Efficient Collision Detection for Spherical Blend Skinning

Ladislav Kavan*
CTU in Prague / Trinity College Dublin

Carol O'Sullivan
Trinity College Dublin

Jiří Žára
CTU in Prague

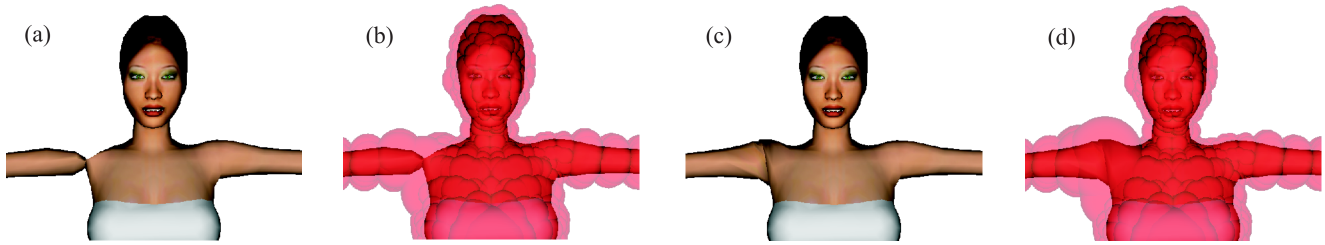

Figure 1: (a) shoulder twist deformed by linear blend skinning produces the candy-wrapper artifact, (b) bounding spheres for linear blend skinning refitted by [Kavan and Zara 2005a], (c) the same posture deformed by spherical blend skinning [Kavan and Zara 2005b], (d) bounding spheres for spherical blend skinning refitted using the algorithm introduced in this paper. Efficient refitting of bounding spheres is a crucial component of our fast collision detection algorithm.

Abstract

Recently, two algorithms improving the real-time simulation of articulated models in virtual environments have been published: 1) fast collision detection for linear blend skinning and 2) spherical blend skinning. Both linear and spherical blending solve the skinning problem of a skeletally controlled 3D model (e.g., an avatar), but only spherical blending avoids artifacts such as the candy-wrapper. However, to date, fast collision detection has been limited to linear blending. This paper describes how to perform collision detection for models skinned with the more sophisticated spherical method. As a result, both high-quality skinning and fast and exact collision detection can be achieved – there is no longer any need for a trade-off. The generalization from linear to spherical blending involves the construction of rotation bounds, derived using a quaternion representation. The resulting algorithm is simple to implement and fast enough for real-time virtual reality applications.

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

Keywords: collision detection, on-demand refitting, sphere refitting, SBS, spherical blending

1 Introduction

Collision detection (CD) is a challenging problem, mainly because of its high computational complexity for typical models. Collision queries are necessary in order to resolve interactions between a virtual character and its environment, as well as among several virtual characters themselves. Since a fast response time is essential for real-time virtual reality applications, most current systems perform CD only with a considerably simplified geometry (e.g., a character replaced by an articulated structure of boxes). This, of course, provides only a rough estimate of actual collisions, making some artifacts visible (such as interpenetration or bouncing before contact). In many situations a precise (mesh-exact) CD is much more desirable, and in certain cases is unavoidable, e.g., in medical applications or virtual prototyping.

Recently, a CD algorithm has been developed, which offers exact and fast CD for 3D models deformed by linear blend skinning [Kavan and Zara 2005a]. Unfortunately, linear blend skinning (also known as skeleton subspace deformation, vertex blending or enveloping) is infamous for its failures, such as the candy-wrapper (shoulder-twist) artifact, see Figure 1(a). This problem can be solved by spherical blend skinning [Kavan and Zara 2005b]: an algorithm not only useful for deforming virtual creatures and avatars, but also for cloth and other objects. Spherical blending works by blending rotations (represented by quaternions) instead of blending vertex positions as in linear blend skinning. The result of spherical blending applied to the same data can be seen in Figure 1(c).

In a practical application, however, we need both high-quality skinning and fast CD simultaneously. Unfortunately, the previous CD algorithm for skeletally deformable objects substantially exploits the linearity of linear blending. Specifically, it relies on the fact that linear blending interpolates always within the convex hull of the input vertices. Since this condition is not true in spherical blending, it is not possible to use the previous collision detection algorithm – bounding spheres constructed by the previous algorithm no longer enclose the skin. This is because spherical blending does not collapse the skin during deformation, as does linear blending.

In this paper we show that, using bounds on the unit quaternion sphere, it is possible to achieve fast CD also for spherical blending. Although it is of course a little more difficult than in the linear case, the resulting algorithm is almost as easy to implement and the computational complexity of the presented algorithm is almost the same as that of the linear version.

The proposed CD algorithm is based on a bounding-volume hierarchy (BVH). This concept has been proven to be very efficient for both rigid and deformable objects, even though it has its limitations, e.g., efficient self-collision detection requires additional treatment [Volino and Magnenat Thalmann 1995]. Concerning deformable objects, it is essential to find a way to refit the bounding volumes

*e-mail: kavanl1@fel.cvut.cz

when the object deforms. It has been shown recently [Larsson and Akenine-Moller 2003; James and Pai 2004; Kavan and Zara 2005a] that one of the most efficient approaches is on-demand (lazy) refitting, which refits the spheres only when they are required by the CD algorithm. Actually, the main difference between the abovementioned methods is in the refitting operation. The reason is that the refitting operation must exploit the properties of the specific deformation model, so that it can work efficiently.

The main contribution of this paper is a procedure for refitting of bounding spheres for spherical blend skinning with sublinear time complexity (with respect to the number of vertices). This refitting operation is actually an extension of the previous refitting for linear blending, because spherical blending decomposes to a linear and rotational component. However, the resulting algorithm is almost as fast as in the case of linear blending, and thus provides a way of exact and efficient CD for 3D models deformed by spherical blending.

2 Related Work

For a survey of CD methods, see [Ericson 2004; Jiménez et al. 2001]. Teschner et al. present a survey specialized on deformable CD [2004]. In this paper, we focus on CD algorithms based on a BVH. Common bounding volumes are: spheres [Quinlan 1994; Guibas et al. 2002], AABBs [van den Bergen 1997], *k*-DOPs [Klosowski et al. 1998], and OBBs [Gottschalk et al. 1996]. Other interesting BVH are QuOSPO-trees [He 1999], BoxTrees [Zachmann 2002] and sphere-swept BVHs [Larsen et al. 1999].

A restriction of most CD algorithms (including ours) is that they consider collisions only at discrete time intervals. It means that some collisions may be missed, especially for small, fast moving objects. This shortcoming has been addressed by proposing so-called *continuous* CD algorithms. Based on a simplified motion model, they find a first time of contact for a given period of time [Redon et al. 2002]. Continuous CD algorithms have been proposed also for articulated models [Redon et al. 2004], but only articulated objects composed of rigid parts were considered (such as robots).

In this paper, we focus on spherical blend skinning and exact CD algorithms based on a BVH, specifically hierarchies of bounding spheres. Spheres were first used for rigid body CD in [Quinlan 1994]. The extension of BVH-based CD for deformable objects was presented first for AABBs by van den Bergen [1997] and later improved by Larsson and Akenine-Moller [2001]. The basic idea of those algorithms is to refit all the bounding volumes during a bottom-up traversal of the tree. A similar algorithm for spheres is presented in [Brown et al. 2001]. Bottom-up refitting is considerably faster than rebuilding the complete BVH, but it obviously wastes time, because not all bounding volumes are necessary for subsequent CD query.

Much more efficient is an on-demand refitting, which recomputes bounding volumes only when required by the CD algorithm. This was first applied for *k*-DOPs and linear morphing by Larsson and Akenine-Moller [2003] and improved later by Klug and Alexa [2004]. The on-demand refitting procedure must be very fast to evaluate – it cannot work by considering the actual vertex displacements (which would be slower than the bottom-up refitting). The refitting should work in time sublinear with respect to the number of vertices, which can only be done by exploiting the properties of the actual deformation model. This is also the main drawback of this approach: it does not work for general deformations – each deformation model needs a special refitting procedure. Such refitting of spheres is presented in [James and Pai 2004] for reduced deformation model and in [Kavan and Zara 2005a] for linear blend skinning. Another collision detection method specialized for skeletally deformable 3D models can be found in [Heim et al. 2004], but their refitting procedure is limited only to a single layer of bounding volumes (no hierarchy), which is restrictive especially for detailed 3D models.

Even though specialized refitting operations ensure very efficient CD, alternative CD algorithms have also been proposed for deformable models, which are not based on a BVH. Instead, they make use of spatial hashing [Teschner et al. 2003], image-space techniques [Heidelberger et al. 2004; Govindaraju et al. 2003], or chromatic decomposition [Govindaraju et al. 2005]. The big advantage of these methods is their generality, i.e., they do not depend on any specific deformation model. Sometimes this is necessary, for example when the deformations are computed during a complex run-time simulation. On the other hand, making no assumptions about the deformation model restricts the time complexity to be at least linear, i.e., disables sublinear time complexity. The reason is obvious: if we allow arbitrary deformations, we must check the displacements of all vertices at runtime. This is especially painful if the deformations are computed on another processor. Therefore, we find it advantageous to apply on-demand refitting whenever possible, not only because of the sublinear execution time, but also because it allows CD to be executed in parallel with the computation of vertex displacements (typically done on a GPU).

As an alternative to spherical blending, it would be possible to consider a different skinning method which does not produce artifacts. For example, [Mohr and Gleicher 2003] reduce the artifacts of linear blending by adding auxiliary joints and recomputing the vertex weights using examples. Even though in this case we could apply the CD solution for linear blending [Kavan and Zara 2005a], there would be associated drawbacks. First, the runtime complexity of Kavan and Zara's algorithm depends on the number of joints, thus addition of auxiliary joints increases the time complexity. In fact, the more joints we add, the better approximation of spherical blending we obtain. Second, Mohr and Gleicher's approach [2003] requires example skins, whose production can be costly. Spherical blend skinning, on the other hand, uses the same input data as linear blending – no extra work is necessary.

A skinning similar to spherical blending is described by [Hejl 2004]. This is actually a simpler algorithm which works correctly only if vertices are influenced solely by neighbouring bones, i.e., bones that share a common joint, an assumption that is only valid for some 3D models. The collision detection method described in this paper can be directly applied for Hejl's skinning (no modifications are necessary as spherical blending is more general).

Other advanced skinning techniques have been proposed recently, allowing realistic simulation of muscle bulges and other effects. Even though delivering high-quality skin deformations, they are usually much more complicated (and slower) than linear or spherical blend skinning. The existence of efficient sphere refitting operation for such deformation models is therefore questionable.

Our Contribution: In this paper, we present a novel CD algorithm especially designed for spherical blend skinning. At its core is an efficient sphere refitting operation, similar in spirit to Kavan and Zara's [2005a]. However, since spherical blending works with rotations, the generalization from this previous linear algorithm is not trivial. This is achieved by the construction of bounds on the unit quaternion sphere. Despite the fact that the derivation and justification of the sphere refitting is not straightforward, the resulting algorithm is simple to implement and fast to execute. We demonstrate this on several practical examples of character animation.

Conventions: We denote the d-dimensional Euclidean space as R^d

and we write its elements (vectors) in bold. The zero vector is denoted as $\mathbf{0}$. The vector $\mathbf{v} \in R^d$ consists of components $(v_1, ..., v_d)^T$. In order to simplify notation, we introduce the set of all possible convex weights:

$$W_d = \{\mathbf{x} \in R^d : x_1 \geq 0, \dots, x_d \geq 0, \sum_{i=1}^{d} x_i = 1\}$$

The convex hull of set $A \subseteq R^d$, i.e., the smallest convex set containing A, is denoted as $CH(A)$. We denote the dot product of two vectors $\mathbf{v}_1 \in R^d, \mathbf{v}_2 \in R^d$ as $(\mathbf{v}_1, \mathbf{v}_2)$ and the norm $\|\mathbf{v}_1\|$ as a shortcut for $\sqrt{(\mathbf{v}_1, \mathbf{v}_1)}$. The 3-dimensional sphere surface of unit quaternions is denoted as $S_3 = \{\mathbf{x} \in R^4 : \|\mathbf{x}\| = 1\}$ (note that we identify quaternions with R^4 vectors).

3 Skinning and Collision Detection

This section recapitulates the spherical blend skinning algorithm and the basics of collision detection based on a BVH, as described in [Kavan and Zara 2005b; Kavan and Zara 2005a]. The input of both linear and spherical blend skinning consists of a skin, a skeleton, and vertex weights for each vertex-joint pair. The skin is just a 3D triangular mesh and the skeleton is a rooted tree. The nodes of this tree represent joints and the edges can be interpreted as bones. The vertex weights describe the amount of influence of individual joints on the position of the vertex in the deformed skin. Note that the input for linear blend skinning is the same, i.e., identical input files are used for both linear and spherical skinning.

Let us assume that the joints are stored in an array, with every joint referenced by an integer number, starting from zero. In the reference posture, each joint has an associated local coordinate system. During animation, the joints rotate – we do not consider translation or scale. The transformation from the reference coordinate system of joint j to its coordinate system in the animated posture can be expressed as a rigid transformation matrix. We can compute this matrix as a multiplication of successive joint transformations. We denote this matrix as C_j (like the "complete" transformation matrix).

We assume that vertex $\mathbf{v}$ is attached to joints $j_1, \dots, j_n$ with weights $\mathbf{w} = (w_1, \dots, w_n)$ (The indices $j_1, \dots, j_n$ are integers referring to joints that influence a given vertex – they can be interpreted as indices into the array of joints.) In order to have properly defined blending, the weights must be convex: $\mathbf{w} \in W_n$. The set of joints $\{j_1, \dots, j_n\}$ is called the *joint-set* influencing the vertex $\mathbf{v}$ and is denoted as $J(\mathbf{v})$. The i-th component of vector $\mathbf{w}$, w_i, represents the amount of influence of joint j_i.

Let us denote the unit quaternions corresponding to the rotation parts of matrices $C_{j_1}, \dots, C_{j_n}$ as $\mathbf{q}_1, \dots, \mathbf{q}_n$ (the conversion procedure can be found in [Eberly 2001]). During skin deformation, quaternions are blended using the QLERP method (Quaternion Linear Interpolation), computing $\frac{w_1\mathbf{q}_1 + \dots + w_n\mathbf{q}_n}{\|w_1\mathbf{q}_1 + \dots + w_n\mathbf{q}_n\|}$ and converting the result to a rotation matrix $Q_\mathbf{w}$. The vertex position in the deformed mesh is then computed as:

$$\mathbf{v}' = Q_\mathbf{w}(\mathbf{v} - \mathbf{r}_c) + \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c \tag{1}$$

where $\mathbf{r}_c$ is the *rotation center*, defined as the point whose transformations by matrices $C_{j_1}, \dots, C_{j_n}$ are as close as possible. For the purpose of on-demand refitting, it is sufficient to take the $\mathbf{r}_c$ as computed by spherical blending; for details please see [Kavan and Zara 2005b].

The interpretation of the spherical blending Equation (1) is as follows: we call the first term $Q_\mathbf{w}(\mathbf{v} - \mathbf{r}_c)$ the *spherical* part and the second one $\sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c$ the *linear* part. The linear part is actually nothing but a linear blending applied to the rotation center $\mathbf{r}_c$. This part becomes important if the set of influencing bones is not simple, i.e., if it contains more than two non-neighbouring joints. In this case the translation between the joints must be interpolated, which is exactly the role of the linear part of Equation (1). The spherical part, on the other hand, interpolates rotation of the influencing joints. The linear quaternion blending (QLERP) used to produce $Q_\mathbf{w}$ is the reason why spherical blend skinning does not exhibit artifacts (which would appear if we blended matrices instead of quaternions, just like in linear blend skinning). More details can be found in [Kavan and Zara 2005b].

For on-demand refitting, it is important that the final vertex position $\mathbf{v}'$ will no longer lie in $CH(C_{j_1}\mathbf{v}, \dots, C_{j_n}\mathbf{v})$, which was true for linear blending (and subsequently exploited in the sphere refitting for linear blend skinning). In spherical blending, $\mathbf{v}'$ lies instead on a helical surface A, given as $A = \left\{Q_\mathbf{w}(\mathbf{v} - \mathbf{r}_c) + \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c : \mathbf{w} \in W_n\right\}$. In simple situations, the set A becomes a spherical arc or surface. The main problem of the on-demand refitting for spherical blending is to find an efficient bound for a subset of set A. This is obviously not as simple as in the case of linear blending, and none of the previous refitting methods can be applied for this task.

For our collision detection algorithm we need a tree of bounding spheres. We build this tree for the reference position of the 3D model, using the same algorithm as in [Kavan and Zara 2005a], thus enabling a fair comparison of the results. The sphere tree is built by a top-bottom algorithm, which starts by first bounding the whole 3D model within one sphere. In the next steps, we split the geometry into two parts and proceed recursively, computing the minimal enclosing sphere with Gaertner's algorithm [1999]. When the complete binary tree is constructed, some nodes are pruned using the same heuristics as in [Kavan and Zara 2005a]. Generally, we prune a node if its bounding sphere has a similar size to the bounding sphere of the parent node, thus discarding less useful nodes and obtaining a general order tree instead of a binary one.

For the actual collision detection, we apply the standard algorithm based on a BVH. As input we have two 3D models, each equipped with a BVH. The task is to either find all colliding triangles, or find at least one (if any). The algorithm proceeds as follows: first, we test the root spheres for intersection. If disjoint, we end up with no collisions. If intersecting, we move to the next level in one of the hierarchies and continue recursively. In the final level we perform intersection tests on individual triangles. The only modification we make to this standard algorithm is the sphere refitting, which is inserted just before the intersection test. In this way we ensure that we work with correct bounding spheres, even though the 3D model has been deformed.

4 Efficient Refitting of Spheres

The crucial part of a BVH-based CD algorithm for deformable objects is the refitting operation. As mentioned in the introduction, the refitting procedure must work in an on-demand way, so that it can be executed on spheres in any order. Moreover, the refitting algorithm must be sublinear with respect to the number of vertices. To achieve this, we exploit the fact that the animation is only controlled by the joint transformation matrices C_j (all other data are constant). It is therefore possible to base the refitting procedure solely on some precomputed information and on the actual joint transformations. This is of course much more efficient than driving the refitting by vertex displacements, because the number of joints is usually orders of magnitude smaller than the number of vertices.

4.1 Problem Decomposition

We assume that the skin of the input 3D model consists of triangles. Since triangles and bounding spheres are always convex, it is sufficient to enclose only the vertices of a triangle in order to bound the whole triangle. During sphere tree construction, we ensure that the bounding spheres in the children nodes enclose the same geometry as the bounding sphere in the parent node. Moreover, we require that each triangle is bounded by a single sphere (it is not sufficient that a triangle is covered only by a union of spheres). Let us assume that we are refitting a sphere S with center $\mathbf{p}$ and radius r, which encloses some set of triangles. We denote all vertices of those triangles as $\mathbf{v}_1, \ldots, \mathbf{v}_t$. For simplicity of notation, we first assume that all these vertices are influenced by the same joint-set $J = \{j_1, \ldots, j_n\}$, that is $J = J(\mathbf{v}_1) = \ldots J(\mathbf{v}_t)$. The task is to compute a new sphere which will enclose vertices $\mathbf{v}'_1, \ldots, \mathbf{v}'_t$, computed by Equation (1).

The trick to obtaining an algorithm sublinear in t is to replace the set of vertices $\{\mathbf{v}_1, \ldots \mathbf{v}_t\}$ by the bounding sphere S, which is correct because $\{\mathbf{v}_1, \ldots, \mathbf{v}_t\} \subseteq S$. That is, instead of bounding $\mathbf{v}'_1, \ldots, \mathbf{v}'_t$ we could bound the set

$$\bigcup_{\mathbf{w} \in W_n} \left(Q_{\mathbf{w}}(S - \mathbf{r}_c) + \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c \right) \tag{2}$$

Considering the whole sphere S of points instead of only one point is not a big problem, as shown in Section 4.3. A more serious problem is that the set (2) is very conservative, because it disregards the actual vertex weights of $\mathbf{v}_1, \ldots \mathbf{v}_t$. This means that (2) actually bounds all skin deformations that could be ever produced by spherical blending for the given posture. Obviously, this would produce a very loose bounding sphere, useless for collision detection. It is therefore necessary to take the actual vertex weights into account. This is done by computing low and high bounds of the vertex weights for all joints in our joint-set J. For every joint $j \in J$ we denote the weight bound as $\langle l_j, h_j \rangle$, making sure that the weight of each vertex $\mathbf{v}_1, \ldots \mathbf{v}_t$ with respect to joint j is within this interval. Using the weight bounds l_i and h_i, we define the set W'_n of limited convex combinations:

$$W'_n = \{\mathbf{w} \in R^n : l_i \leq w_i \leq h_i,\ i = 1, \ldots n,\ \sum_{i=1}^{n} w_i = 1\}$$

and apply this set in (2) instead of W_n. The final set to be bounded by the refitted sphere is therefore

$$\bigcup_{\mathbf{w} \in W'_n} \left(Q_{\mathbf{w}}(S - \mathbf{r}_c) + \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c \right) \tag{3}$$

The set W'_n has a nice geometric interpretation: it is the intersection of an n-dimensional box $\{\mathbf{w} \in R^n : l_i \leq w_i \leq h_i,\ i = 1, \ldots n\}$ with a hyperplane $\{\mathbf{w} \in R^n : \sum_{i=1}^{n} w_i = 1\}$. It means that W'_n is a bounded convex set in R^n and can therefore be expressed as a convex hull of m points (because of the equivalency of bounded H-polytopes and V-polytopes [Matousek 2002]). We denote these m points as $\mathbf{c}_1, \ldots, \mathbf{c}_m$ and call them *corners* (where m is the number of vertices of the corresponding V-polytope). The corners depend only on vertex weights, which means that they can be precomputed during the sphere tree construction. The expression of set W'_n exploiting corners is as follows:

$$W'_n = CH(\mathbf{c}_1, \ldots, \mathbf{c}_m) = \left\{ \sum_{k=1}^{m} u_k \mathbf{c}_k : \mathbf{u} \in W_m \right\} \tag{4}$$

We derive the bound of set (3) in three steps. In the first step, we bound the linear component $\sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c$ (Section 4.2) and in the second step, we bound the spherical part $Q_{\mathbf{w}}(S - \mathbf{r}_c)$ (Section 4.3). Finally, both of these bounds are simply added together (Section 4.4) to create the resulting refitted sphere. The resulting algorithm is presented in Section 4.5. If the reader is not interested in the derivation and justification of our approach, he or she is encouraged to skip the following sections and proceed directly to Section 4.5. Even though the material in Sections 4.2, 4.3, and 4.4 is essential to show the validity of our algorithm, it is not necessary for a practical implementation.

4.2 Bounding the Linear Part

Unlike the bound of the spherical part, the bound of the linear part of set (3) can be done in a way similar to [Kavan and Zara 2005a] (the situation is actually more simple here, because we are bounding points instead of spheres as in the previous article). This is because the linear part is actually nothing but a linear blending applied to the rotation center $\mathbf{r}_c$, which is a point computed by the spherical blend skinning algorithm for a given skeleton posture: it is independent of the vertex weights.

Thanks to the expression of W'_n using corners (Equation (4)), we can rewrite the linear part of set (3) as

$$\left\{ \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c : \mathbf{w} \in W'_n \right\} = \left\{ \sum_{i=1}^{n} \left(\sum_{k=1}^{m} u_k c_{ki} \right) C_{j_i} \mathbf{r}_c : \mathbf{u} \in W_m \right\}$$

where c_{ki} denotes i-th component of vector $\mathbf{c}_k$. In the latter term, we can swap the sums, because

$$\sum_{i=1}^{n} \left(\sum_{k=1}^{m} u_k c_{ki} \right) C_{j_i} \mathbf{r}_c = \sum_{k=1}^{m} u_k \left(\sum_{i=1}^{n} c_{ki} C_{j_i} \mathbf{r}_c \right)$$

We denote the transformations of the rotation center as:

$$\mathbf{r}'_k = \sum_{i=1}^{n} c_{ki} C_{j_i} \mathbf{r}_c, \quad k = 1, \ldots, m \tag{5}$$

which is correct because $\mathbf{c}_k \in W'_n$ and thus $\sum_{i=1}^{n} c_{ki} = 1$. Equation (5) is actually nothing but linear blending applied to $\mathbf{r}_c$ with weight vector $\mathbf{c}_k$. If we put the equations together, we can write the resulting bound of the linear part as

$$\left\{ \sum_{i=1}^{n} w_i C_{j_i} \mathbf{r}_c : \mathbf{w} \in W'_n \right\} = \left\{ \sum_{k=1}^{m} u_k \mathbf{r}'_k : \mathbf{u} \in W_m \right\} = CH(\mathbf{r}'_1, \ldots, \mathbf{r}'_m)$$

To conclude: the bound of the linear part is just a convex hull of several 3D points. These points are given by the precomputed corners and Equation (5). Please note that, although we use the concept of the convex hull in our derivation, the convex hull is actually never computed in our algorithm (bounding spheres are used instead, see Section 4.5).

4.3 Bounding the Spherical Part

Bounding the spherical part of set (3) is a little bit more tricky, because we must deal with the linear quaternion interpolation (QLERP) hidden in $Q_{\mathbf{w}}$. Recall that we are refitting sphere S with center $\mathbf{p}$ and radius r, expressed in the reference position. First, we replace the sphere S by its center $\mathbf{p}$:

$$\bigcup_{\mathbf{w} \in W'_n} Q_{\mathbf{w}}(S - \mathbf{r}_c) = \left\{ Q_{\mathbf{w}}(\mathbf{p} - \mathbf{r}_c) : \mathbf{w} \in W'_n \right\} \oplus \left\{ \mathbf{x} \in R^3 : \|x\| \leq r \right\}$$

where r is the radius of sphere S and $\oplus$ denotes the Minkowski sum. However, the Minkowski sum in the previous equation is actually nothing but a convolution of set $\{Q_{\mathbf{w}}(\mathbf{p}-\mathbf{r}_c) : \mathbf{w} \in W_n'\}$ with a zero center sphere of radius r. In the following, we derive the bounding sphere of $\{Q_{\mathbf{w}}(\mathbf{p}-\mathbf{r}_c) : \mathbf{w} \in W_n'\}$. At the end, we account for the Minkowski sum (convolution) by simply increasing the radius of the resulting sphere by r (line 16 of Algorithm 1). The rest of this section is organized as follows: first, we compute bound on the set of rotations $Q_{\mathbf{w}}$ and express it as a subset of unit quaternion sphere S_3. Second, we apply all rotations from this set to rotate the vector $\mathbf{p}-\mathbf{r}_c$. The result is some subset of R^3, which is enclosed by a final bounding sphere of the spherical part. The reader should not get confused by the fact that the bounds in both steps will have the same shape (a spherical cap, defined below). The difference is that the bounding of rotations occurs in R^4, whereas the bounding of rotated vectors takes place in R^3.

Recall that we denoted the quaternions corresponding to the rotational parts of matrices $C_{j_1}, \ldots, C_{j_n}$ by $\mathbf{q}_1, \ldots, \mathbf{q}_n$. Then $Q_{\mathbf{w}}$ is a rotation matrix given by the quaternion $w_1\mathbf{q}_1 + \ldots + w_n\mathbf{q}_n$. This is correct, because every non-zero quaternion determines a unique 3D rotation (even though not vice-versa). We proceed by constructing a bound of all rotations given by the set of quaternions $\{w_1\mathbf{q}_1 + \ldots + w_n\mathbf{q}_n : \mathbf{w} \in W_n'\}$. We exploit the fact that QLERP applies linear combinations of quaternions, and that quaternions can be interpreted as R^4 vectors. The first step will be therefore similar to that described in Section 4.2, just in R^4 instead of R^3. Using the same corners $\mathbf{c}_1, \ldots, \mathbf{c}_m$ as in Section 4.2, we compute another set of quaternions $\mathbf{q}_1', \ldots, \mathbf{q}_m'$ given by

$$\mathbf{q}_k' = \sum_{i=1}^{n} c_{ki}\mathbf{q}_i, \quad k = 1, \ldots, m$$

which satisfy the property

$$\left\{w_1\mathbf{q}_1 + \ldots + w_n\mathbf{q}_n : \mathbf{w} \in W_n'\right\} = \left\{u_1\mathbf{q}_1' + \ldots + u_m\mathbf{q}_m' : \mathbf{u} \in W_m\right\}$$

This can also be proven by swapping sums as in Section 4.2. It is therefore sufficient to construct a bound for rotations corresponding to quaternions from $CH(\mathbf{q}_1', \ldots, \mathbf{q}_m')$.

Unfortunately, this cannot be done simply by a convex bounding volume, as in Section 4.2, because in this case, we are working in a non-linear space (spherical surface). Linear bounds, such as convex hull, obviously cannot work in curved spaces. For example, it is not correct to just rotate vector $\mathbf{p}-\mathbf{r}_c$ by quaternions $\mathbf{q}_1', \ldots, \mathbf{q}_m'$ and bound the results by a 3D enclosing sphere. In order to obtain a valid bounding volume, we have to appropriately bound the set of rotations given by $CH(\mathbf{q}_1', \ldots, \mathbf{q}_m')$.

We have chosen only a simple bound of rotation sets: a spherical cap on S_3 (the sphere of all unit quaternions). Generally, we define a *cap* in any dimension as a non-empty intersection of a sphere surface with a halfspace. In the following, we will also need another definition of cap, given by the center $\mathbf{c}_s$ of the sphere, point $\mathbf{a}_s$ on the sphere's surface (the cap's apex) and an angle $\alpha_s \in \langle 0, \pi \rangle$. If we denote the radius of the sphere as $r_s = \|\mathbf{a}_s - \mathbf{c}_s\|$, then the cap according to the second definition is expressed as

$$\left\{\mathbf{x} \in R^d : \|\mathbf{x}-\mathbf{c}_s\| = r_s,\ (\mathbf{x}-\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s) \geq r_s^2\cos(\alpha_s)\right\}$$

It is not difficult to prove that both definitions are equivalent, see Lemma 1 in the Appendix. An example of a cap is shown in Figure 2.

The bound of a set of rotations expressed by a cap C on S_3 has a nice geometric interpretation. Let us denote the apex of cap C as $\mathbf{a}_C$ and the angle as α_C (in this case, the center $\mathbf{c}_C = \mathbf{0}$ and radius

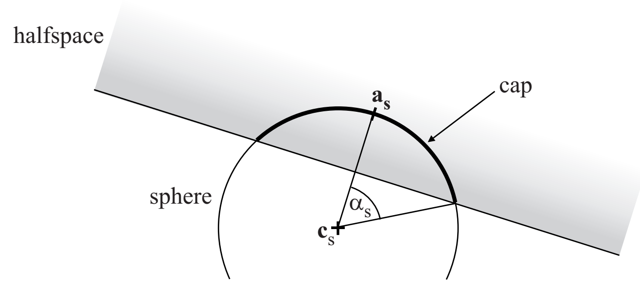

Figure 2: Example of a cap in R^2 with center $\mathbf{c}_s$, apex $\mathbf{a}_s$ and angle α_s. In this case the cap is just a spherical arc.

$r_C = 1$, because S_3 is a zero centered sphere with unit radius). Since we are considering S_3, the apex $\mathbf{a}_C$ is a unit quaternion representing some rotation R_C. Then all rotations represented by cap C can be obtained by composing R_C with a rotation about an arbitrary axis and angle within $\langle 0, 2\alpha_C \rangle$ ($2\alpha_C$ because quaternions work with *half* of the angle of rotation, see for example [Eberly 2001]). If we have the set of rotations bound by cap $C \subseteq S_3$, we can bound our original set $\{Q_{\mathbf{w}}(\mathbf{p}-\mathbf{r}_c) : \mathbf{w} \in W_n'\}$ by $\rho = \{R(\mathbf{p}-\mathbf{r}_c) : R \in C\}$, where C is interpreted as a set of rotations. But the set ρ is nothing but the set of all possible rotations of vector $R_C(\mathbf{p}-\mathbf{r}_c)$ along an arbitrary axis and angle within $\langle 0, 2\alpha_C \rangle$. It means that ρ is nothing but another spherical cap (but now in R^3)! The apex of cap ρ is $R_C(\mathbf{p}-\mathbf{r}_c)$, center is $\mathbf{0}$ and angle $2\alpha_C$.

Since we are constructing a cap on S_3, we normalize our quaternions $\mathbf{q}_1', \ldots, \mathbf{q}_m'$ to unit quaternions: $\mathbf{q}_k'' = \mathbf{q}_k'/\|\mathbf{q}_k'\|,\ k = 1, \ldots, m$. We can work with $\mathbf{q}_k''$ instead of $\mathbf{q}_k'$, because Lemma 2 from the Appendix shows that the sets of rotations corresponding to $CH(\mathbf{q}_1', \ldots, \mathbf{q}_m')$ and $CH(\mathbf{q}_1'', \ldots, \mathbf{q}_m'')$ are the same. What remains is to construct a cap on S_3 containing our quaternions $\mathbf{q}_1'', \ldots, \mathbf{q}_m''$.

In order to construct this cap, we bound $\mathbf{q}_1'', \ldots, \mathbf{q}_m''$ by an enclosing sphere $E \subseteq R^4$ with center $\mathbf{c}_E$ and radius r_E. It would be possible to use again the randomized algorithm [Gaertner 1999] which finds the smallest enclosing sphere, but we found that just an approximate enclosing sphere performs better. The approximate enclosing sphere is computed in the same way as in [James and Pai 2004] (but in R^4 in our case), i.e., by taking the average of $\mathbf{q}_1'', \ldots, \mathbf{q}_m''$ as the center and then determining the smallest possible radius which still gives a correct enclosing sphere.

After computing the enclosing sphere E we are almost done, because $E \cap S_3$ is the desired cap. This is illustrated in Figure 3 and verified in Lemma 3 and Lemma 4 in the Appendix. We can assume that the radius of sphere E satisfies $r_E < 1$: if not, we can simply consider the whole S_3 (of radius 1) which bounds all rotations, as the bounding cap (although it should be noted that this situation never occurred during practical experiments).

Now it is straightforward to derive the resulting cap $C' \subseteq R^3$ such that $\{Q_{\mathbf{w}}(\mathbf{p}-\mathbf{r}_c) : \mathbf{w} \in W_n'\} \subseteq C'$. We denote by Q_c the rotation corresponding to $\mathbf{c}_E$. The apex of the cap C' is then $Q_c(\mathbf{p}-\mathbf{r}_c)$, the center is $\mathbf{0}$ and the angle is

$$\alpha = 2\arccos(d_H),\ d_H = \frac{1 + \|\mathbf{c}_E\|^2 - r_E^2}{2\|\mathbf{c}_E\|} \qquad (6)$$

where d_H denotes the distance from H to $\mathbf{0}$, as computed in Lemma 3 and illustrated in Figure 3. Note that, since the sphere E intersects S_3, it cannot happen that E is strictly inside S_3, i.e., $\|\mathbf{c}_E\| + r_E < 1$ cannot be true. Therefore, $\|\mathbf{c}_E\| + r_E \geq 1$, thus

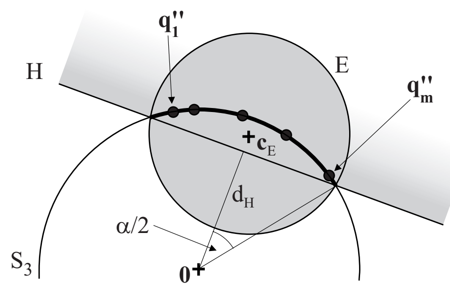

Figure 3: To construct the bounding cap (in this picture just the spherical arc) for quaternions $\mathbf{q}''_1, \ldots, \mathbf{q}''_m$, we first create an enclosing sphere E (not necessarily the smallest one). The cap is then given as $E \cap S_3$, which can be equally expressed as $H \cap S_3$, where H is a halfspace from Lemma 3. A formula for distance d_H from H to $\mathbf{0}$ is also derived in Lemma 3. The distance d_H is used to compute the angle α.

$-r_E^2 \leq -(1-\|\mathbf{c}_E\|)^2 = -1+2\|\mathbf{c}_E\| - \|\mathbf{c}_E\|^2$ from which follows:

$$d_H \leq \frac{1+\|\mathbf{c}_E\|^2 - 1 + 2\|\mathbf{c}_E\| - \|\mathbf{c}_E\|^2}{2\|\mathbf{c}_E\|} = \frac{2\|\mathbf{c}_E\|}{2\|\mathbf{c}_E\|} = 1$$

Since obviously $0 \leq d_H$, the arccos in Equation 6 is well defined and $\alpha \in \langle 0, \pi \rangle$.

The resulting sphere returned for the bound of the spherical part is nothing but a minimal enclosing sphere of cap C'. We denote this minimal enclosing sphere as $F \subseteq R^3$ and we compute it easily: its center is $\cos(\alpha) Q_c(\mathbf{p} - \mathbf{r}_c)$ and radius is $\sin(\alpha)\|Q_c(\mathbf{p}-\mathbf{r}_c)\| = \sin(\alpha)\|\mathbf{p}-\mathbf{r}_c\|$, where α is given by Equation 6. This is proven in Lemma 5 and illustrated in Figure 4.

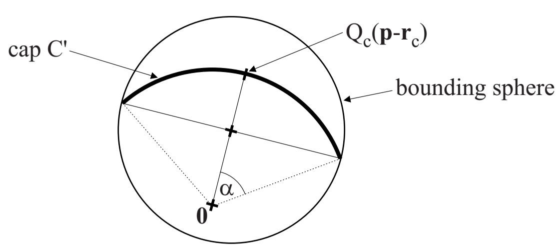

Figure 4: If the cap C' centered in the origin has apex $Q_c(\mathbf{p} - \mathbf{r}_c)$ and angle α, then its minimal enclosing sphere has center $\cos(\alpha) Q_c(\mathbf{p}-\mathbf{r}_c)$ and radius $\sin(\alpha)\|\mathbf{p}-\mathbf{r}_c\|$.

4.4 Putting the Bounds Together

To construct the final bounding sphere of set (3), it remains just to combine the bound of the linear part and the bound of the spherical part. Recall that, in Section 4.2, the bound of the linear part was expressed as $CH(\mathbf{r}'_1, \ldots, \mathbf{r}'_m)$ and, in Section 4.3, the spherical part was bound by sphere F. The bound of both parts can therefore be expressed as $CH(\mathbf{r}'_1 \oplus F, \ldots, \mathbf{r}'_m \oplus F)$, i.e., the final bounding sphere encloses $\mathbf{r}'_1 \oplus F, \ldots, \mathbf{r}'_m \oplus F$. This enclosing sphere is again computed only by a simple approximation (the same as before): the center of the enclosing sphere is set to the average of the centers of $\mathbf{r}'_1 \oplus F, \ldots, \mathbf{r}'_m \oplus F$, and the smallest possible radius is computed in a straightforward way. Note that this enclosing sphere is not the smallest possible enclosing sphere. However, as suggested already by [James and Pai 2004; Kavan and Zara 2005a], it is more efficient for collision detection than computation of the minimal enclosing sphere. The correctness of collision detection is of course not affected by employing bigger-than-necessary spheres.

In the beginning of Section 4, we assumed that all vertices $\mathbf{v}_1, \ldots, \mathbf{v}_t$ of the reference sphere S are assigned to only one joint-set J. If this is not the case, i.e., the vertices $\mathbf{v}_1, \ldots, \mathbf{v}_t$ are influenced by more joint-sets $J_1, \ldots, J_z$, we simply repeat the same algorithm for each of these joint-sets. This way, we obtain spheres $\mathbf{r}'_{1,1} \oplus F_1, \ldots, \mathbf{r}'_{m_1,1} \oplus F_1, \mathbf{r}'_{1,2} \oplus F_2, \ldots, \mathbf{r}'_{m_2,2} \oplus F_2, \ldots, \mathbf{r}'_{1,z} \oplus F_z, \ldots, \mathbf{r}'_{m_z,z} \oplus F_z$ and enclose them by one bounding sphere as before.

4.5 Final Algorithm

This section presents the final sphere refitting algorithm. For simplicity, we write $[\mathbf{c}, r]$ to denote a data structure describing a sphere with center $\mathbf{c}$ and radius r. The symbol c_{ki} on lines (7) and (8) denotes the i-th component of an n-dimensional vector $\mathbf{c}_k$. In list L_2 are actually stored points $\mathbf{q}''_k$, but for convenience, we treat them as spheres with zero radius: $[\mathbf{q}''_l, 0]$.

Algorithm 1: Sphere Refitting for Spherical Blending
Input: $S = [\mathbf{p}, r]$ – sphere to be refitted
$C_1, \ldots, C_N$ – joint transformation matrices
J – list of joint-sets influencing sphere S
$\mathbf{c}_1, \ldots, \mathbf{c}_m$ – corners describing the bound of weights
$\mathbf{r}_c$ – rotation center
Output: sphere S refitted for current skin deformation
SPHEREREFIT(S)
(1) L_1 = empty list
(2) **for** $k = 1$ **to** m
(3) $\mathbf{q}_i$ = MATRIX2QUAT(C_i)
(4) **foreach** joint-set $\{j_1, \ldots, j_n\} \in J$
(5) L_2 = empty list
(6) **for** $k = 1$ **to** m
(7) $\mathbf{r}'_k = \sum_{i=1}^{n} c_{ki} C_{j_i} \mathbf{r}_c$
(8) $\mathbf{q}'_k = \sum_{i=1}^{n} c_{ki} \mathbf{q}_i$
(9) $\mathbf{q}''_k = \mathbf{q}'_k / \|\mathbf{q}'_k\|$
(10) insert sphere $[\mathbf{q}''_l, 0]$ into list L_2
(11) $[\mathbf{c}_E, r_E]$ = BOUNDINGSPHERE(L_2)
(12) Q_c = QUAT2MATRIX($\mathbf{c}_E / \|\mathbf{c}_E\|$)
(13) $\alpha = 2\arccos(\frac{1+\|\mathbf{c}_E\|^2 - r_E^2}{2\|\mathbf{c}_E\|})$
(14) **for** $k = 1$ **to** m
(15) insert sphere $[\mathbf{r}'_k + \cos(\alpha) Q_c(\mathbf{p}-\mathbf{r}_c), \sin(\alpha)\|\mathbf{p} - \mathbf{r}_c\|]$ into list L_1
(16) **return** BOUNDINGSPHERE(L_1) + $[\mathbf{0}, r]$

The addition of $[\mathbf{0}, r]$ on line (16) simply inflates the radius of the resulting sphere by r. The sphere refitting algorithm uses three subroutines: MATRIX2QUAT, QUAT2MATRIX, and BOUNDINGSPHERE. The first two convert between quaternion and matrix representation (note that this would not be necessary if our application worked internally with quaternions instead of matrices). These routines are usually a standard part of mathematical libraries [Eberly 2001]. Concerning BOUNDINGSPHERE, we use only the simple approximate algorithm for the bounding sphere of spheres mentioned before: setting center as the average of centers of input spheres and computing the minimal possible radius straigtforwardly.

Level	Reference			Linear Blending			Spherical Blending			Optimal		
1	34.55	34.55	34.55	72.17	72.17	72.17	79.55	79.55	79.55	34.41	34.41	34.41
2	18.11	18.86	19.74	26.17	31.58	34.48	26.23	33.95	37.84	17.97	22.05	26.57
3	3.11	7.38	10.47	3.11	8.40	14.89	3.11	8.89	17.33	3.11	7.39	10.13
4	0.98	3.68	6.23	0.98	4.03	7.96	0.98	4.12	9.23	0.98	3.69	6.47
5	0.24	1.69	3.69	0.24	1.80	5.05	0.24	1.83	5.28	0.24	1.70	4.33
6	0.20	0.91	2.84	0.20	0.97	3.50	0.20	0.98	4.01	0.20	0.92	2.89
7	0.15	0.61	2.09	0.15	0.65	2.68	0.15	0.66	3.58	0.15	0.61	2.09
8	0.15	0.42	1.16	0.15	0.43	1.59	0.15	0.43	1.71	0.15	0.42	1.29
9	0.12	0.30	0.54	0.12	0.30	0.59	0.12	0.30	0.78	0.12	0.30	0.54

Table 1: This table lists minimal, average and maximal radii of a spheres in the creature model's sphere tree. **Reference:** spheres for the reference posture (Figure 5 top). **Linear Blending:** animated posture in Figure 5 middle, skin deformed by linear blending, spheres refitted by Kavan and Zara's algorithm [2005a]. **Spherical Blending:** animated posture in Figure 5 bottom, skin deformed by spherical blending, spheres refitted by Algorithm 1. **Best:** animated posture in Figure 5 bottom, skin deformed by spherical blending, spheres refitted by an exact minimal enclosing sphere algorithm [Gaertner 1999].

5 Results

In order to provide comparative measurements, we execute the tests on models from [Kavan and Zara 2005a]: the man model with 4435 vertices, 8270 triangles and 27 joints, and the creature model with 6682 vertices, 13590 triangles and 56 joints. The sphere tree is also constructed in the same way. First, we investigate the tightness of the refitted spheres, see Figure 5 and Table 1.

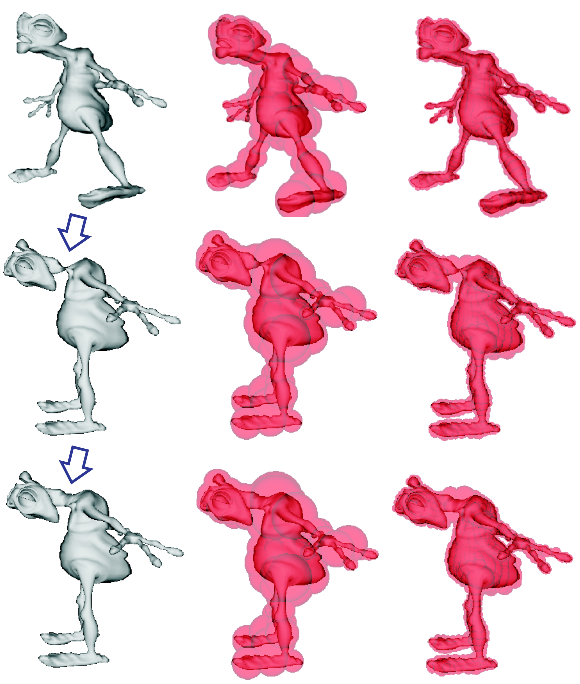

Figure 5: Bounding spheres on levels 4 and 6 of the tree: **Top:** reference posture, minimal enclosing spheres. **Middle:** animated posture deformed with linear blend skinning (note the candy-wrapper artifact in the neck), spheres refitted by [Kavan and Zara 2005a]. **Bottom:** the same posture as in the middle row deformed by spherical blend skinning, spheres refitted by Algorithm 1.

We observe that the size of the refitted spheres is almost the same as the size of the spheres in the reference posture, which are optimal because they are computed by an exact minimal enclosing sphere algorithm [Gaertner 1999]. Also, the size of the spheres refitted for linear and spherical blending is similar – even though the geometries of the deformed skins are different (observe the candy-wrapper artifact in the middle row of Figure 5). The average radii of spheres are reported in Table 1.

We measured the speed of the collision detection and sphere refitting on a 2.5GHz Athlon PC under normal working conditions. The animations are adapted from [Kavan and Zara 2005a], except that spherical blending is used instead of linear. Please note that the comparison of timings of sphere refitting for linear and spherical blending cannot be exact, because both skinning methods produce slightly different geometry (and thus possibly also a different number of intersections). However, since the test animations involve only moderate joint rotations, the difference between linear and spherical skinning is not very big (the difference is obvious only for large joint rotations, e.g., the neck twist in Figure 5). The first scenario is an animation of two walking men, shown in Figure 6.

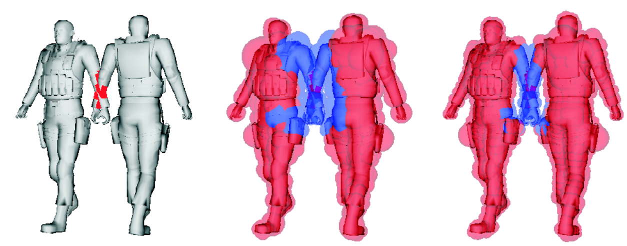

Figure 6: One frame from the first test animation. On the right we see spheres on levels 5 and 6 of the tree refitted by Algorithm 1. Thanks to the on-demand refitting, only 19% of spheres on the fifth level are refitted (9% on the sixth level).

Results for this animation are reported in the first row of Table 2. We use a new version of mathematical libraries, thus the results for linear blending differ slightly from those reported in [Kavan and Zara 2005a]. We see that the slowdown caused by an advanced skinning algorithm is really negligible. The refitting of all 15339 spheres requires a total time of 20.15ms, which is 1.31μs per sphere. This is almost as good as sphere refitting for linear blending, which requires in average 1.21μs per sphere. In practical situations, of course, only a small fraction of all those spheres is refitted, therefore times for collision detection are much smaller, see Table 2.

The next testing scenario is called a "worst-case scenario", because of the many colliding triangles (much more than in practical situations, where the collision response routines prevent such extreme interpenetration). In this animation, we measured besides the standard full CD query also the average time for yes/no CD task,

Scenario	LBS	SBS	Bottom-up
Men (Full)	0.27	0.31	16.70
Creatures (Full)	6.14	7.47	35.17
Creatures (Yes/no)	0.72	1.44	28.67

Table 2: Average times in milliseconds for one collision detection query in various settings. **Full:** CD returns set of all colliding triangles, **Yes/no:** CD returns only one pair of colliding triangles, if any. **LBS:** on-demand refitting for linear blend skinning [Kavan and Zara 2005a]. **SBS:** on-demand refitting for spherical blending skinning presented in this paper. **Bottom-up:** general bottom-up refitting [Brown et al. 2001], applied for model deformed by spherical blend skinning.

which reports only whether the objects are colliding or not, without searching for all colliding triangles (collision detection algorithm in this case stops when the first intersecting triangle pair is found). Measurements are reported in the second and third row of Table 2. We see that, even in difficult situations, the overhead for spherical skinning is fortunately very low, and thus its performance is comparable to refitting for linear blend skinning.

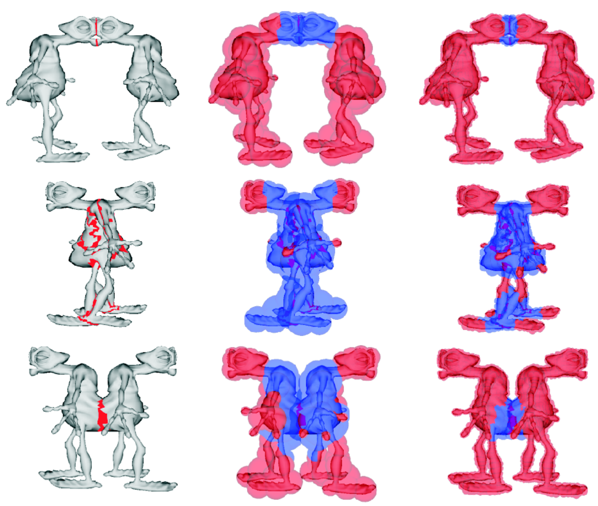

Figure 7: A "walk-through" animation, involving a lot of collisions. In the second and third column we see spheres on level 4 and 6 refitted by our method. This scenario demonstrates that our algorithm is suitable even in difficult situations.

6 Conclusions

In this paper, we propose an efficient collision detection algorithm for 3D models deformed by spherical blend skinning. As a result, it is no longer necessary to consider a trade-off between an artifact free skinning and fast and exact collision detection. The experiments demonstrate that the performance overhead required by spherical blending is very low. The proposed approach is robust and fully compatible with other on-demand methods, e.g., it is possible to detect collisions between models deformed by both spherical skinning and reduced deformations [James and Pai 2004]. The key technique developed in this paper is bounding of rotations on the unit quaternion sphere. On-demand sphere refitting is only one possible application of this technique: we believe that other applications will be discovered in the future.

One of the limitations of the proposed method is that it has at least linear complexity with respect to the number of joints (it is sublinear only with respect to the number of vertices). An algorithm sublinear also in the number of joints would be advantageous for certain kinds of 3D models [Redon et al. 2005]. Other promising future work is to consider more advanced bounding volumes, such as OBBs [Gottschalk et al. 1996] or k-DOPs [Klosowski et al. 1998].

7 Acknowledgements

This work has been partly supported by the Ministry of Education of the Czech Republic under the research programs LC-06008 (Center for Computer Graphics) and MSM 6840770014. We also thank the anonymous reviewers for their valuable suggestions and Štěpán Prokop for donating his 3D models.

Appendix

Lemma 1. *The two following definitions of a cap of sphere with center* $\mathbf{c}_s$ *and radius* r_s *are equivalent:*

(i) A non-empty intersection of a half-space with a surface of sphere with center $\mathbf{c}_s \in R^d$ *and radius* $r_s \in R$

(ii) $\left\{\mathbf{x} \in R^d : \|\mathbf{x}-\mathbf{c}_s\| = r_s,\ (\mathbf{x}-\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s) \geq r_s^2\cos(\alpha_s)\right\}$, *where* $\mathbf{a}_s$ *is the apex, and* α_s *the angle.*

Proof. The half-space from (i) can be written as $\left\{\mathbf{x} \in R^d : (\mathbf{x}, \mathbf{d}) \geq D\right\}$ for some $\mathbf{d} \in R^d, D \in R$, and the surface of the sphere from (i) can be expressed as $\left\{\mathbf{x} \in R^d : \|\mathbf{x}-\mathbf{c}_s\| = r_s\right\}$. It is therefore sufficient to show that the set (ii) can be written as

$$C = \left\{\mathbf{x} \in R^d : \|\mathbf{x}-\mathbf{c}_s\| = r_s,\ (\mathbf{x}, \mathbf{d}) \geq D\right\}$$

The first part is straightforward: if we have a set (ii), we can rewrite $(\mathbf{x}-\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s) \geq r_s^2\cos(\alpha_s)$ as $(\mathbf{x}, \mathbf{a}_s-\mathbf{c}_s) \geq r_s^2\cos(\alpha_s) + (\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s)$. Then it is sufficient to let $\mathbf{d} = \mathbf{a}_s - \mathbf{c}_s$ and $D = r_s^2\cos(\alpha_s) + (\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s)$.

The second part requires showing that $(\mathbf{x}, \mathbf{d}) \geq D$ can be written as $(\mathbf{x}, \mathbf{a}_s-\mathbf{c}_s) \geq r_s^2\cos(\alpha_s) + (\mathbf{c}_s, \mathbf{a}_s-\mathbf{c}_s)$ for some $\mathbf{a}_s \in R^d, \alpha_s \in R$. Without loss of generality, we can assume that $\|\mathbf{d}\| = r_s$ (because $(\mathbf{x}, \mathbf{d}) \geq D$ can be multiplied by any non-zero scalar and still represents the same half-space). The apex is then given simply as $\mathbf{a}_s = \mathbf{d} + \mathbf{c}_s$. It remains to find an angle α_s satisfying equation $D = r_s^2\cos(\alpha_s) + (\mathbf{c}_s, \mathbf{d})$. To complete the proof, it is thus sufficient to verify that $|D - (\mathbf{c}_s, \mathbf{d})| \leq r_s^2$ (so that $\cos(\alpha_s)$ is properly defined by the equation $D = r_s^2\cos(\alpha_s) + (\mathbf{c}_s, \mathbf{d})$). To show this, we use the fact that the half-space $(\mathbf{x}, \mathbf{d}) \geq D$ intersects the spherical surface (here we use the non-emptiness of the intersection in definition (i)). This means that the distance from plane $(\mathbf{x}, \mathbf{d}) = D$ to center $\mathbf{c}_s$ is less than or equal to r_s. The distance from $(\mathbf{x}, \mathbf{d}) = D$ to $\mathbf{c}_s$ is given by the formula $\frac{|D-(\mathbf{c}_s, \mathbf{d})|}{r_s}$, so the previous condition can be written as

$$\frac{|D-(\mathbf{c}_s, \mathbf{d})|}{r_s} \leq r_s \ \Rightarrow\ |D-(\mathbf{c}_s, \mathbf{d})| \leq r_s^2$$

as we wanted to prove. $\square$

Lemma 2. *Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be non-zero quaternions and $\mathbf{n}_1 = \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|}, \ldots, \mathbf{n}_m = \frac{\mathbf{q}_m}{\|\mathbf{q}_m\|}$ the corresponding unit quaternions. Then the set $M = CH(\mathbf{q}_1, \ldots, \mathbf{q}_m)$ represents the same rotations as the set $N = CH(\mathbf{n}_1, \ldots, \mathbf{n}_m)$.*

Proof. We know that two non-zero quaternions $\mathbf{p}, \mathbf{q}$ represent the same rotation iff there exists $k \in R$, $k \neq 0$ such that $\mathbf{p} = k\mathbf{q}$. First, we show that any rotation from M is also present in N. Let us choose an arbitrary $\mathbf{a} \in M$, i.e. $\mathbf{a} = \sum w_i \mathbf{q}_i$ for some $\mathbf{w} \in W_m$. We define $K = \sum w_i \|\mathbf{q}_i\|$ and $u_i = \frac{w_i \|\mathbf{q}_i\|}{K}$. Obviously $K > 0$, $u_i \geq 0$ and $\sum u_i = 1$, that is $\mathbf{u} \in W_m$. Hence $\sum u_i \mathbf{n}_i \in N$ and to finish the first part of the proof it is sufficient to show that $K \cdot \sum u_i \mathbf{n}_i = \mathbf{a}$ (i.e. that $\sum u_i \mathbf{n}_i$ represents the same rotation as $\mathbf{a}$). So, we show that:

$$K \cdot \sum u_i \mathbf{n}_i = K \cdot \sum \frac{w_i \|\mathbf{q}_i\| \mathbf{q}_i}{K \|\mathbf{q}_i\|} = \sum w_i \mathbf{q}_i = \mathbf{a}$$

Second, we show that any rotation from N is also present in M. Let us choose an arbitrary $\mathbf{b} \in N$, i.e. $\mathbf{b} = \sum t_i \mathbf{n}_i$ for some $\mathbf{t} \in W_m$. We define $L = \sum \frac{t_i}{\|\mathbf{q}_i\|}$ and $s_i = \frac{t_i}{\|\mathbf{q}_i\| L}$. Again, $L > 0$ and $\mathbf{s} \in W_m$, therefore $\sum s_i \mathbf{q}_i \in M$. In a similar way as before we see that

$$L \cdot \sum s_i \mathbf{q}_i = L \cdot \sum \frac{t_i \mathbf{q}_i}{\|\mathbf{q}_i\| L} = \sum t_i \mathbf{n}_i = \mathbf{b}$$

which completes the proof. $\square$

Lemma 3. *Let S_a be a spherical surface in R^d with center $\mathbf{a}$ and radius r_a. Let S_b be a sphere in R^d with center $\mathbf{b}$ and radius r_b. Then the intersection $S_a \cap S_b$ is a cap, i.e. $S_a \cap S_b = S_a \cap H$, where H is a halfspace in R^d. Moreover, if $\mathbf{a} = \mathbf{0}, r_a = 1$ and $r_b < 1$, then $\mathbf{0} \notin H$ and the distance from $\mathbf{0}$ to H is $\frac{1 + \|\mathbf{b}\|^2 - r_b^2}{2\|\mathbf{b}\|}$.*

Proof. The set S_a can be written as $\left\{\mathbf{x} \in R^d : \sum (x_i - a_i)^2 = r_a^2\right\}$ and $S_b = \left\{\mathbf{x} \in R^d : \sum (x_i - b_i)^2 \leq r_b^2\right\}$. Therefore the intersection $S_a \cap S_b = \left\{\mathbf{x} \in R^d : \sum (x_i - a_i)^2 = r_a^2,\ \sum (x_i - b_i)^2 \leq r_b^2\right\}$. The system of these two formulae can be written as

$$\sum (x_i^2 - 2x_i a_i + a_i^2) \quad = \quad r_a^2 \tag{7}$$
$$\sum (x_i^2 - 2x_i b_i + b_i^2) \quad \leq \quad r_b^2 \tag{8}$$

which is equivalent to the system

$$\sum (x_i - a_i)^2 \quad = \quad r_a^2 \tag{9}$$
$$\sum (2(a_i - b_i)x_i + b_i^2 - a_i^2) \quad \leq \quad r_b^2 - r_a^2 \tag{10}$$

because (10) is simply (8) minus (7). However, (10) is an equation describing a halfspace, which we can denote as H. This proves the first part of the statement. To show the second part, we substitute $\mathbf{x} = \mathbf{0}$, $\mathbf{a} = 0$, $r_a = 1$ into (10) and we obtain $\sum b_i^2 \leq r_b^2 - 1$. Since we supposed $r_b < 1$, this equation obviously cannot be satisfied, which means that $\mathbf{x} = \mathbf{0}$ cannot be in H. Therefore, the distance from H to $\mathbf{0}$ is the same as the distance from the hyperplane determining H to $\mathbf{0}$. Generally, the distance of hyperplane $(\mathbf{x}, \mathbf{d}) = D$, $\mathbf{d} \in R^d, D \in R$ from $\mathbf{0}$ is $|D| / \|\mathbf{d}\|$. In our case, we have $|D| = |r_b^2 - 1 - \|\mathbf{b}\|^2| = 1 + \|\mathbf{b}\|^2 - r_b^2$ and $\|\mathbf{d}\| = 2\|\mathbf{b}\|$, which proves the last part of the statement. $\square$

Lemma 4. *Let $\mathbf{n}_1, \ldots, \mathbf{n}_m$ be unit quaternions enclosed by sphere $E \subseteq R^4$ with radius < 1. Then the set $C = E \cap S_3$ is a cap such that*

$$\forall \mathbf{w} \in W_m : \ \frac{w_1 \mathbf{n}_1 + \ldots + w_m \mathbf{n}_m}{\|w_1 \mathbf{n}_1 + \ldots + w_m \mathbf{n}_m\|} \in C$$

Proof. From Lemma 3 we know that C is really a cap and can be written as $C = H \cap S_3$, where H is some halfspace not containing the zero vector. Obviously $\mathbf{n}_i \in C$ (because $\mathbf{n}_i \in S_3$ and $\mathbf{n}_i \in E$), therefore it must be also true that $\mathbf{n}_i \in H$. Since a halfspace is always convex, we have $w_1 \mathbf{n}_1 + \ldots + w_m \mathbf{n}_m \in H$. We denote $\mathbf{n}' = w_1 \mathbf{n}_1 + \ldots + w_m \mathbf{n}_m$. Since obviously $\mathbf{n}' / \|\mathbf{n}'\| \in S_3$, it remains to show only that $\mathbf{n}' / \|\mathbf{n}'\| \in H$. First, we apply the triangle inequality to obtain

$$\|\mathbf{n}'\| = \|\sum w_i \mathbf{n}_i\| \leq \sum \|w_i \mathbf{n}_i\| = \sum w_i \|\mathbf{n}_i\| = \sum w_i = 1$$

that is $1 / \|\mathbf{n}'\| \geq 1$. Second, we show that $\gamma \mathbf{n}' \in H$ for any $\gamma \geq 1$, especially for $\gamma = 1 / \|\mathbf{n}'\|$. Since $\mathbf{0} \notin H$, the halfspace H can be expressed as $H = \left\{\mathbf{x} \in R^4 : (\mathbf{x}, \mathbf{d}) \geq 1\right\}$ for some vector $\mathbf{d} \in R^4$. We know that $\mathbf{n}' \in H$, which means that $(\mathbf{n}', \mathbf{d}) \geq 1$ and therefore also $(\gamma \mathbf{n}', \mathbf{d}) \geq 1$, i.e. $\gamma \mathbf{n}' \in H$, which is what we wanted to prove. $\square$

Corollary 1. *If $\mathbf{n}_1, \ldots, \mathbf{n}_m$ and C are as above, then all rotations represented by $CH(\mathbf{n}_1, \ldots, \mathbf{n}_m)$ are also present in C.*

Lemma 5. *Let $C \subseteq R^d$ be a cap with center $\mathbf{0}$, radius r, apex $\mathbf{a}$ and angle $\alpha \in \langle 0, \pi \rangle$. Then the smallest enclosing sphere S of cap C has center $\mathbf{a} \cos \alpha$ and radius $\|\mathbf{a}\| \sin \alpha$.*

Proof. First, we show that cap C cannot be enclosed by a sphere with smaller radius than $\|\mathbf{a}\| \sin \alpha$. This is because there exist two vectors $\mathbf{v}_1, \mathbf{v}_2 \in C$ whose distance is $2\|\mathbf{a}\| \sin \alpha$. To construct those vectors, pick an arbitrary vector $\mathbf{v}$ such that $(\mathbf{v}, \mathbf{a}) = 0$ and $\|\mathbf{v}\| = \|\mathbf{a}\| = r$. Now we can define $\mathbf{v}_1 = \mathbf{a} \cos \alpha + \mathbf{v} \sin \alpha$ and $\mathbf{v}_2 = \mathbf{a} \cos \alpha - \mathbf{v} \sin \alpha$. Their distance is obviously $2\|\mathbf{v}\| \sin \alpha = 2\|\mathbf{a}\| \sin \alpha$, so it remains to verify that really $\mathbf{v}_1, \mathbf{v}_2 \in C$. We compute that

$$\|\mathbf{v}_1\| = \sqrt{\|\mathbf{a}\|^2 \cos^2 \alpha + \|\mathbf{v}\|^2 \sin^2 \alpha} = \sqrt{r^2 (\cos^2 \alpha + \sin^2 \alpha)} = r$$

and

$$(\mathbf{v}_1, \mathbf{a}) = (\mathbf{a} \cos \alpha + \mathbf{v} \sin \alpha, \mathbf{a}) = (\mathbf{a}, \mathbf{a}) \cos \alpha + (\mathbf{v}, \mathbf{a}) \sin \alpha = r^2 \cos \alpha$$

which shows that $\mathbf{v}_1 \in C$. The same reasoning can be repeated for $\mathbf{v}_2$, showing that also $\mathbf{v}_2 \in C$.

In the rest of the proof, we have to verify that $C \subseteq S$. Therefore, let us pick an arbitrary $\mathbf{x} \in C$. According to the definition of the cap, it means that $\|\mathbf{x}\| = r$ and $(\mathbf{x}, \mathbf{a}) \geq r^2 \cos \alpha$. To show that $\mathbf{x} \in S$, we compute

$$\begin{aligned} \|\mathbf{x} - \mathbf{a} \cos \alpha\|^2 \quad &= \quad (\mathbf{x} - \mathbf{a} \cos \alpha, \mathbf{x} - \mathbf{a} \cos \alpha) \\ &= \quad \|\mathbf{x}\|^2 - 2(\mathbf{x}, \mathbf{a} \cos \alpha) + \cos^2 \alpha \|\mathbf{a}\|^2 \end{aligned}$$

Now we use the fact that $\|\mathbf{a}\| = \|\mathbf{x}\| = r$ and $-2(\mathbf{x}, \mathbf{a} \cos \alpha) \leq -2r^2 \cos^2 \alpha$ to obtain

$$\|\mathbf{x}\|^2 - 2(\mathbf{x}, \mathbf{a} \cos \alpha) + \cos^2 \alpha \|\mathbf{a}\|^2 \leq r^2 - 2r^2 \cos^2 \alpha + r^2 \cos^2 \alpha$$

which can be simplified to $r^2 (1 - \cos^2 \alpha) = r^2 \sin^2 \alpha$. Taking the square root on both sides yields $\|\mathbf{x} - \mathbf{a} \cos \alpha\| \leq r \sin \alpha$, that is $\mathbf{x} \in S$. $\square$

References

BROWN, J., SORKIN, S., BRUYNS, C., LATOMBE, J.-C., MONTGOMERY, K., AND STEPHANIDES, M. 2001. Real-time simulation of deformable objects: Tools and application. In *Computer Animation 2001*, 228–236.

EBERLY, D. 2001. *3D game engine design: a practical approach to real-time computer graphics*. Morgan Kaufmann Publishers Inc.

ERICSON, C. 2004. *Real-Time Collision Detection*. Morgan Kaufmann Publishers Inc.

GAERTNER, B. 1999. Fast and robust smallest enclosing balls. In *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms*, Springer-Verlag, 325–338.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *Computer Graphics 30*, Annual Conference Series, 171–180.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, 25–32.

GOVINDARAJU, N. K., KNOTT, D., JAIN, N., KABUL, I., TAMSTORF, R., GAYLE, R., LIN, M. C., AND MANOCHA, D. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3, 991–999.

GUIBAS, L., NGUYEN, A., RUSSEL, D., AND ZHANG, L. 2002. Collision detection for deforming necklaces. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, ACM Press, New York, NY, USA, 33–42.

HE, T. 1999. Fast collision detection using QuOSPO trees. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, 55–62.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. In *Proceedings of Computer Graphics, Visualization and Computer Vision WSCG'04*, 145–152.

HEIM, O., MARSHALL, C. S., AND LAKE, A., 2004. Fast collision detection for 3D bones-based articulated characters. *Game Programming Gems 4*, Charles River Media, 503–514.

HEJL, J., 2004. Hardware skinning with quaternions. *Game Programming Gems 4*, Charles River Media, 487–495.

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph. 23*, 3, 393–398.

JIMÉNEZ, P., THOMAS, F., AND TORRAS, C. 2001. 3D collision detection: a survey. *Computers & Graphics 25*, 2, 269–285.

KAVAN, L., AND ZARA, J. 2005. Fast collision detection for skeletally deformable models. *Computer Graphics Forum 24*, 3, 363–372.

KAVAN, L., AND ZARA, J. 2005. Spherical blend skinning: A real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 9–16.

KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S. B., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics 4*, 1, 21–36.

KLUG, T., AND ALEXA, M. 2004. Bounding volumes for linearly interpolated shapes. In *Computer Graphics International*, 134–139.

LARSEN, E., GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D., 1999. Fast proximity queries with swept sphere volumes. Technical report TR99-018, University of N. Carolina, Chapel Hill.

LARSSON, T., AND AKENINE-MOLLER, T. 2001. Collision detection for continuously deforming bodies. In *Eurographics 2001, Short Presentations*, Eurographics Association, 325–333.

LARSSON, T., AND AKENINE-MOLLER, T. 2003. Efficient collision detection for models deformed by morphing. *The Visual Computer 19*, 2–3, 164–174.

MATOUSEK, J. 2002. *Lectures on Discrete Geometry*. Springer, April.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3, 562–568.

QUINLAN, S. 1994. Efficient distance computation between non-convex objects. In *ICRA*, 3324–3329.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Comput. Graph. Forum 21*, 3.

REDON, S., KIM, Y. J., LIN, M. C., MANOCHA, D., AND TEMPLEMAN, J. 2004. Interactive and continuous collision detection for avatars in virtual environments. In *VR '04: Proceedings of the IEEE Virtual Reality 2004 (VR'04)*, IEEE Computer Society, 117–124.

REDON, S., GALOPPO, N., AND LIN, M. C. 2005. Adaptive dynamics of articulated bodies. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3, 936–945.

TESCHNER, M., HEIDELBERGER, B., MUELLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV'03*, 47–54.

TESCHNER, M., KIMMERLE, S., ZACHMANN, G., HEIDELBERGER, B., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNETAT-THALMANN, N., AND STRASSER, W. 2004. Collision detection for deformable objects. In *Proc. Eurographics, State-of-the-Art Report*, Eurographics Association, Grenoble, France, 119–135.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT 2*, 4, 1–14.

VOLINO, P., AND MAGNENAT THALMANN, N. 1995. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In *Computer Animation and Simulation '95*, Springer-Verlag, D. Terzopoulos and D. Thalmann, Eds., 55–65.

ZACHMANN, G. 2002. Minimal hierarchical collision detection. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, New York, NY, USA, 121–128.