Geodesic Binding for Degenerate Character Geometry Using Sparse Voxelization

Olivier Dionne and Martin de Lasa

Abstract—We propose a fully automatic method for specifying influence weights for closed-form skinning methods, such as linear blend or dual quaternion skinning. Our method is designed to work with production meshes that may contain non-manifold geometry, be non-watertight, have intersecting triangles, or be comprised of multiple connected components. Starting from a character rest pose mesh and skeleton hierarchy, we first voxelize the input geometry. The resulting sparse 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.

Index Terms—Character Animation, Skinning, Sparse Voxelization, Deformation

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 deformations 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 game engines. 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. [1] for an in-depth discussion. Additionally, since the skeleton and mesh are required to be in rest configurations 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 many advanced 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 (e.g., [2], [3], [4], [5], [6]) 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 examples.

In this paper, we propose a fully automatic method for specifying influence weights for closed-form skinning methods, designed to work with production geometry. Our method, Geodesic Voxel Binding (GVB), leverages a novel voxelization algorithm (Section 5) that is well suited for parallelization on commodity hardware. We propose a simple weighting scheme, based on discrete geodesic distances, that leverages the resulting voxelization to generate smooth weights. The method requires no time-constants, iteration parameters, or optimization at bind or pose time (Section 7). Additionally, by decoupling weight assignment from distance computation it is possible to modify weights interactively, at pose time, without requiring additional pre-processing or computation. Several interesting applications result from our approach which we discuss in Section 9.

• *Authors are with Autodesk Inc.*
E-mail: [olivier.dionne, martin.delasa]@autodesk.com

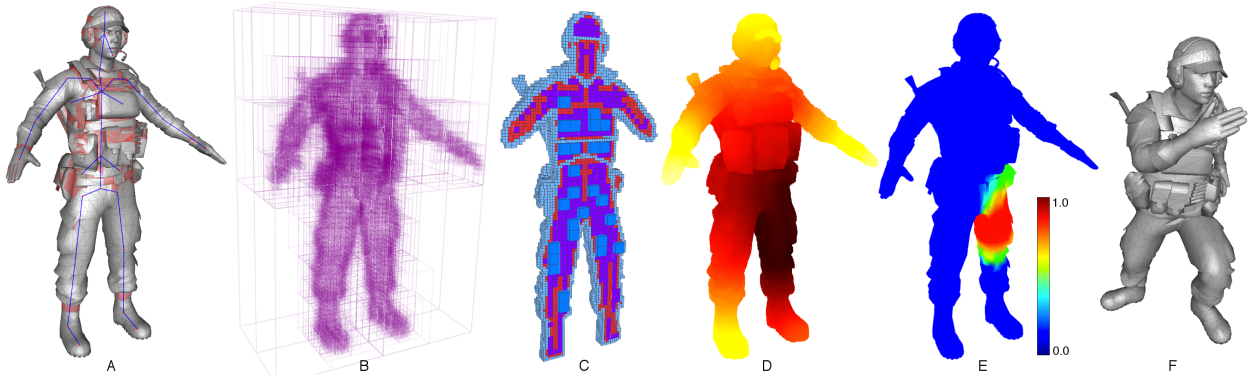

Fig. 1. Given a character skeleton and mesh, which may contain degenerate geometry (in red) (A), we preprocess the input and compute an octree (B) prior to generating a sparse voxelization using graphics hardware (C). From this voxelization, geodesic distances are computed (D) and mapped to bind weights (E) which can be applied to closed-form skinning methods to deform character geometry during animation (F).

2 Related Work

2.1 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 [7], approximate elastic models that may consider contact [8], [9], [10], reduced-order methods that use key deformation modes [11], [12], methods that rely on examples to compute corrections [1], [13], [14], [15], [16], volume-based methods that embed geometry in cages [17], [18], [19], and closed-form methods that deform geometry based on character pose [20], [21], [22], [23].

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 [24] propose a resolution-dependent mesh segmentation strategy that can be applied to skinning. Wade and Parent [25] use voxelization to extract a skeleton but resulting weights do not vary smoothly, causing artifacts on high-resolution meshes. Baran and Popović [2] 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 [3], [17] or lighting models [6], iteration parameters, time-constants, or convergence thresholds must be specified for use as termination criteria; our method requires no such parameters.

Some recent methods have been proposed that select weights by optimization. Bounded biharmonic weights (BBW) [4] uses a quadratic program (QP) to minimize Laplacian energy subject to interpolation constraints. This yields smooth weights and supports a number of influence types, enabling interesting applications for 2D and 3D shapes. However, the method is very sensitive to the quality of input geometry; generating the required tetrahedral meshing often fails for watertight geometry due to the presence of self-intersecting triangles. Kavan and colleagues [22] use elastic energy and perform computations on a voxel grid. Hence, their method is also capable of handling many types of geometry encountered in production. Though results are impressive, both approaches are quite costly; reported results take several minutes to compute. A more detailed comparison with BBW follows in Section 8.

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 cost introduced by optimization. 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 (Figure 2) 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 [5]. Our approach could ease authoring of new weight templates for such systems.

2.2 Voxelization

Although numerous real-time voxelization algorithms have been devised most focus on building surface voxelizations (e.g., [26], [27], [28]). 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 [27] use a slice-wise approach, while Eisemann and Décoret [29] process all slices simultaneously using fixed-function pipeline commands.

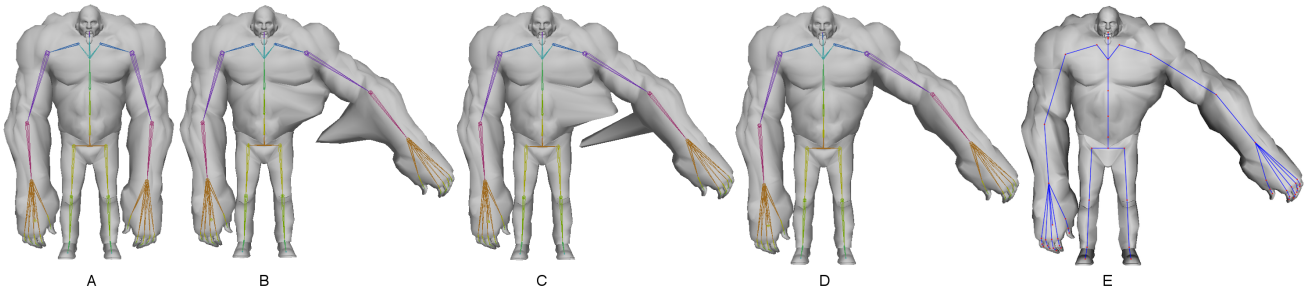

Fig. 2. Comparison of other automatic weighting schemes (B-D) with our method (E). (A) Initial bind pose. (B) Closest distance (Maya) (C) Closest hierarchy (Maya) (D) Heat map weighting [2] (E) Geodesic voxel binding. Distance-based methods that ignore mesh structure (B,C) can produce severe artifacts. For watertight meshes we obtain similar results to Heat map weighting (D,E). See Figure 11 for a more complete discussion.

Our voting scheme is inspired by the work of Nooruddin and Turk [30] who use multiple views and ray stabbing to classify 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 and slices geometry on graphics hardware. For low scoring voxels, that are challenging to classify, we use generalized winding numbers [31] (Section 5). This is significantly faster than relying exclusively on winding numbers as costly computation is limited to a small set of hard to categorise voxels.

Recent GPU accelerated approaches have also been proposed that generate sparse voxelizations [32]. To use these approaches on degenerate geometry, it is often necessary to first compute an intermediate watertight mesh using a signed distance field, which easily dominates computation. Starting from a uniform voxelization, we propose a method that builds a sparse voxel representation, at minimal additional cost. The resulting voxelization includes fine details not only on the mesh boundary, but also at interior bone voxels. It is straightforward to exclude fine interior details if applications do not require them.

3 Overview

We describe an algorithm that automatically computes deformation weights for interactive skin deformation algorithms, such as linear blend skinning (LBS). In addition to LBS, our approach can also be used with more advanced skinning algorithms, such as the recent work of Kavan et al. [22] or as a preprocessing step to the implicit method of Vaillant et al. [33].

The input to our system is a user specified rest-pose mesh $\mathbb{M}$ with vertex positions:

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

and corresponding skeleton $\mathbb{S}$, represented as a hierarchy of transformations:

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

Skinning methods require a weight function:

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

that specifies the amount of influence $\mathbf{T}_j$ has on a vertex with position $\mathbf{p}$. Hence, vertices on $\mathbb{M}$ will deform 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 deformation 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 [2].

To determine weights, our method first performs preprocessing operations on $\mathbb{M}$ and $\mathbb{S}$, including scaling and octree $\mathbb{O}$ computation. From the scaled mesh $\mathbb{M}'$, scaled skeleton $\mathbb{S}'$, and $\mathbb{O}$, a uniform voxel representation of the geometry $\mathbb{U}$ is then computed (Section 5). A voting scheme classifies resulting voxels as interior or exterior. From $\mathbb{O}$ and $\mathbb{U}$, a sparse voxelization $\mathbb{V}$ is then calculated and used to compute geodesic distances $\mathbf{d}$ between voxels falling on skeleton bones and all non-exterior voxels. This step is repeated for each skeleton bone (Section 6). Lastly, skinning weights are computed using a distance-based falloff function. Figure 1 shows key steps for our method; pseudocode is provided in Algorithm 1.

Algorithm 1: Method Overview

input : Mesh $\mathbb{M}$, skeleton $\mathbb{S}$, and resolution r
output: Skinning weights ω

1 $\mathbb{M}', \mathbb{S}', \mathbb{O} \leftarrow$ preprocess($\mathbb{M}, \mathbb{S}, r$) § 4
2 $\mathbb{V} \leftarrow$ voxelization($\mathbb{M}', \mathbb{S}', \mathbb{O}, r$) § 5
3 $\mathbf{d} \leftarrow$ computeDistance($\mathbb{V}$) § 6
4 $\omega \leftarrow$ computeWeights($\mathbb{M}', \mathbb{O}, \mathbf{d}$) § 7

4 PREPROCESSING

To reduce scale dependent artifacts from our distance-based binding metric (Section 6), and force computation to occur in well behaved interval of the floating point number range, we perform a few preprocessing operations on the input mesh $\mathbb{M}$ and skeleton $\mathbb{S}$.

First, we uniformly rescale $\mathbb{M}$ and $\mathbb{S}$ such that the axis-aligned bounding box (AABB) of the resulting scaled mesh $\mathbb{M}'$ and skeleton $\mathbb{S}'$ fit inside the unit cube $[0,1]^3$.

Second, we compute an octree $\mathbb{O}$, to a user specified maximum resolution r, for all geometric primitives in $\mathbb{M}'$ and line segments corresponding to bones in $\mathbb{S}'$. We evaluate intersections between voxels and input primitives using accelerated tests based on the separating-axis theorem [34]. This yields a list of boundary and bone voxels, respectively. $\mathbb{O}$ is reused in multiple subsequent steps in our pipeline to accelerate computation and lookups, by focusing computation on areas of high-detail.

Once preprocessing is complete, we proceed to compute a sparse voxelization for $\mathbb{M}'$ and $\mathbb{S}'$.

5 VOXELIZATION

To voxelize the input geometry, we propose an approach based on z-buffer slicing that yields a uniform voxelization. Our hardware-accelerated method is inspired by the approach of Fang et al. [27], with a voting scheme adapted from Nooruddin and Turk's [30] multiple view parity counting method. Low scoring voxels can optionally be classified using generalized winding numbers [31]. Once all voxels have been categorized, we use the octree $\mathbb{O}$, obtained during preprocessing (Section 4), to generate a sparse voxelization of the mesh volume.

5.1 Uniform Voxelization

Our uniform voxelization algorithm starts from the orthographic view volume of the normalized mesh obtained during preprocessing (Section 4). 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 offscreen buffer, initialized to black and of same slice dimensions with no filtering, followed by front faces in black. Once all faces are rendered, white pixels (which correspond to voxels in the slice) are tagged as internal (1), while black pixels are tagged as external (0). This process is repeated until the entire mesh bounding volume has been processed for all pairs of x, y, and z view directions (Figure 3).

To classify a voxel v_i, we compile votes from each $\pm x$, $\pm y$, $\pm z$ view pair according to:

$$\begin{aligned} v_{i_{(+x,-x)}} &: \{0,1\} \leftarrow v_{i_{+x}} \parallel v_{i_{-x}} \\ v_{i_{(+y,-y)}} &: \{0,1\} \leftarrow v_{i_{+y}} \parallel v_{i_{-y}} \\ v_{i_{(+z,-z)}} &: \{0,1\} \leftarrow v_{i_{+z}} \parallel v_{i_{-z}}. \end{aligned} \quad (5)$$

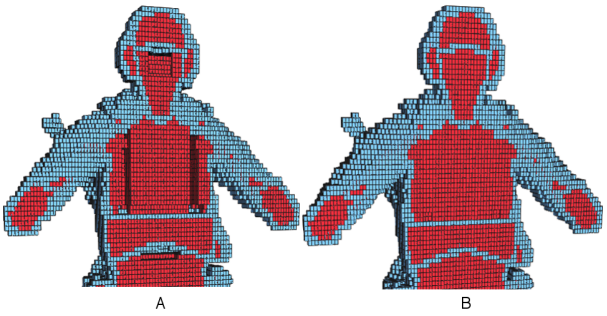

Fig. 4. Complex models may have hard to categorise interior voxels. A uniform voxelization generated by the majority voting scheme (A). Corrected voxelization from voting method and generalized winding numbers [31] applied to low scoring voxels (B).

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:

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

Remaining untagged voxels are considered external to the mesh domain.

For all test models, the proposed majority voting scheme correctly categorizes voxels with 2 or more votes. However, when processing badly degenerate meshes with many intersecting triangles, low scoring voxels may be misclassified. Although having a few mislabelled voxels typically has a negligeable impact on bind quality, since multiple paths can exist through the voxelized domain, interior gaps may affect distance value smoothness (Section 6). Additionally, if entire bones lie in mislabelled interior regions they will not be bound to the geometry, as no path will exist through the volume to the mesh surface.

We propose to use generalized winding numbers [31], to classify voxels with a single vote. Using this accurate check on a small set of ambiguous voxels performs significantly faster than just using winding numbers to create a uniform voxelization (Section 8).

5.2 Sparse Voxelization

To make calculation of geodesic distances (Section 6) more computationally and memory efficient, we propose an algorithm, inspired by texture mipmaps, that generates a sparse voxelization $\mathbb{V}$ from the uniform voxel grid $\mathbb{U}$ and octree $\mathbb{O}$.

Algorithm 2 begins by recursively creating $n = \log_2(r)$, representations of $\mathbb{U}$ at different scales $\mathbb{G}_n, \ldots, \mathbb{G}_0$, where $\mathbb{G}_n = \mathbb{U}$. Each $\mathbb{G}_i$ contains voxels v_j tagged as interior or exterior. If at least one of the eight higher resolution voxels is tagged as non-exterior then v_j is marked as interior (lines 11-14).

At each resolution, $i \in \{n-1 \ldots 0\}$, we initialize all voxels in $\mathbb{G}_i$ (line 5), as exterior and compute offsets $\mathbf{O}$, corresponding to the voxel half-width at resolution $i+1$ (line 6). Next, for each voxel v_j of $\mathbb{G}_i$, we test if

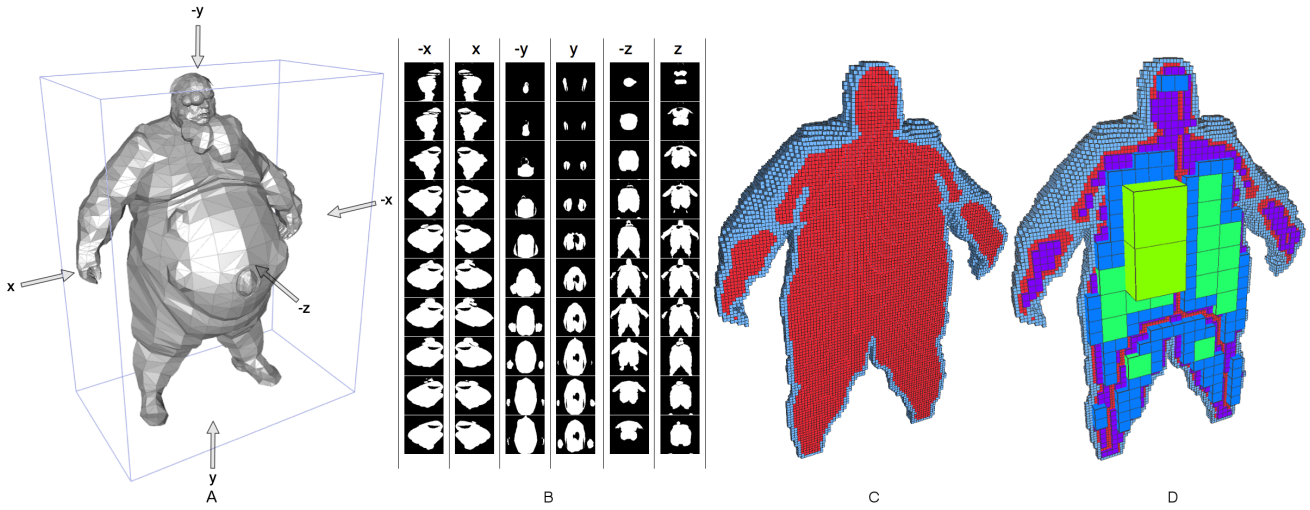

Fig. 3. Starting from the normalized character world-space AABB, defining the initial orthographic view volume (A), the camera is positioned along every pair of x, y, z view directions and the model is sliced by moving the near plane (B). After compiling votes for each view using equations (5) and (6), misclassified voxels are corrected. This yields a uniform voxelization of the input volume (C). Internal/boundary voxels are shown in red/blue respectively. A sparse voxelization is then created using Algorithm 2 (D).

any of the voxels it embeds, from $\mathbb{G}_{i+1}$, are interior. For each embedded voxel, we check the class of voxels at positions $\mathbf{p}_j^k = \mathbf{p}_j + \mathbf{O}(:,k)$ in $\mathbb{G}_{i+1}$, where, $\mathbf{p}_j$ is the position of the center of voxel v_j and $\mathbf{O}(:,k)$ is the k^{th} column $\mathbf{O}$. Note that since all voxel checks are independent, lines 7-16 can be executed in parallel.

Given that the computed multi-resolution grids $\mathbb{G}_0, \ldots, \mathbb{G}_n$ map directly to the octants of $\mathbb{O}$ at different depth levels, the sparse mesh voxelization $\mathbb{V}$ can be extracted by traversing all octree leaves (line 18) and checking if the corresponding octant is non-exterior (lines 22-24). Once $\mathbb{V}$ has been created $\mathbb{G}$'s can be discarded.

Although the proposed scheme is quite simple, we have found it to be robust, efficient, and straightforward to implement.

6 Distance Computation

To compute the shortest distance between interior bone and boundary voxels, we use a form of Djikstra's algorithm (Algorithm 3).

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 compute the distance $dist$ between the voxel's center $\mathbf{p}_{v_i}$ and it's neighboring voxels v_j (line 17). To encourage walks through the interior voxel volume, distances to boundary voxels are penalized by a user-specified parameter, $\epsilon_{penalty} > 1$ (line 15). Since we seek the shortest path through the voxel representation, if the stored distance d_{v_j} is greater than $dist$, we update d_{v_j} and add v_j to the queue (lines 10-23). The algorithm proceeds recursively yielding the discrete geodesic distance from bone b_i to every voxel of the mesh domain.

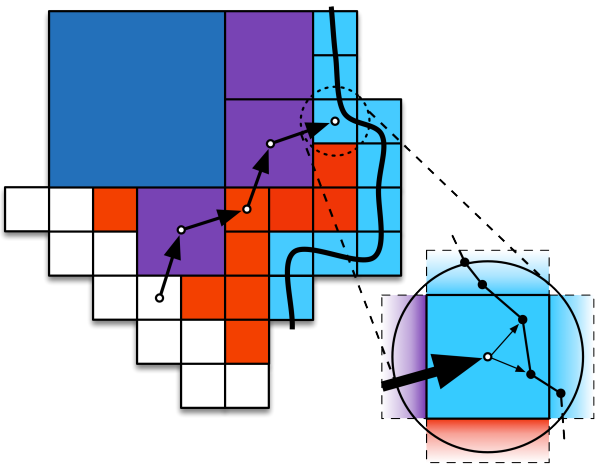

Fig. 5. Distances through the voxelized volume, between bone (white) and boundary voxels (light blue), are computed using Algorithm 3. We account for the distance between the voxel center and the embedded mesh vertex once a boundary voxel is reached.

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

7 Weight Computation

Once distances for non-exterior voxels $\mathbf{d}$ are calculated we can compute skinning weights.

Algorithm 2: Sparse Voxelization Computation

input : Uniform voxelization $\mathbb{U}$ with resolution r, Octree $\mathbb{O}$
output: Sparse voxelization $\mathbb{V}$

$\mathbf{S} \leftarrow \begin{bmatrix} -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{bmatrix}$
$n \leftarrow \log_2(r)$
$\mathbb{G}_n \leftarrow \mathbb{U}$
// Compute lower resolution grids
for $i \leftarrow n-1$ **to** 0 **do**
  // Create lower resolution grid
  $\mathbb{G}_i \leftarrow$ createGrid(2^i)
  // Compute high resolution grid offsets
  $\mathbf{O} \leftarrow \mathbf{S}/2^{i+2}$
  // Fill lower resolution grid
  foreach *voxel* v_j *of* $\mathbb{G}_i$ **do**
    $\mathbf{p}_j \leftarrow$ voxelCenter(v_j)
    for $k \leftarrow 0$ **to** 7 **do**
      $\mathbf{p}_j^k \leftarrow \mathbf{p}_j + \mathbf{O}(:, k)$
      if *voxelAt*$(\mathbf{p}_j^k, \mathbb{G}_{i+1})$ *is not exterior* **then**
        $v_j \leftarrow$ *interior*
        break
      end
    end
  end
end
// Populate $\mathbb{V}$ by traversing octree
foreach *leaf* l *of* $\mathbb{O}$ **do**
  $\mathbf{p} \leftarrow$ octantCenter(l)
  $d \leftarrow$ octantDepth(l)
  $v \leftarrow$ voxelAt$(\mathbf{p}, \mathbb{G}_d)$
  if v *is not exterior* **then**
    Push v *to* $\mathbb{V}$
  end
end

Algorithm 3: Distance Computation

input : Character skeleton $\mathbb{S}$ and voxelized mesh $\mathbb{V}$
output: Distance between all bone and boundary voxels $\mathbf{d}$

foreach *bone* b_i *of* $\mathbb{S}$ **do**
  // Initialize voxel distance values
  foreach *non-exterior voxel* v_i *of* $\mathbb{V}$ **do**
    $d_{v_i} \leftarrow \infty$
  end
  Create empty voxel queue $\mathbf{Q}$
  // Initialize and enqueue bone voxels
  foreach *non-exterior voxel* v_i *of* $\mathbb{V}$ *intersecting with* b_i **do**
    $d_{v_i} \leftarrow 0$
    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**
      $\beta \leftarrow 1$
      if v_j *is boundary* **then**
        $\beta \leftarrow \epsilon_{penalty}$
      end
      $dist \leftarrow d_{v_i} + \beta|\mathbf{p}_{v_i} - \mathbf{p}_{v_j}|$
      if $d_{v_j} > dist$ **then**
        $d_{v_j} \leftarrow dist$
        Push v_j *to* $\mathbf{Q}$
      end
    end
  end
end

To compute mesh vertex weights (Equation 4) we use $\mathbb{O}$ to identify voxels containing vertices. 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 center of voxel v_j and the mesh vertex position $\mathbf{p}_{vertex}$ to the current bone distances d_v^i in the voxel, as shown in Figure 5.

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

$$d_j^i \leftarrow d_v^i + |\mathbf{p}_{vertex} - \mathbf{p}_{v_j}| \quad (7)$$

$$\lambda \leftarrow (1-\alpha)\lambda_{min} + \alpha\lambda_{max} \quad (8)$$

$$\omega_j^i \leftarrow max(d_{tol}, d_j^i)^{-\lambda} \quad (9)$$

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 more local bind. We found that using $\lambda_{min} = 5$ and $\lambda_{max} = 30$ gave good results. To ensure Equation 9 is well behaved, we ensure $d_j^i \geq d_{tol}$, where d_{tol} is some small minimum distance to avoid numerical problems. Figure 6 shows a visualization of the chosen falloff function.

Though Equation 9 works well on the characters we tested, providing the ability to control bind stiffness, other falloff functions could also have been employed. When the number of influence 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 11.

Once all weights have been computed they are normalized to ensure $|\omega| = 1$ prior to skinning. To normalize weights we use a maximum number of user-specified influences. If the number of maximum

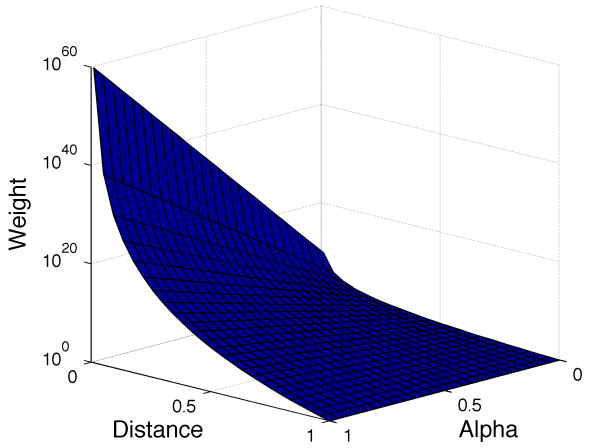

Fig. 6. Visualization of proposed weighting function ω_j^i (Equation 9). Note that weights are normalized, based on the number of user specified influences.

influences is not explicitly limited, weights are normalized based on the total number of joints/bones affecting geometry (Section 9).

8 Results

We tested our method on a variety of meshes obtained from http://tf3dm.com. Figure 8 shows some of our test models with intersecting triangles illustrated in red. No post processing or clean-up was performed on the input geometry. Due to the quick turnaround and aggressive schedules of most films and games, it is common to find production meshes with artifacts, including multiple disjoint parts, non-watertight geometry, and non-manifold edges/vertices. Table 1 lists statistics for our test set.

After downloading geometry, we manually created skeletons for each character and used GVB to calculate skinning weights for each set of input meshes/skeletons. The resulting weights were then used to skin characters using LBS.

To test quality of the resulting bind, we animated all test characters using a small library of motion clips. To account for differences in skeletal proportions between the source and destination characters all motions were retargeted using Autodesk Maya HumanIK. In all cases, we obtained good default skinning weights without manual adjustment. Limiting the number of bone influences per vertex did not significantly impact bind quality.

Skinning results for each test mesh can be found in the accompanying video. Unless noted, examples use 4 influences per vertex, a maximum resolution of $r = 256^3$, $\epsilon_{penalty} = 4$, $\alpha = 0.1$, as well as generalized winding number correction for low scoring voxels.

Although our voxelization algorithm can be implemented on the CPU, we use a hardware accelerated implementation for improved performance. This required modest commodity hardware; our implementation uses only fixed-function pipeline commands

TABLE 1
Statistics for 11 test models. Only Beast 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	853

and relies on framebuffer objects (available since OpenGL 1.5) to manage bounding box slice images. For systems limited to OpenGL 1.1, an alternative implementation based on pbuffers could yield similar performance gains.

A major bottleneck in the voxelization process (Section 5) is the GPU to CPU transfer of buffer slice images for each view direction. A straightforward strategy to reduce this overhead is to bit slice individual images into a single texture. For $r = 256^3$, using a standard RGBA texture, with 8 bits per component, reduces the number of readbacks from 1536 to only 48. Using more advanced texture formats, with higher numbers of bits per component, could further decrease transfers.

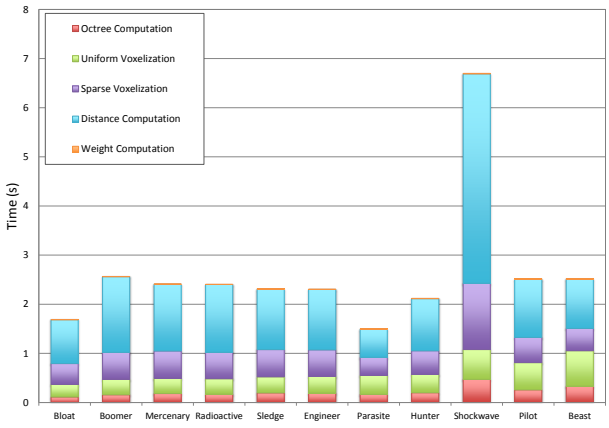

Fig. 7. Breakdown of algorithm run times for test models at 256^3 maximum voxel resolution. Generalized winding number classification was not used.

A detailed breakdown of GVB run times, on the 11 test meshes, can be found in Figure 7. With the exception of Shockwave, all test models are bound in under 3 seconds. Shockwave takes around 7 seconds due to its many small surface details, which limit voxelization sparsity. This results in longer walks through many uniformly voxelized volume regions. All tests were conducted on a MacBook Pro (2.2 GHz Intel Core i7) with 8GB of RAM and an AMD Radeon

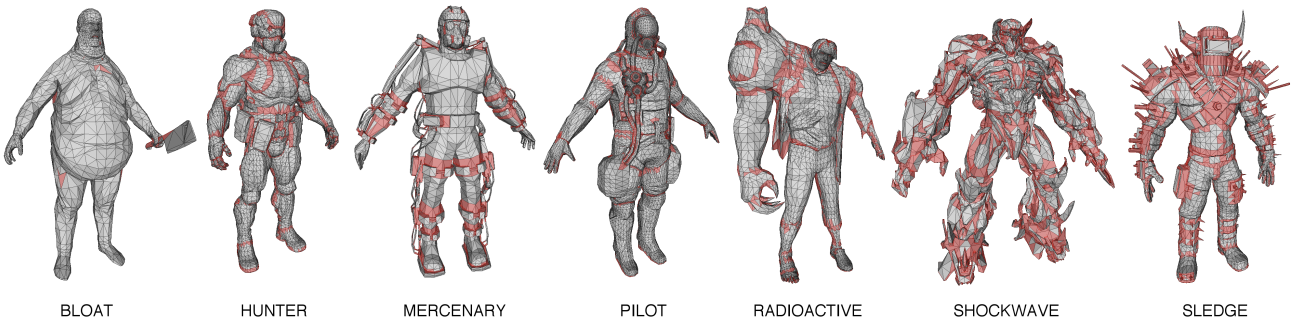

Fig. 8. Illustrations for 7 out of 11 test models with intersecting triangles shown in red. Figure 1, Figure 3, and Figure 11 show Engineer, Boomer, and Beast respectively. Parasite is shown in the accompanying video. Table 1 lists test model statistics.

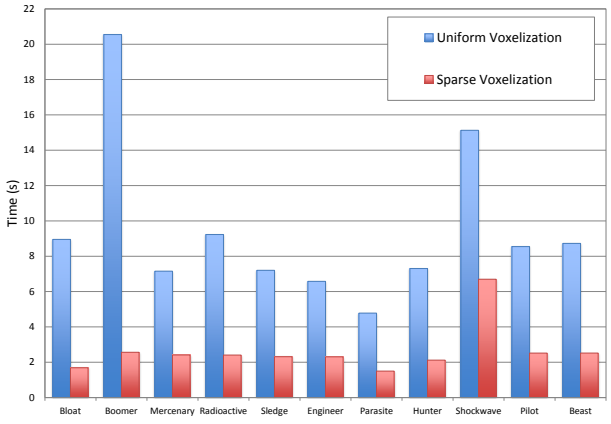

Fig. 9. Algorithm run times for sparse vs. uniform voxelization. The sparse voxelization provides significant speed/memory savings when calculting geodesic distances.

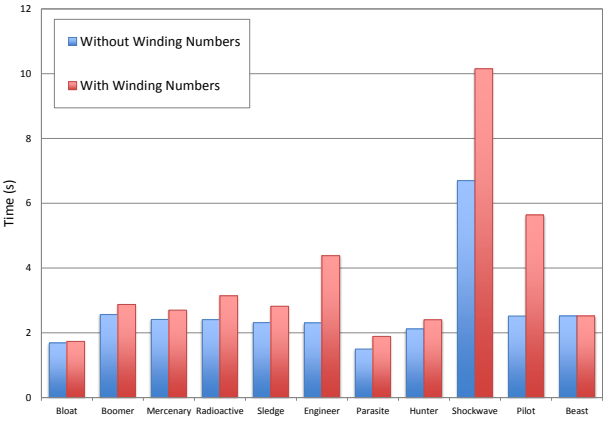

Fig. 10. GVB run times with/without generalized winding number correction. Using generalized winding numbers on low-scoring voxels improves voxelization quality but may increase processing time.

HD 6750M 1024 MB graphics card.

To assess the impact of sparse voxelization on GVB performance, binding was repeated with and without the sparse voxelization (Figure 9). By creating the sparse representation, which took between $0.3 - 1.3$ seconds, run times for all test models improved significantly. For models with large regions of interior voxels, such as Boomer, using the sparse voxelization produces binding results approximately 5 times faster than using the uniform representation.

Figure 10 shows GVB run times with and without generalized winding number classification of low-scoring voxels. For models with many interpenetrating triangles (Figure 8), using winding numbers can have a significant impact on overall performance; Pilot takes roughly twice as long to bind, while Shockwave and Engineer take approximately 30% longer. Other models, including Bloat, Boomer, Hunter, and Mercenary, are impacted more moderately. As expected, run times for well-structured watertight meshes, such as Beast, do not vary significantly with/without winding numbers since the proposed voxelization correctly classifies interior voxels.

Geodesic voxel binding performance and bind quality was compared to two existing weight selection methods: heat diffusion (HD) [2] and BBW [4], using implementations found in Autodesk Maya and libigl [35] respectively. As both HD and BBW are limited to single watertight meshes, we focus testing on compatible input geometry. For HD we use the watertight Beast model (Figure 11). For BBW, which requires a tetrahedral meshing of the mesh, we use the Stanford Armadillo (Figure 12); despite being watertight we cannot use Beast to test BBW as Tetgen [36] fails on this model due to intersecting triangles (Table 1). Table 2 summarizes run times for all 3 methods.

Figure 11 shows a comparison of HD and GVB, with and without influence weights truncation. Generally HD produces smooth results, with many small weights assigned to distant influences. As shown in Figure 11c, truncating and renormalizing HD weights, to enable vectorization for interactive applications, can result in artifacts. Due to the local support of calculated weights, truncating GVB results does not

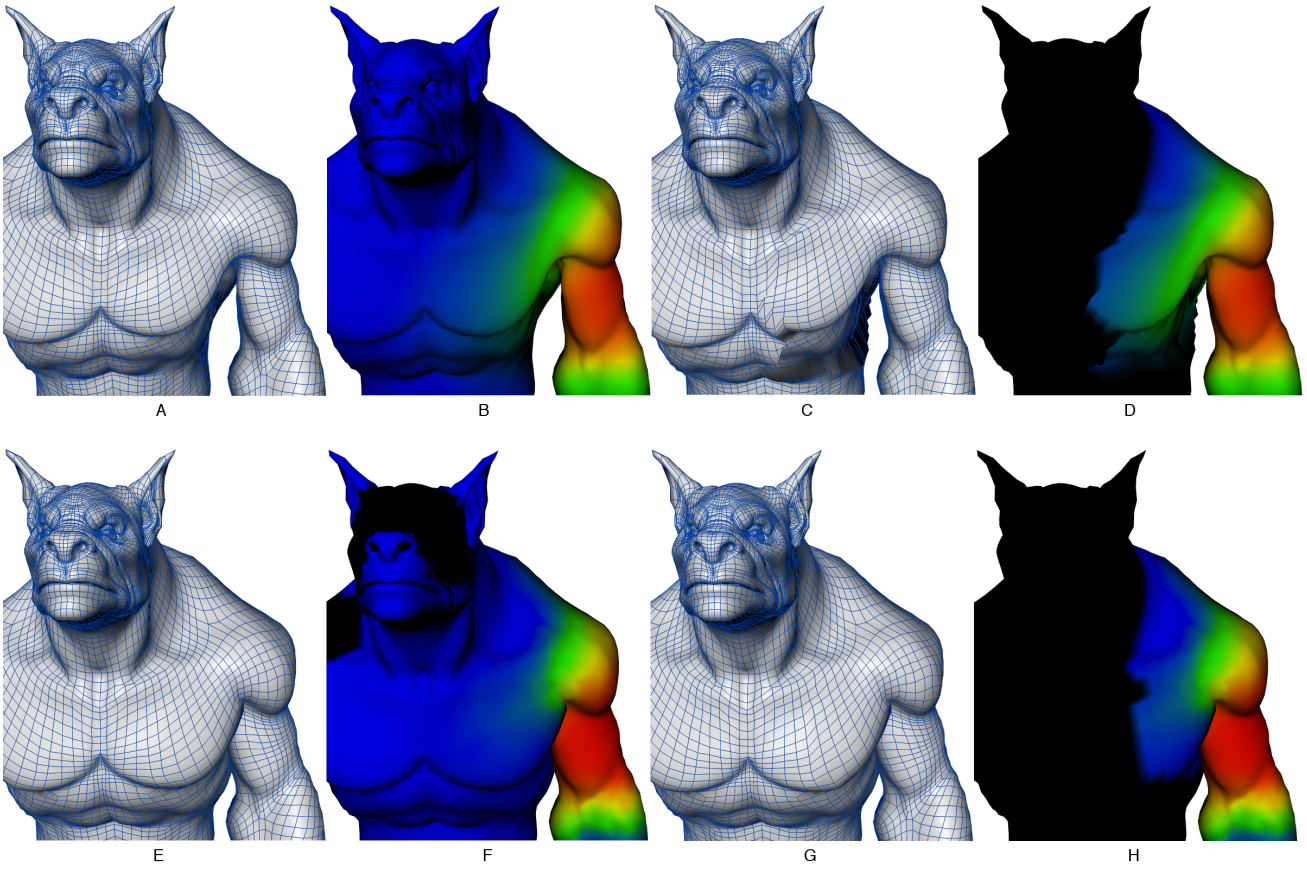

Fig. 11. Comparison of HD [2] (A-D) with GVB (E-H). On watertight meshes HD generates appealing results (A) but yields many small weights far from the joint of interest (B). When the number of influences is limited (D), as is often done to enable vectorization, artifacts may become visible (C) (e.g., see armpit region). GVB produces visually indistinguishable results to HD (E) with few small weights (F). Because we concentrate most influences on bones near the joints (H), limiting influences does not introduce problems (G).

have this problem. Run times for HD were comparable to GVB; although GVB is about twice as fast at resolutions of $r = 256^3$ it is about three times slower for $r = 512^3$.

Figure 12 shows bind results for BBW and GVB on Armadillo. Although resulting bind weights from both methods are qualitatively similar, BBW tends to produce smoother weights with more localized binds. For Armadillo this is especially noticeable around the hands and neck area (see Figure 12c). For high resolution meshes, using BBW is significantly slower than GVB. For Armadillo, GVB computes bind weights in under 30 seconds, BBW takes nearly 12 minutes. Processing time is dominated by the high-dimensional constrained QPs that arise from tetrahedralizing meshes with high surface detail.

TABLE 2
GVB, HD, and BBW bind times in seconds for Beast and Armadillo. Armadillo is shown in Figure 12.

Model	GVB		HD	BBW
	$r = 256^3$	$r = 512^3$		
Beast	2.5	14.6	5.2	-
Armadillo	6.7	29.6	7.1	721.0

9 Discussion

A key assumption of our method is that front-face surface normals are oriented correctly. If this is not the case, white pixels may not correspond to back faces. Hence, pixels may be misclassified. In practice, this assumption is not a significant problem as inverted surfaces are readily apparent when viewed with conventional lighting models.

Despite using a multi-resolution representation, quality and performance of GVB depends on the user-specified voxel resolution r. For simplicity, all presented examples use the same maximal resolution ($r = 256^3$). We found this straightforward approach worked well despite large differences in test model proportions, smoothness, and number of vertices. Alternatively, r could be automatically determined from input geometry features, such as the smallest distance between two vertices. We leave this as future work.

The impact of maximum voxelization resolution is

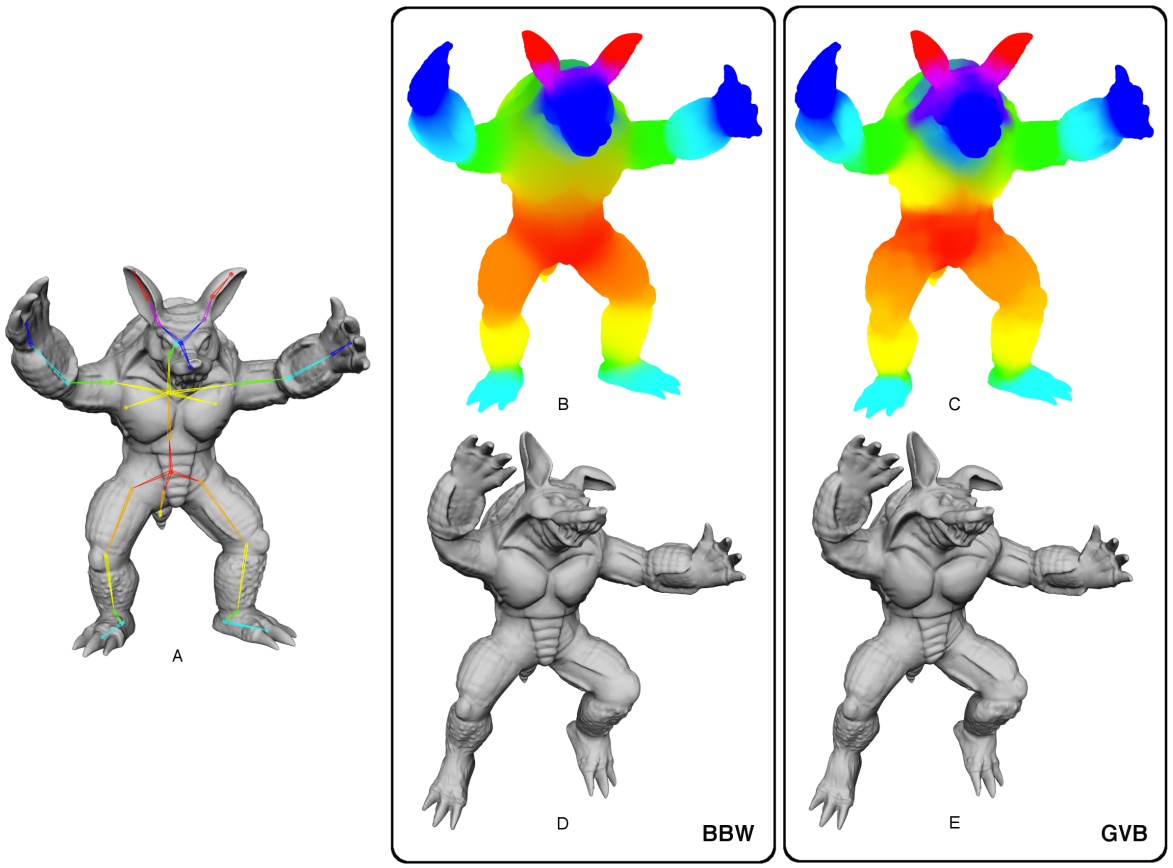

Fig. 12. Comparison of BBW [4] (B,D) with GVB (C,E) on Stanford Armadillo mesh/skeleton (A). GVB results use $\alpha = 0.2$. DQS is used to deform Armadillo geometry in selected pose (D, E). All influences are used without truncation.

particularly noticeable for A-stance rest pose meshes, such as Shockwave (Figure 13). As resolution decreases voxels surrounding the hands and legs become connected, altering the topology of the voxel domain and impacting distance computations. Other areas requiring fine detail, such as the fingers, can also be affected by the choice of voxelization resolution. A visual inspection of bind results is typically sufficient to catch these sorts of problems. If additional detail is needed, it is straightforward to increase voxelization resolution until desired results are obtained.

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

To encourage walks through interior voxels, algorithm 3 applies a distance penalty, $\epsilon_{penalty} > 1$, whenever boundary voxels are encountered. This helps eliminate binding artifacts that may arise when rest-pose geometry contains sharp folds, such as the arm pits. Biasing the distance algorithm towards interior voxels produces tighter, more natural looking, binds between joint and nearby geometry. Figure 14 shows sample results that benefit from the addition of $\epsilon_{penalty}$.

A major motivation for the development of our voxelization algorithm came from experimenting with two alternate weight assignment schemes. In the first method, we started by computing a discrete signed distance field in the domain of the axis-aligned bounding box surrounding the skin mesh [37]. 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 [19]. Although we often obtained good results using this approach, it introduced a costly level of indirection at pose time. Additionally, since there is no straightforward method for determining skinning weights for the original mesh, this made the method unusable for applications that use existing closed-form methods. To overcome this limitation, we

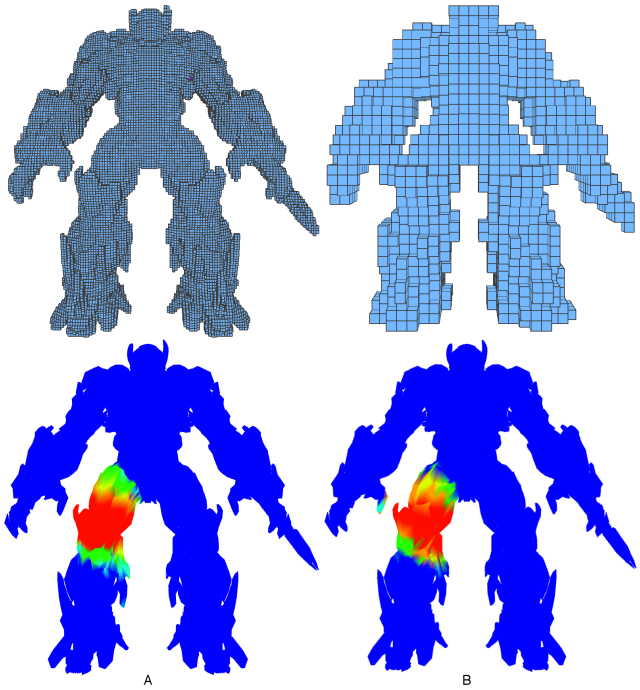

Fig. 13. Shockwave voxelized at $r = 128^3$ (A) and $r = 32^3$ (B). As resolution decreases topology may change, connecting disparate limbs and altering the bind.

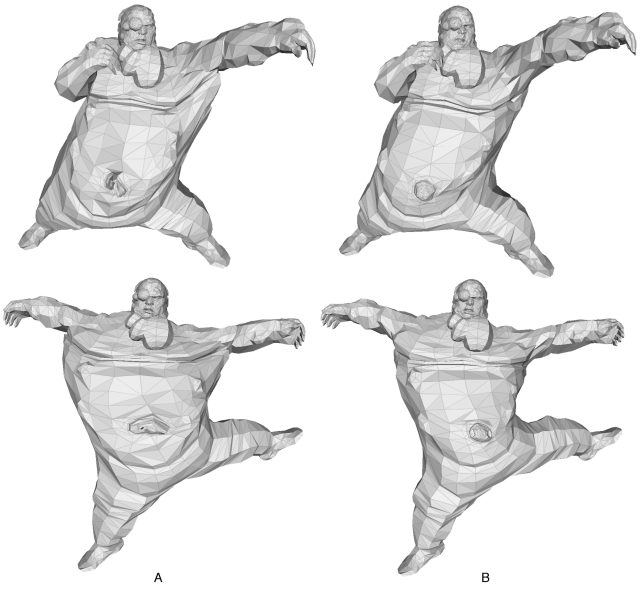

Fig. 14. Bind results for different boundary penalties ($\epsilon_{penalty} = 0$ (A) $\epsilon_{penalty} = 20$ (B)). Boundary penalties discourage walks through surface voxels, producing tighter binds, as shown here for Boomer.

explored a second option that voxelized the resulting (watertight) isosurface, using the method of Crane et al. [38]. This worked well in many cases, but the test used to tag discrete points in the signed distance field as interior or 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 bottleneck, often taking several minutes.

Several interesting applications arise from our voxelization based weighting algorithm. Since the process of calculating weights is extremely fast (Figure 7) 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 injuries or joining the undead). Additionally, since our weighting function has a simple parameterization depending only on α, it is possible for users 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 making α, for different mesh regions, dependent on joint orientation. Furthermore, as our weight parameterization comes from a distance field, it is straightforward to specify regions of the mesh that should not be deformed. This could be useful for characters with a mix of rigid and deformable equipment. In such cases bounding regions could be defined that override computed distances, assigning infinite distance to all bones except for user-specified drivers.

As with BBW, our method does not require a fully connected skeleton and can be applied to disjoint hierarchies. Additionally, though we focus on bone influences, our method can handle a variety of different handle types, including point and volumetric influences. For influences to be compatible, one need only define additional octree splitting elements (Section 4) such that we can define interior voxels from which distance computations would originate.

GVB uses a well-structured intermediate representation to isolate weight selection from mesh characteristics that can be problematic for other methods. For example, although BBW [4] can yield appealing results, formulating the needed QP requires a tetrahedral meshing, greatly limiting classes of valid input. Our proposed voxelization could be leveraged for BBW computation by using voxels instead of tetrahedra. One approach would be to use the uniform voxelization (Section 5.1). However, this would yield extremely large QPs that would be resource intensive and slow to solve; even at moderate resolutions tens of thousands of nodes can arise. A more promising future direction might be to use the sparse voxelization (Section 5.2). This would require updating the Laplacian used by BBW to account for voxels at multiple resolutions.

10 Conclusion

We present a novel system for determining influence weights for closed-form skinning methods. Our method is specifically designed to circumvent problems encountered with previous approaches requiring 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 GVB 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 [38], model simplification [30], or could be used to automatically extract skeletons, as part of a complete auto-rigging system [25]. Our method is also complementary to the recent subspace energy minimization method of Jacobson and colleagues [39] and to the implicit method of Vaillant et al. [33] which require initial weights to be specified.

Acknowledgments

We thank reviewers for valuable feedback and suggestions. Thanks to Alec Jacobson, Daniele Panozzo, and other members of the libIGL team for their Bounded Biharmonic Weights implementation. Special thanks to Martin Bisson for many insightful conversations and suggestions regarding use of winding numbers and potential algorithm optimizations.

References

[1] J. P. Lewis, M. Cordner, and N. Fong, "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*, 2000, pp. 165–172.

[2] I. Baran and J. Popović, "Automatic rigging and animation of 3d characters," *ACM Trans. on Graphics*, vol. 26, no. 3, 2007.

[3] C.-H. Chen, I.-C. Lin, M.-H. Tsai, and P.-H. Lu, "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*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 306–312.

[4] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Trans. on Graphics*, vol. 30, no. 4, 2011.

[5] C. Miller, O. Arikan, and D. Fussell, "Frankenrigs: Building character rigs from multiple sources," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 8, pp. 1060–1070, 2011.

[6] R. Wareham and J. Lasenby, "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*, Berlin, Heidelberg, 2008, pp. 63–71.

[7] S.-H. Lee, E. Sifakis, and D. Terzopoulos, "Comprehensive biomechanical modeling and simulation of the upper body," *ACM Trans. on Graphics*, vol. 28, no. 4, 2009.

[8] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović, "Interactive skeleton-driven dynamic deformations," *ACM Trans. on Graphics*, vol. 21, no. 3, 2002.

[9] S. Capell, M. Burkhart, B. Curless, T. Duchamp, and Z. Popović, "Physically based rigging for deformable characters," in *Proc. of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2005, pp. 301–310.

[10] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis, "Efficient elasticity for characer skinning with contact and collisions," *ACM Trans. on Graphics*, vol. 30, no. 4, 2011.

[11] J. Barbič and Y. Zhao, "Real-time large-deformation substructuring," *ACM Trans. on Graphics*, vol. 30, no. 4, 2011.

[12] T. Kim and D. L. James, "Physics-based character skinning using multi-domain subspace deformations," in *Proc. of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2011, pp. 63–72.

[13] P. G. Kry, D. L. James, and D. K. Pai, "Eigenskin: real time large deformation character skinning in hardware," in *Proc. of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2002, pp. 153–159.

[14] A. Mohr and M. Gleicher, "Building efficient, accurate character skins from examples," *ACM Trans. on Graphics*, vol. 22, no. 3, 2003.

[15] P.-P. J. Sloan, C. F. Rose, III, and M. F. Cohen, "Shape by example," in *Proc. of the 2001 symposium on Interactive 3D graphics*, 2001, pp. 135–143.

[16] R. Y. Wang, K. Pulli, and J. Popović, "Real-time enveloping with rotational regression," *ACM Trans. on Graphics*, vol. 26, no. 3, 2007.

[17] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki, "Harmonic coordinates for character articulation," *ACM Trans. on Graphics*, vol. 26, no. 3, 2007.

[18] T. Ju, Q.-Y. Zhou, M. van de Panne, D. Cohen-Or, and U. Neumann, "Reusable skinning templates using cage-based deformations," *ACM Trans. on Graphics*, vol. 27, no. 5, 2008.

[19] Y. Lipman, D. Levin, and D. Cohen-Or, "Green coordinates," *ACM Trans. on Graphics*, vol. 27, no. 3, 2008.

[20] A. Jacobson and O. Sorkine, "Stretchable and twistable bones for skeletal shape deformation," *ACM Trans. on Graphics*, vol. 30, no. 6, 2011.

[21] L. Kavan, S. Collins, J. Zara, and C. O'Sullivan, "Skinning with dual quaternions," in *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM Press, 2007, pp. 39–46.

[22] L. Kavan and O. Sorkine, "Elasticity-inspired deformers for character articulation," *ACM Trans. on Graphics*, vol. 31, no. 6, 2012.

[23] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann, "Joint-dependent local deformations for hand animation and object grasping," in *Proceedings on Graphics interface '88*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 1988, pp. 26–33.

[24] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Trans. on Graphics*, vol. 22, no. 3, 2003.

[25] L. Wade and R. E. Parent, "Automated generation of control skeletons for use in animation," *The Visual Computer*, vol. 18, no. 2, pp. 97–110, 2002.

[26] Z. Dong, W. Chen, H. Bao, H. Zhang, and Q. Peng, "Real-time voxelization for complex polygonal models," in *Proc. of the Computer Graphics and Applications, 12th Pacific Conference*. IEEE Computer Society, 2004, pp. 43–50.

[27] S. Fang, S. Fang, H. Chen, and H. Chen, "Hardware accelerated voxelization," *Computers and Graphics*, vol. 24, pp. 200–0, 2000.

[28] W. Li, Z. Fan, X. Wei, and A. Kaufman, "Flow simulation with complex boundaries," in *GPU Gems 2*, M. Pharr, Ed. Addison-Wesley, 2005, pp. 747–764.

[29] E. Eisemann and X. Décoret, "Single-pass gpu solid voxelization for real-time applications," in *Proc. of graphics interface 2008*, 2008, pp. 73–80.

[30] F. S. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Trans. on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.

[31] A. Jacobson, L. Kavan, , and O. Sorkine-Hornung, "Robust inside-outside segmentation using generalized winding numbers," *ACM Trans. on Graphics*, vol. 32, no. 4, 2013.

[32] M. Schwarz and H.-P. Seidel, "Fast parallel surface and solid voxelization on gpus," *ACM Trans. on Graphics*, vol. 29, no. 6, 2010.
[33] R. Vaillant, L. Barthe, G. Guennebaud, M.-P. Cani, D. Rohmer, B. Wyvill, O. Gourmel, and M. Paulin, "Implicit skinning: real-time skin deformation with contact modeling," *ACM Trans. on Graphics*, vol. 32, no. 4, 2013.
[34] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering 3rd Edition*. Natick, MA, USA: A. K. Peters, Ltd., 2008.
[35] A. Jacobson, D. Panozzo *et al.*, "libigl: A simple C++ geometry processing library," 2013, http://igl.ethz.ch/projects/libigl/.
[36] H. Si, "TetGen User Manual," 2006, http://tetgen.org.
[37] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, "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*, 2000, pp. 249–254.
[38] K. Crane, I. Llamas, and S. Tariq, "Real-time simulation and rendering of 3d fluids," in *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 2008, pp. 633–675.
[39] A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine, "Fast automatic skinning transformations," *ACM Trans. on Graphics*, vol. 31, no. 4, 2012.

Olivier Dionne received both the B.Eng. (2004) and M.A.Sc. (2009) degrees in Software Engineering at École Polytechnique in Montreal. His master's research focused on realistic simulation of interactive soft tissue deformations for an interventional scoliosis surgery simulator at Saint-Justine University Hospital Center in Montreal. He is Sr. Software Engineer on the Autodesk M&E Animation Team. Prior to joining Autodesk and completing his masters, he worked for various start-ups in the game industry developing high-performance real-time animation and rendering engines on embedded platforms. His research interests include computer graphics, imaging processing, and geometric/physics-based deformation modeling.

Martin de Lasa received the BESc degree in Mechanical Engineering and the BSc degree in Computer Science from the University of Western Ontario in 1998, the MESc degree in Electrical Engineering from McGill University in 2000, and the PhD degree in Computer Science from the University of Toronto in 2011. He is Technical Lead and Principal Engineer at Autodesk where he leads animation activities. Prior to his doctoral studies he worked at Boston Dynamics where he played a leading role in human simulation and robotics projects, including: Digital Biomechanics, BigDog, and LittleDog. His research interests span the areas of computer graphics and robotics, focusing on leveraging optimal control, optimization, and machine learning methods to build physically-based models of motion to ease animation/control of simulated characters and robotic systems. He is the recipient of the CAIAC doctoral dissertation award for the top Canadian dissertation in artificial intelligence (2011).