Direct Manipulation of Interactive Character Skins

Alex Mohr * Luke Tokheim * Michael Gleicher *

University of Wisconsin, Madison

Abstract

Geometry deformations for interactive animated characters are most commonly achieved using a skeleton-driven deformation technique called linear blend skinning. To deform a vertex, linear blend skinning computes a weighted average of that vertex rigidly transformed by each bone that influences it. Authoring a character for linear blend skinning involves explicitly setting the weights used to compute deformed vertex positions. This process is tedious, repetitive, and frustrating not only because the deformed vertex positions are not intuitively related to the vertex weights, but also because the range of possible deformations is unclear. In this paper, we present a method that lets users directly manipulate the deformed vertex positions in a linear blend skin. We compute the subspace of possible deformed vertex positions, display it for users, and let them place the vertex anywhere in this space. Our algorithm then computes the correct weights automatically. This method lets us provide a skin editing interface that gives users as much direct control as possible and makes explicit what deformations are possible.

Keywords: Skinning, Direct Manipulation

1 Introduction

Linear blend skinning is an important form of character display for interactive systems. It is standard in video games and virtual environments because it produces reasonably good results, is simple to implement and is extremely efficient. In fact, consumer graphics hardware natively accelerates it [12]. While flaws of linear blend skinning are well known [11], many of the methods that address these issues present new techniques that do not meet the strict performance demands of interactive systems. Therefore, artists are forced to work within the limitations of linear blend skinning when creating characters for interactive environments.

Despite the wide use of linear blend skinning, authoring them is a daunting task. The linear blend skinning algorithm attaches geometry to characters by assigning to each vertex a set of bones (and the associated transformation matrices) that should affect it, known as *influences*, and a *weight* for each influence. A deformed vertex position is computed by transforming an initial vertex rigidly by each of its influences and taking a linear combination of these transformed positions using the weights as coefficients. Therefore, all control of the deformation of a linear blend skin is achieved by varying each vertex's influences and weights. Unfortunately, setting these parameters is difficult because deformed vertex positions are only *implicitly* related to the vertex weights. Further, the range of possible deformed vertex positions is not obvious. In fact, inexperienced users often believe that if they edit the weights enough, they can obtain any desired deformation. This leads to frustrating and tedious work with no knowledge of whether further improvement can be made [11]. Finally, since vertex weights are fixed, editing the weights while viewing one pose can change the geometry in all other poses. This creates a tradeoff problem where making the deformation look good in one pose can ruin it in another. Despite all these user interaction drawbacks, most commercial systems only provide a weight manipulation interface.

This paper presents a method that allows explicit manipulation of deformed mesh vertices. Skin authors may directly position deformed vertices using any vertex manipulation or mesh sculpting tool they like. Our algorithm interactively computes the range of possible deformations, projects these desired vertex positions onto this legal range, and computes the blending weights automatically. This approach frees users from having to manually set blending weights and lets them adjust deformations directly. We also provide visualization tools that help users see and understand the range of possible deformations and evaluate the results of their edits across several poses concurrently.

We begin by reviewing linear blend skinning and other character skinning approaches. Analysis of linear blend skinning's problems leads to a method for visualizing the possible deformed vertex positions for the full range of potential skin parameters. We then present our method to automatically compute blend weights from manipulated vertices. Finally, we discuss the results of our approach.

2 Linear Blend Skinning

Character skin deformations are fundamental to character animation. One of the earliest skeleton-driven techniques was introduced by Catmull [5]. This paper introduces rigid skinning to a hierarchically structured articulated figure. A 2D skeletal bilinear deformation method was presented by Burtnyk and Wein [3]. An early 3D skeleton-driven technique used custom programmed algorithms to deform character meshes [13].

Linear blend skinning is unpublished in the literature because techniques like [13] are more general. However, the technique is widely used for interactive applications. An excellent description is found in [11]. Linear blend skinning goes by many names. Several authors use the acronym SSD for Skeleton Subspace Deformation [11] while Alias|wavefront's Maya calls it "smooth skinning".

To create a linear blend skin, one begins with a static model of a character, typically in some neutral pose. A hierarchical skeleton of joints connected by rigid links is created to fit inside the geometry. This initial character pose is referred to as "dress pose". Then, each vertex is assigned a set of influences and a blending weight for each influence. To compute the deformed skin, each dress pose vertex is rigidly transformed by all of its influences and the blending weights are used to take a convex combination of these rigidly transformed

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

positions. The deformed vertex position, $\bar{\mathbf{v}}$ is

$$\bar{\mathbf{v}} = \sum_{i=1}^{n} w_i M_i D_i^{-1} \mathbf{v}_d$$

where w_i are the influence weights, $\mathbf{v}_d$ is the dress-pose location of a particular vertex $\mathbf{v}$, M_i is the transformation matrix associated with the ith influence, and D_i^{-1} is the inverse of the dress-pose matrix associated with the ith influence. Taken together, $D_i^{-1}\mathbf{v}_d$ represents the location of $\mathbf{v}_d$ in the local coordinate frame of the ith influence.

Linear blend skinning is notorious in character animation. It has two primary failings. First, the method is incapable of expressing complex deformations. Second, authoring linear blend skins is difficult and frustrating for users.

The first failing is due to the problems inherent in the linear blend skinning algorithm, resulting in characteristic deformation problems. First, taking a convex combination of the *entries* in the transformation matrices for each influence and applying this "blended matrix" to the dress pose vertex can produce unnatural results. While this kind of transformation blending may be acceptable for combining transformations that are nearly the same, it does not work well for blending very different transformations. Unfortunately, this happens often with articulated figures. For example, consider the case of a human wrist. The wrist can twist through an angle of nearly 180 degrees. A linear blend of these transformations produces a completely degenerate transformation that results in the familiar "candy-wrapper" collapse effect. Similar problems involving shrinkage occur around bent elbows, shoulders, and knees.

The range of linear blend skins is also very restricted. If an artist desires a particular mesh vertex to be in a certain location for a particular pose, it is only possible if some weight setting places the vertex there. This happens only if the desired location is representable by some convex combination of the rigidly transformed dress-pose vertex positions, $M_i D_i^{-1}\mathbf{v}_d$.

Much of the literature has introduced novel skinning methods to solve these problems. Many of these techniques can be viewed as corrections to linear blend skinning. A relatively new technique involves combining linear blend or rigid skinning with radial basis example interpolation [11, 17]. The multi-weight enveloping technique [19] adds many more weights to the linear blend model (one per transformation matrix entry) and solves for them using a number of example poses and linear least squares. EigenSkin [10] presents a different example-driven linear blend skin correction technique. Recently, range scan data has been used to develop interactive character skins [1]. Singh and Kokkevis present a surface-oriented free-form deformation technique for character skinning [16]. Other authors have used dynamics for interactive deformations, especially secondary animation [9, 4].

Despite the quality of deformations afforded by these new techniques, many of them are still inappropriate for interactive systems. Some require a large amount of memory to store examples while others are slow to compute in comparison with linear blend skins. For these reasons, linear blend skinning remains popular for computer games and virtual environments.

Unlike previous work, this paper does not address the fundamental problems with linear blend skinning. Instead, we focus on the difficulty of authoring linear blend skins.

3 Authoring Skins

Assuming a skeleton has been placed inside a character mesh, authoring a linear blend skin amounts to choosing the influence set and weights for each vertex. Most animation systems provide weight "painting" interfaces that authors use to directly manipulate influence sets and weights. Since changing skin weights directly only *indirectly* changes vertex positions, the authoring experience is considered tedious, difficult, and unpredictable [11].

Typical user interaction begins with the system providing an initial guess for influences and weights based on some vertex to joint proximity measure. For example, Maya makes an initial guess using Euclidean distance. Shin and Shin present a method based on Delaunay tetrahedronalization [15].

Both influence sets and weights are then edited via a painting interface. To adjust weights and influences, users select a joint, and all the vertices that are influenced by it are highlighted, often with a color gradation to indicate the weight of that influence on each vertex. The user may then "paint" over the mesh vertices to change their influences or to change the relative weight of that influence on vertices. Other image editing inspired interactions are common. For example, a "smoothing brush" or "flood fill" may be provided.

While the painting interface seems attractive, it is difficult to control skin deformations with it. To move a single vertex, the user must guess at which influence should be changed to move in the desired direction. Then the user must guess at how that influence's weight should be adjusted. After setting the paint tool with both of these options, the user paints the vertex. Since the connection between weight values and vertex positions is indirect, the two guesses are often wrong. This leads to many iterations of tweaking and trial-and-error. Worse, there is not only no easy way to know whether a desired vertex position can be achieved at all, but also no easy way to know if the deformed vertex is as close as possible to the target location, making it difficult to know when to stop iterating.

The main contributions of this paper are the methods for direct manipulation of deformed vertices and visualization of the possible range of deformation. These tools can significantly reduce the guesswork and tedium involved in authoring linear blend skins.

Direct manipulation methods have been used since the earliest interactive systems. The technique was pioneered by many authors [18, 14, 8]. Since then, direct manipulation techniques have been applied to a wide variety of tasks.

Much work has focused on providing direct manipulation for objects whose internal parameters are not always most conveniently set directly. For example, there has been some work on direct manipulation for parametric curves and surfaces [7, 6]. These techniques let users manipulate parts of the curve directly. Our scheme is similar to these approaches since we are solving for the underlying skin weight parameters based on direct vertex manipulation.

4 Direct Manipulation of Geometry

Our direct manipulation method works by determining the range of possible deformed vertex positions and drawing this subspace for the user. Then we allow users to position vertices anywhere they like and project back onto the valid subspace. Finally, we compute the appropriate skin weights from the resulting vertex positions.

As mentioned earlier, a vertex's deformed position is the a convex combination of rigidly transformed positions. This means that the entire possible range of a deformed vertex is simply the convex hull of these rigidly transformed positions. Since the number of weights per vertex in most interactive skins is small (usually five or fewer), the convex hull is at worst a small convex polyhedron. In many cases, it is even simpler. For example, in the cases of two and three influences, the valid subspace is a line segment and a triangle.

As an aside, if negative weights are allowed, the combination becomes affine so the range of deformations becomes the affine hull. Although negative weights can afford more freedom in deformation, we do not use them in our system because they have produced

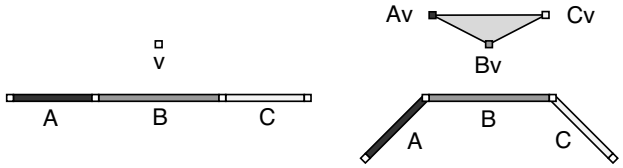

Figure 1: On the left is a skeleton in dress pose (A, B, C) and a single vertex v influenced by all three joints. On the right, the skeleton is in a different pose. Av, Bv, and Cv, show the vertex v rigidly transformed by each of its influences. The shaded triangle is the convex hull of these points and represents all possible deformed positions for v with convex weights.

undesirable results in our experience. Therefore, our discussion focuses on the convex case.

Drawing the convex polyhedron that represents the possible range of a deformed vertex is valuable to users. It lets them determine whether further weight modification can improve their results. Figure 1 shows the simple case of a vertex influenced by three bones and the shape of the valid subspace in a particular pose.

The geometric interpretation of the possible deformation range suggests a vertex manipulation interface. We let users manipulate skin vertices any way they like and project back onto the valid regions. This step ensures that the desired positions are reachable with some set of convex weights and guarantees that the deformed positions are as close as possible to the desired positions. Finally, we find a set of weights that achieves the projected position.

We use a standard technique to do this projection. If the desired location is already inside the convex hull, we are finished. Otherwise, we project the desired point onto the planes containing the faces of the convex hull. We keep the best of all these projections that lie inside the faces. Next, we project onto the lines containing each edge and keep the best of those that lie on the edges. Finally, we check each vertex, and keep the overall closest projection. While this general technique scales poorly in high dimensions, we are in three dimensions and have small convex hulls (since the number of influences per vertex is generally small) so this is not an issue.

4.1 Computing Skin Weights

Once the projection of the desired vertex position is known, we must find a set of compatible vertex weights that reach this goal. There is a difficulty in solving for convex weights when the legal subspace is degenerate or there are five or more influences for a vertex. These cases are treated in Section 4.2. For subspaces that have four or less influences and are non-degenerate, there is always a single unique set of convex weights. We find these weights by solving a linear system of equations.

$$\left[(R_2 - R_1)\mathbf{v}_d \cdots (R_n - R_1)\mathbf{v}_d\right] <w_2, w_3 \cdots w_n>^{\mathrm{T}} = \mathbf{t} - R_1\mathbf{v}_d$$

Where R_n is $M_i D_i^{-1}$, or the rigid transformation matrix for influence i, and $\mathbf{t}$ is the target vertex position, already projected onto the valid subspace. Finally, we set $w_1 = 1 - \sum_{i=2}^{n} w_i$. This guarantees that the weights are affine.

Note that in general, this algorithm does not guarantee that all weights will be positive. However, we know that the solution is unique and all-positive from the earlier projection step. More precisely, the projected target position is on a non-degenerate simplex so its barycentric coordinates are unique and convex.

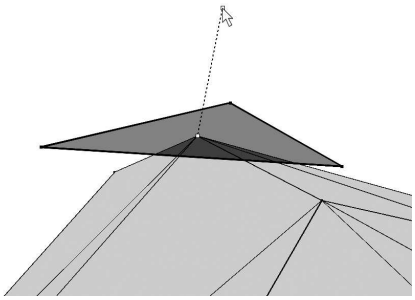

Figure 2: Here a user is manipulating a single skin vertex. The vertex has three influences. The valid subspace is shown by the dark shaded triangle. The desired vertex position is at the cursor but is unattainable. Our algorithm finds the closest point on the valid subspace to the desired location and places the skin vertex there. A dashed line shows the difference between the target location and the closest valid location.

4.2 Underdetermined Cases

While there is always exactly one solution in the cases described above, there can be many possible solutions in underdetermined cases. This occurs when a joint has more than four influences or when the legal subspace is degenerate. More precisely, degeneracies arise when the vectors that trace the edges from a single vertex of the subspace to all other vertices of the subspace are linearly dependent. Solving for weights using the method given in Section 4.1 in these situations can produce negative weights.

This problem could be avoided by certain measures. For instance, an animation system may only allow four or less influences per vertex. This ensures that the subspace is always a simplex in three dimensions (either a tetrahedron, triangle, line segment, or point). To handle the case of a degenerate *simplex*, a system could detect this condition and perturb joint angles to force the simplex to be non-degenerate. It is possible, however, to deal with these cases directly and robustly.

Our method finds valid weights in underdetermined cases by solving a linear program. We know there must exist at least one convex solution because we initially projected the target onto the subspace reachable by convex weights. In general, however, there will be several convex solutions. In such cases, we assume that there was a previous valid weight setting for the vertex being manipulated. Then we choose the convex solution that minimizes the change between the new and previous weights in an L_1 sense.

To set up the linear program, we must supply a set of inequality constraints and an objective function to minimize. First we enforce

$$\left[R_1\mathbf{v}_d \cdots R_n\mathbf{v}_d\right] \mathbf{w} = \mathbf{t}$$

to ensure that we actually reach the target point. Next, we constrain $\sum_{i=1}^{n} w_i = 1$ and $w_i \geq 0$. To minimize the difference between the current and new weights in an L_1 sense, we introduce new variables $\mathbf{y}$ and add the constraints $-\mathbf{y} \leq \mathbf{w} - \mathbf{w}_p \leq \mathbf{y}$, where $\mathbf{w}_p$ are the previous weights. The new variables, $\mathbf{y}$, are meant to squeeze the difference between the new and previous weights. To do this, we minimize the objective function $\sum_{i=1}^{n} y_i$.

We solve this linear program using the freely available lp_solve library [2]. This computation is interactive even for many vertices at once because the size of the problem scales with the number of influences for a single vertex.

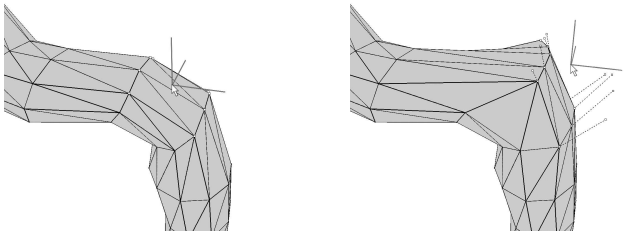

Figure 3: Using our technique, skin authors may manipulate skin vertices using any mesh editing tool they like. Here, a user pulls several vertices of a skin simultaneously. The desired locations are unattainable but our system places the vertices as closed as possible. The dashed lines indicate to the user the difference between the desired and actual locations.

5 User Interaction

We have implemented our direct manipulation method inside a prototype skinning system that we use to author characters. The system provides both a weight painting and direct manipulation interface. We evaluate our direct manipulation technique in this system.

Authoring a skin in our system begins with a mesh and a skeleton. First, initial influence sets and skin weights are determined automatically using a geodesic distance metric.

Next, the user may tune influence sets using the painting interface. It is important to note that even though the user paints on the mesh, it is only to determine influence sets, not to set weights or move vertices. It is in fact beneficial to make influence sets larger than required to provide more freedom for vertex editing.

Once the influences and initial weights are determined, the user may edit the vertex weights. The character is set to a pose and the vertices that deform improperly may be fixed. To manipulate a single vertex, the user simply selects it and drags it to a new location using a direct-manipulation jack [20]. The user interface shows the desired point location and the projection of that point onto the valid subspace, connecting them with a line. The user may also view the valid subspace itself to so the possible range of deformation is made completely explicit. The resulting weights are computed on the fly so the skin is automatically updated. Users may also select sets of points and translate them or scale them about their centroid. Our algorithm ensures that the deformed vertices come as close as possible to their desired locations. This is shown in Figure 3. We note that any vertex manipulation tool that a skin author wants can be used to manipulate the vertices. The goal positions are simply projected onto their respective subspaces.

Editing weights changes the skin deformation in all poses, so we let users view the results of their edits for several poses concurrently.

It is important to note that direct manipulation does not conflict with weight painting. The ability to use both methods is useful.

6 Discussion

Using our skinning system, we have found that we prefer our direct manipulation interface for our skinning tasks, all but abandoning the painting interface. The ability to visualize the possible range of deformations and to manipulate the vertices directly makes the skin authoring process much more straightforward—especially for inexperienced users.

Our method improves the linear blend skin authoring process. This is valuable since this skinning technique is widely used for interactive characters. We stress that our direct manipulation algorithm does not fix linear blend skinning. In fact, from using our system, we believe that the direct manipulation technique makes the fundamental limitations of linear blend skinning more explicit.

Acknowledgments

Lucas Kovar helped formulate the linear program. The UW Graphics Group assisted with paper and video production. This research is supported by a Wisconsin University-Industry Relations Grant, NSF grants CCR-9984506 and CCR-0204372, and equipment donations from Intel.

References

[1] Brett Allen, Brian Curless, and Zoran Popović. Articulated body deformation from range scan data. *ACM Transactions on Graphics*, 21(3):612–619, 2002.

[2] Michel Berkelaar. lp_solve linear programming library. http://www.cs.sunysb.edu/ algorith/implement/lpsolve/implement.shtml.

[3] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM*, 19(10), 1976.

[4] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics*, 21(3):586–593, 2002.

[5] Edwin E. Catmull. A system for computer generated movies. In *Proc. ACM Annual Conf.*, pages 422–431. August 1972.

[6] Barry Fowler. Geometric manipulation of tensor product surfaces. In *Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 101–108, 1992.

[7] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 177–184, July 1992.

[8] E.L. Hutchins, J.D. Hollan, and D.A. Norman. Direct manipulation interfaces. pages 243–262, 1986.

[9] Doug L. James and Dinesh K. Pai. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Transactions on Graphics*, 21(3):582–585, 2002.

[10] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: Real time large deformation character skinning in hardware. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 153–160, July 2002.

[11] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH 2000*, July 2000.

[12] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of ACM SIGGRAPH 2001*, pages 149–158, 2001.

[13] N. Magnenat-Thalmann, R. Laperrire, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33, 1988.

[14] Ben Schneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16:57–63, August 1983.

[15] Seung-Hyup Shin and Sung Yong Shin. Real-time human body deformation based on rotation angle interpolation. In *Proceedings of The 27th KISS Spring Conference*, pages 625–627, 2000.

[16] Karan Singh and Evangelos Kokkevis. Skinning characters using surface oriented free-form deformations. In *Graphics Interface*, pages 35–42, 2000.

[17] Peter-Pike J. Sloan, III Charles F. Rose, and Michael F. Cohen. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143, 2001.

[18] I. E. Sutherland. Sketchpad: A man-machine graphical communication system. In *AFIPS Spring Joint Computer Conference*, pages 329–346. 1963.

[19] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 129–138, July 2002.

[20] Robert C. Zeleznik, Kenneth P. Herndon, Daniel C. Robbins, Nate Huang, Tom Meyer, Noah Parker, and John F. Hughes. An interactive 3d toolkit for constructing 3d widgets. In *Proceedings of SIGGRAPH 1993*.