions on Graphics (TOG) 27*, 3 (2008). 5

[HF06] HORMANN K., FLOATER M. S.: Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph. 25*, 4 (2006), 1424–1441. 1, 3

[HS08] HORMANN K., SUKUMAR N.: Maximum entropy coordinates for arbitrary polytopes. *Computer Graphics Forum 27*, 5 (2008), 1513–1520. 3

[HXS08] HE Y., XIAO X., SEAH H.: Example based skeletonization using harmonic one-forms. In *IEEE International Conference on Shape Modeling and Applications* (2008), pp. 53–61. 5

[JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), p. 71. 1, 2, 3

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (2005), 561–566. 1, 2, 3

[JT05] JAMES D., TWIGG C.: Skinning mesh animations. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 399–407. 4, 5

[LH74] LAWSON C., HANSON R.: *Solving Least Squares Problems*. Prentice Hall, Englewood Cliffs, NJ, 1974. 4

[LKCOL07] LIPMAN Y., KOPF J., COHEN-OR D., LEVIN D.: Gpu-assisted positive mean value coordinates for mesh deformations. In *Proceedings of the Eurographics Symposium on Geometry processing* (2007), pp. 117–123. 3

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (2003), 562–568. 5

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2* (1993), 15–36. 4

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 151–160. 3

[SZGP05] SUMNER R., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* (2005), pp. 488–495. 5

[TLHD03] TONG Y., LOMBEYDA S., HIRANI A., DESBRUN M.: Discrete multiscale vector field decomposition. *ACM Trans. Graph. 22*, 3 (2003), 445–452. 5

[Wac75] WACHSPRESS E.: *A Rational Finite Element Basis*. Academic Press, New York, 1975. 3

[War96] WARREN J.: Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics 6* (1996). 3

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 644–651. 5