Skeletal Rigid Skinning with Blending Patches on the GPU

Andreas A. Vasilakis, Ioannis Fudos
Department of Computer Science - University of Ioannina
Ioannina, Greece
email: $\{abasilak, fudos\}$@cs.uoi.gr

Abstract

In this paper, we present a novel skeletal rigid skinning approach. First, we introduce a skeleton extraction technique that produces refined skeletons appropriate for animation from decomposed solid models. Then, to avoid the artifacts generated in previous rigid 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 implemented all steps of our rigid skinning algorithm on the GPU. Finally, we present an evaluation of our techniques against four criteria: efficiency, quality, scope and robustness.

Keywords: skeleton extraction, rigid skinning, character animation, re-meshing, GPU implementation, real time

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. 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 representation model consists of at least two main layers: 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*) providing a kinematic model of the character. Quite often, it is common practice to add a third layer to support realistic effects of the character's musculature. A skeleton has usually much simpler structure than the original object that aims at simplifying the skinning process by avoiding the tedious task of animating each vertex independently. The process of extracting a skeleton is called *skeletonization*.

There are two 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 (see e.g. [49]).

- 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. Then we compute a skeleton that is compatible with this decomposition. We follow this approach because it can performed fully automated without any user intervention.

A skeleton acts as a special type of deformer transferring its motion to the skin by assigning each skin vertex one (*rigid skinning*) or more (*linear blend skinning* - LBS) joints 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 of LBS (also known as *skeleton subspace deformation (SSD)*, *vertex blending* or *enveloping*), each skin vertex is assigned multiple influences with associated blending weights for each joint. This scheme provides more detailed control over the results. The deformed vertex is computed by a convex combination of the corresponding joints. The generated meshes exhibit volume loss as joints are rotated to extreme angles producing non-natural deformations, such as the collapsing joint and the candy wrapper effect(Figure 1). Despite these, variations of this method are widely used in real time computer graphics applications because they are simple and easy to implement on GPUs.

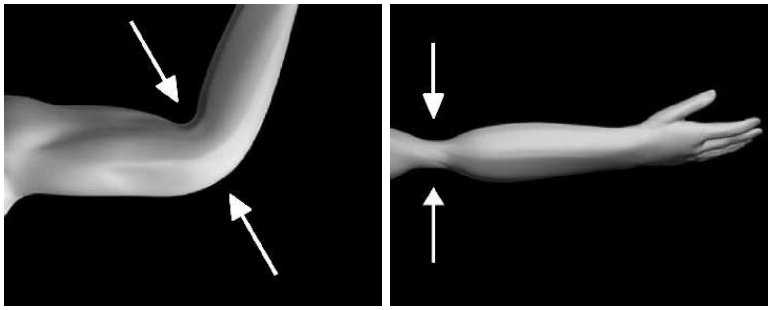

Figure 1: “Collapsing elbow” defect (left) and “candy-wrapper” defect (right) [45].

Vertex weight tuning tends to be a tedious and cumbersome task which is applied on a single mesh. [55] presented a direct weight manipulation tool which is time-consuming and not easily applicable to other skinning methods. Other methods result in serious flaws on characters with high-resolution meshes due to the irregularity of the surface weights [69] and the skin resolution dependence [35]. More recently, example-based approaches have been proposed to infer the character articulation from a training set of example poses [33, 63]. Example-based fitting is a complicated iterative process due to the implicit dependencies of the deformed vertex positions. Researchers have identified as potential drawbacks of these schemes the need for multiple example meshes and the complexity of avoiding *over-fitting* caused by under-determined weights. Also, such techniques require decomposition of the articulated figures into several component meshes. This step however does not affect the real time feasibility of the process, since it can be performed beforehand as a preprocessing step and stored as part of the character representation.

For the purposes of seamless character animation we aim at building a system that eliminates the skin attachment (weight fitting) part and overcomes LBS shortcomings by producing a single mesh for each frame. To support this functionality, we present an integrated skeletal rigid-skinning framework that is given as input a character representation and a motion description of its extracted skeleton, and produces natural deformations. Given a static character mesh, an effective decomposition method [5, 61, 19] or an experienced user partitions the mesh into visually meaningful or otherwise appropriate with regards to the application components. Additionally, a functional skeleton is extracted using centroids and principal axes [47] of the character’s components by performing a depth-first traversal of the skeleton hierarchy tree. We refine the produced skeleton segments to derive better skeletal representations that are more appropriate for our application. Then, we use rigid skinning

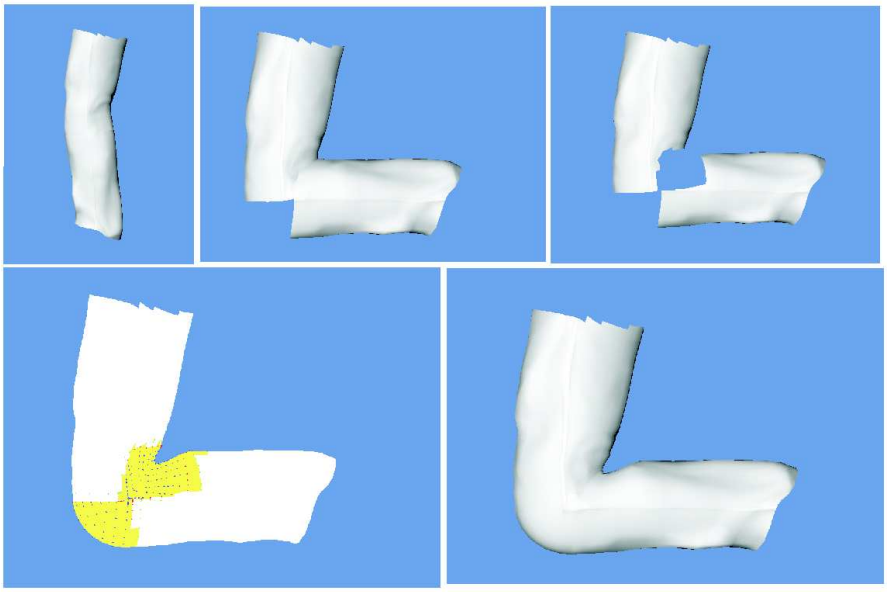

Figure 2: Robust rigid skinning. From top left to bottom right: the initial model; rotating the lower part; removing points; adding points; constructing a robust blending patch between the two components.

by assigning each skin vertex to an influence bone to achieve mesh animation avoiding thus weight fitting and training pose set production costs. We introduce a novel 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's modules.
- 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 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 working on the overall mesh. We introduce novel blending patch construction techniques that preserve the initial volume of the joint 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 version of this work was presented in [66]. We have added a more complete description of the skeleton extraction methods offering background material for principal axis and kernel centroid computations. Moreover, we present a detailed evaluation of four quality measures of the proposed deformation technique addressing volume conservation and mesh smoothness. We have further improved the testing process accuracy by including one extra plane-point test in the step of removing overlapping meshes. Finally, we provide a full GPU realization of the entire skinning algorithm, which enables us to achieve real time performance while increasing significantly the quality of the produced result as compared to previous methods.

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 basic rigid-skinning method along with its GPU implementation is outlined 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

There is an abundance of research work in the literature that tackles the skeleton extraction and skinning of 3D objects from different perspectives. We focus on recent developments most closely related to 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)* [12] is a popular topological skeletal representation technique which consists of a set of curves which roughly run along the object's middle. MAT-based representations suffer from perturbation and noise dependence, high computation cost for 3D [18] ($O(n^2 \log n)$ in worst case [62]), shape complexity (because in 3D they consist of surface elements). Researchers have proposed approximate MAT to overcome some of the above inefficiencies, using Voronoi diagram [2, 21] or dual Delaunay triangulation [4]. However MAT based skeletons are still not well fitted for character animation, i.e. for representing efficiently the bone hierarchy of an articulated figure. *Reeb graphs* are a fundamental 1-D data structure for representing the configuration of critical points and their relationships in an attempt to capture the intrinsic topological shape of the object [64, 67, 57, 8]. However, a re-meshing technique [6, 31] is usually required to generate accurate skeletons.

Volumetric methods. Several methods generate skeletons by constructing discrete field functions by means of the object's volume. In this direction researchers have proposed using distance transform [69], repulsive force field [48, 17] or radial basis [51]. Other volumetric-based techniques make use of a multi-resolution thinning process applied on the model's voxelized representation [27, 50]. Although accurate, such 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 use different approaches to achieve more accurate and efficient skeletonization. [60] presents a method for extracting a hierarchical, rigid skeleton from a set of example poses. Researchers also generate skeletons based on mesh contraction [46, 7]. [35] extracts a skeleton using a hierarchical mesh decomposition algorithm. Finally,

[47] proposed an iterative approach that simultaneously generates hierarchical shape decomposition and a corresponding set of multi-resolution skeletons. We have adapted the technique presented in [47], that was initially targeted to reverse engineering, for the purposes of character animation.

2.2 Skinning

We focus on skeleton-driven skinning introduced by [56]. Linear blend skinning (LBS) is the most widely used technique for real-time animation in spite of its limitations [72] due to its computational efficiency and straightforward GPU implementation [44]. LBS was initially presented in the game development community [41, 42]. LBS determines the new vertex position by linearly combining the results of the vertex transformed rigidly with each influence bone. A very high performance LBS method using a vertex shader program is proposed in [24]. Further parallelism achieved in [59] using multiple passes to a fragment shader. GPU progressive skinning [58] extends LBS to support a continuous level of animation detail.

Recent skinning algorithms are classified based on whether they use a single input mesh (*Geometric methods*) or a training set of poses (*Example-based methods*) of input models. Further, many methods have been devised for detailed geometric deformation combined with dynamic deformation using physical constraints (*Physics inspired methods*). In the first case, vertex weighting is usually specified manually. In the second case, researchers have proposed to automatically approximate realistic skin deformation by training weights with one [9] or multiple input meshes [34, 33]. Our rigid skinning approach falls under geometric class of algorithms introducing a novel versatile, robust and efficient blending approach based on rational quadratic Bezier patches.

Geometric methods revert to non-linear blending of rigid transformations since deformation is inherently spherical. Numerous proposed methods have replaced the linear blending domain with simple quaternions [30], [1] or have proposed matrix operators [52], log-matrix [16] operators, spherical blending (SBS) [39] and dual quaternions (DQS) [37]. SBS and DQS has been implemented on the GPU achieving performance comparable to LBS [3, 37]. The drawback of DQS is that dual quaternions do not exist in most existing graphics libraries yet. A promising research direction with the same principle (no weights used) as our approach was proposed by [75]. They used the skeleton to control the simplices

defining the model instead of the vertices yielding smooth transitions near joints. Another alternative recently explored technique is the use of 3D free-form character articulation using sweep-based [32, 77] or spline-based techniques [76, 25]. Kanan et al. [38] introduce a method for automatically constructing skinning approximations of arbitrary precomputed animations, such as those of cloth or elastic materials. Finally, [36] demonstrates that any nonlinear skinning technique can be approximated to an arbitrary degree of accuracy by linear skinning, using just a few samples of the nonlinear blending function (virtual bones).

Example-based methods remove artifacts by correcting LBS errors while the storage and computation costs increase. Initial approaches combined rigid skinning with interpolation examples using radial basis functions [45, 65]. EigenSkin [40] used principal component analysis for approximating the original deformation model based on GPU vertex programming. Multi-Weight Enveloping (MWE) [71] and Animation Space [53] are similar methods that introduce more weights per influence bone to provide more precise approximations and additional flexibility. [33] found the Animation Space technique to consistently perform better than LBS and MWE while MWE also suffered from over-fitting. In addition, [54] introduced extra bones to capture richer deformations than the standard LBS model. A recent research work proposes a replacement of linear with rotational regression when examples are available [70, 73]. Finally, [15] 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 [14, 29, 78] or add secondary deformations enriching skeleton driven animations [43, 68, 63], by paying the price of increasing considerably the computational complexity.

3 Preliminaries

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 $m^k = \frac{1}{2}(p^k + q^k + r^k)$. Finally, the total area of the convex hull is denoted by $A^{CH(C)} = \sum a^k$.

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

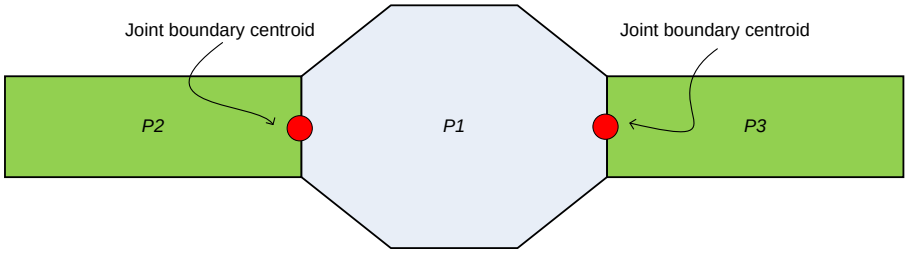

Figure 3: The generated joint boundary centroids between P1 and the adjacent components P2 and P3.

of a component C, $BC(C) = \{bc_1, \ldots, bc_n\}$, where n is the number of joint boundaries of C.

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 divides the 3D space. The kernel is the intersection of all such inner half-spaces. The simple centroid computation of a kernel may return nonintuitive results when applied on non-uniformly distributed points. Instead, we evaluate the kernel centroid as the weighted average of the triangle centroids of its convex hull (since kernel is a convex set) weighted by their area. Hence, the centroid of the convex hull is denoted by $m^{CH(C)}$ and is given by $m^{CH(C)} = \frac{\sum a^k m^k}{A^{CH(C)}}$. 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 [10].

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 the entire faces of the primitives instead of using only the vertices [28]. The principal axis is the eigenvector of the covariance matrix which corresponds to the largest eigenvalue. The covariance matrix for the aforementioned case is given by:

$$C_{ij} = \frac{\sum_{k=0}^{n} a^k (9 m_i^k m_j^k + T_i^k \cdot T_j^k)}{12 A^{CH(C)}} - m_i^{ch} m_j^{CH(C)} \tag{1}$$

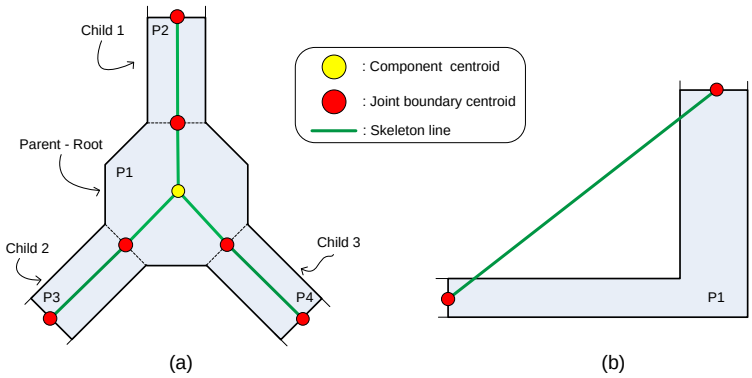

Figure 4: (a) A simple example of a skeletal representation using the Boundary Method. (b) An example that illustrates the inappropriate skeleton extraction that arises when this method is applied to non convex components.

4 Skeleton Extraction

We have built on techniques introduced by [47] that produces 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 produced skeletons. The global skeleton of the character is then reconstructed by connecting the refined skeleton components.

4.1 Previous Approaches to Skeleton Extraction

4.1.1 Boundary Method

A straightforward approach to building the skeleton of a component is to connect the joint boundary centroids. Note that for the component identified as root 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 (see for example Figure 4).

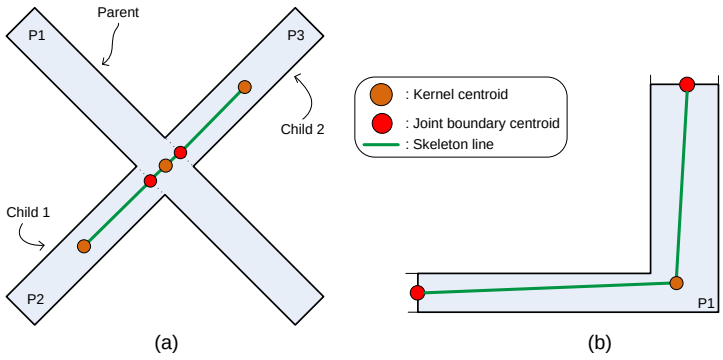

Figure 5: (a) A simple example of a skeletal representation using the Centroid Method. Skeleton of component P_1 does not capture its shape accurately. (b) The Boundary Method artifact of Figure 3(b) addressed using the Centroid Method.

4.1.2 Centroid Method

An improvement over the boundary method is to build local skeleton segments by connecting joint boundary centroids to the center of the component's mass [47]. 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 kernel)(Figure 5).

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

4.1.3 Principal Axis Method

In [47], 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.

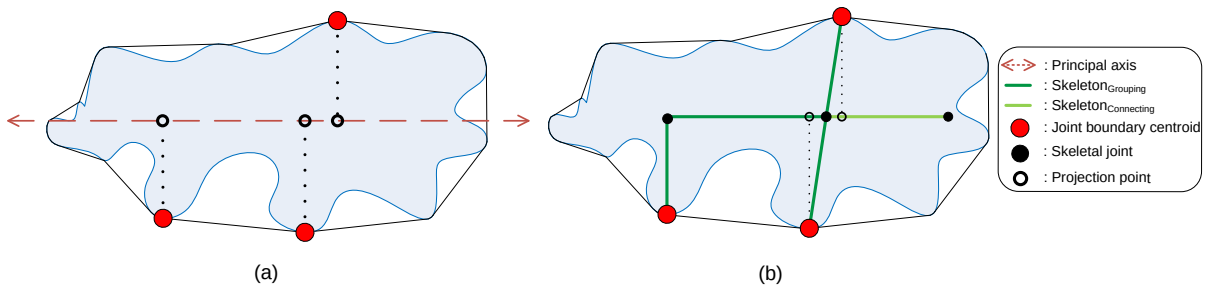

Figure 6: Skeleton extraction based by the principal axis method.

4.2 Optimized Skeleton Extraction

Our approach adopts and improves this skeleton extraction technique for use in an animation context. 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. We denote this major axis segment as pa. 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.

Moreover, the method outlined in [47] 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 $|BC|$, we propose to subdivide pa in a number of uneven segments by picking as link points the projections of the joint boundary centroids on pa. Let $P_{pa}(bc_i)$ be the projection of a joint boundary centroid bc_i on pa. If this projection lies outside pa then we use instead the end point of 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 6).

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 pa (Section 4.2.1).

After grouping has been performed, we create the set of skeleton edges. Beyond the standard connections between the joint boundary centroids and the link points and line segments that interconnect link points, we use extra skeleton edges based on the mapping result (Section 4.2.2).

Finally, we propose to add four more joints which represent the two ends of the other axes segments hence adding more topological information to skeletal representation.

The principal axis algorithm is more expensive to compute than the other two methods due to the principal axis and kernel centroid computation but achieves a higher quality skeleton by encoding the topology of the character's 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 a joint boundary centroid.

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

First, we compute their link point on the principal axis as the mean of their projections. By doing so, the sum of their distances to 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 evaluated as the normalized distance of the joint boundary centroid to the principal axis,

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

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

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

So, the total mapping length cost among these 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 normalized length of the principal axis which is vanished after merging these joint boundary centroids. This factor equals to the length of the principal axis which is generated among their projections divided by the length of the maximum principal axis segment that can be constructed,

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

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 projection set bc_{ij} is defined as the average of the total distance cost plus the ratio of length of the principal axis we lose,

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

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,

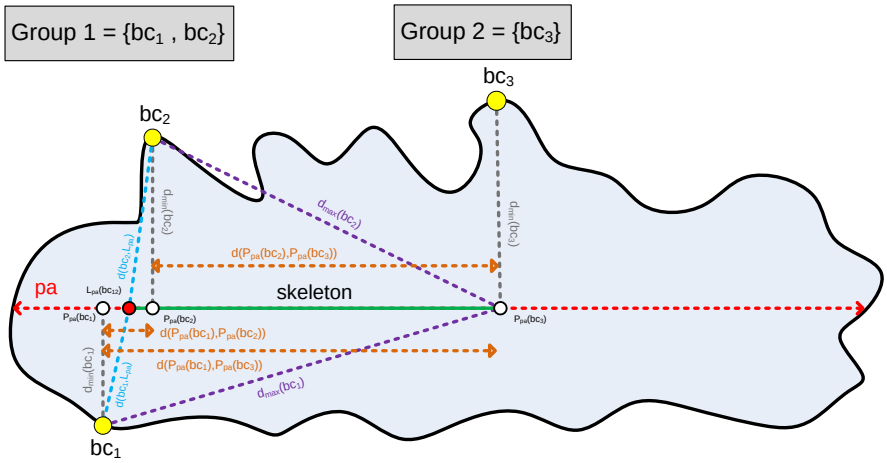

Figure 7: An example of grouping with three joint boundary centroids.

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

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 7 demonstrates a grouping example of a component with three joint boundary centroids. Algorithm 1 summarizes the grouping process. By $G[i, j]$ we denote the optimal solution for the sub-problem $\langle oc_i, \ldots, oc_j \rangle$.

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. Joint boundary centroid grouping often generates skeletons that do not capture important topological information. Therefore, the connection algorithm works with the following rules:

1. If all joint boundary centroids are grouped to one link point and this point is closer to:

   - pa's centroid, we connect it with the pa's end points on both sides of its centroid.

Algorithm 1 group(BC, pa)

1. **for each** $bc_i \in BC$ **do** $G[i,i] = bc_i$;
2. **for** $l = 2$ **to** $|BC|$ **do**
3. **for** $i = 1$ **to** $|BC| - l + 1$ **do**
4. $j = i + l - 1$;
5. $G[i,j] = \langle bc_i, \ldots, bc_j \rangle$;
6. $s_1 = F_1(G[i,j], pa)$;
7. **for** $k = 1$ **to** $j - l$ **do**
8. $s_2 = F_2(G[i,k], G[k+1,j], pa)$;
9. **if** $s_2 < s_1$ **then**
10. $G[i,j] = G[i,k] \cup G[k+1,j]$;
11. $s_1 = s_2$;
12. **return** $G[0, |BC|]$;

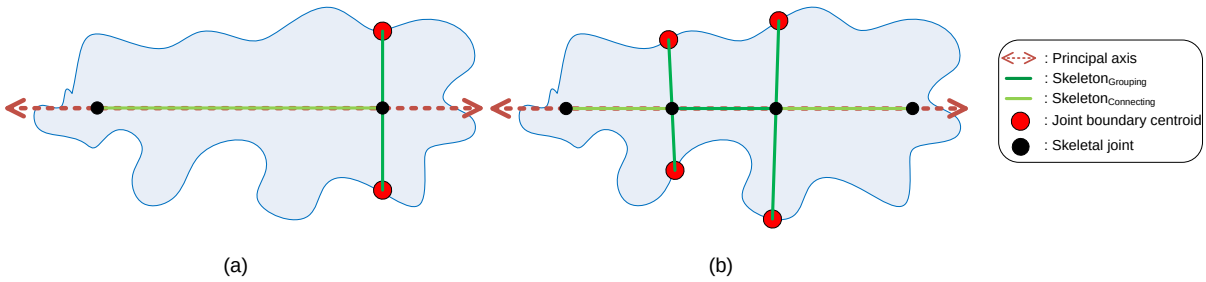

Figure 8: The refined skeletons obtained by the connecting algorithm capture more information about the shape of the components.

- one of pa's end points, we connect it with the pa's end point on the other side of the pa's centroid.

2. If all joint boundary centroids are connecting to more than one link point and these points are closer to:

   - pa's centroid, we connect the first and the last link points with the pa's end points on both sides of pa's centroid, respectively.
   - one of pa's end points, we connect the first or the last link point with the pa's end point on the other side of pa's centroid.

Figure 8 shows the resulting skeletons of two components using this technique.

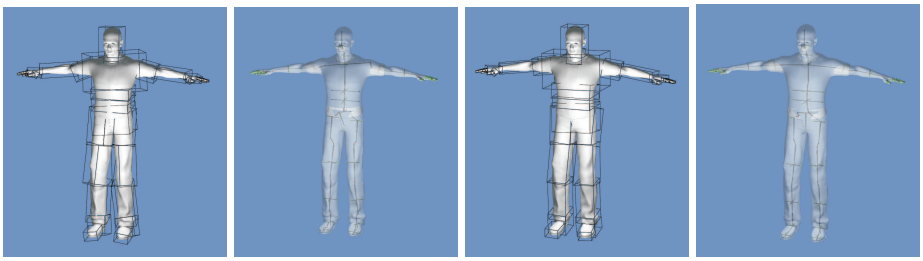

Figure 9: Oriented bounding boxes and skeletons of "Human" model components: (left) Original configuration (right) After alignment.

5 Skeleton Refinement

Covariance-aligned principal directions are not optimal due to the used approximation algorithm. In this section we introduce a process that improves the alignment through small corrective adjustments. Clearly, the PCA algorithm fails to derive a unique valid solution when applied to point clouds that are characterized by a large degree of symmetry. However, it can provide useful topological information that if combined with qualitative features may improve the initial solution considerably. Our approach adjusts the principal direction orientation using two individual processes; the first uses local characteristics while the second uses inherited hierarchical information, i.e. knowledge derived from the ancestors. Figure 9 illustrates the qualitative improvement of the skeletons and the corresponding OBBs 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 define these features as the extracted skeletons segments by the centroid method; i.e. vectors defined by connecting the kernel's 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 10).

If the component has more than one joint boundary centroids, 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,

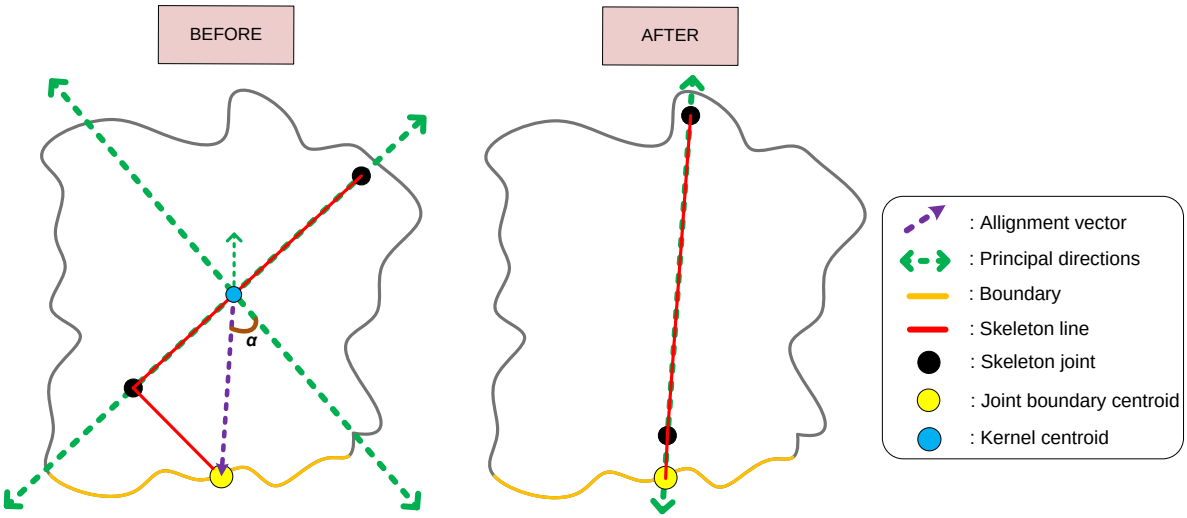

Figure 10: This example illustrates the result of a local refinement operation on a component with one joint boundary centroid: (left) the initial principal axes orientation and (right) the produced skeleton after local alignment.

we weighted 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 $5^\circ - 20^\circ$ degrees interactively. Modifications that require rotations beyond this upper bound are rejected.

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,

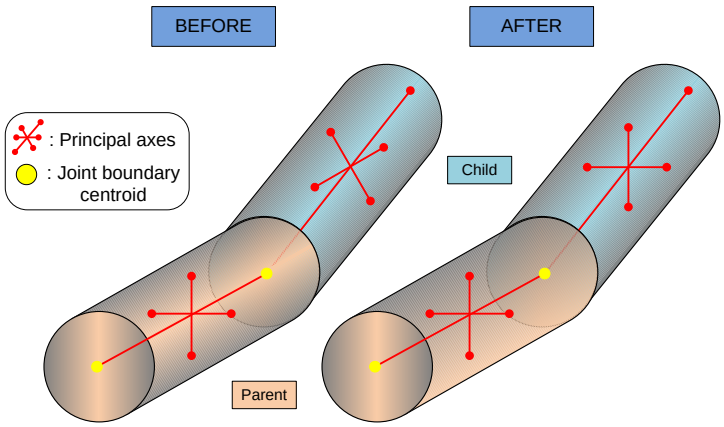

Figure 11: Comparison of the extracted skeletons (left) before and (right) after using the hierarchical refinement algorithm.

from the rest of the child and parent principal directions we detect those that are closest and aligns them (Figure 11) accordingly. 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's surface by assigning each skin vertex one joint as driver. A skin vertex is pinned in the local coordinate system of the corresponding joint, which is the one derived during the skeletonization procedure. By using homogeneous coordinates, a joint transformation is represented through a 4×4 matrix W. The vertex then repeats whatever motion this joint 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 joint. Similarly, the normal of each skin vertex is transformed by rotating it with the upper 3×3 rotation matrix R of W: $n' = Rn$. Despite its simplicity and computational efficiency, rigid skinning makes it difficult to obtain sufficiently smooth skin deformation in areas around the joints. This has led to the emergence of more specialized techniques that improve the undesirable rough skinning results. We have develop a novel technique to address these major flaws. This technique is carried out in four steps:

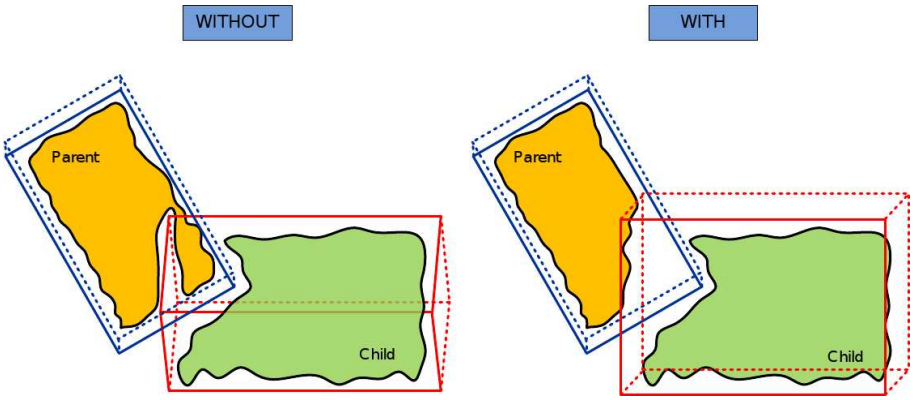

Figure 12: The difference of the cutting result without and with applying hierarchical refinement.

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 6 plane-point tests result having 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 13). The use of the joint boundary plane decreases significantly the number of removed points by 20% - 50% (see Figure 14).

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

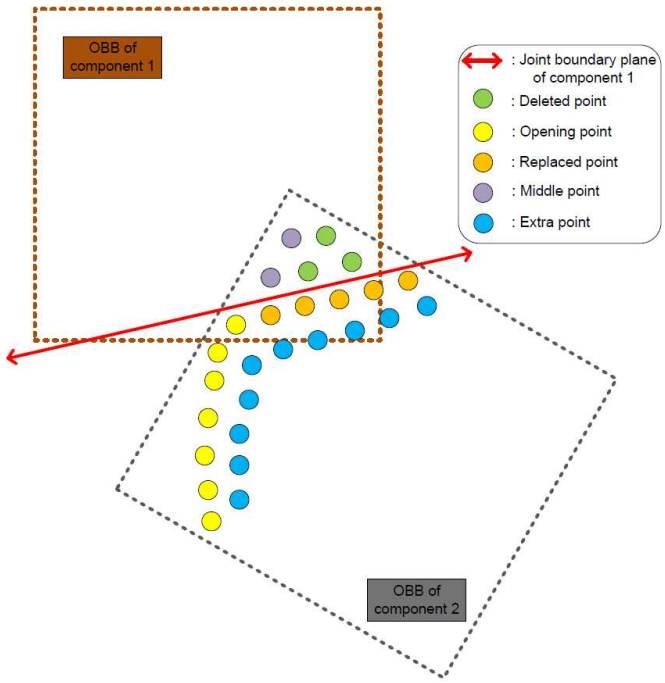

Figure 13: This is an example which shows the point classification for component 2 after removing the unnecessary points when the joint boundary plane is used.

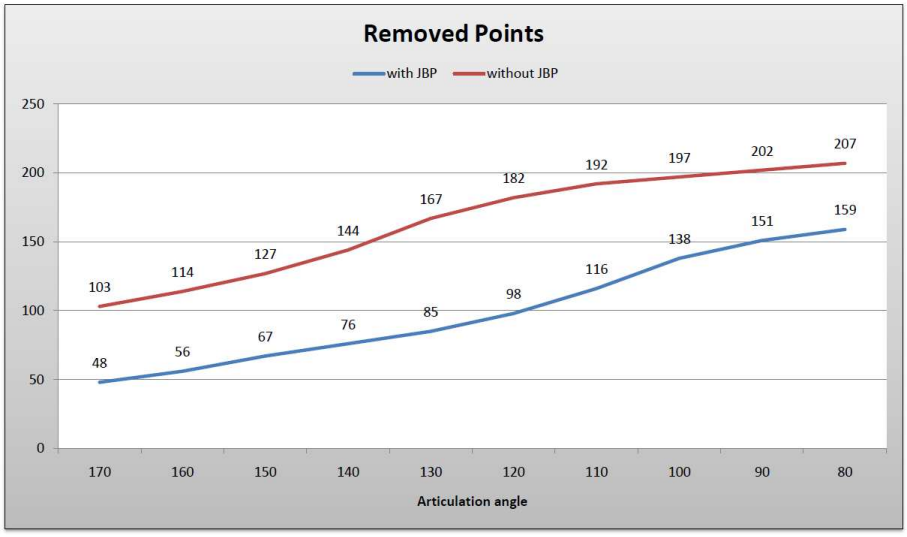

Figure 14: Reduction of removed points by introducing the joint boundary plane test.

The refined and aligned OBBs derived during the skeletonization process provide reduced overlapping between the moving adjacent components. An extreme example of the result of not using hierarchical refinement is illustrated in Figure 12.

6.2 Adding Vertices

Definition 3. To facilitate the description of the blending mesh we define as:

- *Opening* is the set of joint boundary points of the component that have not been removed.
- *Middle* is the point set that we obtain if we subtract Opening set from the joint boundary point set.
- *Replaced* is the point subset of the component that corresponds to new end points created by the removed points that will be used to fill in the gap.
- *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 them with B_i^s and B_i^f, respectively).
- *Extra* is the point set that consists of all 1-ring neighbors of the points that belong to $Opening \cup Replaced$ set.

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 results in movement of every point $O_j \in Opening$ on a circular arc. The task is to find new points by interpolating from the point's position $(O_j^s = q_s O_j q_s^{-1})$ before quaternion rotation q_s to the point's position $(O_j^f = q_f O_j q_f^{-1})$ after quaternion rotation q_f (see Figure 15).

We have used fast linear interpolation of quaternions (QLERP) since the interpolation along the shortest segment hardly causes any observable skin defect [39]. QLERP is computed as

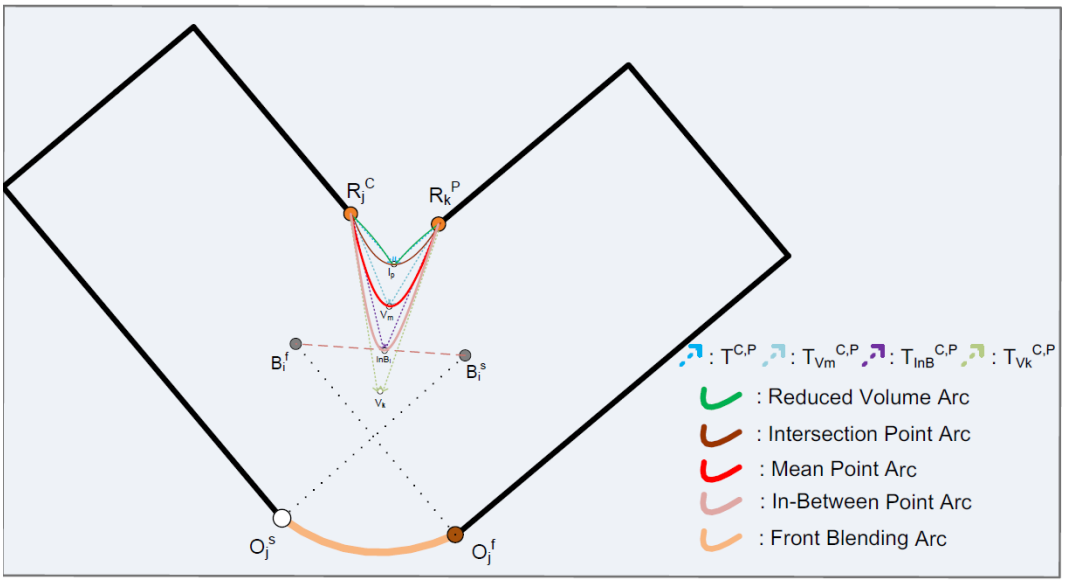

Figure 15: Blending Patch Construction processes

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

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

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

where I is a 3×3 identity matrix. Finally, a new point is constructed using:

$$v' = TQO_j^s T^{-1} \tag{14}$$

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. Thus, we introduce a new point set, called *In-Between* points.

A grouping algorithm is used to create *triplets* from each $InB_i \in$ *In-Between* and one point from each of the child and parent *Replaced* sets (R_j^C, R_k^P selected points, 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 of the best fitting triplet ($\approx O(n^3)$), 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 i*-th* step of the iterative algorithm, y_i^C and y_i^P are evaluated as the next position of previous step selected candidates (j, k) of R^C, R^P sets. Algorithm 2 summarizes the fitting triplet process, where N^{PL} is the normal vector of *PL* plane. The number of steps for each component are initialized to $lS^C = 0.1 * |R^C|$ and $lS^P = 0.1 * |R^P|$. Finally, the tangent vectors at R_j^C, R_k^P are defined as the projections on the triplet plane of the outgoing vectors from the child and parent centroids to the joint position *J*.

If we construct the patch with interpolated points of a circle equation, artifacts will arise due to the 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 (v_x^C, v_y^C) and (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 $U = (v_x^C M_2 - v_x^P M_1, v_y M_2 - u_y M_1)$, 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 [26]:

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

Algorithm 2 triplet(*In-B*, R^C, R^P, N^{PL})

1. **Sort***(In-B);* **Sort**(R^C); **Sort**(R^P);
2. $j \leftarrow 0; k \leftarrow 0;$
3. $lS^C \leftarrow 0.1 * |R^C|; lS^P \leftarrow 0.1 * |R^P|$;
4. **for each** $InB_i \in$ *In-B* **do**
5. $\quad minA \leftarrow MAX_{float}; i^C \leftarrow 0; y^C \leftarrow j;$
6. $\quad$ **while** $i^C++ < lS^C \;\&\&\; y^C < |R^C|$ **do**
7. $\quad\quad D^C \leftarrow R^C_{y^C} - InB_i;$
8. $\quad\quad i^P \leftarrow 0; y^P \leftarrow k;$
9. $\quad\quad$ **while** $i^P++ < lS^P \;\&\&\; y^P < |R^P|$ **do**
10. $\quad\quad\quad D^P \leftarrow R^P_{y^P} - InB_i;$
11. $\quad\quad\quad N^{C,P} \leftarrow normalize(cross(D^C, D^P));$
12. $\quad\quad\quad angle \leftarrow \arccos(dot(N^{PL}, N^{C,P}));$
12. $\quad\quad\quad$ **if** $angle < minA$ **then**
13. $\quad\quad\quad\quad minA \leftarrow angle;$
14. $\quad\quad\quad\quad j \leftarrow y^C; k \leftarrow y^P;$
15. $\quad\quad\quad y^P++;$
16. $\quad\quad y^C++;$
17. $\quad$ *rationalBezierInterpolation*$(InB_i, R^C_j, R^P_k);$

where

$$W = \frac{Area(P^C, P^I, P^P)U}{R}, w = \frac{Area(P^C, P^I,)P^P\vec{v^C} \times \vec{v^P})}{R}$$
$$R = \sqrt{d(P^P, \vec{v^C}, P^C)d(P^I, \vec{v^C}, P^C)d(P^C, \vec{v^P}, P^P)d(P^I, \vec{v^P}, P^P)}$$
$$d(S, \vec{r}, R) = (S_y - R_y)r_x - (S_x - R_x)r_y$$
$Area(C, P, D)$: 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 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^C_{InB}, T^P_{InB} are the vectors going from R^C_j, R^P_k to InB_i, respectively.
- $T^C_{V_m}, T^P_{V_m}$ are the vectors going from R^C_j, R^P_k 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 15 and 16). These four alternative selections are evaluated in Section 7 in terms of quality and robustness.

1. : V_m with T^C_{InB}, T^P_{InB}, and the resulting gap filling process is called *Mean Point Interpolation.*
2. : I_p with T^C_{InB}, T^P_{InB}, and the resulting process is called *Intersection Point Interpolation.*
3. : I_p with $T^C_{V_m}, T^P_{V_m}$, and the resulting process is called *Reduced Volume Intersection Point Interpolation.*
4. : InB with $T^C_{V_k}, T^P_{V_k}$, and the resulting process is called *In-Between Point Interpolation.*

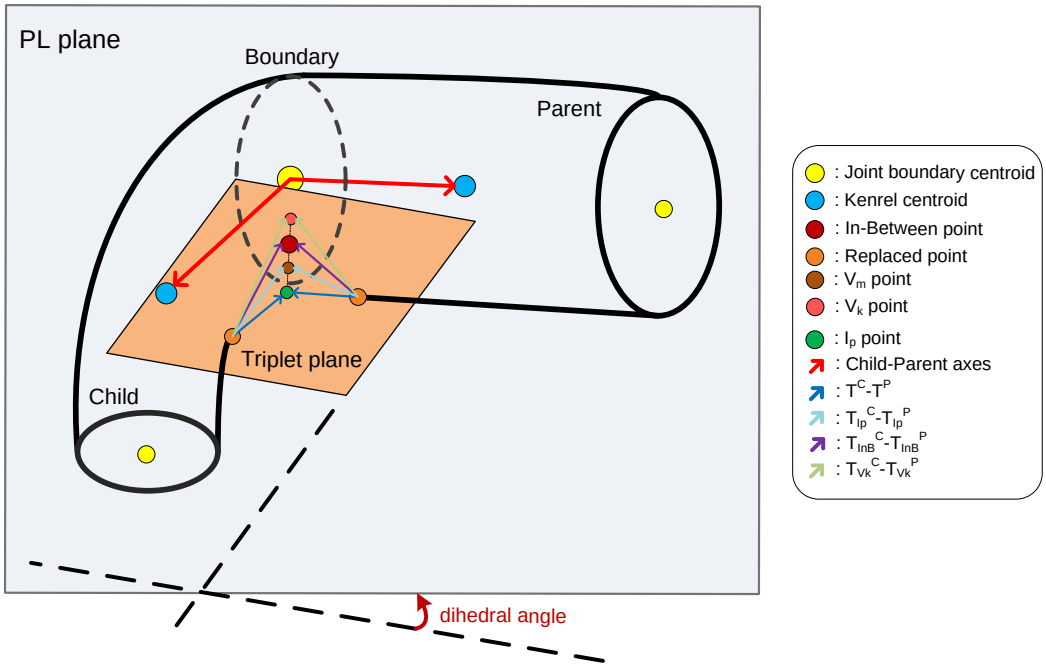

Figure 16: Rear Blending Patch Construction process alternatives and notation.

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* [20] algorithm to build water-tight surfaces around the newly inserted points. Since we need only two parts of the generated spheroid to fill the holes, we must eliminate a number of unnecessary faces to derive optimal lighting results. To this end, we adjust the normal vector orientation 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

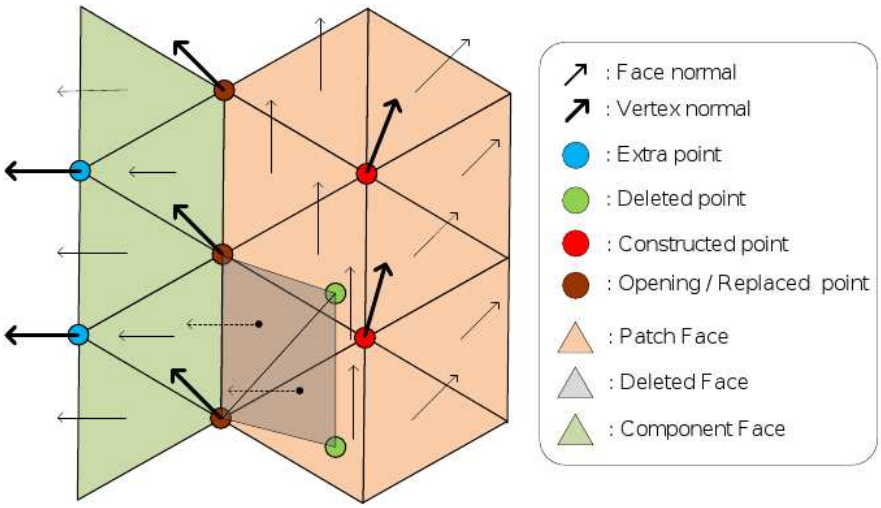

Figure 17: The normal vector evaluation of all possible mesh points.

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 17).

6.4 Porting the Rigid Skinning Algorithm on a GPU Platform

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 be parallelized on a GPU architecture using five render stages. We have significantly improved the overall performance of the proposed method by employing GPU which takes advantage of the SIMD programming model (Single Instruction Multiple Data) to accelerate the whole rigid skinning process with minimal CPU involvement. 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 [23], 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 trasformations on large data sets as compared to standard GPU data communication schemes [74].

Geometry information from the rest pose is stored into *vertex buffer objects* (VBO) to reduce data transfer between CPU and GPU. Figure 18 illustrates the GPU work flow for our implementation. The vertex, geometry and fragment shader programs were written in GLSL. The framework was developed using OpenGL as graphics library under C++. Unfortunately for porting these complicated graph-based operation on the GPU we have used extensions that are supported only on NVIDIA Geforce 8800 and later series. As geometry shaders will be supported by other GPUs as well it is straightforward to port this code to other GPU brands, since we use only the extensions of OpenGL 2.1 GLSL shaders.

6.4.1 Removing Vertices

Vertices VBOs of the overlapping components along with an array of each joint boundary point indices and the corresponding BBs are transfered to the *Remove Points* vertex shader program. If a point is inside the BB 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 BB collision detection, this shader will also characterize the exported point using one float value [1,2,0] as *Middle*, *Opening*, or neither of them, by checking whether it belongs to the joint boundary point array and storing this information into *Boundary Vertices* VBO. Finally, we construct a *texture buffer object* (TBO) to maintain a large global data store of vertex attributes that will be accessed afterwards by the *Replaced* and *Extra* 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* 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* 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 *Opening* or *Replaced* point.

All the computation process is done using vertex shader programming. Only *Replaced* or *Extra* points are emitted by the corresponding geometry shader. For both *Replaced* and *Extra* vertex shader programs, we have used ”unnormalized” integer texture formats [23] looking up the neighbor indices of each point.

6.4.2 Adding Vertices

At first, a vertex shader program defines if testing vertex is *Opening* or *Middle* point using *Boundary Vertices* VBO. If it belongs to the *Opening* point set then we generate new points using the equations presented in detail in Section 6.2.1. 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 components info stored in the *Replaced* TBOs. The difference from the CPU-based implementation is that we cannot use Algorithm 2 due to its sequential nature. After this step, a geometry shader is used to construct points interpolating rational Bezier arcs given the triplet vertices as control points 6.2.2. Finally, we copy the *Extra* TBO to the generated *Patch Vertices* VBO for the purposes of expanding the blending point cloud to achieve better robustness.

6.4.3 Triangulation

To triangulate the newly constructed unorganized point cloud we adapt a GPU interpolating reconstruction algorithm based on local Delaunay triangulation [13]. Triangulation is undertaken locally around each point, so that points do not depend on each other. Hence, it can be performed in a parallel. We have used a similar algorithm with [13], but we have made the following modifications to best suit the purposes of our application and to allow faster execution and utilization of the state of the art hardware used:

- We do not divide the point cloud into smaller sets or passes, which are independently processed in the GPU, the graphics memory of the new GPUs can easily accommodate all data.

- We have implemented the k-nearest neighbor algorithm in *Create Patch* vertex shader

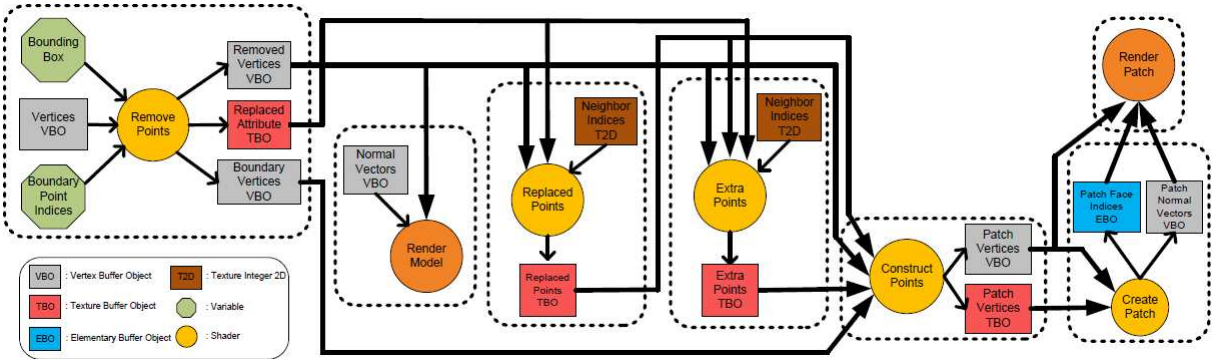

Figure 18: The GPU implementation work flow.

program, thus maximizing parallelization. Normal estimation, rotational projection, angle computation and radial sorting are also computed in the same shader.

- We have used the simple Bubble sort for finding k-NN neighbors and Radial sorting generated neighbors, due to the small number of each point's neighbors.

- 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 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* (*http://www.blender.org/*) software and animate them using manually supplied motion data. Table 1 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 2.6GHz Intel Core Duo with 3GB 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.

Model	Components	Vertices	Triangles
Cow	16	3825	7646
Horse	17	8964	17924
Dilo	37	26214	48924
Human	31	74014	140544

Table 1: Number of components, vertices and triangles of the models used in the experimental evaluation.

7.1 Performance Evaluation

Table 2 gives computation times for extracting skeletons from the above models. The largest computation time corresponds to the principal directions and kernel centroid calculation 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 2, we observe that the average overall skeleton extraction complexity appears to be almost linear on the number of triangles.

It would be meaningless to quantitatively compare our method with other competitive skeleton extraction methods, since most such methods are bound with a suitable decomposition method. In general, our method generates refined skeletons in less than half a minute for dense models (Figures 9, 20, 19). Thus, if our method is used in conjunction with a fast decomposition method, it will be a very efficient overall process.

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

Table 2: Time performance of skeletonization steps measured in seconds.

We have compared the performance of the GPU realization over our optimal sequential CPU implementation [66] for rigid skinning animation of a human knee (see Figure 2)

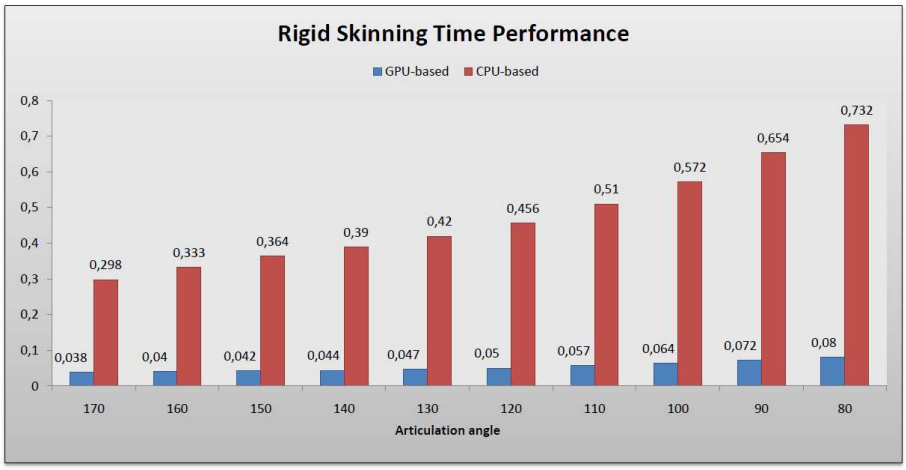

Table 3: Comparison of GPU vs CPU implementation.

from the initial pose to extreme angles (articulation angle going from $170°$ to $80°$). where the parent and child components consist of 891 and 1000 vertices, respectively. Table 3 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.

7.2 Quality Evaluation

Figures 19 and 9 illustrate the qualitative superiority of our refined skeletons and aligned OBBs over the original principal axis algorithm. Figure 20 illustrates the similarity between the default skeletons generated by *Poser* [22] software and skeletons extracted by our method.

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 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 ($N(V_i^a)$),

$$Curv(V) = \max_{1 \leq i \leq k} \{\arccos(N(V) \cdot N(V_i^n))\} \tag{16}$$

Then, the total curvature of a bone component can be defined as the median of curvature of

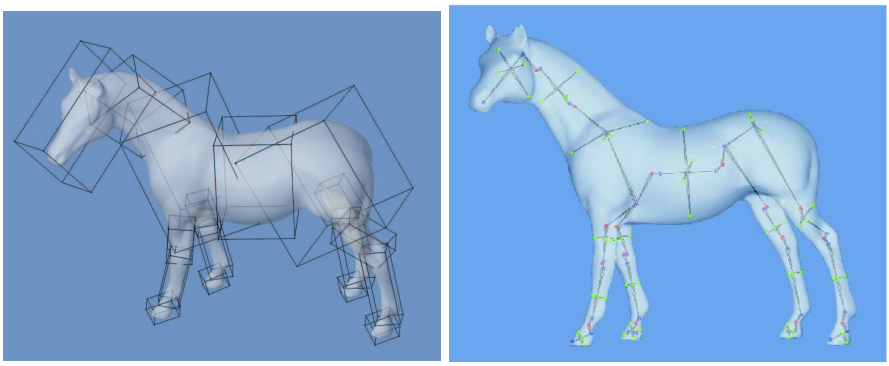

Figure 19: Refined oriented bounding boxes and skeletons of the components of the horse model.

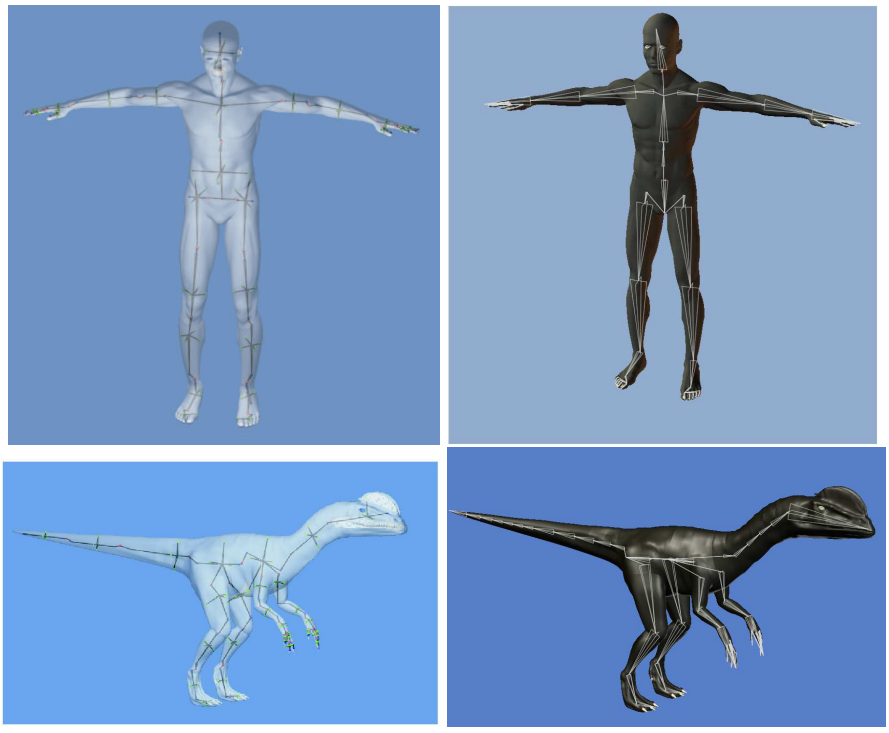

Figure 20: Comparison on human and dino skeletons (left) derived by applying our refinement process and (right) built manually by the creator of the models.

all the vertices connected to this bone. Lower curvature measure numbers means smoother meshes. As the model segment approaches to the mathematically smooth surface, the curvature approaches to zero.

A graph of the curvature variation of the components involved in the human knee joint movement is illustrated in Figure 21. 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. Figure 22 illustrates the volume variation percentage computed as: $100 \cdot \frac{|Vol_{new} - Vol_{ref}|}{Vol_{ref}}$, where Vol_{ref} is the volume of the two components at the reference posture and Vol_{new} is the volume of the two patched components at the corresponding articulation angle. The deformed poses retain the same volume as the reference pose within a small deviation of up to 0.1%.

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 [11] to the skeletons extracted from the original meshes. This format describes each motion frame as a 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.

The main limitation when using geometry shaders to perform point and triangle amplification is that the level of geometry upsampling is limited by the hardware. For instance,

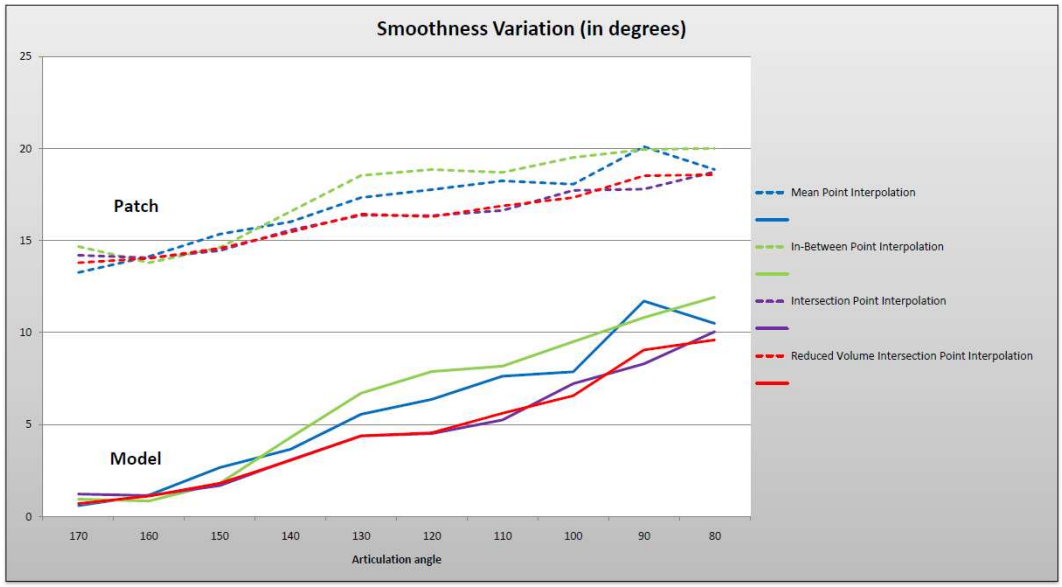

Figure 21: Skin curvature variation (% percentage with regards to the reference posture) during articulation: (top) for the blending patch and (bottom) for the two involved components

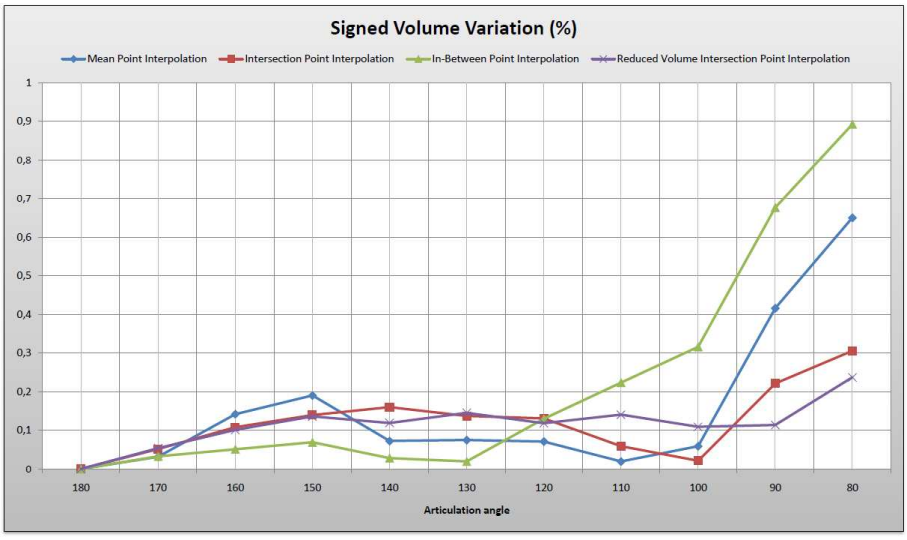

Figure 22: Volume variation of blending patch construction alternatives

only 1024 floating point numbers can be used with the current settings which sets a bound of 256 on the number of the constructed points or triangles.

Figures 23 and 24 show animated poses using our rigid skinning technique. We have applied our technique to all types of joints of several models decomposed manually 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.

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 increase the appropriateness and expressiveness of the produced skeletons. Then, to avoid artifacts that occur in linear blending 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 carried out regarding our improved principal axis skeletonization technique. There is need for further investigating in a more quantitative manner the grouping functions. Further, one could try replacing the angle-weighted algorithm with an optimization approach which will compute the principal axis orientation more accurately.

Finally, since geometry shader was not designed to handle large-scale complex geometric operations, we could benefit from splitting the geometry amplification into multiple passes. A partial solution for this problem would be be the independent execution of the FBCP and RBCP on the GPU, would lead to the triangulation process being divided into

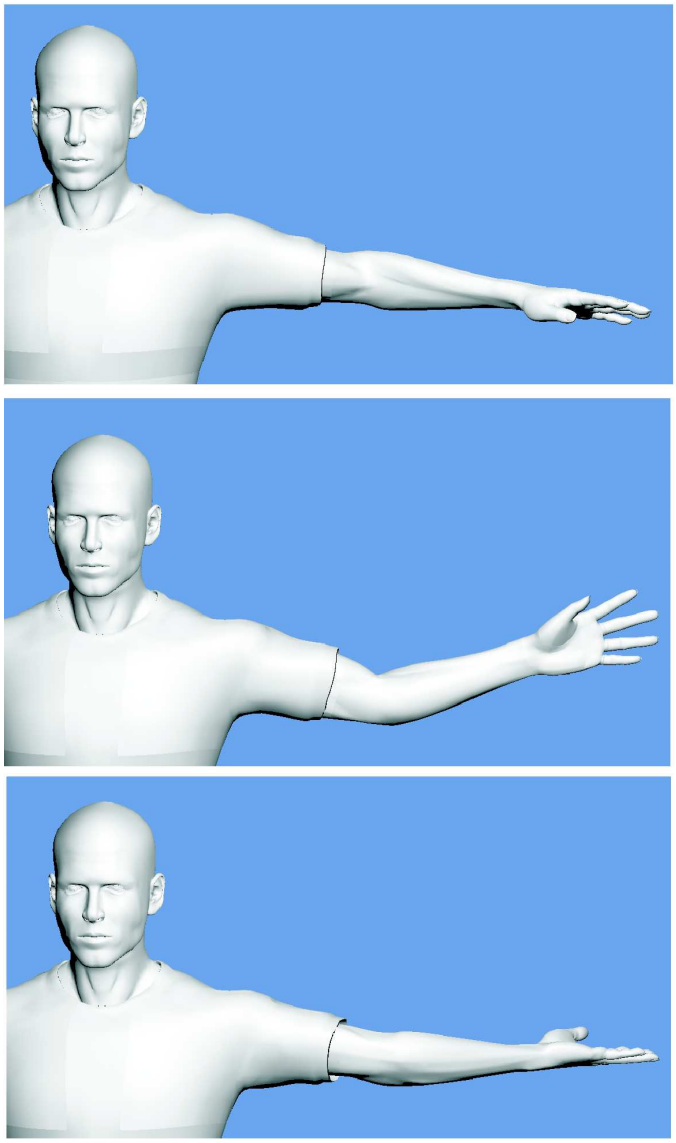

Figure 23: Avoidance of the typical LBS “candy-wrapper” artifact. (Left) is a reference pose, (middle) upper arm is rotated 90^o about its axis, and on (right) upper arm is rotated 180^o about its axis.

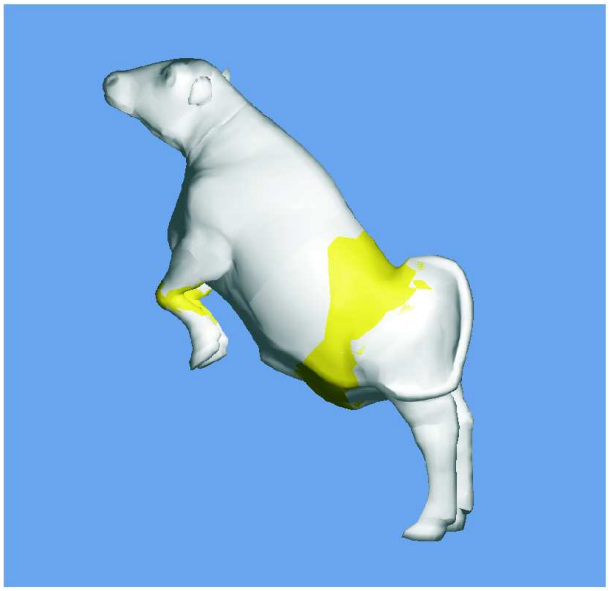

Figure 24: Multiple part animation of the “Cow” model. Patched parts are highlighted in yellow.

two smaller independent subtasks.

References

[1] Marc Alexa. Linear combination of transformations. *ACM Trans. Graph.*, 21(3):380–387, 2002.

[2] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *SMA ’01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, New York, NY, USA, 2001. ACM.

[3] Kasper Amstrup Andersen. Spherical blend skinning on gpu. -, 2007.

[4] Dominique Attali and Jean-Daniel Boissonnat. Approximation of the medial axis. Technical report, INRIA, 2002.

[5] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. In *SMI ’06: Proceedings of the IEEE International*

Conference on Shape Modeling and Applications 2006, page 7, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Marco Attene, Silvia Biasotti, and Michela Spagnuolo. Re-meshing techniques for topological analysis. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 142, Washington, DC, USA, 2001. IEEE Computer Society.

[7] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, New York, NY, USA, 2008. ACM.

[8] Grégoire Aujay, Franck Hétroy, Francis Lazarus, and Christine Depraz. Harmonic skeleton for realistic character animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 151–160, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[9] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72, 2007.

[10] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[11] Biovision. Bvh format.

[12] Harry Blum. A transformation for extracting new descriptors of shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[13] C. Buchart, D. Borro, and A. Amundarain. Gpu local triangulation: an interpolating surface reconstruction algorithm. *Computer Graphics Forum*, 27:807–814(8), 2008.

[14] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 586–593, New York, NY, USA, 2002. ACM.

[15] P. Chaudhuri, G. Papagiannakis, and N. Magnenat-Thalmann. Self adaptive animation based on user perspective. *The Visual Computer*, 24(7–9):525–533, 2008.

[16] Frederic Cordier and Nadia Magnenat-Thalmann. A data-driven approach for real-time clothes simulation. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 257–266, Washington, DC, USA, 2004. IEEE Computer Society.

[17] Nicu D. Cornea, Deborah Silver, Xiaosong Yuan, and Raman Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005.

[18] Tim Culver, John Keyser, and Dinesh Manocha. Exact computation of the medial axis of a polyhedron. *Comput. Aided Geom. Des.*, 21(1):65–98, 2004.

[19] Fernando de Goes, Siome Goldenstein, and Luiz Velh. A hierarchical segmentation of articulated bodies. *Computer Graphics Forum*, 27(5):–, 2008.

[20] Tamal K. Dey and Samrat Goswami. Tight cocone: a water-tight surface reconstructor. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134, New York, NY, USA, 2003. ACM.

[21] Tamal K. Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 143–152, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[22] e frontier. Poser 7, 2006.

[23] OPENGL: Opengl extension registry.

[24] R. Fernardo. Gpu gems: Programming techniques, tips, and tricks for real-time graphics. nvidia corporation.

[25] Sven Forstmann, Jun Ohya, Artus Krohn-Grimberghe, and Ryan McDougall. Deformation styles for spline-based skeletal animation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–150, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[26] Ioannis Fudos and Christoph M. Hoffmann. Constraint-based parametric conics for CAD. *Computer-aided Design*, 28(2):91–100, 1996.

[27] Nikhil Gagvani and Deborah Silver. Animating volumetric models. *Graph. Models*, 63(6):443–458, 2001.

[28] Stefan Aric Gottschalk. *Collision queries using oriented bounding boxes*. PhD thesis, -, 2000. Director-Dinesh Manocha and Director-Ming C. Lin.

[29] Zheng Guo and Kok Cheong Wong. Skinning with deformable chunks. *Computer Graphics Forum*, 24(3):373–381, 2005.

[30] Jim Hejl. Hardware skinning with quaternions. In *Game Programming Gems*, volume 4, pages 487–495. Charles River Media, 2004.

[31] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212, New York, NY, USA, 2001. ACM.

[32] Dae-Eun Hyun, Seung-Hyun Yoon, Jung-Woo Chang, Joon-Kyung Seong, Myung-Soo Kim, and Bert Juttler. Sweep-based human deformation. *The Visual Computer*, 21(8–10):542–550, 2005.

[33] David Jacka, Ashley Reid, Bruce Merry, and James Gain. 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*, pages 177–186, New York, NY, USA, 2007. ACM.

[34] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. Graph.*, 24(3):399–407, 2005.

[35] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, 22(3):954–961, 2003.

[36] Ladislav Kavan, Steven Collins, and Carol O'Sullivan. Automatic linearization of nonlinear skinning. In *2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page to appear. ACM Press, February/March 2009.

[37] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):1–23, 2008.

[38] Ladislav Kavan, Rachel McDonnell, Simon Dobbyn, Jiří Žára, and Carol O'Sullivan. Skinning arbitrary deformations. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 53–60, New York, NY, USA, 2007. ACM.

[39] Ladislav Kavan and Jiri Zara. Spherical blend skinning: A real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 9–16. ACM Press, April 2005.

[40] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: real time large deformation character skinning in hardware. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–159, New York, NY, USA, 2002. ACM.

[41] Jeff Lander. Slashing through realtime character animation. In *Game Developer Magazine*, pages 13–15. -, 1998.

[42] Jeff Lander. Over my dead polygonal body. In *Game Developer Magazine*, pages 17–22. -, 1999.

[43] Caroline Larboulette, Marie-Paule Cani, and Bruno Arnaldi. Dynamic skinning: adding real-time dynamic effects to an existing character animation. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 87–93, New York, NY, USA, 2005. ACM.

[44] Matt Lee. Seven ways to skin a mesh character skinning revisted for modern gpus, 2007.

[45] J. P. Lewis, Matt Cordner, and Nickson Fong. 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*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[46] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 35–42, New York, NY, USA, 2001. ACM.

[47] Jyh-Ming Lien. *Approximate Convex decomposition and its Applications*. PhD thesis, National ChengChi University, 2006.

[48] Pin-Chou Liu, Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, and Ming Ouhyoung. Automatic animation skeleton construction using repulsive force field. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 409, Washington, DC, USA, 2003. IEEE Computer Society.

[49] Lin Lu, Franck Hetroy, Cedric Gerot, and Boris Thibert. Atlas-based character skinning with automatic mesh decomposition. Technical report, INRIA, 2008.

[50] Cherng-Min Ma, Shu-Yen Wan, and Jiann-Der Lee. Three-dimensional topology preserving reduction on the 4-subfields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1594–1605, 2002.

[51] Wan-Chun Ma, Fu-Che Wu, and Ming Ouhyoung. Skeleton extraction of 3d objects with radial basis functions. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 207, Washington, DC, USA, 2003. IEEE Computer Society.

[52] N. Magnenat-Thalmann, F. Cordier, H. Seo, and G. Papagianakis. Modeling of bodies and clothes for virtual environments. *International Conference on Cyberworlds*, -:201–208, Nov. 2004.

[53] Bruce Merry, Patrick Marais, and James Gain. Animation space: A truly linear framework for character animation. *ACM Trans. Graph.*, 25(4):1400–1423, 2006.

[54] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 562–568, New York, NY, USA, 2003. ACM.

[55] Alex Mohr, Luke Tokheim, and Michael Gleicher. Direct manipulation of interactive character skins. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 27–30, New York, NY, USA, 2003. ACM.

[56] R. Laperriere N. Magnenat-Thalmann and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface 1988: Proceedings of the GI 88 Canadian Graphics Conference*, pages 26–33. Canadian Human-Computer Communications Society, 1988.

[57] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[58] Simon Pilgrim, Anthony Steed, and Alberto Aguado. Progressive skinning for character animation. *Comput. Animat. Virtual Worlds*, 18(4-5):473–481, 2007.

[59] Taehyun Rhee, J.P. Lewis, and Ulrich Neumann. Real-time weighted pose-space deformation on the gpu. *Computer Graphics Forum*, 25(3):439–448, 2006.

[60] S. Schaefer and C. Yuksel. Example-based skeleton extraction. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 153–162, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[61] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27:1539–1556(18), 2008.

[62] Evan C. Sherbrooke, Nicholas M. Patrikalakis, and Erik Brisson. An algorithm for the medial axis transform of 3d polyhedral solids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):44–61, 1996.

[63] Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. Example-based dynamic skinning in real time. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[64] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. Surface coding based on morse theory. *IEEE Comput. Graph. Appl.*, 11(5):66–78, 1991.

[65] Peter-Pike J. Sloan, III Charles F. Rose, and Michael F. Cohen. Shape by example. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143, New York, NY, USA, 2001. ACM.

[66] Andreas Vasilakis and Ioannis Fudos. Skeleton-based rigid skinning for character animation. In *GRAPP*, pages 302–308, 2009.

[67] A. Verroust and F. Lazarus. Extracting skeletal curves from 3d scattered data. *Shape Modeling and Applications, 1999. Proceedings. Shape Modeling International '99. International Conference on*, -:194–201, Mar 1999.

[68] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. Elastic secondary deformations by vector field integration. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[69] Lawson Wade and Richard E Parent. Automated generation of control skeletons for use in animation. *Visual Computer*, 18(2):97–110, 2002.

[70] Robert Y. Wang, Kari Pulli, and Jovan Popović. Real-time enveloping with rotational regression. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 73, New York, NY, USA, 2007. ACM.

[71] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138, New York, NY, USA, 2002. ACM.

[72] Jason Weber. Run-time skin deformation. Game Developers Conference, 2000.

[73] Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. Context-aware skeletal shape deformation. *Computer Graphics Forum (Proceedings of Eurographics)*, 26(3):265–274, 2007.

[74] GPGPU with OpenGL and VisualWorks. http://www.cincomsmalltalk.com/userblogs/mls/blogview?entry=3412284871.

[75] Han-Bing Yan, Shimin Hu, Ralph R. Martin, and Yong-Liang Yang. Shape deformation using a skeleton to drive simplex transformations. *IEEE Transactions on Visualization and Computer Graphics*, 14(3):693–706, 2008.

[76] Xiaosong Yang, Arun Somasekharan, and Jian J. Zhang. Curve skeleton skinning for human and creature characters: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):281–292, 2006.

[77] Seung-Hyun Yoon and Myung-Soo Kim. Sweep-based freeform deformations. *Computer Graphics Forum*, 25(3):487–496, 2006.

[78] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. Skeleton-based variational mesh deformations. *Computer Graphics Forum*, 26(3):255–264, 2007.