COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2011; **22**:27–36
Published online 8 November 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.382

RESEARCH ARTICLE

GPU rigid skinning based on a refined skeletonization method

Andreas A. Vasilakis and Ioannis Fudos*

Department of Computer Science, University of Ioannina, GR45110 Ioannina, Greece

ABSTRACT

In this paper, we present a skeletal rigid skinning approach. First, we describe a skeleton extraction technique that produces refined skeletons appropriate for animation from decomposed character models. Then, to avoid the artifacts generated in previous skinning approaches and the associated high training costs, we develop an efficient and robust rigid skinning technique that applies blending patches around joints. To achieve real time animation, we have adapted all steps of our rigid skinning algorithm so that they are performed efficiently on the GPU. Finally, we present an evaluation of our methods against four criteria: efficiency, quality, scope, and robustness. Copyright © 2010 John Wiley & Sons, Ltd.

KEYWORDS

skeleton extraction; rigid skinning; character animation; re-meshing; GPU implementation; real time

***Correspondence**

I. Fudos, Department of Computer Science, University of Ioannina, GR45110 Ioannina, Greece. E-mail: fudos@cs.uoi.gr

1. INTRODUCTION

Rapid realistic animation of articulated characters is a key issue in video games, crowd simulations, computer generated imagery films, and other applications of 3D computer graphics [1,2]. To achieve natural animation of articulated characters, we seek intuitive mesh deformations, often called in this context skinning techniques, that improve visual fidelity, computational efficiency and robustness of the character animation process. *Skeletal animation*, due to its versatility and ease of use, is one of the most popular animation techniques. Hence, in this paper we focus on skinning techniques for skeletal animation of articulated objects.

In skeletal animation, a kinematic model of the character is provided that consists of a highly detailed 3D surface representing the character's *skin*, and an underlying *skeleton* which is a hierarchical tree structure of *joints* connected with rigid links (*bones*). A skeleton usually has a much simpler structure than the original object that simplifies the skinning process by avoiding the tedious task of animating each vertex independently. The process of extracting a skeleton is called *skeletonization*.

There are several approaches to obtaining a representation appropriate for skeletal animation:

- An expert provides a skeleton and a mesh for the object. Then, we compute a decomposition based on the skeleton, so that the decomposition is compatible with the skeleton (e.g., Reference [3]). Subsequently, a skeleton refinement technique can optionally be used to adapt the skeleton for use with a rigid skinning method. This process is compatible with our method and can also be applied to data collected by motion capture systems.
- The user provides a mesh and a decomposition is either provided by an expert or is extracted automatically based on the morphology of the overall mesh (e.g., Reference [4]). Then we compute a skeleton that is compatible with this decomposition.
- Finally, if the approach being used does not require mesh segmentation, a skeleton is provided which is then embedded in the model and adapted accordingly (e.g., Reference [5]).

A skeleton acts as a special type of deformer transferring its motion to the skin by assigning to each skin vertex one (*rigid skinning*) or more (*smooth skinning*) bones as guides. In the former case, a skin vertex is fixed in the local coordinate system of the corresponding bone following whatever motion this bone is subjected to. This technique suffers from inherent flaws caused by elongations and inter-penetrations, especially in areas around joints. In the latter case, Linear Blend Skinning (LBS) is often used (also known as *skeleton subspace deformation* (SSD), *vertex blending* or *enveloping*), in which each skin vertex is assigned multiple influences in the form of blending weights that associate each such vertex with a number of bones. This scheme provides more detailed control over the results. The deformed vertex position is computed as a convex combination of the rigidly transformed influencing bones. The generated meshes exhibit volume loss as joints are rotated to extreme angles

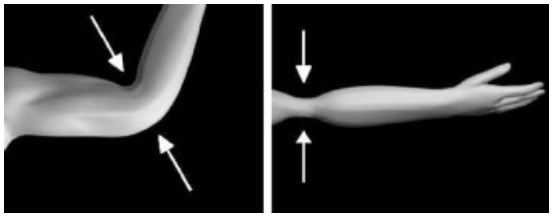


Figure 1. (left)"Collapsing elbow" and (right) "candy-wrapper" defects [6].

producing non-natural deformations, such as the collapsing joint and the candy wrapper defects (Figure 1). Despite of these shortcomings, variations of this method are widely used in real time computer graphics applications because they are simple and easy to implement on GPUs.

Vertex weight tuning tends to be a tedious and cumbersome task when applied on a single mesh. Existing 3D modeling applications expose the task to artists in the form of *weight painting* operations whereby the artists paint the weights for each bone with a virtual airbrush. Example-based approaches have been proposed to infer the character articulation from a training set of example poses. Some recent methods [7–9] automatically extract skinning weights from the deformed meshes over a set of reference poses or mesh animation frames. However, example-based fitting is a complicated iterative process due to the implicit dependencies of the deformed vertex positions. Researchers have identified as other potential drawbacks of these schemes the need for multiple example meshes and the complexity of avoiding over-fitting caused by under-determined weights [10].

For the purposes of seamless character animation, we aim at building a system that eliminates the skin attachment (weight fitting) process and overcomes LBS shortcomings by producing a high quality overall mesh. To this end, we present a method to eliminate self-intersection flaws by performing alternative intuitive deformations in four steps:

(i) Remove skin vertices of the overlapping components.
(ii) Add new vertices in this area to fill in the generated gap by approximately preserving local volume.
(iii) Construct a blending mesh that produces a smooth skin surface by using a robust triangulation method.
(iv) Compute and adjust surface normal vectors.

Figure 2 illustrates this process.

We have designed and evaluated our framework against four criteria: scope, quality, performance, and robustness for the proposed skeletonization and skinning algorithms. In a nutshell, our work makes the following technical contributions:

- We explore a skeletonization strategy that uses a dynamic programming algorithm to extract high-quality skeletal morphs using kernel centroids and principal axes of the character components.
- We introduce new approximate refinement methods to improve the produced skeleton morphs using local and global criteria.
- Our skeletonization method is independent of the character's component size, has no convexity requirements and is invariant under distortion and deformation of the

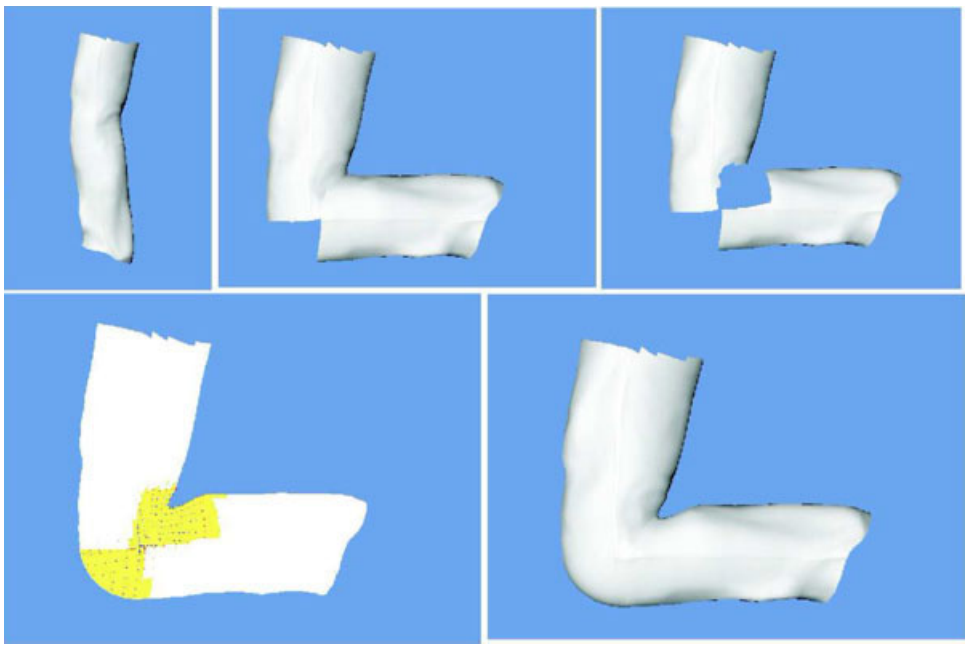


Figure 2. Robust rigid skinning. From top left to bottom right: the initial model; rotating the lower part; removing points; adding points; constructing a robust blending patch between the two components and adjusting normal vectors.

DOI: 10.1002/cav

input model which makes it applicable to several research areas such as character animation, shape recognition and collision detection.
- We present an advanced real time rigid skinning method that overcomes LBS artifacts by working on the overall mesh. We introduce novel blending patch construction techniques that preserve the initial volume and ensure mesh robustness.
- All the skinning and animation steps have been implemented on the GPU. This realization achieves considerable parallelization and hence real-time performance.

A preliminary short version of the main skeleton refinement process and the blending patch construction technique was presented in Reference [11].

The rest of this paper is organized as follows. Section 2 provides a survey of related work while Section 3 gives a brief overview of background definitions and tools. Sections 4 and 5 describe the skeleton extraction and refinement techniques developed for the purposes of character animation. The core rigid-skinning method along with its GPU implementation is introduced in Section 6. Section 7 provides an analytical and experimental evaluation of several quantitative and qualitative characteristics of our methods. Finally, Section 8 offers conclusions.

2. RELATED WORK

An abundance of research work in the literature tackles the skeleton extraction and skinning of 3D objects from different perspectives. We focus on recent developments most closely related to character animation.

2.1. Skeletonization

Skeletonization algorithms are roughly classified based on whether they process the boundary surface (*surface methods*) or the inner volume (*volumetric methods*).

Surface Methods: *Medial Axis Transform (MAT)* is a popular topological skeletal representation technique which consists of a set of curves which roughly run along the middle of the object surface. MAT-based representations suffer from perturbation and noise dependence, high computation cost and shape complexity in 3D. Researchers have proposed approximate MAT to overcome some of the above inefficiencies, however MAT-based skeletons are still not well fitted for character animation. *Reeb graphs* are a fundamental 1D data structure for representing the configuration of critical points and their relationships in an attempt to capture the intrinsic topological shape of the object [12,13]. However, a re-meshing technique [14] is usually required to generate accurate skeletons.

Volumetric Methods: Several methods generate skeletons by constructing discrete field functions by means of the object volume [15–17]. Such accurate methods are usually very time consuming and they cannot be applied to animation since the volumetric information needed is not usually part of the animation model.

Other Methods: Other methods have introduced alternative approaches to achieving more accurate and efficient skeletonization [8]. presents a method for extracting a hierarchical, rigid skeleton from a set of example poses [18]. proposes an iterative approach that simultaneously generates hierarchical shape decomposition and a corresponding set of multi-resolution skeletons. We have adapted the technique presented in Reference [18], that was initially targeted to reverse engineering, for the purposes of character animation.

2.2. Skinning

We focus on skeleton-driven skinning introduced by Reference [19]. LBS is the most widely used technique for real-time animation in spite of its limitations [20] due to its computational efficiency and straightforward GPU implementation [21]. LBS determines the new vertex position by linearly combining the results of the vertex transformed rigidly with each influence bone. An efficient LBS implementation using a vertex shader program is proposed in Reference [22].

Recent skinning algorithms can be classified based on whether they use a single input mesh (*Geometric methods*) or a training set of poses (*Example-based methods*). Our rigid skinning approach falls under the geometric class of algorithms introducing a novel versatile, robust, and efficient blending approach based on rational quadratic Bezier patches.

Geometric methods revert to nonlinear blending of rigid transformations since deformation is inherently spherical. Numerous proposed methods have replaced the linear blending domain with simple quaternions [23], log-matrix blending [24], spherical blending (SBS) [25] and dual quaternions (DQS) [26]. SBS and DQS have been implemented on the GPU achieving performance comparable to LBS [27,26].

Example-based methods remove artifacts by correcting LBS errors by paying the price of increased space and time requirements. Initial approaches combined LBS with interpolation examples using radial basis functions [6,28]. EigenSkin [29] used principal component analysis (PCA) for approximating the original deformation model based on GPU vertex programming. Multi-Weight Enveloping (MWE) [30] and Animation Space [31] are similar methods that introduce more weights per influence bone to provide more precise approximations and additional flexibility. Animation Space consistently performs better than LBS and MWE while MWE also suffers from overfitting [10]. In addition, Reference [32] introduced extra bones to capture richer deformations than the standard LBS model. Kavan *et al*. [33] proposed an algorithm which can be seen as a generalization of Reference [32], allowing to add an arbitrary number of virtual bones and blend them using a general transformation blending technique.

Reference [34] introduced a method for automatically constructing skinning approximations of arbitrary precomputed animations, such as those of cloth or elastic materials. Finally, Reference [35] employs efficient dual-quaternion transformation blending to achieve real time viewpoint adaptable animation.

Physics Inspired Methods: Physics inspired methods simulate realistic skin motion with high degree of realism (see e.g., Reference [36]) or add secondary deformations enriching skeleton driven animations [9], by paying the price of increasing considerably the computational complexity. Reference [37] addresses the problem of adding simplified dynamic effects to a skinned character without trying to correct the flaws of classic skinning. Dyrt [38] uses a heavy precomputation phase employing modal analysis of dynamic elastic methods using finite elements.

3. PRELIMINARIES

In this section, we introduce the notation and terminology to be used in the subsequent sections. Let $CH(C)$ be the convex hull of a component C that consists of n triangles: $T_k = \{(p_k, q_k, r_k), k = 0, 1, \ldots, n\}$. Then, the area and the centroid of the kth triangle are given respectively by $a_k = \frac{1}{2}|(q_k - p_k) \times (r_k - p_k)|$ and $m_k = \frac{1}{3}(p_k + q_k + r_k)$.

3.1. Joint Boundary

Joint boundaries are created when a component is split into sub-components during mesh segmentation. We define as *joint boundary* the common (joining) points of two adjacent components (Figure 3). Let $BC(C)$ be the set of the joint boundary centroids bc_i of a component C, $BC(C) = \{bc_1, \ldots, bc_n\}$, where n is the number of joint boundaries of C.

3.2. Kernel Centroid

The *kernel* of a component is the locus of its internal points from which all vertices are visible. Each facet of the component defines an interior *half-space* where half-space is either of two parts into which a plane partitions the 3D space. The kernel is the intersection of all such inner half-spaces. The simple centroid computation of a kernel may return non-intuitive results when applied to non-uniformly distributed points. Instead, we evaluate the kernel centroid as the weighted average of the triangle centroids weighted by their area (note that the kernel is a convex set). We denote the centroid of a convex hull by m_{CH} and is given by $m_{CH} = \frac{\sum a_k m_k}{\sum a_k}$. Efficient algorithms for computing the convex hull of a point set and the intersection of a set of half-spaces are publicly available. In this work, we have used the *qhull* implementation [39].

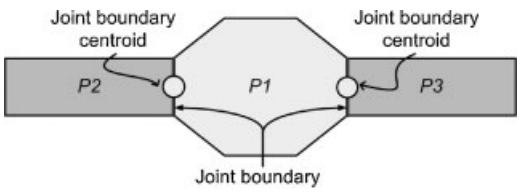


Figure 3. The generated joint boundary centroids between P1 and the adjacent components P2 and P3.

3.3. Principal Axis

The principal directions are three mutually perpendicular axes of a rigid body and may be used to find an approximation of the object's minimal bounding box axes. To derive principal directions, we use Principal Component Analysis in a covariance matrix computed across all the faces of the primitives instead of using only the vertices [40]. The principal axis is the eigenvector of the covariance matrix which corresponds to the largest eigenvalue. The other two principal directions correspond to the two eigenvalues with the second and third largest values. The covariance matrix for the aforementioned case is given by

$$C_{ij} = \frac{\sum_{k=0}^{n} a_k(9m_{ki}m_{kj} + T_{ki} \cdot T_{kj})}{12\sum a_k} - m_{CHi}m_{CHj} \quad (1)$$

4. SKELETON EXTRACTION

We have built on techniques introduced by Reference [18] that produce refined local skeletal morphs from the components of modular models. We have adapted these techniques for the purposes of our application and we have introduced appropriate refinements of the derived skeletons. The global skeleton of the character is then reconstructed by connecting the refined skeleton segments.

4.1. Previous Approaches to Skeleton Extraction

A common approach to building the skeleton of a component is to connect the joint boundary centroids. This technique is often called the *boundary method*. Note that for the root component, the skeleton is derived by connecting the component centroid with the joint boundary centroids. Although the method is very efficient it may generate unacceptable skeletons even for simple non-convex objects. Its major drawback is that it may produce skeletal segments that intersect arbitrarily the component boundary.

An improvement over the boundary method is to build local skeleton segments by connecting joint boundary centroids to the center of mass of the component [18]. Sometimes, however, the centroid may lie outside the component boundary producing erroneous skeletons. To overcome this shortcoming, we propose to use the centroid of its kernel for *star-shaped* components (polyhedra with non-empty

DOI: 10.1002/cav

kernel), where the kernel is computed as described Section 'Preliminaries'.

For non-star-shaped components the center of mass is used instead. This approach is efficient and addresses some of the flaws that occur in the boundary method. However, quite often it does not capture the component shape accurately.

In Reference [18], the authors proposed to overcome failures of previous methods by extracting a skeleton from a component by connecting the joint boundary centroids to a principal axis segment. To achieve this, the principal axis is segmented in line segments of equal length by defining a number of equidistant *link points* on the axis. This method maps the joint boundary centroids on the principal axis by minimizing the total interconnection length and the number of link points used. This is reduced to an optimization matching problem. The final skeleton of the component contains line segments that connect joint boundary centroids to link points and line segments that interconnect link points.

4.2. Optimized Skeleton Extraction

Our approach improves previous skeleton extraction techniques for use in character animation. To ensure that skeleton segments have the least possible intersection with the component border, we use as major axis an axis parallel to the principal axis that goes through the kernel centroid of the component. In addition, we select the principal axis segment of the convex hull that resides within the interior of the component instead of the one that lies within the convex hull. We denote this major axis segment as $\mathbf{pa}$.

Moreover, the method outlined in Reference [18] depends on a scalar that defines the minimum skeleton linkage length, making it not versatile. Since the cardinality of the link points varies from 1 to $|BC(C)|$, we propose to subdivide $\mathbf{pa}$ in a number of uneven segments by picking as link points the projections of the joint boundary centroids on $\mathbf{pa}$. Let $P_{pa}(bc_i)$ be the projection of a joint boundary centroid bc_i on $\mathbf{pa}$. If this projection lies outside $\mathbf{pa}$ then we use instead the end point of $\mathbf{pa}$ that lies closer to the projection. Then, we sort the joint boundary centroids according to their closest projection points by observing that two centroids are likely to be grouped when their projected points are close enough (see Figure 4).

Our matching algorithm uses a dynamic programming concept, based on novel score functions which aim at minimizing the total interconnection length and the number of link points used and maximizing the length of the utilized $\mathbf{pa}$ (Section 'Grouping Algorithm').

After grouping has been performed, we create the set of skeleton segments. Beyond the standard connections between the joint boundary centroids and the link points and line segments that interconnect link points, we use extra skeleton segments based on the mapping result (Section 'Connecting Algorithm'). Finally, we propose to add four more joints which represent the two ends of the other principal direction segments hence augmenting the topological information captured by the skeletal representation.

The principal axis algorithm is more expensive to compute than the previous approaches (see also corresponding section) due to the principal axis and kernel centroid computations but derives better quality skeletons by capturing the topology of the character shape more effectively.

4.2.1. Grouping Algorithm.

In this step, our goal is to group the sorted joint boundary centroids so as to minimize the total mapping length and the cardinality of the set of link points (score function F_{bc}), while at the same time maximize the length of the used principal axis (score function F_{gr}). First, each group contains only one joint boundary centroid.

The main concept behind the score function for grouping a joint boundary centroid set, $bc_{ij} = \langle bc_i, \ldots, bc_j \rangle$, is that it should be monotone on the distance among joint boundary centroids.

First, we compute their link point on the principal axis, named $L_{pa}(bc_{ij})$, as the mean of their projections. By doing so, the sum of their distances from the principal axis is minimized,

$$L_{pa}(bc_{ij}) = \frac{\sum_{k=i}^{j} P_{pa}(bc_k)}{|bc_{ij}|} \tag{2}$$

Let $d(\mathbf{p_1}, \mathbf{p_2})$ be the Euclidean distance between point $\mathbf{p_1}$ and point $\mathbf{p_2}$. Then $\forall\ bc_k \in bc_{ij}$, we measure their interconnection cost with respect to this grouping. We denote this as *normalized variation* of the joint boundary centroid and is the distance variation of the joint boundary centroid from the corresponding link point on the principal axis over the maximum such variation. Both quantities are computed with respect to the minimum distance (i.e., the distance of the joint boundary centroid from the principal axis).

$$V(bc_k) = \frac{d(bc_k, L_{pa}(bc_{ij})) - d_{\min}(bc_k)}{d_{\max}(bc_k) - d_{\min}(bc_k)}, \tag{3}$$

$$d_{\min}(bc_k) = d(bc_k, P_{pa}(bc_k)), \tag{4}$$

$$d_{\max}(bc_k) = \max_{1 \le l \le |BC|} \{d(bc_k, bc_l)\} \tag{5}$$

So, the total interconnection length cost among the joint boundary centroids and the principal axis is computed as the average of their normalized variations,

$$V(bc_{ij}) = \frac{\sum_{k=i}^{j} V(bc_k)}{|bc_{ij}|} \tag{6}$$

Moreover, we compute the ratio of the principal axis length that vanishes after merging these joint boundary

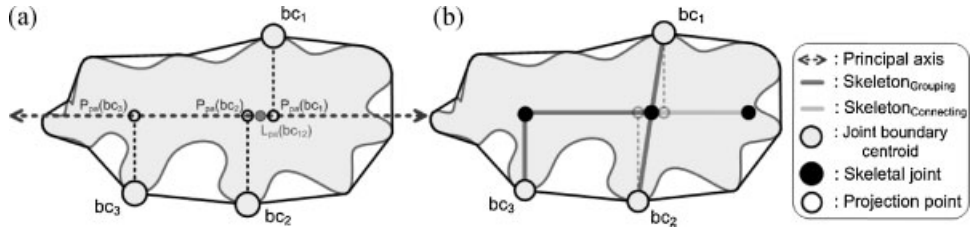


Figure 4. Skeleton extraction by the principal axis method.

centroids over the length of the maximal principal axis segment that can be constructed:

$$pa_{\text{lost}}(bc_{ij}) = \frac{d(P_{pa}(bc_i),\, P_{pa}(bc_j))}{d(P_{pa}(bc_1),\, P_{pa}(bc_{|BC|})} \tag{7}$$

Sometimes, the value of the maximum constructible principal axis length is very small as compared to the actual length of **pa**. In that particular case, the value of the lost skeleton score will be very large, which is undesirable since the best solution is to group joint boundary centroids to one link point. Hence, we set the maximum constructible principal axis length to $c|pa|$, where c is an experimentally determined constant in [0.1, 0.2].

Definition 1. The normalized merging score function F_{bc} for a joint boundary centroid set bc_{ij} is defined as the average of the normalized variation and the ratio of length of the principal axis that vanishes,

$$F_{bc}(bc_{ij}) = \frac{V(bc_{ij}) + pa_{\text{lost}}(bc_{ij})}{2} \tag{8}$$

The main idea of the score function for not merging two sequential groups $G_x = \langle bc_i^x, \ldots, bc_l^x \rangle$ and $G_y = \langle bc_{l+1}^y, \ldots, bc_j^y \rangle$, is that as the distance between these groups becomes very small the corresponding score function value increases.

The total mapping length cost between the joint boundary centroids of these groups and the principal axis is computed again as the average of their normalized variations which have been computed in previous steps (dynamic programming),

$$V(G_x, G_y) = V(bc_{il}) + V(bc_{l+1j}) \tag{9}$$

Moreover, we define a factor that expresses how these two groups participate quantitatively to the skeleton construction. This factor equals the normalized length of the principal axis which is generated between these groups which is the distance from the last projection point of the first group to the first projection point of the second group divided by the length of the maximum principal axis segment that can be constructed,

$$gR_{pa} = \frac{d(P_{pa}(bc_l^x),\, P_{pa}(bc_{l+1}^y))}{d(P_{pa}(bc_1),\, P_{pa}(bc_{|BC|})} \tag{10}$$

Definition 2. The normalized separating score function F_{gr} for two groups is defined as the average of the sums of their total distance cost plus the complement of the normalized utilized principal axis,

$$F_{gr}(G_x, G_y) = \frac{V(G_x, G_y) + (1 - gR_{pa})}{3} \tag{11}$$

Figure 5 illustrates a grouping example of a component with three joint boundary centroids. Moreover, it describes the sequence of grouping steps over time. Specifically, at

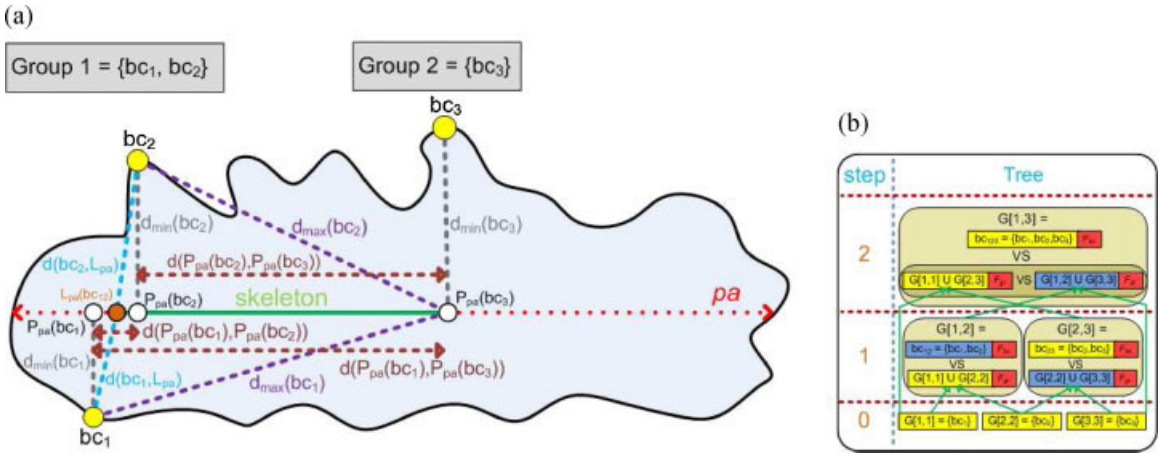


Figure 5. (a) An example of grouping with three joint boundary centroids and (b) the corresponding comparison-decision steps.

DOI: 10.1002/cav

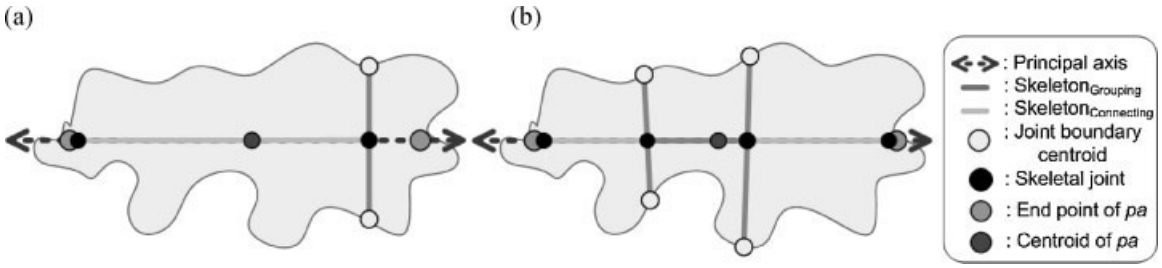


Figure 6. The refined skeletons obtained by the connecting algorithm by (a) case 1(ii) and (b) case 2(i) that both capture more information about the shape of the components.

the initialization step each group contains only one joint boundary centroid. At the next step, we compute the sub-grouping result that consists of two joint boundary centroids by comparing the values F_{bc} and F_{gr} of the joint boundary centroid sets bc_{12} and bc_{23} and by storing the ones that prevail. Winner groups are highlighted in light blue. Finally, we compute the final grouping result of the example component using the same strategy. Note that the solution for small sub-problems is combined to derive the solution for larger sub-problems.

By $G[i, j]$ we denote the optimal solution for grouping the bc_{ij} subset.

4.2.2. Connecting Algorithm.

The final skeleton of the component contains line segments that connect joint boundary centroids to link points and line segments that interconnect link points. The joint boundary centroid grouping often generates skeletons that do not capture important topological information. Therefore, the connection algorithm works according to the following cases:

(1) If all joint boundary centroids are grouped to one link point and this point is closer to
   (i) the centroid of $\mathbf{pa}$ rather than to one of the end points, then we connect it to both end points of $\mathbf{pa}$;
   (ii) the one of the end points of $\mathbf{pa}$ rather than to the centroid, then we connect it to the other end point of $\mathbf{pa}$.

(2) If all joint boundary centroids are connected to more than one link point and these are closer to
   (i) the centroid of $\mathbf{pa}$, then we connect the first and the last link points to both end points of $\mathbf{pa}$;
   (ii) one of the end points of $\mathbf{pa}$, then we connect the first or the last link point to the other end point of $\mathbf{pa}$'s.

Figure 6 shows the resulting skeletons of two components using this technique.

5. SKELETON REFINEMENT

In this section, we introduce a process that improves the alignment through small corrective adjustments. Our approach adjusts the principal directions using two individual processes; the first uses local characteristics while the second uses inherited hierarchical information, i.e., knowledge derived from the ancestors. Figure 7 illustrates the qualitative improvement of the oriented bounding boxes over the original principal axis algorithm result.

5.1. Local Refinement

To achieve nearly optimal and visually satisfactory orientation results, we approximately align principal directions through slight modifications using qualitative features. We select these features to be the skeleton segments extracted

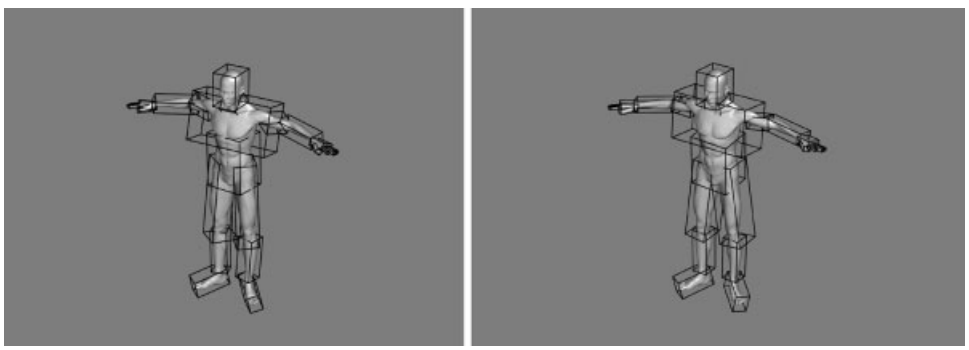


Figure 7. Oriented bounding boxes of the "Human" model components: (left) original configuration and (right) after alignment.

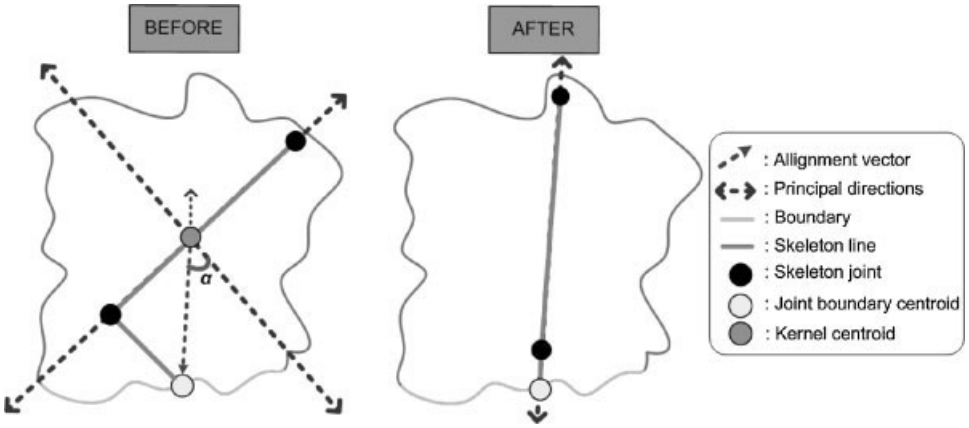


Figure 8. This example illustrates the result of a local refinement operation on a component with one joint boundary centroid: (left) the initial principal axes orientation and (right) the produced skeleton after local alignment.

by the centroid method. These are the vectors defined by connecting the kernel centroid to the joint boundary centroids. The proposed algorithm performs *weighted* vector alignment for each joint boundary centroid so that each adjustment will not undo the benefits acquired during previous improvements (Figure 8).

If the component has more than one joint boundary centroid, we align with regards to the closest principal direction by $angle = \alpha w$, where α is the angle between the given vector and its closest axis and weight $w = \frac{1}{|BC|+1}$ except from the joint boundary centroid which lies at the root. Considering that this has more importance to the overall refinement, we weight it by $2w$, such that $\sum_{i=1}^{|BC|} w_i = 1$. Closest is the principal direction that forms the minimal angle with the given vector. When the component has only one joint boundary centroid, we simply match its corresponding vector with its closest principal direction. If the closest principal direction is different from the principal axis then the former will be the selected as the principal axis. The upper bound for the rotation angle is a user determined parameter and in practice is set in the range $5 - 20^{\circ}$ interactively. Modifications that require rotations beyond this upper bound are rejected.

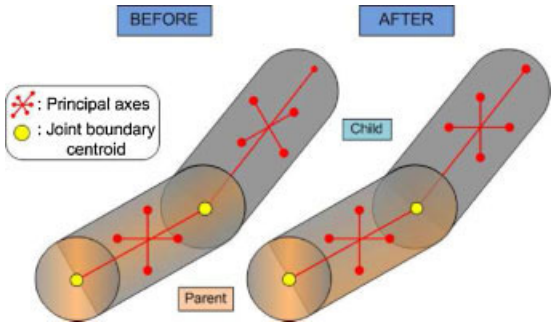


Figure 9. Comparison of the extracted skeletons (left) before and (right) after applying the hierarchical refinement algorithm.

5.2. Hierarchical Refinement

We extend the local refinement process by performing optimal fitting of the local principal directions of neighbor components to achieve skeletal uniformity.

The algorithm starts from the children of the root node of the skeleton tree performing the same process to their children and so on, until it reaches the tree leaves. An inherent drawback of this method is the dependence of the result on the selection of the root component. For each component, the *Hierarchical Refinement* process first aligns with regards to the parent principal direction which is closest to the child principal axis. Subsequently, from the rest of the child and parent principal directions we detect those that are closest and align them accordingly (Figure 9). Note that fitting two principal directions from different components, results in rotating the other principal directions too.

6. RIGID SKINNING WITH BLENDING PATCHES

With Rigid Skinning every motion is transferred to the character surface by assigning to each skin vertex one bone as a driver. The skin vertex is pinned to the local coordinate system of the corresponding joint, which is the one derived during the skeletonization process. By using homogeneous coordinates, a joint transformation is represented through a 4×4 matrix W. The vertex then repeats whatever motion the corresponding bone experiences, and its position in world coordinates is determined by forward kinematics, i.e it is transformed by the joint matrix W: $v' = Wv$. This will give the skin vertex position when moving rigidly with the bone. Similarly, the normal of each skin vertex is transformed by rotating it with the upper 3×3 rotation matrix R of W: $n' = Rn$. Although simple and efficient, rigid skinning often produces non-smooth skin deformations in areas around joints. This has led to the emergence of more specialized techniques that improve the undesirable rough

skinning results. We have developed a technique to address these major flaws. This technique is carried out in four steps:

(1) Remove skin vertices of overlapping components.
(2) Add new vertices in this area to fill the gap.
(3) Construct a blending mesh that produces a smooth skin surface by using a robust triangulation method around the new vertices area.
(4) Compute and adjust the normal vectors of the patch faces.

6.1. Removing Vertices

First, we detect which vertices from each of the collided components are located inside each other. A fast approximate test is to use the oriented bounding box (OBB) since accurate methods require a lot of computational resources. A point is inside an OBB if all six plane-point tests have the same sign. We further improve the testing process accuracy by making one extra plane-point test. Each point is tested against the plane which fits best to the joint boundary points of the components. We denote this as *joint boundary plane* (see Figure 10). The use of the joint boundary plane decreases significantly the number of removed points by 10–50% (see Table 1).

The algorithm starts from the joint boundary points to perform as few tests as possible since they have the highest probability to lie inside the other component's OBB and goes through their 1-ring of neighbors until it reaches points which lie in the outer side of the OBB. Two points are 1-ring neighbors if they belong to the same triangle.

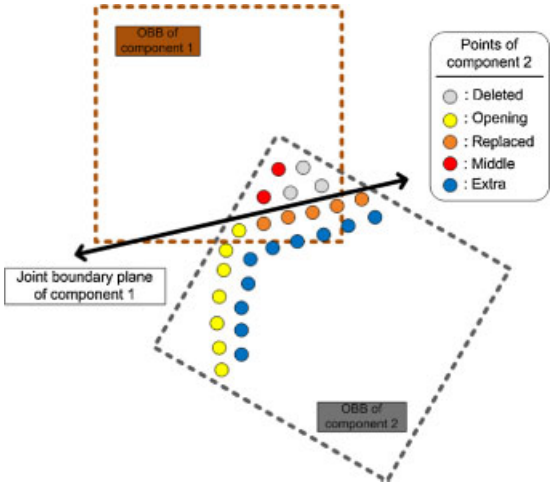


Figure 10. An example that illustrates the point classification for component 2 after removing points using the joint boundary plane.

The refined and aligned OBBs derived during the skeletonization process provide reduced overlapping between the moving adjacent components.

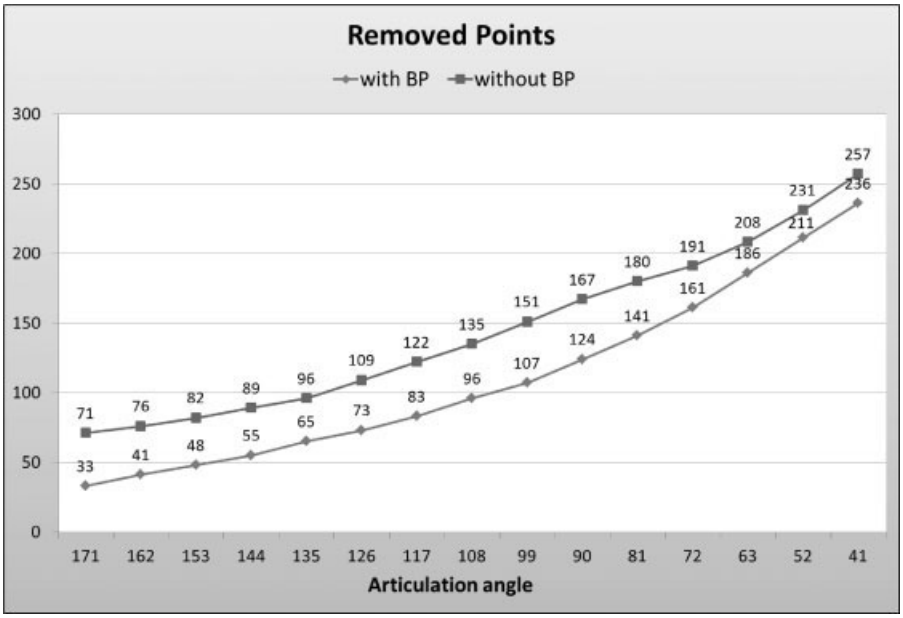
6.2. Adding Vertices

To facilitate the description of the blending mesh we define as:

Definition 3. *We define as*

- Opening *is the set of joint boundary points of the component that have not been removed (see Figure 10).*

Table 1. Reducing the number of removed points by introducing the joint boundary plane test.



- Middle *is the point set that we obtain if we subtract the* Opening *set from the joint boundary point set (see Figure 10).*
- Replaced *is the point subset of the component that corresponds to the new boundary points that appear after applying the* Removing Vertices *process. (see Figure 10).*
- In-Between *is the point set which consists of the average of the initial and final positions of the joint boundary points of the component that have been removed, which we denote by* B_i^s *and* B_i^f*, respectively (see Figure 11).*
- Extra *is the point set that consists of all 1-ring neighbors of the points that belong to the Opening* $\cup$ *Replaced set (see Figure 10).*

We present two blending techniques; the first one is called *front blending patch construction* and patches only the *Opening* set and the second is called *rear blending patch construction* and patches the *Replaced* and *In-Between* point sets.

6.2.1. Front Blending Patch Construction—FBPC.

The observation enabling this mechanism is that the rotation of a component around a joint results in movement of every point $O_j \in Opening$ on a circular arc centered on the joint point. The task is to find new points by interpolating from the point position ($O_j^s = q_s O_j q_s^{-1}$) before quaternion rotation q_s to the point position ($O_j^f = q_f O_j q_f^{-1}$) after quaternion rotation q_f (see Figure 11).

We have used fast linear interpolation of quaternions (QLERP) since the interpolation along the shortest segment hardly causes any observable skin defect [25]. QLERP is computed as

$$q(t; q_s, q_f) = \frac{q'}{\| q' \|}, q' = (1-t)q_s + tq_f \tag{12}$$

where the values assigned to t determine the interpolation step and depend on the proper distance which is enforced between consecutive constructed points. We compute this as the average of the median edge distances of the two components. To avoid square root operations (involved in quaternion normalization), we convert q' to a homogeneous rotation matrix Q' and then normalize subsequently by dividing with $q'q'$ to obtain Q. To translate the center of rotation from the joint position (J) to the origin, we use the following homogeneous matrix:

$$T = \begin{bmatrix} I & -\vec{J} \\ \vec{0^T} & 1 \end{bmatrix} \tag{13}$$

where I is a 3×3 identity matrix. Finally, new points are acquired using

$$v' = \mathrm{TQO}_j^s T^{-1} \tag{14}$$

6.2.2. Rear Blending Patch Construction—RBPC.

We have to fill in the gap in the area where we have removed points from both components. It is central to establish a correspondence between a pair of *Replaced points* from each component and then fill in the in between gap.

A grouping algorithm is used to create *triplets* from each $InB_i \in In\text{-}Between$ and one point from each of the child and parent *Replaced* sets (R_j^C, R_k^P selected points,

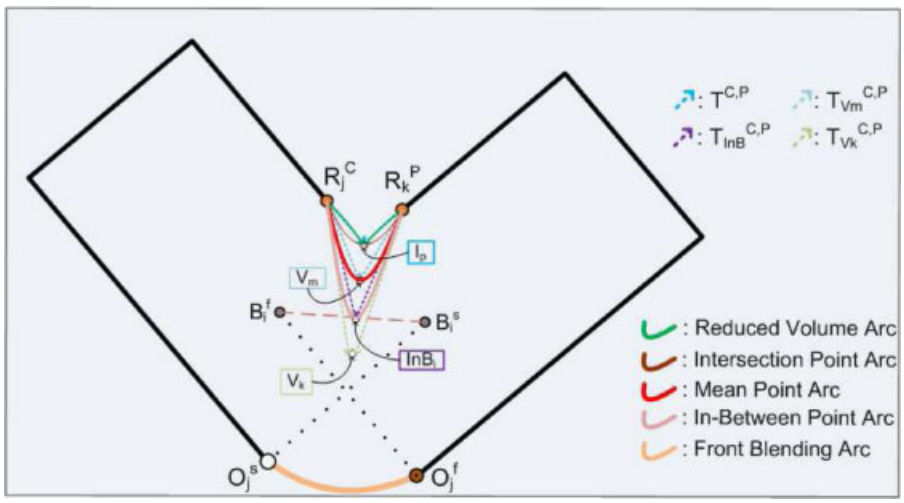


Figure 11. Blending patch construction processes.

DOI: 10.1002/cav

respectively). We seek triplets with the following property: the plane defined by the three points, and has the minimum dihedral angle with the plane defined by the joint position and the kernel centroids of the participating components (plane PL). To reduce the computational burden of brute-force search for finding the best fitting triplet ($\approx O(n^3)$, where $n = \max\{|InB|, |R^C|, |R^P|\}$), we reorder the search space by sorting the three point sets with respect to the distance from plane PL. Then, we search for the best triplet which contains InB_i by traversing the child and parent *Replaced* sets (R^C, R^P) from the y_i^C and y_i^P start positions and continue for a limited number of steps (lS^C and lS^P, respectively). Evidently, y_0^C and y_0^P values are initialized to zero. At the ith step of the iterative algorithm, y_i^C and y_i^P are evaluated as the next position of the selected candidates (j, k) of R^C, R^P sets of the previous step.

If we construct the patch with interpolated points derived by a circle equation, artifacts will arise due to the G^1 discontinuities at the end points of the two components. To avoid these, we construct blending arcs given the triplet vertices as control points using a Rational Bezier representation: Let P^C and P^P be the two end points of the conic arc, let $\vec{v^C} = (v_x^C, v_y^C)$ and $\vec{v^P} = (v_x^P, v_y^P)$ be the tangent vectors that we wish to enforce at the end points and let P^I be the third in between point we wish to interpolate. Furthermore, let $U = (v_x^C M_2 - v_x^P M_1, v_y^C M_2 - u_y^P M_1)$, where $M_1 = P_x^C v_y^C - P_y^C v_x^C$ and $M_2 = P_y^P v_y^P - P_x^P v_x^P$.

Then the rational quadratic Bezier curve segment has the form [41]:

$$c(t) = \frac{(1-t)^2 P^C + 2t(1-t)W + t^2 P^P}{(1-t)^2 + 2wt(1-t) + t^2} \qquad (15)$$

where

$$W = \frac{\text{Area}(P^C, P^I, P^P)U}{R}, \; w = \frac{\text{Area}(P^C, P^I, P^P)(\vec{v^C} \times \vec{v^P})}{R}$$

$$R = \sqrt{d(P^P, \vec{v^C}, P^C)d(P^I, \vec{v^C}, P^C)d(P^C, \vec{v^P}, P^P)d(P^I, \vec{v^P}, P^P)}$$

$$d(S, \vec{r}, R) = (S_y - R_y)r_x - (S_x - R_x)r_y$$

Area(C, P, D) : The signed area of the triangle $\widehat{CPD}$

To prevent the generated meshes from exhibiting volume loss and from decreasing mesh smoothness as joints rotate to extreme angles we are using the alternative InB point sets and tangent vectors. To illustrate this we use the following notation:

- I_p is the intersection of T^C and T^P vectors.
- V_m is the average of InB_i and I_p points.
- V_k is the terminal point of the negative vector going from InB_i to V_m.
- T_{InB}^C, T_{InB}^P are the vectors going from R_j^C, R_k^P to InB_i, respectively.
- $T_{V_m}^C$, $T_{V_m}^P$ are the vectors going from R_j^C, R_k^P to V_m, respectively.

To this end, we present and evaluate four alternatives for replacing the InB_i points and the associated tangent vectors (see Figures 11 and 12). These four options evaluated in Section 'Evaluation Results' in terms of quality and robustness.

(1) V_m with T_{InB}^C, T_{InB}^P, and the resulting gap filling process is called *Mean Point Interpolation.*
(2) I_p with T_{InB}^C, T_{InB}^P, and the resulting process is called *Intersection Point Interpolation.*

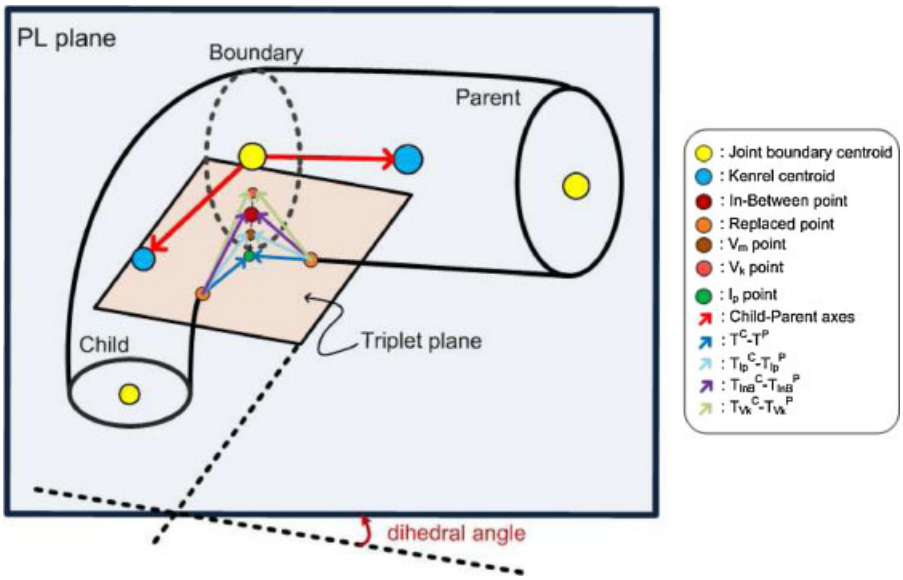


Figure 12. Rear blending patch construction process alternatives and notation.

(3) I_p with $T^C_{V_m}$, $T^P_{V_m}$, and the resulting process is called *Reduced Volume Intersection Point Interpolation.*
(4) *In B* with $T^C_{V_k}$, $T^P_{V_k}$, and the resulting process is called *In-Between Point Interpolation.*

Finally, we ensure that the constructed points are uniformly sampled by adjusting the interpolation step. In the special case of *twisting*, adding vertices process is not necessary since the articulation remains constant, so we omit this step.

6.3. Blending Mesh

The points derived through the two blending processes constitute the blending 3D point cloud which will be triangulated to form the blending patch between the two components. To ensure the robustness of the whole mesh, we expand the blending point cloud to include the 1-ring neighbors of the component's *Replaced* points (part of the *Extra* point set). In the CPU version of this work, we have used the *Tight Cocone* [42] algorithm to build water-tight surfaces around the newly inserted points. Since we need only two parts of the generated spheroid to patch the holes, we must eliminate a number of unnecessary faces to derive optimal lighting results. To this end, we adjust the normal vectors and then we remove the faces that belong exclusively to either the child or the parent component and whose normal vector is not nearly parallel with all normal vectors of its defining points. The evaluation of the patch point normal vector is achieved by averaging the normals of the output surface faces that contain this vertex. We adjust the normal values of the *Opening* and *Replaced* points by averaging the normals of the newly built surface faces and the normals of the original component faces that share this vertex. In this process, we take care as to not include component faces that have been removed. Finally, the normal vectors of extra inserted points remain unaffected since the above adjustment is not necessary (Figure 13). Finally, the patch texture coordinates can easily be approximated. Specifically, for the patch vertices constructed from the *FBPC* process, we may similarly interpolate the UV coordinates of the component points. On the other hand, texture information may be missed due to vertex clusters being removed. In this case, texture coordinates for a point constructed during *RBPC* can be approximated by interpolating between the *Removed* point from each component which has the minimum distance from it.

6.4. Porting the Rigid Skinning Algorithm on the GPU

The performance of the GPU is potentially much faster than the CPU to handle streaming input data working on large amounts of data in parallel in conjunction with the reduced data transfer advantage of the graphics hardware. The proposed rigid skinning algorithm is based on per-vertex computations which is independent for each vertex and so can

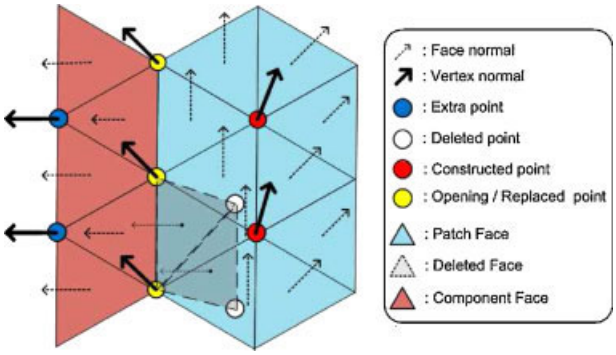


Figure 13. The normal vector estimation.

be parallelized on a GPU architecture using five rendering stages. By doing so, we achieve real time performance, while maintaining the quality of our rigid skinning result.

We distinguish between two types of data, varying and uniform data. Varying data is the streaming input data, while uniform data is data independent from the input stream that do not change between stream elements. Further, we have exploited an advanced usage of the *Transform Feedback* extension [43], which stores selected vertex attributes for each primitive and writes into buffer objects, thus feeding the data through one shader program to another. Furthermore, according to recent benchmarks, the use of the transform feedback is proven to be highly efficient even for simple transformations on large data sets as compared to standard GPU data communication schemes [44].

Geometry information from the rest pose is stored into *vertex buffer objects* (VBO) to reduce data transfer between CPU and GPU. Figure 14 illustrates the GPU work flow for our implementation. The vertex, geometry and fragment shader programs are written in GLSL. The framework was developed using OpenGL 2.1 as graphics library under C++. Unfortunately for porting these complicated graph-based operation on the GPU we have used extensions that are currently supported only on NVIDIA GeForce 8800 and later series.

6.4.1. Removing Vertices.

Vertices VBOs of the overlapping components along with an array containing the joint boundary point indices and the corresponding OBBs are transferred to the *Remove Points* vertex shader program. If a point is inside the OBB then we set its w position coordinate to zero, so that the *Render Model* geometry shader program will not emit it (*Removed Vertices* VBO). Moreover, besides the OBB collision detection, this shader will also characterize the exported point as *Middle*, *Opening*, or neither of them, by checking whether it belongs to the joint boundary point array and will subsequently store this information in the *Boundary Vertices* VBO. Finally, we construct a *texture buffer object* (TBO) to maintain a large global data collection of vertex attributes that will be accessed afterwards by the *Replaced* and *Extra*

DOI: 10.1002/cav

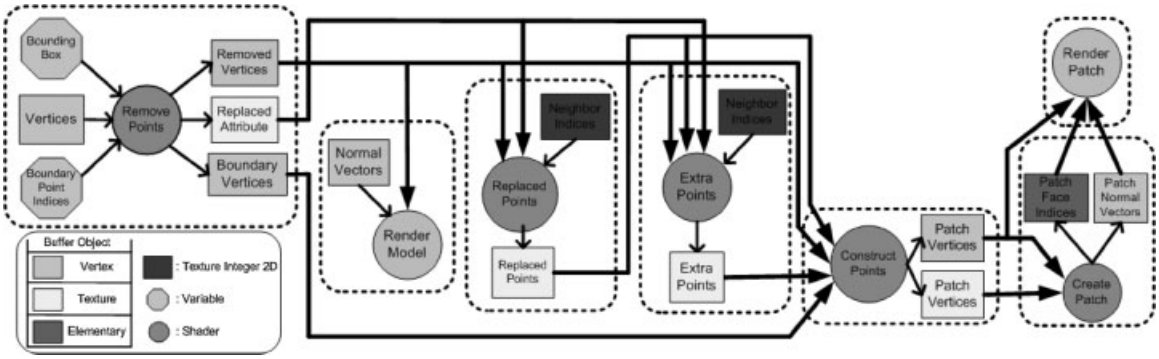


Figure 14. The GPU implementation work flow.

shader programs. This TBO is called *Replaced attribute* and stores for each point a 2D vector whose x and y coordinates determine whether this point is removed and whether it is an *Opening* point, respectively.

The *Replaced Points* shader programs are used to obtain a TBO which contains the *Replaced* vertices. The generated output will be used to extract the *Extra* point set and to assist with the construction process. A point is a *Replaced* point if it is not removed, is not an *Opening* point and one of its neighbors is removed. Similarly, the *Extra Points* shader programs (vertex and geometry) are responsible to export a TBO which contains the *Extra* vertices which will expand the blending point cloud. An *Extra* point is returned if is not removed, all of its neighbors are not removed and one of them is either an *Opening* or a *Replaced* point.

All the computation process is performed using vertex shader programming. Only *Replaced* and *Extra* points are emitted by the corresponding geometry shader. For both *Replaced* and *Extra* vertex shader programs, we have used "unnormalized" integer texture formats [43] looking up the neighbor indices for each point.

6.4.2. Adding Vertices.

At first, a vertex shader program determines whether the testing vertex is *Opening* or *Middle* point using the *Boundary Vertices* VBO. If it belongs to the *Opening* point set then we generate new points using the equations presented in detail in Section 'Front Blending Patch Construction'. On the other hand, if it belongs to the *Middle* point set, we first compute its corresponding *In-Between* point and then the grouping algorithm is carried out to create the best *triplet* using the overlapping component information stored in the *Replaced Points* TBOs. We use a brute-force search algorithm to compute the best fitting triplet without sorting the three point sets, since the fitting triplet process employed for the CPU implementation cannot be parallelized due to its sequential nature. After this step, a geometry shader is used to produce points interpolating rational Bezier arcs given the triplet vertices as control points (Section 'Rear Blending Patch Construction'). Finally, we copy the *Extra Points* TBO to the generated *Patch Vertices* VBO for the purposes of expanding the blending point cloud to achieve a more robust overall mesh.

6.4.3. Triangulation.

To triangulate the newly constructed unorganized point cloud we adapt a GPU interpolating reconstruction algorithm based on local Delaunay triangulation [45]. Triangulation is undertaken locally around each point, so that points do not depend on each other. Hence, it can be performed in parallel. We have made the following modifications to Reference [45] to best suit the purposes of our application and to allow faster execution and utilization of current hardware:

- We do not divide the point cloud into smaller sets or passes which are independently processed in the GPU, since the graphics memory of the new GPUs can easily accommodate all data.
- We have implemented the k-nearest neighbor algorithm in the *Create Patch* vertex shader program, thus maximizing parallelism. Normal estimation, rotational projection, angle computation and radial sorting are also performed on the same shader.
- We have used the simple bubble sort for finding k-NN neighbors and radial sorting of the generated neighbors, due to the small number of neighbors per point.
- The geometry shader program computes the valid Delaunay neighbors and outputs zero or more point primitives per-vertex. Triangle indices are stored into emitted point position vectors. Since the generated primitive stream is in the same order as given in the input, we can recompute the normal vectors for each vertex by averaging the normals of neighboring constructed facets.

7. EVALUATION RESULTS

The input to our algorithm is a segmented polyhedron model and its associated hierarchy information. Then, the character components are set to motion using forward

Table 2. Number of components, vertices, and triangles of the models used in the experimental evaluation.

Model	Components	Vertices	Triangles
Cow	16	3825	7646
Horse	17	8964	17 924
Dilo	37	26 214	48 924
Human	31	74 014	140 544

kinematics guided by a key-framing animation controller by altering joint angles of the embedded skeleton. In the following experiments we have decomposed characters into components using the *Blender* [46] software and animate them using manually supplied motion data. Table 2 summarizes the characteristics of the solid models used in the experiments.

The experiments were performed on a MS Windows XP Professional 64-bit Platform with a 64 bit 2.6 GHz Intel Core Duo with 3 GB of main memory and a nVidia GeForce 8800 Ultra graphics card. We evaluate our proposed techniques with respect to four criteria: performance, quality, versatility, and robustness.

7.1. Performance Evaluation

Table 3 gives computation times for extracting skeletons from the above models. The largest time corresponds to the principal directions and kernel centroid computations which are performed in $\sum_{i=0}^{k} O(n_i \log n_i)$ and $\sum_{i=0}^{k} O(n_i \log n_i) + O(r_i \log r_i)$ time respectively, for an articulated character model with k components where the ith component has n_i vertices and r_i kernel points. The refinement times are negligible as compared to the overall performance. In Table 3, we observe that the average overall skeleton extraction complexity appears to be almost linear on the number of triangles.

In general, our method generates refined skeletons in less than half a minute for dense models (Figures 15 and 16). Thus, if our method is used in conjunction with a fast decomposition method, it may establish a very efficient overall process. In Table 4 we present a comparison of our implementation with two recent automatic weight fitting techniques using one [5] or more example poses [8] for the whole preprocessing step that consists of segmentation, skeletonization and skin attachment. The model that was used in this experiment consists of 74 k vertices (the human model). For segmentation we have used the approach presented in Reference [4]. For skeleton driven decomposition the results are comparable.

Table 3. Time performance of skeletonization steps measured in seconds.

Model	Kernel centroid	Principal axis	Total
Cow	1.982	0.868	2.995
Horse	2.086	0.924	3.178
Dilo	4.701	2.022	7.150
Human	10.244	4.649	14.893

We have compared the performance of the GPU realization over our optimal sequential CPU implementation [11] for rigid skinning animation of a human knee (see Figure 2) from the initial pose to extreme angles (articulation angle going from 170 to 40° according to Reference [47]). where the parent and child components consist of 891 and 1000 vertices, respectively. Table 5 illustrates the computation times for both implementations which shows that using the GPU we obtain an average speed-up around 10x compared to the CPU implementation.

The patch construction process can be precomputed and used in the animation in a way much similar to the weight adjustment by examples step. This is performed by storing all the constructed blending patches as triangulated meshes including vertices, normals and facets for the entire animation. More specifically, for each elementary joint movement step we precompute the blending patch. If the number of steps increases we compute the interpolated blending patch from the two closest precomputed patches. Then, using the precomputed blending patches improved significantly th animation/rendering runtime of our approach that becomes up to 8% better for large animation configurations with multiple joints as compared to LBS and up to 20% better as compared to dual quaternion blending. These results are summarized by Table 6 using the same rendering algorithm.

7.2. Quality Evaluation

Figures 7 and 15 illustrate the qualitative superiority of our refined skeletons and aligned OBBs over the original principal axis algorithm. Figure 16 illustrates the similarity between the default skeletons generated by *Poser* [48] and the skeletons extracted by our method. Note that the joints in the refined skeletons are highlighted in red.

To measure the *smoothness* of the deformed meshes derived from our skinning technique, we define:

Definition 4. *Let v be a vertex with an associated set V^a of k adjacent vertices $V^a = \{v_1^a, \ldots, v_k^a\}$. Then we define the approximate measure Curv for the curvature of*

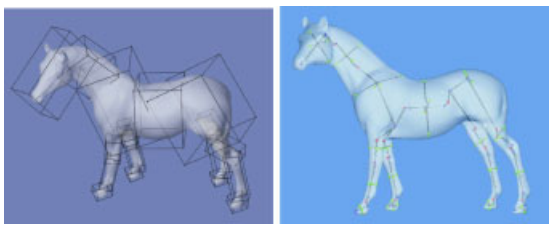


Figure 15. Refined oriented bounding boxes and skeletons of the components of the horse model.

DOI: 10.1002/cav

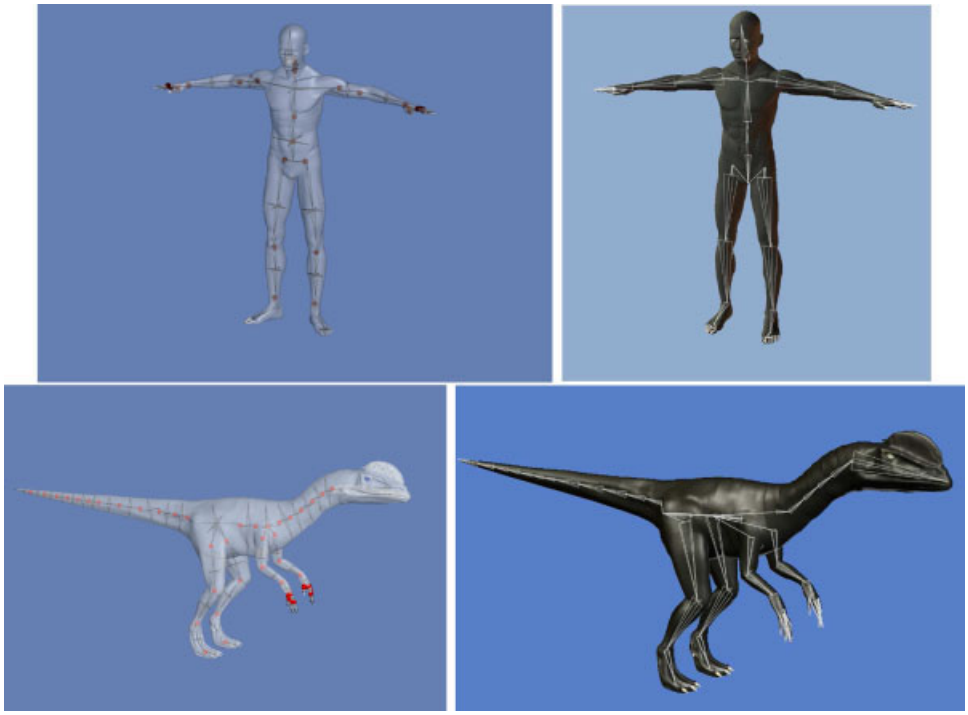


Figure 16. Comparison based on human and dino skeletons (left) derived by applying our refinement process and (right) built manually by the user/artist.

Table 4. Preprocessing time in seconds as compared to References [5] and [8].

Method	Segmentation	Skeletonization	Skin attachment
Refined principal axis	2.1	14.9	NA
Automatic character rigging[5]	NA	80	5
Example-based rigging[8]	30	0.02	24

Table 5. Comparison of GPU vs CPU implementation.

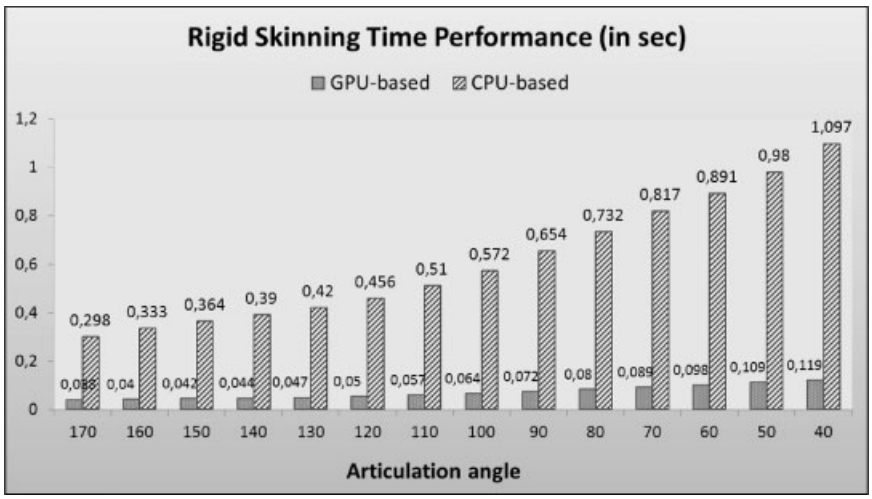


Table 6. Average FPS for several different animations of our method as compared to LBS and Dual Quaternion Blending (all on the GPU).

Models	Moving limb pairs	Realtime patch construction	Precomputed patches	LBS	DQS [26]
Cow	6	25	430	425	422
Horse	6	21	385	370	348
Dilo	6	16	270	253	225
Human	4	10	145	135	119

v as the maximum of the angles between the normal vector of v ($N(v)$) with the normal vectors of the vertices of V^a ($N(v_i^a)$),

$$Curv(v) = \max_{1 \le i \le k} \{\arccos(N(v) \cdot N(v_i^a))\} \quad (16)$$

Then, the total curvature of a bone component can be defined as the median curvature over all vertices connected to this bone. Lower curvature means smoother meshes. More detailed segment representations yield lower curvature values.

A graph of the curvature variation of the components involved in the human knee joint movement is illustrated in Table 7. From this figure we deduce that the total deformed skin smoothness of the participating components exhibits negligible variation from the smoothness of the reference posture during large articulations. Moreover, curvature results of the patch component are increased since the change in the curvature of the blending patch increases as the joint is imposed to large angle movement.

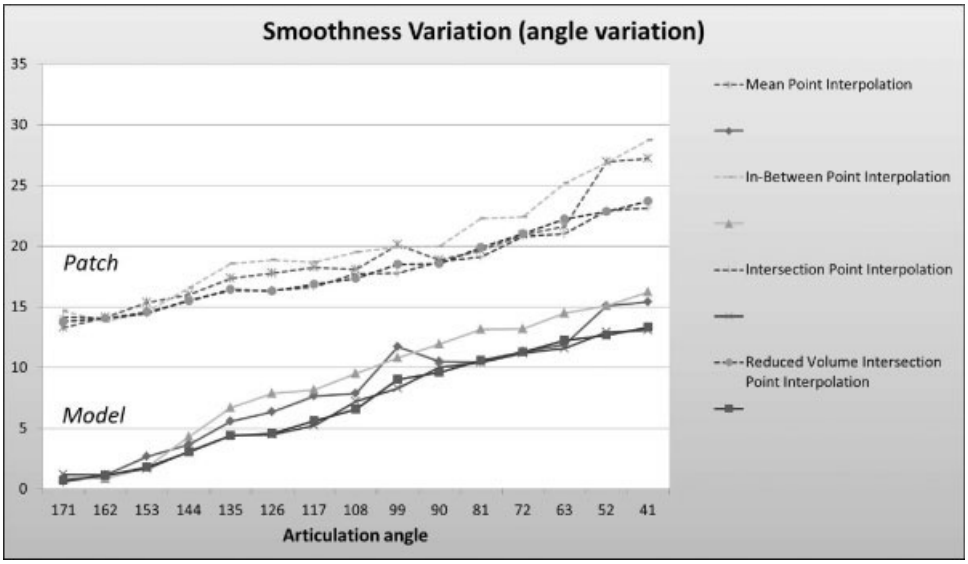
In addition, despite the fact that exact volume preservation of the surrounding skinned mesh was not among our primary goals, our method exhibits a very good behavior in terms of preserving the original volume. Table 8 illustrates the volume variation percentage computed as: $100 \cdot \frac{|vol_t - vol_r|}{vol_r}$, where vol_r is the volume of the two components at the reference posture and vol_t is the volume of the two patched components at the t animation frame. The deformed poses retain the same volume as the reference pose within a small deviation below 1% for usual angles and up to 4.5% (below 3% for the best method) for extreme angles.

Overall, the *Reduced Volume Intersection Point Interpolation* exhibits very good results in extreme angles and overall good results in terms of both volume preservation and curvature variation. The *In-Between Point Interpolation* exhibits the best volume preservation for small angles, while the *Mean Point Interpolation* exhibits better smoothness for smaller angles.

7.3. Versatility and Robustness Evaluation

We create animations by re-targeting hand-made BVH motion sequences [49] to the skeletons extracted from the original meshes. This format describes each motion frame as a

Table 7. Skin curvature variation (% percentage with regards to the reference posture) during articulation: (top) for the blending patch and (bottom) for the two involved components.



DOI: 10.1002/cav

Figure 17. Avoidance of the typical LBS "candy-wrapper" artifact. (Left) is a reference pose, (middle) upper arm is rotated 90° about its axis, and on (right) upper arm is rotated 180° about its axis.

sequence of the Euler angles of the skeleton joints. The sequence of parameters can then be applied to each bone component transformation matrix. We adjust the local matrices in the beginning to reflect the initial pose of the motion capture data, and then refresh the angle deformation to produce the different animation frames.

Figures 17 and 18 show animated poses using our rigid skinning technique. We have applied our technique to all types of joints of several models decomposed manually and automatically in various ways. We have not encountered any incompatible types of joints or solid models. This fact experimentally establishes the versatility of our approach.

We also provide snapshots depicting a closer view of a human knee mesh to demonstrate the robustness of the skinning process (Figure 2). The visual results confirm that our method is indeed free of all the unnatural rigid skinning flaws modeling realistic skin motion without distortion artifacts. As compared to the method presented in Reference [26], our method does not suffer from the flipping artifact that occurs when we rotate in the area of $179 - 181^{\circ}$.

A geometry shader is limited in the number of vertices it may emit per invocation. The maximum number of vertices

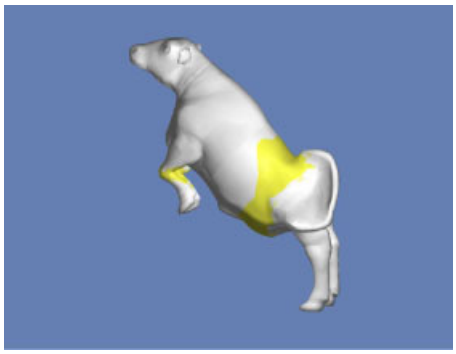
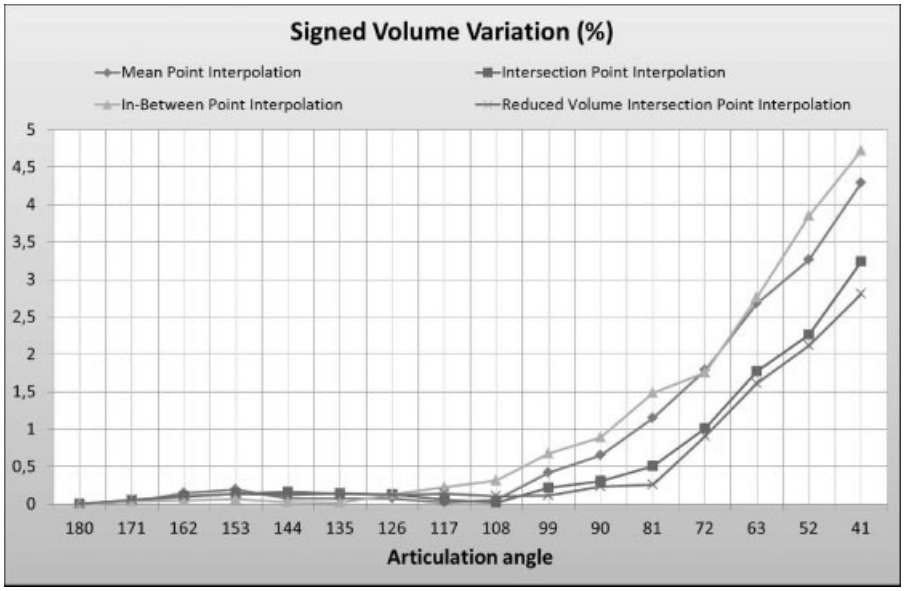


Figure 18. Multiple part animation of the "Cow" model. Patched parts are highlighted in yellow.

a geometry shader can possibly emit needs to be set as a parameter of the object that contains the geometry shader. The maximum value for any GPU can be estimated by dividing the value of *GEOMETRY_VERTICES_OUT_EXT* (set to 4096 bytes for the latest GPUs) by the vertex representation size. Since we have used a vertex size of three bytes, each patch can have at most around 1365 triangles.

Table 8. Volume variation of blending patch construction alternatives.



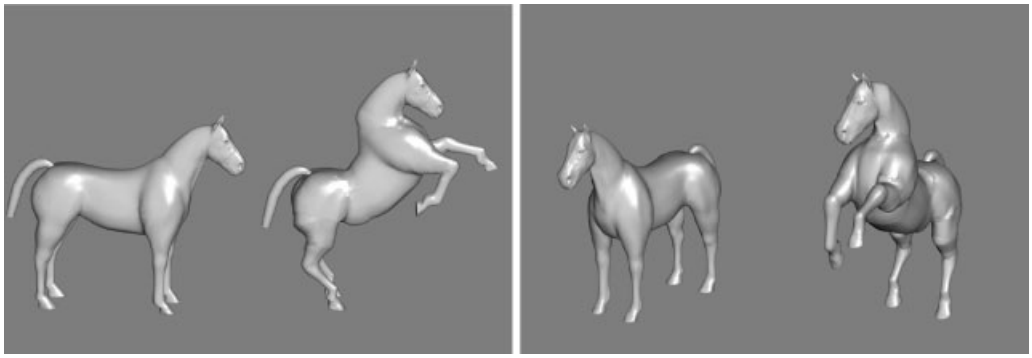


Figure 19. Horse model with mocap data all from Reference [50] animated using our technique.

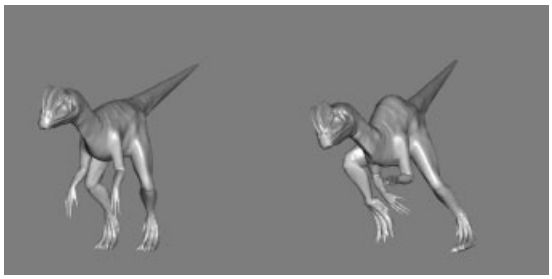


Figure 20. The dino model animated with human mocap data from Reference [51].

Finally, to substantiate the versatility of our approach we have applied our method to: (i) Mocap data including joint hierarchy information derived for a horse model [50]. The skeleton was then refined using our techniques and rigid skinning was obtained for the mocap data (see Figure 19). (ii) Mocap data for a human model along with the corresponding joint hierarchy derived from Reference [51]. This was adapted to the dino mesh, the mesh was decomposed, and the skeleton was refined appropriately. The results are illustrated in Figure 20.

8. CONCLUSIONS

We have introduced a suite of techniques for robust skeletal animation of 3D characters. The contribution of this work can be summarized in three directions. First we have presented a method for producing refined skeletons appropriate for articulated character animation. To this end, we have developed an improved local skeletonization method which in conjunction with approximate refinement algorithms increases the appropriateness and expressiveness of the produced skeletons. Then, to avoid artifacts that occur in linear blend skinning and the associated sample pose generation costs, we have developed a robust skinning method that eliminates the potential shortcomings from self-intersections, providing plausible character mesh deformations using blending patches around joints. Finally, to achieve real-time performance, we have developed GPU compatible software for all steps of our rigid skinning method.

Considering completeness there are many subtasks that could be examined regarding our improved principal axis skeletonization technique. There is need for further investigating in a more quantitative manner the grouping functions. Furthermore, one could try replacing the angle-weighted algorithm with an optimization approach which will compute the principal axis orientation more accurately.

Also, since the geometry shader was not designed to handle large-scale complex geometric operations, we could benefit from splitting the geometry processing into multiple steps that could be carried out in parallel. A partial solution for this problem would be the independent execution of the FBCP and RBCP on the GPU which would lead to the triangulation process being divided into two smaller independent subtasks that may be performed on a dual GPU graphics card without any considerable merging overhead.

Finally, an interesting extension to our approach is to introduce an automated placement of virtual bones in critical parts of the model [52,33]. This would enable advanced effects such as muscle bulging being captured efficiently with a slight time and space overhead.

REFERENCES

1. Kwatra N, Wojtan C, Carlson M, Essa IA, Mucha PJ, Turk G. Fluid Simulation with Articulated Bodies. *IEEE Transactions on Visualization and Computer Graphics* 2010; **16**(1): 70–80.
2. Watt A, Policarpo F. *3D Games, Volume 2, Animation and Advanced Real-time Rendering*. Addison-Wesley, Pearson Education Ltd. 2003.
3. Hétroy F, Gérot C, Lu L. Simple flexible skinning based on manifold modeling. *International Conference on Computer Graphics Theory and Applications*, GRAPP 2009; 259–265.

4. Lai Y-K, Hu S-M, Martin R-R, Rosin P-L. Fast mesh segmentation using random walks. In *SPM '08: Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, New York, NY, USA, 2008; 183–191 (ACM).
5. Baran I, Popović J. Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics* 2007; **26**(3): 72.
6. Lewis J-P, Cordner M, Fong N. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques*, New York, NY, USA, 2000; 165–172 (ACM Press/Addison-Wesley Publishing Co.).
7. James D-L, Twigg C-D. Skinning mesh animations. *ACM Transactions on Graphics* 2005; **24**(3): 399–407.
8. Schaefer S, Yuksel C. Example-based skeleton extraction. In *SGP '07: Proceedings of the 5th Eurographics Symposium on Geometry Processing*, Aire-la-Ville, Switzerland, Switzerland, 2007; 153–162 (Eurographics Association).
9. Shi X, Zhou K, Tong Y, Desbrun M, Bao H, Guo B. Example-based dynamic skinning in real time. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, New York, NY, USA, 2008; 1–8 (ACM).
10. Jacka D, Reid A, Merry B, Gain J. A comparison of linear skinning techniques for character animation. In *AFRIGRAPH '07: Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and interaction in Africa*, New York, NY, USA, 2007; 177–186 (ACM).
11. Vasilakis A, Fudos I. Skeleton-based rigid skinning for character animation. In *Proceedings of the 4th International Conference on Computer Graphics Theory and Applications, GRAPP 2009*, Lisboa, Portugal, 2009; 302–308.
12. Verroust A, Lazarus F. Extracting skeletal curves from 3d scattered data. In *In Proceedings of the International Conference on Shape Modeling and Applications*, 1999; 194–201.
13. Aujay G, Hétroy F, Lazarus F, Depraz C. Harmonic skeleton for realistic character animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aire-la-Ville, Switzerland, 2007; 151–160 (Eurographics Association).
14. Attene, M., Biasotti, S., Spagnuolo, M. Re-meshing techniques for topological analysis. In *Proceedings of the international Conference on Shape Modeling & Applications*, Genova, Italy, 2001; 142–151 (IEEE Computer Society Press).
15. Cornea N-D, Silver D, Yuan X, Balasubramanian R. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer* 2005; **21**(11): 945–955.
16. Ma W-C, Wu F-C, Ouhyoung M. Skeleton extraction of 3d objects with radial basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003*, Washington, DC, USA, 2003; 207 (IEEE Computer Society).
17. Wade L, Parent R-E. Automated generation of control skeletons for use in animation. *The Visual Computer* 2002; **18**(2): 97–110.
18. Lien J-M, Keyser J, Amato N-M. Simultaneous shape decomposition and skeletonization. In *SPM '06: Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, New York, NY, USA, 2006; 219–228 (ACM).
19. Laperriere R, Magnenat-Thalmann N, Thalmann D. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface*, Canadian Information Processing Society, Toronto, Ontario, Canada, 1988; 26–33.
20. Weber J. Run-time skin deformation. *Game Developers Conference*, 2000.
21. Lee M. Seven ways to skin a mesh character skinning revisted for modern GPUs. In *Proceedings of GameFest, Microsoft Game Technology Conference*, 2007.
22. Fernando R. GPU Gems: *Programming Techniques, Tips, and Tricks for Real-time Graphics*. Addison Wesley 2004.
23. Hejl J. Hardware Skinning with quaternions. In Game Programming Gems 4, 2004; Cengage Learning, Florence, KY .
24. Cordier F, Magnenat-Thalmann N. A data-driven approach for real-time clothes simulation. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, Washington, DC, USA, 2004; 257–266 (IEEE Computer Society).
25. Kavan L, Žára J. Spherical blend skinning: a real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, April 2005; 9–16 (ACM Press).
26. Kavan L, Collins S, Žára J, O'Sullivan C. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics* 2008; **27**(4): 1–23.
27. Andersen K. Spherical blend skinning on GPU. *Project Report*, Department of Computer Science, University of Copenhagen, 2007.
28. Sloan P-PJ, Rose C-F, III, Cohen M-F. Shape by example. In *I3D '01: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, New York, NY, USA, 2001; 135–143 (ACM).
29. Kry P-G, James D-L, Pai D-K. EigenSkin: real time large deformation character skinning in hardware. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, New York, NY, USA, 2002; 153–159 (ACM).

30. Wang X-C, Phillips C. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM Siggraph/Eurographics Symposium on Computer Animation*, New York, NY, USA, 2002; 129–138 (ACM).
31. Merry B, Marais P, Gain J. Animation space: a truly linear framework for character animation. *ACM Transactions on Graphics* 2006; **25**(4): 1400–1423.
32. Mohr A, Gleicher M. Building efficient, accurate character skins from examples. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, New York, NY, USA, 2003; 562–568 (ACM).
33. Kavan, L., Collins, S., O'Sullivan, C. Automatic linearization of nonlinear skinning. In *Proceedings of the 2009 Symposium on interactive 3D Graphics and Games, I3D'09*, Boston, Massachusetts, 2009; 49–56 (ACM).
34. Kavan L, McDonnell R, Dobbyn S, Žára J, O'Sullivan C. Skinning arbitrary deformations. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D Graphics and Games*, New York, NY, USA, 2007; 53–60 (ACM).
35. Chaudhuri P, Papagiannakis G, Magnenat-Thalmann N. Self adaptive animation based on user perspective. *The Visual Computer* 2008; **24**(7): 525–533.
36. Yoshizawa S, Belyaev A, Seidel H-P. Skeleton-based variational mesh deformations. *Computer Graphics Forum* 2007; **26**(3): 255–264.
37. Larboulette C, Cani M-P, Arnaldi B. Dynamic skinning: adding real-time dynamic effects to an existing character animation. In *SCCG '05: Proceedings of the 21st Spring Conference on Computer Graphics*, New York, NY, USA, 2005; 87–93 (ACM).
38. James D-L, Pai D-K. DyRT: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 2002; 582–585 (ACM).
39. Barber C-B, Dobkin D-P, Huhdanpaa H. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software* 1996; **22**(4): 469–483.
40. Gottschalk S-A. Collision queries using oriented bounding boxes. *Doctoral Thesis*, UMI Order Number: AAI9993311, The University of North Carolina at Chapel Hill, 2000.
41. Fudos I, Hoffmann C-M. Constraint-based parametric conics for CAD. *Computer-aided Design* 1996; **28**(2): 91–100.
42. Dey T-K, Goswami S. Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the 8th ACM Symposium on Solid Modeling and Applications*, New York, NY, USA, 2003; 127–134 (ACM).
43. NVIDIA Corporation: NVIDIA OpenGL Extension Specifications for the GeForce 8 Series Architecture (G8x), 2008. Available at http://developer.nvidia.com/object/nvidia_opengl_specs.html
44. Smith M-L. GPGPU with OpenGL and VisualWorks. Available at: http://www.cincomsmalltalk.com/userblogs/mls/blogview?entry=3412284871
45. Buchart C, Borro D, Amundarain A. GPU local triangulation: an interpolating surface reconstruction algorithm algorithm. *Computer Graphics Forum* 2008; 27(3): 807–814.
46. Blender Foundation. *Blender: An Open Source 3D Content Creation Suite*. Available at http://www.blender.org
47. Luttgens K, Hamilton N. *Scientific Basis of Human Motion*, 9th edn. Brown & Benchmark: Madison, WI, 1997.
48. e-frontier Inc. Poser 7 3D Figure Design and Animation. Available at http://www.e-frontier.com.
49. Biovision. BVH Format. Available at: http://www.biovision.com/bvh.html
50. PlanIt 3D. The Ultimate Poser Horse Morph Set. Available at http://www.planit3d.com
51. CMU Graphics Lab. Motion Capture Database. Available at: http://mocap.cs.cmu.edu/
52. Wang R-Y, Pulli K, Popović J. Real-time enveloping with rotational regression. In *ACM SIGGRAPH 2007 Papers*, New York, NY, USA, 2007; 73 (ACM).

AUTHORS' BIOGRAPHIES

Andreas-Alexandros Vasilakis was born on 12 October 1983 in Corfu. He received his BSc and MSc degrees in 2006 and 2008, respectively from the Department of Computer Science of the University of Ioannina, Greece. He is currently a PhD candidate at the same department. His research interests include geometry algorithms, object modeling, character animation, and GPU programming.

Ioannis Fudos is an Assistant Professor at the Department of Computer Science of the University of Ioannina. He received a diploma in computer engineering and informatics from University of Patras, Greece in 1990 and an MSc and PhD in Computer Science both from Purdue University, USA in 1993 and 1995, respectively. His research interests include animation, rendering, morphing, CAD systems, reverse engineering, geometry compilers, solid modelling, and image retrieval. He has published in well-established journals and conferences and has served as reviewer in various conferences and journals. He has received funding from EC, the General Secretariat of Research and Technology, Greece, and the Greek Ministry of National Education and Religious Affairs.