Geodesic Voxel Binding for Production Character Meshes

Olivier Dionne and Martin de Lasa*
Autodesk Inc.

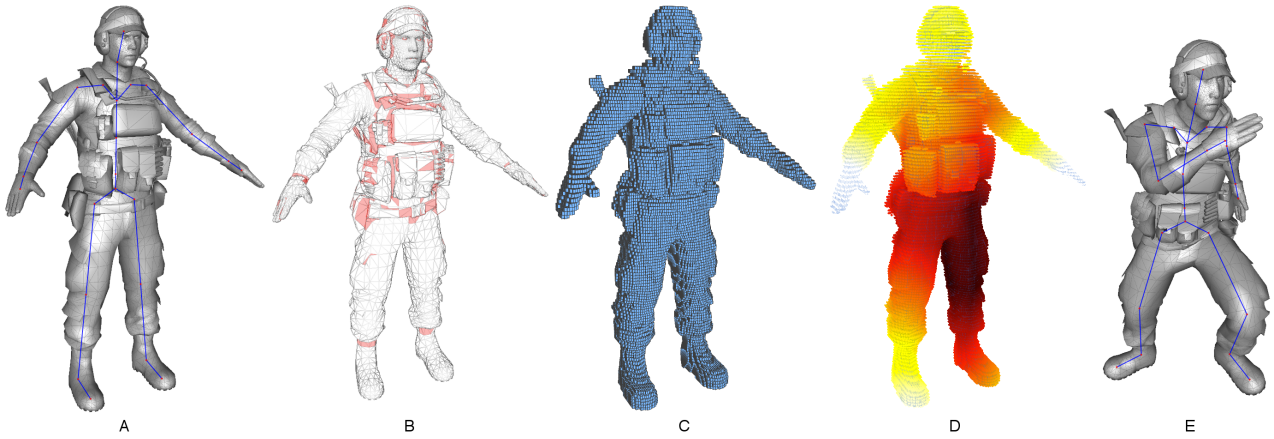

Figure 1: *Starting from a character skeleton and mesh (A), which may contain degenerate geometry (in red) (B), we voxelize the mesh using graphics hardware (C), and compute bind weights using geodesic distances from each bone (D). Resulting weights are applied to existing closed-form skinning methods to deform character geometry (E).*

Abstract

We propose a fully automatic method for specifying influence weights for closed-form skinning methods, such as linear blend skinning. Our method is designed to work with production meshes that may contain non-manifold geometry, be non-watertight, have intersecting triangles, or be comprise of multiple connected components. Starting from a character rest pose mesh and skeleton hierarchy, we first voxelize the input geometry. The resulting voxelization is then used to calculate binding weights, based on the geodesic distance between each voxel lying on a skeleton "bone" and all non-exterior voxels. This yields smooth weights at interactive rates, without time-constants, iteration parameters, or costly optimization at bind or pose time. By decoupling weight assignment from distance computation we make it possible to modify weights interactively, at pose time, without additional pre-processing or computation. This allows artists to assess impact of weight selection in the context in which they are used.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Skinning, Voxelization, Animation, Deformations

*e-mail:olivier.dionne|martin.delasa@autodesk.com

1 Introduction

Creating high quality virtual characters for feature film and games is a time-consuming process. In a typical character authoring pipeline, a modeler first creates a mesh and image textures to represent the character's "skin". Next, a rig comprised of a transformation hierarchy (i.e., the skeleton) and constraints is specified. Skinning weights are then painted by hand onto the mesh to determine how the skin should deform during animations. Additional deformers may also be layered to create more realism, such as muscle bulges. Since interactions between different elements may produce unexpected results, it is often necessary to repeat each step multiple times until the desired outcome is obtained.

One particularly challenging step in the character creation pipeline is painting skin weights. Painting weights is unintuitive since artists must compensate for deficiencies in skinning algorithms found in most commercial animation packages and run-times. For example, linear blend skinning (LBS), the *de-facto* standard closed-form method for joint-based deformations, suffers from many well know artifacts, such as the "candy-wrapper" effect that is visible when joints are twisted; see Lewis et al. [2000] for an in-depth discussion. Additionally since the skeleton is required to be in a rest pose (e.g., a T-stance) when weights are specified, the impact of different choices is not immediately clear; users must paint weights in one pose and evaluate results in another.

Despite these drawbacks and the myriad skinning methods that have been proposed to date (Section 2), closed-form methods, such as LBS and dual quaternion skinning (DQS), continue to be widely used. This is due to the large number of efficient implementations found in authoring and runtime environments. Recent auto-rigging systems [Baran and Popović 2007; Chen et al. 2011; Jacobson et al. 2011; Miller et al. 2011; Wareham and Lasenby 2008] have also made it possible to define skeletons and find good skinning weights with little manual intervention. Unfortunately, these methods are either unable to handle production meshes, which often contain multiple connected components and non-manifold geometry, or require

numerous, costly to create, hand-authored input examples.

In this paper, we propose a fully automatic method for specifying influence weights for closed-form skinning methods, designed to work with production meshes. Our method leverages a novel voxelization algorithm (Section 4) that is well suited for parallelization on commodity hardware. We propose a simple weighting scheme that leverages the resulting voxelization to generate smooth weights, based on geodesic voxel distances. The method requires no time-constants, iteration parameters, or optimization at bind or pose time (Section 6). Additionally, by decoupling weight assignment from distance computation it is possible to modify weights interactively, at pose time, without requiring additional preprocessing or computation. Several interesting applications also result from our approach which we discuss in Section 8.

2 Related Work

Weight Selection There is a large body of research on generating appealing deformations for animated characters. To date, several approaches have emerged including detailed anatomical models of skin and underlying tissues [Lee et al. 2009], approximate elastic models that may consider contact [Capell et al. 2002; Capell et al. 2005; McAdams et al. 2011], reduced-order methods that use key deformation modes [Barbič and Zhao 2011; Kim and James 2011], methods that rely on examples to compute corrections [Kry et al. 2002; Lewis et al. 2000; Mohr and Gleicher 2003; Sloan et al. 2001; Wang et al. 2007], volume-based methods that embed geometry in cages [Joshi et al. 2007; Ju et al. 2008; Lipman et al. 2008], and closed-form methods that deform geometry based on character pose [Jacobson and Sorkine 2011; Kavan et al. 2007; Kavan and Sorkine 2012; Magnenat-Thalmann et al. 1988].

For closed-form skinning methods, only a handful of approaches have attempted to select good skinning weights from a single mesh and skeleton. Katz and Tal [2003] propose a resolution-dependent mesh segmentation strategy that can be applied to skinning. Wade and Parent [2002] use voxelization to extract a skeleton but resulting weights do not vary smoothly, causing artifacts on high-resolution meshes. Baran and Popović [2007] compute skinning weights by formulating a linear system based on equilibrium heat equations over the mesh surface. To be well behaved and robustly generate the needed tri-diagonal system, this approach relies on a distance smoothing operation which limits input to manifold geometry. Additionally, as with other methods based on diffusion [Chen et al. 2011; Joshi et al. 2007] or lighting models [Wareham and Lasenby 2008], iteration parameters, time-constants, or convergence thresholds must be specified for use as termination criteria; our method requires no such parameters.

Kavan and colleagues [2012] formulate weight selection as a quadratic optimization, using an objective based on elastic energy. Since computations are performed on a voxel grid, their method is also capable of handling many types of geometry encountered in production. Though results are impressive, the proposed formulation is quite costly; reported results take several minutes to compute. As our goal is to develop an interactive method that produces good results, on real-world input, and is straightforward to tune if additional changes are desired, we avoid the indirection and runtime cost introduced by optimization [Jacobson et al. 2011]. As with commercial packages, such as Autodesk Maya, we use a weighting scheme based on the distance between vertices and bones. However, we avoid common artifacts of those methods (cf. Figure 5) by accounting for the mesh's structure.

An alternative, to recomputing weights on each new input mesh, is to transfer weights from existing hand-painted examples [Miller et al. 2011]. Our approach could be used to ease authoring of new weight templates for such systems.

Voxelization Although numerous real-time voxelization algorithms have been devised most focus on building surface voxelizations (e.g., [Dong et al. 2004; Fang et al. 2000; Li et al. 2005]). Methods for solid voxelization are typically restricted to closed watertight geometry and rely on parity tests to determine voxel classification. Surface intersections tests are performed using rays originating at each voxel; an odd count indicates the voxel is interior, while an even count indicates it is exterior. Fang and Chen [2000] use a slice-wise approach, while Eisemann and Décoret [2008] process all slices simultaneously using fixed-function pipeline commands. Recent GPU accelerated approaches have also been proposed that generate sparse voxelizations [Schwarz and Seidel 2010]. To use these approaches on degenerate geometry, one can compute an intermediate watertight mesh, but required processing can quickly dominate computation; see Section 8 for more details.

Our voting scheme is inspired by the work of Nooruddin and Turk [2003] who use multiple views and ray stabbing to generate candidate classifications for voxels. To break ties, their algorithm uses a large odd number of views, aligning cameras with icosahedron face normals. Our approach uses a small number of orthographic views, slicing the model on graphics hardware. This leads to a simple implementation that is very fast (cf. Table 2).

3 Overview

We describe an algorithm that automatically computes deformation weights for interactive skin deformation algorithms, such as linear blend skinning (LBS). Although we limit our discussion to LBS, due to its widespread use in interactive applications, our approach can also be used with more advanced skinning algorithms, such as the recent work of Kavan et al.[2012].

The input to our system is a user specified rest-pose mesh M with vertex positions:

$$\mathbf{p}_1, \quad \mathbf{p}_2, \quad \ldots, \quad \mathbf{p}_n \quad \in \Re^3. \tag{1}$$

and corresponding skeleton, represented as a hierarchy of transformations:

$$\mathbf{T}_1, \quad \mathbf{T}_2, \quad \ldots, \quad \mathbf{T}_m \quad \in \Re^{3\times 4}. \tag{2}$$

Methods such as LBS require a weight function:

$$\omega_j(\mathbf{p}) : M \rightarrow \Re \tag{3}$$

that specifies the amount of influence $\mathbf{T}_j$ has on a vertex with position $\mathbf{p}$. Hence, LBS will deform each vertex on M according to:

$$\mathbf{p}_i{}' = \sum_j^m \omega_j(\mathbf{p}_i)\mathbf{T}_j \begin{bmatrix} \mathbf{p}_i \\ 1 \end{bmatrix}. \tag{4}$$

Selecting good weights is critical if deformations artifacts are to be avoided; desirable criteria for weights include being independent of mesh resolution, varying smoothly along the surface, and handling transitions between joints to avoid creases [Baran and Popović 2007].

To determine weights, our method first computes a voxelized representation of the input mesh (Section 4). This divides the bounding volume surrounding the mesh into voxels, which we classify as interior, boundary, and exterior voxels. Next, geodesic distances d_j^i

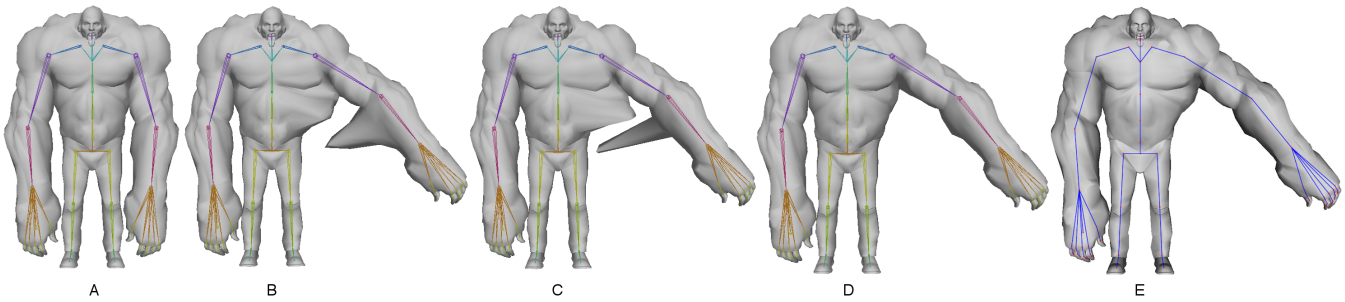

Figure 2: *Comparison of other automatic weighting schemes (B-D) with our method (E). (A) Initial bind pose. (B) Closest distance (Maya) (C) Closest hierarchy (Maya) (D) Heat map weighting [Baran and Popović 2007] (E) Geodesic voxel binding. Distance-based methods that ignore mesh structure (B,C) can produce severe artifacts. For watertight meshes we obtain similar results to Heat map weighting (D,E). See Figure 8 for more complete discussion.*

between voxels v_i falling on skeleton "bones" and all non-exterior voxels v_j are computed. This step is repeated for each bone in the skeleton (Section 5). Once complete, we perform a hit test to determine boundary voxels containing geometry vertices and calculate weight values based on a simple falloff function (Section 6). Figure 1 shows an illustration of the key steps for our algorithm.

4 Voxelization

To voxelize the input geometry, we propose an approach based on z-buffer slicing. Our hardware-accelerated method is inspired by the approach of Fang et al. [2000], with a voting scheme adapted from Nooruddin and Turk's [2003] multiple view parity counting method.

Our algorithm first computes the world space axis-aligned bounding box for the input geometry and uses calculated extents to initialize the orthographic view volume. Then, for each volume slice, the near clip plane is adjusted to match the current slice depth while the far clip plane position is kept fixed. Back faces of the mesh are first rendered in white to an off screen buffer, of same slice dimensions with no filtering, followed by front faces in black. A white pixel, corresponding to a voxel in the slice, is thus voted as internal to the mesh volume. This process is repeated until the whole mesh bounding volume has been processed for all pairs of view directions in x, y and z as shown in Figure 3.

Finally, to classify a voxel v_i as internal (1) or external (0) we compile the votes from each view (x,$-x$,y,$-y$,z and $-z$) using the following scheme:

$$\begin{aligned} v_{i_{(x,-x)}} &: \{0,1\} \rightarrow v_{i_x} \parallel v_{i_{-x}} \\ v_{i_{(y,-y)}} &: \{0,1\} \rightarrow v_{i_y} \parallel v_{i_{-y}} \\ v_{i_{(z,-z)}} &: \{0,1\} \rightarrow v_{i_z} \parallel v_{i_{-z}} \end{aligned} \tag{5}$$

If at least two of the three pairs of views agree that v_i is internal to the mesh domain, it is tagged as such:

$$v_i : \{0,1\} \rightarrow (v_{i_{(x,-x)}} + v_{i_{(y,-y)}} + v_{i_{(z,-z)}}) > 1. \tag{6}$$

We perform a final post-processing step to extract mesh boundary voxels. This is achieved by computing an octree from the mesh and testing for intersections against each voxel using Akenine-Möller's separating axis theorem triangle/box overlap test [2001]. Remaining untagged voxels are considered external to the mesh domain.

The proposed majority voting scheme gave us satisfactory result for all production meshes we tested. For badly degenerate meshes, we cannot guarantee results will always be perfect; some potential failure cases might include labeling interior voxels as exterior, and so on. In such cases, it would be possible to include simple voxel volume painting tools for touch up or make it possible for the user to exclude views from the voting scheme in Equation 6.

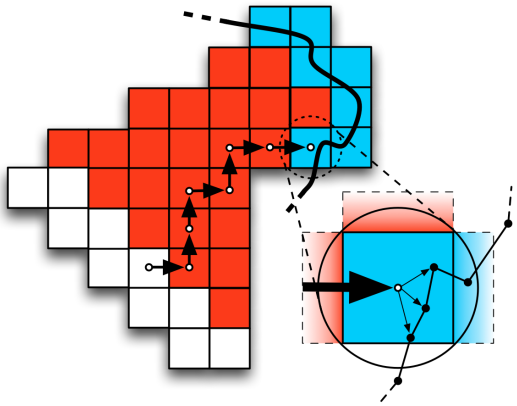

Figure 4: *Computing distances through the voxelized model where white, red and blue colors represent skeleton, interior and boundary voxels.*

5 Distance Computation

To compute the shortest distance between interior voxels falling on skeleton bones and boundary voxels, we use a form of Djikstra's algorithm (cf. Algorithm 1).

For each skeleton bone b_i, we start by initializing the distance value of each non-exterior voxel to infinity (lines 2-4). Next, we identify all voxels intersecting the bone (i.e., skeleton voxels) and set their distance to zero, prior to pushing these voxels onto a working queue (lines 6-9). While the queue is not empty, we dequeue a voxel v_i and for every neighboring voxels v_j with a stored distance d_{v_j} greater than $dist$, we update v_j and add it to the queue (lines 10-19). Here, $\mathbf{p}_v$ represents a voxel's center position. Once processed, we obtain the geodesic distance to bone b_i for every voxel of the mesh domain.

In our implementation we optimize this operation by distributing the distance computation for each skeleton bone b_i across multiple cores.

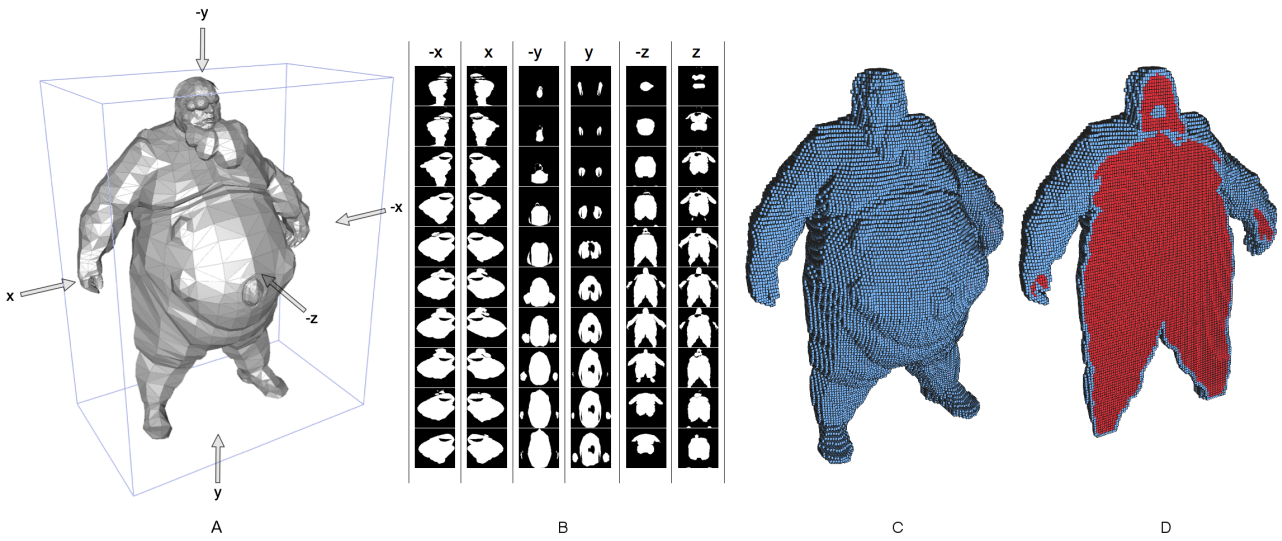

Figure 3: *Voxelization overview for "Boomer" model. (A) The character world space axis-aligned bounding box defines the initial orthographic view volume. (B) For every pair of view directions in x, y and z we position the camera in the proper axis and slice the model by moving the near plane. (C) After compiling votes for each view using equations* (5) *and* (6) *and performing an octree hit test we obtain a voxelization of the input geometry. (D) A cross section view of the resulting voxelization. Red and blue represent internal and boundary voxels respectively.*

Algorithm 1: Distance Computation

input: Character skeleton $\mathbb{S}$ and voxelized mesh $\mathbb{V}$

foreach *bone* b_i *of* $\mathbb{S}$ **do**
  `// Initialize voxel distance values`
  foreach *non-exterior voxel* v_i *of* $\mathbb{V}$ **do**
    $d_{v_i} = \infty$;
  end
  Create empty voxel queue $\mathbf{Q}$;
  `// Initialize bone voxels and`
  `// add to queue.`
  foreach *non-exterior voxel* v_i *of* $\mathbb{V}$ *intersecting with* b_i **do**
    $d_{v_i} = 0$;
    Push v_i *to* $\mathbf{Q}$;
  end
  `// Compute geodesic distances`
  while $\mathbf{Q}$ *not empty* **do**
    Pop v_i *from* $\mathbf{Q}$;
    foreach *non-exterior voxel neighbor* v_j *to* v_i **do**
      $dist = d_{v_i} + |\mathbf{p}_{v_i} - \mathbf{p}_{v_j}|$;
      if $d_{v_j} > dist$ **then**
        $d_{v_j} = dist$;
        Push v_j *to* $\mathbf{Q}$;
      end
    end
  end
end

6 Weight Computation

Once distances for non-exterior voxels are calculated we can compute skinning weights. To compute final mesh weights for each vertex used by Equation 4 we first start by performing a hit test, using our previous computed mesh octree (cf. Section 4). This identifies the corresponding voxel for each vertex. To compensate for the voxel grid's coarseness, and the fact that multiple vertices may fall in the same voxel, we add the distance between the voxel center $\mathbf{p}_{voxel}$ and mesh vertex position $\mathbf{p}_{vertex}$ to the current bone distances in the voxel d_v^i, as shown in Figure 4.

From this final distance value d_j^i we compute a weight influence ω_j^i of bone i for vertex j as follows:

$$d_j^i = \frac{d_v^i + |\mathbf{p}_{vertex} - \mathbf{p}_{voxel}|}{D} \tag{7}$$

$$\omega_j^i = \left(\frac{1}{(1-\alpha)(d_j^i) + \alpha(d_j^i)^2} \right)^2 \tag{8}$$

where α is a parameter in the range $[0, 1]$ allowing animators to control bind smoothness. Increasing this parameter has the effect of reducing the overall influence of distant bones to the vertex, creating a stiffer local bind. Note that distance values are normalized using the product of bounding box extents D, such that $\epsilon \leq d_j^i \leq 1$ to make weights independent of mesh scale. We use the minimum distance ϵ to avoid numerical problems with Equation 8.

Many other falloff functions could have been employed, however, we found Equation 8 works well on the characters we tested while leaving some binding flexibility to animators. When the number of influencing weights needs to be limited, as is often done in games to enable vectorization, adjusting binding stiffness is crucial to maintaining quality as observed in Figure 8.

Once all weights have been computed for a given vertex they are normalized, based on the number of user-specified influences, prior to skinning.

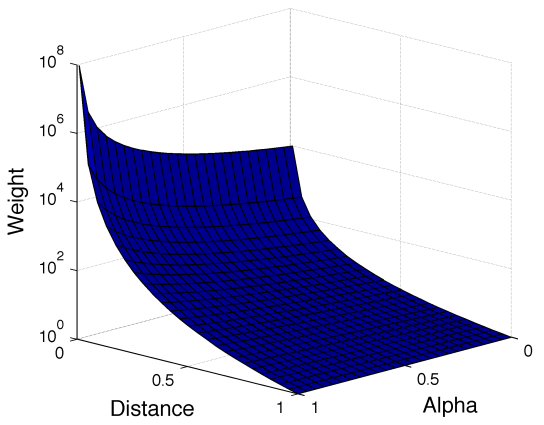

Figure 5: *Visualization of proposed weighting function ω_j^i (cf. Equation 8). Note that weights are normalized, based on the number of user specified influences.*

7 Results

We tested our method on a variety of meshes obtained from the Internet (e.g., http://thefree3dmodels.com). No post processing or clean-up was performed on geometry. Due to the quick turnaround and aggressive time lines of most films and games, it is common to find production geometry with numerous artifacts. Figure 6 shows a few of the models we used for testing, with intersecting triangles shown in red. Other types of degeneracies we encountered include meshes composed of multiple disjoint parts, non-watertight meshes, and meshes with non-manifold edges/vertices (cf. Table 1 for test geometry statistics).

After downloading geometry, we manually created skeletons for each mesh. Alternatively, we could have used a method for automatic skeleton creation [Wade and Parent 2002; Baran and Popović 2007], however, this was outside the scope of our work. Once skeletons were defined, we used our method to calculate skinning weights for each set of input meshes/skeletons. Resulting weights were used to skin characters with LBS. All examples use four influences per vertex and $\alpha = 0.7$.

To test the resulting skinning quality, we animated all characters using a small library of motion clips. All motions were retargetted using Autodesk Maya HumanIK to account for differences in skeletal proportions between source and destination characters. In all cases, we obtained good default skinning weights without need for manual adjustment/weight painting. Additionally, we did not find that limiting the number of influences had a large impact on quality of skinning results. Table 2 lists durations for the different stages of our algorithm on 11 test meshes. All experiments were conducted on a MacBook Pro (2.2 GHz Intel Core i7) with 8GB of RAM and an AMD Radeon HD 6750M 1024 MB graphics card. Skinning results for each of these meshes can be found in the accompanying video.

Although our voxelization algorithm can be implemented on the CPU, we used a hardware accelerated implementation for improved performance. This required modest commodity hardware; our implementation used only fixed-function pipeline commands and relied on framebuffer objects (available since OpenGL 1.5) to manage bounding box slice images. For systems supporting OpenGL 1.1, an alternative implementation based on pbuffers could yield similar performance gains.

Table 1: *Statistics for 11 models used to test our method. Only the Beast model is watertight.*

Model	# Faces	# Non-manifold Vertices/Edges	# Separate Mesh Parts	# Intersecting Faces
Bloat	4680	3	6	399
Boomer	5297	0	11	472
Mercenary	9236	1	155	3029
Radioactive	10479	0	97	2842
Sledge	12850	5	390	6772
Engineer	13217	0	108	2845
Parasite	17570	2	88	4763
Hunter	17968	2	140	6758
Shockwave	23181	7	660	14982
Pilot	35644	0	239	9528
Beast	54236	0	1	0

Table 2: *Breakdown of algorithm run times for several example models. Total computation time is broken up according to the major steps used by our system. All values are in seconds.*

Model	Voxelization Internal	Voxelization Border	Distance Computation	Weight Computation
Bloat	6.679	3.832	2.900	0.003
Boomer	6.524	4.449	5.145	0.003
Mercenary	6.652	4.671	2.544	0.008
Radioactive	6.904	4.913	6.291	0.006
Sledge	6.822	5.413	3.712	0.031
Engineer	6.830	4.466	2.126	0.008
Parasite	7.016	3.489	0.582	0.041
Hunter	7.020	5.061	4.151	0.011
Shockwave	7.262	11.480	2.594	0.013
Pilot	7.660	4.528	3.637	0.020
Beast	8.426	4.398	1.343	0.027

8 Discussion

Binding results and performance for our method depend on the voxelization resolution. To simplify the implementation, we used the same maximum number of voxels (i.e., $256 \times 256 \times 128$ in the x, y and z directions) for all models. We found that this worked well despite there being large differences in test model proportions, smoothness, and number of vertices. An alternate strategy could be to base the voxelization resolution on some geometric feature in the input model, such as the smallest distance between two vertices. The impact of voxelization resolution was particularly noticeable for A-stance rest pose meshes, such as Shockwave. Figure 7 shows such a case; as the maximum number of voxels decreases, the hands and legs become connected which impacts distance computations, since character topology changes. For models such as Beast (see accompanying video and Figure 8) that are in a T-stance, the voxelization resolution has a minor impact on the quality of computed weights. Other areas requiring fine detail, such as the fingers will also be sensitive to the choices of voxelization resolution. Practically, we found that a quick visual inspection of the resulting voxelization is sufficient to catch these sorts of problems. If the user wishes to add more detail, it is possible to increase the voxelization resolution until desired results are obtained. An alternate approach, might be to generate a sparse voxelization with additional detail near small features in a manner similar to Schwartz et al. [2010]. We leave this for future work.

We experimented with a number of distance metrics prior to settling on the Euclidean norm between Manhattan voxel neighbor centers. This included using Manhattan distance between voxels, and the Euclidean norm to the center of all adjacent voxels. These alternate

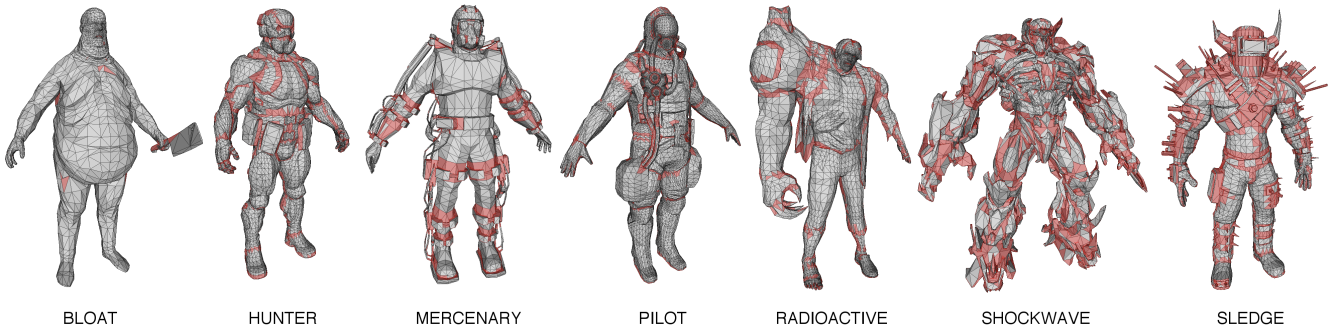

Figure 6: *7 of the 11 models used for testing. Parasite model is shown in the accompanying video. Engineer can be seen in Figure 1, Boomer is shown in Figure 3 while Beast is in Figure 8. Intersecting triangles are shown in red. Statistics for all test models can be found in Table 1.*

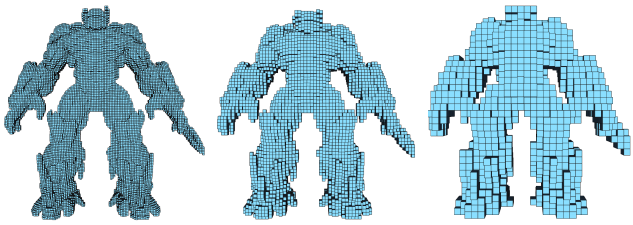

Figure 7: *Shockwave model voxelized at different resolutions (Left:* $128 \times 128 \times 64$*, Middle:* $64 \times 64 \times 32$*, Right:* $32 \times 32 \times 16$*). As resolution decreases character topology can change which alters the minimum distance between voxels falling on bones and non-exterior voxels.*

metrics did not yield significant improvements, but required more memory and complicated implementation. We also found our chosen distance metric to be well suited for domains with non-uniform voxels; by using the Euclidean norm to voxel centers we correctly account for differences in voxel sizes that results from discretizing the bounding box using a fixed number of voxels. For models with few branches, such as Boomer (cf. Figure 3), distance computations can become expensive. This occurs since the queue used to keep track of minimal distances (cf. Algorithm 1) can grow significantly.

A major motivation for the development of our new voxelization algorithm came from experimenting with two alternate weight assignment schemes. In the first method, we started by computing a discreet signed distance field in the domain of the axis-aligned bounding box surrounding the skin mesh [Frisken et al. 2000]. Using this field, we calculated an isosurface at a small positive level set outside the model. We used the resulting isosurface as a deformation cage that could be driven by the animated skeleton, while the original geometry was embedded in the cage using a variant of Green coordinates [Lipman et al. 2008]. Although we often obtained good results using this approach, it introduced a costly level of indirection at pose time. Additionally, since there isn't a straightforward method for determining LBS weights for the original mesh, this made the method unusable by applications requiring skinning weights for existing closed-form methods. To overcome this limitation, we explored a second option that voxelized the resulting (watertight) isosurface, using the method of Crane et al. [2008]. This worked well in many cases, but the test used to tag discreet points in the signed distance field, as interior/exterior, often returned incorrect results. This test is dependent on normal values which can be incorrect for models with multiple disconnected components. In both cases, we found distance field calculation to be a major computational bottleneck, often taking several minutes.

Several interesting applications arise from our voxelization based weighting algorithm. Since the process of calculating weights is extremely fast (cf. Table 2) once the input mesh has been voxelized and distances are computed, it is possible to update geometry with a mesh of similar topology that can be fully enclosed by the original voxelization volume. For example, in a game scenario, skinning weights for geometry can be quickly recalculated as a character's appearance is altered (e.g., to account for being injured or joining the undead). Additionally, since our weighting function has a simple parameterization depending only on α, it is possible for user to interactively alter the falloff of individual areas of the mesh and immediately see how the resulting deformations are affected. A simple extension would be to make skinning weights pose dependent by turning α for different regions of the mesh into a function of joints orientations. Lastly, since our weight parameterization comes from a distance based field, it is straightforward to specify regions of the mesh that should not be deformed at runtime. This could be useful for characters with a mix of rigid equipment and deformable parts/clothing. In such cases bounding regions could be defined that override computed distances, assigning infinite distance to all bones except for a chosen "driver".

9 Conclusions

We present a novel system for determining influence weights for closed-form skinning methods, such as LBS. Our method is specifically designed to circumvent problems we encountered with previous approaches that require watertight geometry. To achieve this, we propose a new voxelization strategy that does not require distance field or isosurface computation and works on real-world degenerate geometry. By not using the stencil buffer and extra clipping planes, our voxelization algorithm is straightforward to implement. The only requirement for our algorithm is a mechanism that renders the view state to an image. We use frame buffers, but other approaches could also work. A flexible voting scheme also helps us robustly handle ambiguities that arise when multiple triangles intersect.

Our voxelization algorithm will benefit numerous applications including collision detection, CSG modeling, fluid simulation [Crane et al. 2008], model simplification [Nooruddin and Turk 2003], or could be used to automatically extract skeletons, as part of a complete auto-rigging systems [Wade and Parent 2002]. Our method

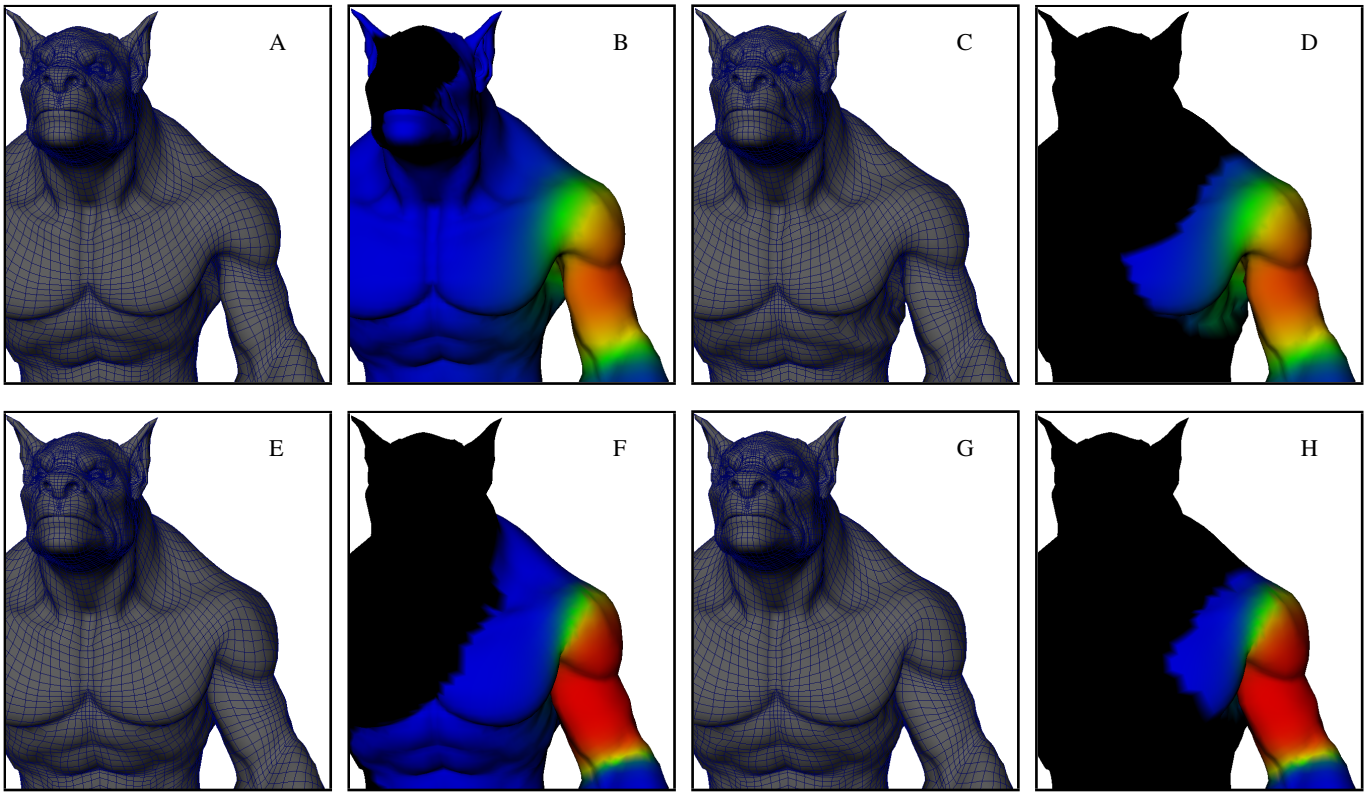

Figure 8: *Comparison with Baran et al. [2007] (A-D) with our method (E-H). On watertight meshes Baran's method generates appealing results (A) but yields many small weights far from the joint of interest (B). When the number of weights is limited (D), as is often done in games to enable vectorization, artifacts may be visible (C) (e.g., see armpit region). Our method produces visually indistinguishable results (E) with few small weights (F). Because we concentrate most influences on bones near the joints (H), limiting influences does not introduce problems (G). Results shown use* $\alpha = 0.7$.

is also complementary to the recent subspace energy minimization method of Jacobson and colleagues [2012] which requires initial weights to be specified for a small number of skeleton bones. Our method does not require a fully connected skeleton and can thus easily be applied to disjoint hierarchies.

References

AKENINE-MÖLLER, T. 2001. Fast 3d triangle-box overlap testing. *Journal of Graphics Tools 6*, 29–33.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3 (July).

BARBIČ, J., AND ZHAO, Y. 2011. Real-time large-deformation substructuring. *ACM Trans. on Graphics 30*, 4, 91:1–91:7.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformations. *ACM Trans. Graph. 21*, 3 (July), 586–593.

CAPELL, S., BURKHART, M., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2005. Physically based rigging for deformable characters. In *Proc. of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA '05, 301–310.

CHEN, C.-H., LIN, I.-C., TSAI, M.-H., AND LU, P.-H. 2011. Lattice-based skinning and deformation for real-time skeleton-driven animation. In *Proc. of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics*, IEEE Computer Society, Washington, DC, USA, CADGRAPHICS '11, 306–312.

CRANE, K., LLAMAS, I., AND TARIQ, S. 2008. Real-time simulation and rendering of 3d fluids. In *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 633–675.

DONG, Z., CHEN, W., BAO, H., ZHANG, H., AND PENG, Q. 2004. Real-time voxelization for complex polygonal models. In *Proc. of the Computer Graphics and Applications, 12th Pacific Conference*, IEEE Computer Society, Washington, DC, USA, PG '04, 43–50.

EISEMANN, E., AND DÉCORET, X. 2008. Single-pass gpu solid voxelization for real-time applications. In *Proc. of graphics interface 2008*, GI '08, 73–80.

FANG, S., FANG, S., CHEN, H., AND CHEN, H. 2000. Hardware accelerated voxelization. *Computers and Graphics 24*, 200–0.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proc. of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, 249–254.

JACOBSON, A., AND SORKINE, O. 2011. Stretchable and twistable bones for skeletal shape deformation. *ACM Trans. on Graphics 30*, 6, 165:1–165:8.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. on Graphics 30*, 4, 78:1–78:8.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (July), 77:1–77:10.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (July).

JU, T., ZHOU, Q.-Y., VAN DE PANNE, M., COHEN-OR, D., AND NEUMANN, U. 2008. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph. 27*, 5 (Dec.), 122:1–122:10.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3 (July), 954–961.

KAVAN, L., AND SORKINE, O. 2012. Elasticity-inspired deformers for character articulation. *ACM Trans. on Graphics 31*, 6, 196:1–196:8.

KAVAN, L., COLLINS, S., ZARA, J., AND O'SULLIVAN, C. 2007. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 39–46.

KIM, T., AND JAMES, D. L. 2011. Physics-based character skinning using multi-domain subspace deformations. In *Proc. of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '11, 63–72.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA '02, 153–159.

LEE, S.-H., SIFAKIS, E., AND TERZOPOULOS, D. 2009. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph. 28*, 4 (Sept.), 99:1–99:17.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proc. of the 27th annual conference on Computer graphics and interactive techniques*, 165–172.

LI, W., FAN, Z., WEI, X., AND KAUFMAN, A. 2005. Flow simulation with complex boundaries. In *GPU Gems 2*, M. Pharr, Ed. Addison-Wesley, 747–764.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph. 27*, 3 (Aug.), 78:1–78:10.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 26–33.

MCADAMS, A., ZHU, Y., SELLE, A., EMPEY, M., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30*, 4 (July), 37:1–37:12.

MILLER, C., ARIKAN, O., AND FUSSELL, D. 2011. Frankenrigs: Building character rigs from multiple sources. *IEEE Trans. Vis. Comput. Graph. 17*, 8, 1060–1070.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Transcations on Graphics 22*, 3 (jul), 562–568. Special Issue: Proc. of ACM SIGGRAPH 2003.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. on Visualization and Computer Graphics 9*, 2 (Apr.), 191–205.

SCHWARZ, M., AND SEIDEL, H.-P. 2010. Fast parallel surface and solid voxelization on gpus. *ACM Trans. Graph. 29*, 6 (Dec.), 179:1–179:10.

SLOAN, P.-P. J., ROSE, III, C. F., AND COHEN, M. F. 2001. Shape by example. In *Proc. of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, I3D '01, 135–143.

WADE, L., AND PARENT, R. E. 2002. Automated generation of control skeletons for use in animation. *The Visual Computer 18*, 2, 97–110.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26*, 3 (July).

WAREHAM, R., AND LASENBY, J. 2008. Bone glow: An improved method for the assignment of weights for mesh deformation. In *Proc. of the 5th international conference on Articulated Motion and Deformable Objects*, AMDO '08, 63–71.