EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

Fast Collision Detection for Skeletally Deformable Models

L. Kavan†, J. Žára

Czech Technical University in Prague

Abstract

We present a new method of collision detection for models deformed by linear blend skinning. The linear blend skinning (also known as skeleton-subspace deformation, vertex-blending, or enveloping) is a popular method to animate believable organic models. We consider an exact collision detection based on a hierarchy of bounding spheres. The main problem with this approach is the update of bounding volumes – they must follow the current deformation of the model. We introduce a new fast method to refit the bounding spheres, which can be executed on spheres in any order. Thanks to this on-demand refitting operation we obtain a collision detection algorithm with speed comparable to the standard rigid body collision detection. The algorithm was tested on a variety of practical situations, including an animated crowd. According to these experiments, the proposed approach is considerably faster than the previous method.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Efficient collision detection (CD) is a crucial part of any application simulating interaction between impenetrable objects. The problem of CD is well studied for the case of rigid bodies moving in space. Unfortunately, these algorithms do not generalize easily to deformable objects, i.e. models whose shape changes during the simulation, for example virtual humans, animals, plants etc.

We focus on CD based on a bounding-volume hierarchy (BVH), which is very efficient for the case of rigid objects. A classical approach how to extend a CD algorithm based on a BVH to deformable objects is to refit all the bounding volumes each time the shape of the object changes [vdB97]. However, the complete refitting of the BVH has inherently at least linear complexity in the number of vertices. It means that the refitting operation is a bottleneck of the CD algorithm, because the CD test itself runs in a sublinear time in common situations.

Therefore, it looks appealing to replace the eager BVH refitting by a lazy method, which refits the bounding volumes in an on-demand way, only prior to an intersection test. An intelligent (on-demand) refitting operation has been published so far only for two classes of deformations: linear morphing [LAM03], and reduced deformable models [JP04].

We present an efficient on-demand refitting operation for another important class of objects, deformed by linear-blend skinning (LBS, also known as skeleton-subspace deformation, vertex-blending, or enveloping). In general, the LBS deforms the model's skin using an auxiliary structure – a skeleton, whose posture influences the shape of the skin. The LBS algorithm is easy to implement and is applied in many virtual reality applications and computer games.

We employ our new refitting operation in a CD algorithm for LBS models. The resulting algorithm is output-sensitive and runs in a sublinear time in configurations without a large number of contacts. The speed of our algorithm is similar to the classic CD algorithms for rigid objects. When compared to the complete BVH refitting, we observe a significant speed-up. The algorithm has been tested on a variety of scenarios, including an animated crowd consisting of 50 models in close proximity. All collisions (without approximation) in more than 650 thousand triangles are detected

† kavanl1@fel.cvut.cz

in real-time, see Figure 1. This was not possible using the previous method.

In the next section we summarize the related work. In section 3 we briefly recapitulate the LBS algorithm and sphere-tree construction. Our proposed refitting operation is presented in section 4, with details elaborated in the Appendix. Section 5 describes various experiments used to test our CD algorithm from different viewpoints.

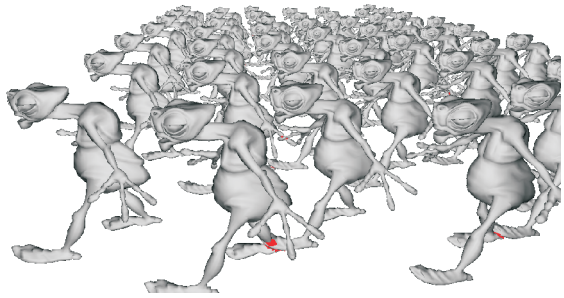

Figure 1: *The scenario for a crowd collision detection experiment: 50 animated creatures with more than 650 thousands triangles together. Our CD algorithm needs an average 52.83ms per frame, whereas the CD based on the complete BVH refitting needs 779.3ms (15-times more).*

2. Related Work

A detailed survey of CD methods is presented in [Eri04]. We focus on CD based on a BVH. The BVH algorithms differ mainly in the choice of bounding volumes. The classic bounding volumes are spheres [Qui94], AABBs [vdB97], OBBs [GLM96], and k-DOPs [KHM*98].

An interesting branch of CD algorithms are so called *time-critical* CD algorithms. Their advantage is that they can be interrupted and asked for an approximate solution, which is a desirable feature in real-time applications. Time-critical CD uses mostly sphere-trees [Hub96, BO02, BO04].

Another important class are *continuous* CD algorithms. They consider not only collisions in discrete time intervals, but search for intersections of moving objects during a time interval [RKC02]. There exist also continuous CD algorithms for articulated models [RKL*04, RKLM04], but working only with objects consisting of rigid parts (without skin deformation, such as robots).

In this paper, we focus on collisions of deformable objects. [TKZ*04] presents a recent survey of deformable CD. The classic approach to deformable CD is based on a complete BVH refitting, which recomputes all the bounding volumes for new vertex positions (after deformation). [vdB97] presents a bottom-up refitting for an AABB-tree, which is reported to be about ten times faster than tree rebuilding. This algorithm is further improved in [LAM01] by combining bottom-up and top-down update and by using trees of higher order than binary. Another method of deformable CD [GDO00] uses a BucketTree, which is based on an octree. The refitting of a sphere-tree for deformable object is studied in [BSB*01]. Their algorithm optimizes the refitting procedure by considering only the vertices with non-zero displacement. Although this optimization can help in some applications, it is useless if the whole model is deformed.

Other approaches to deformable CD are not based on a BVH. This is the case of techniques based on spatial hashing [THM*03] and image-space techniques [HTG04, GRLM03]. The latter can be executed on a GPU, exploiting its efficient visibility queries. The advantage of the non-BVH methods is that they do not need any preprocessing. The disadvantage is the necessity of either hashing or rasterizing all primitives, which disables the sublinear execution time.

In the following, we study the CD based on a BVH. The advantage of a complete BVH refitting is its generality – the CD does not depend on the actual deformation model, because it works directly with the displaced vertices. The drawback is the inefficiency: many bounding volumes may be refitted uselessly, because the subsequent CD query usually needs not all the bounding volumes. As a result of eager refitting, the time complexity is at least linear in the number of vertices. The more efficient strategy is to exploit the properties of the deformation model and avoid checking of all vertices. This makes possible to refit only those bounding volumes that are really necessary for the CD. This has been successfully implemented for models deformed by linear morphing [LAM03]. In this case, the bounding volumes are simply deformed by the same algorithm as the actual object. This is correct, but leads to conservative bounding volumes. A tight bound for linearly combined shapes is achieved in [KA04]. Another model works with so called reduced deformations, which use a linear superposition of displacement fields. An efficient on-demand refitting operation for this model is described in [JP04].

Nevertheless the virtual characters and similar objects are usually modeled and visualized using other techniques, such as the linear blend skinning [MTG03, MG03]. Although it is possible to use the complete BVH refitting for this case, it is prohibitively slow for detailed objects, because of its at least linear complexity. Therefore common applications such as computer games use typically only a crude approximation of the object geometry in order to detect collisions quickly. We demonstrate that using our new on-demand refitting operation, the exact CD of LBS models is possible even in real-time applications.

Conventions

We denote the d-dimensional Euclidean space as R^d, and write its elements (vectors) in bold. The i-th component of

vector $\mathbf{v}$ is denoted as v_i. We work extensively with convex combinations, and therefore we introduce set of convex weights

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

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 length (Euclidean norm) of vector $\mathbf{v}$ as $\|\mathbf{v}\|$.

3. Skeletally Deformable Models

This section briefly reviews the established methods of skeletal animation. The input consists of a triangular mesh representing a digital skin, a skeleton, and vertex weights for each vertex-joint pair. The digital skin is simply a 3D triangular mesh. The skeleton is, formally speaking, a rooted tree, whose nodes are associated with joints and the edges conveniently interpreted as bones. The vertex weights represent the skeleton to skin binding. The triangular mesh and the skeleton are designed in some reference position, e.g. virtual humans are often posed in the da Vinci posture.

We label the joints by integer numbers, assigning zero to the root. Each joint in the reference posture is associated with a local coordinate system. In the animated posture, the joints are transformed by rotation (we do not consider neither translating nor scaling joints). The transformation from the reference position and orientation of joint j to its position and orientation in the animated posture can be described by a homogeneous matrix. This matrix can be expressed as a multiplication of successive joint transformations, and we denote it as C_j, standing for the "complete" matrix.

The most simple skin deformation algorithm assigns each vertex to only one joint and computes

$$\mathbf{v}' = C_j \mathbf{v}$$

where $\mathbf{v}$ is a vertex in the reference skin associated with joint j and $\mathbf{v}'$ is its position in the deformed mesh. Some older computer games animated characters in this way, even though it does not produce nice, smooth deformations.

The linear blend skinning allows assignment of one vertex to multiple bones. Assume that vertex $\mathbf{v}$ is attached to joints $j_1, \ldots, j_n$ with weights $\mathbf{w} = (w_1, \ldots, w_n)$. The n usually is a small number, typically from 1 to 4. The weights are coefficients of a convex combination, which can be expressed using our abbreviation as $\mathbf{w} \in W_n$. We call the joints $j_1, \ldots, j_n$ as the *joint-set* influencing the vertex $\mathbf{v}$ and denote it as $J(\mathbf{v})$. The weight w_i represents the amount of influence of joint j_i. The vertex position in the mesh deformed by LBS is then computed as

$$\mathbf{v}' = \sum_{i=1}^{n} w_i C_{j_i} \mathbf{v} \qquad (1)$$

The interpretation is straightforward: we transform $\mathbf{v}$ by each associated joint's matrix C_{j_i} and take a convex combination of individual transformations. The weights $\mathbf{w}$ are constant during the animation.

It is possible that a triangle has each of its vertices assigned to a completely different joint-set. Note that LBS is non-linear with respect to joint angles, which are the values that actually change during the animation (LBS is linear only in vertex weights). It indicates that computing bounding volumes for the skin deformed by LBS is not trivial. Naive approaches, such as deforming bounding volumes in the same way as the skin, will fail. The methods developed in [LAM03, KA04, JP04] are also not applicable to the LBS, because they consider different deformation models.

3.1. Sphere-tree Construction

The basic step of any BVH algorithm is the construction of the bounding volume tree. We have chosen the most simple bounding volume – a sphere, because of its one unique feature: the invariance under rotation. This property is very advantageous for the on-demand refitting operation. Our sphere-tree is built by a simple top-bottom algorithm. The sphere-tree hierarchy is constructed for the skin in the reference position.

Since our goal is a precise algorithm, we must ensure that each triangle of the skin is covered by a bounding sphere entirely. (We do not allow covering of one triangle by several bounding spheres, although this could be advantageous for long thin triangles.) In the first pass, we construct only a binary sphere-tree in a way similar to [Qui94]. The tree is built in a top to bottom manner, bounding first the whole mesh by one big sphere. Next, we divide the geometry into two parts by a heuristics according to [Qui94] and proceed recursively. In contrast to the previous algorithm, we compute the minimal enclosing sphere for a set of vertices in each node. For this task we use a fast exact algorithm described by [Gae99]. This ensures that our spheres in the reference position are as tight as possible.

In addition to the standard algorithm, we propose a second pass which optimizes the sphere tree. We observed that it may happen that the radii of the children and parent spheres do not differ considerably (see example in section 5). If a node c has a sphere of similar size as its parent node p, it means that it has only a small discriminating power for the CD query. We can therefore remove the node c from the tree and assign its children directly to the node p. It does not violate the correctness of the algorithm, but saves an intersection test and several refitting operations. Therefore in the second pass, we collapse the binary tree to a general n-ary tree, following a simple rule. Let us denote the radii of spheres in nodes c and p as c_r and p_r. The node c is deleted if

$$c_r > C \cdot p_r$$

where C is a constant. We found the number $C = 0.6$ to work well in practice.

4. On-demand Sphere Refitting

In this section we describe our main contribution: the efficient method for an on-demand sphere refitting, which is a key part of fast CD algorithm. Our goal is to detect collisions between models deformed by LBS in an arbitrary posture. During preprocessing, a sphere tree for the model in the reference position is built according to section 3.1. During the CD query, it is necessary to transform the bounding spheres from the reference position to the current (animated) posture preserving their bounding property, i.e. ensuring that a transformed sphere encloses all the geometry that it enclosed in the reference position. It does not affect the result of the CD if the transformed sphere is bigger than necessary, although it may affect the performance.

Observe that the only thing that changes the shape of the deformed skin during the animation are the joint rotations, i.e. the transformation matrices C_j. The number of joints in a typical model is much smaller than the number of vertices. The main idea of our on-demand refitting operation is to update the bounding spheres using only the joint transformations C_j (and some precomputed information which is invariant during the animation). As mentioned in section 3, the LBS does nothing else but a convex combination of individual vertex transformations. In order to enclose the transformed geometry, it is sufficient to enclose only the transformed vertices (since triangles are convex).

Assume that we are refitting a sphere S containing vertices $\mathbf{v}_1,\ldots,\mathbf{v}_l$ in the reference posture. The list of all joint-sets influencing $\mathbf{v}_1,\ldots,\mathbf{v}_l$ is precomputed and stored in the node (together with bounding sphere). First assume for simplicity that all vertices $\mathbf{v}_1,\ldots,\mathbf{v}_l$ are influenced by only one joint-set J, i.e. $J = J(\mathbf{v}_1) = J(\mathbf{v}_2) = \ldots = J(\mathbf{v}_l)$.

The LBS transforms the vertex $\mathbf{v}_i$ to $\mathbf{v}'_i$ according to equation (1). From this equation it can be seen immediately that

$$\mathbf{v}'_i \in CH(\{C_j\mathbf{v}_i : j \in J\})$$

Since $\mathbf{v}_i \in S$, it indicates that we can transform the whole sphere S instead of individual vertices. To transform a sphere by a homogeneous matrix it is sufficient to transform only the center of the sphere and keep the radius intact (thanks to the rotation invariance). Since $C_j\mathbf{v}_i \in C_jS$ for any $j \in J$, we can write

$$\mathbf{v}'_i \in CH\left(\bigcup_{j\in J} C_jS\right) \tag{2}$$

for any $i = 1,\ldots,l$. Note that the bounding volume in (2) depends only on the bounding sphere S and the current joint transformations C_j, i.e. it can be computed in time sublinear to l – the number of vertices. In practice, we work with minimal enclosing sphere instead of a convex hull. This is correct indeed, because the sphere is convex and therefore contains the convex hull.

Unfortunately, the bounding volume according to (2) is very loose (conservative). This is especially apparent if sphere S contains only a few vertices. The bad approximation quality of $CH\left(\bigcup_{j\in J} C_jS\right)$ follows from the fact that it contains points $\mathbf{v}'_i$ for *arbitrary* convex weights. Put in other words, it encloses not only the current LBS deformation, but all possible LBS deformations that could be achieved by varying vertex weights. We can make the bounding spheres much more tight by taking the actual model's vertex weights into account.

This is illustrated in Figure 2. In the left image we see three vertices $\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3$ bounded by sphere S in the reference position. These vertices are influenced by joints j_1 and j_2, let us say that the weights of $\mathbf{v}_1$ are $0.6, 0.4$ (for j_1 and j_2), of $\mathbf{v}_2$: $0.5, 0.5$, and of $\mathbf{v}_3$: $0.4, 0.6$ (indicated by the color). The middle image shows the animated skeleton. All possible positions of the deformed vertex $\mathbf{v}'_i$ form the set

$$L_i = \{wC_{j_1}\mathbf{v}_i + (1-w)C_{j_2}\mathbf{v}_i : w \in \langle 0,1\rangle\}$$

which are the line segments illustrated in the picture. The bounding volume BV_1 given as the convex hull of transformed spheres $C_{j_1}S$ and $C_{j_2}S$ encloses the lines L_1, L_2, L_3 and therefore also the new vertex positions $\mathbf{v}'_1, \mathbf{v}'_2, \mathbf{v}'_3$. A smaller bounding volume BV_2 is depicted in Figure 2 right. It is created by considering that vertex weights for j_1 fall into interval $\langle 0.4, 0.6\rangle$. Therefore it is sufficient to enclose only shorter line segments

$$\{wC_{j_1}\mathbf{v}_i + (1-w)C_{j_2}\mathbf{v}_i : w \in \langle 0.4, 0.6\rangle\}$$

which gives BV_2. The next section describes how to exploit this observation to create tighter refitted spheres.

4.1. Optimized Sphere Refitting

Let us return to the general situation, assuming that $J = \{1,\ldots,n\}$. For any $j \in J$ we define the smallest interval of weights $\langle l_j, h_j\rangle$ such that all vertex weights for joint j fit into this interval. These weight intervals are used to create smaller refitted spheres. The following construction is based on a concept of convex combination of spheres – a generalization of convex combination of points. We discuss and justify this concept in the Appendix. For now it is sufficient to suppose that a convex combination of spheres $C_1S,\ldots,C_nS$ with weights $\mathbf{w} \in W_n$, written as

$$\sum_{j=1}^{n} w_jC_jS$$

is a sphere whose center (resp. radius) is a convex combination of centers (resp. radii) of C_jS with weights $\mathbf{w}$. In order to simplify the notation we define $S_j = C_jS$ for each $j = 1,\ldots,n$.

In the Appendix we also show that it is possible to rewrite the bound (2) to

$$CH\left(\bigcup_{j\in J} S_j\right) = \bigcup_{\mathbf{w}\in W_n} \sum_{j=1}^{n} w_jS_j \tag{3}$$

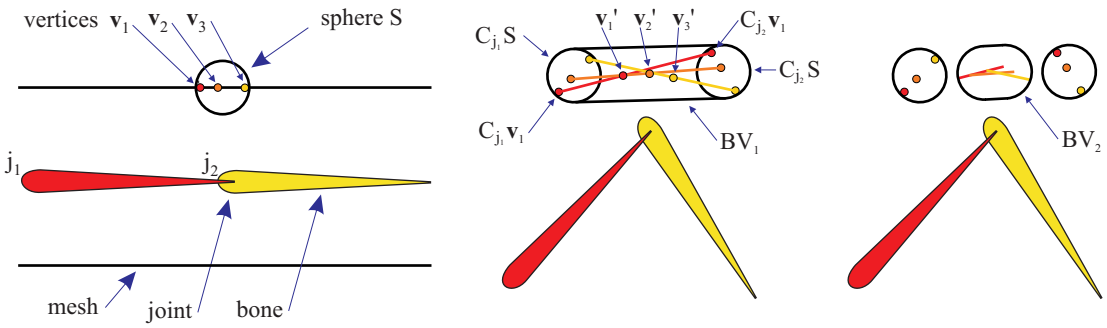

Figure 2: *Left: the reference posture, middle and right: the animated posture. C_{j_1} (resp. C_{j_2}) is the transformation matrix of joint j_1 (resp. j_2). Bounding volume BV_1 is bigger than necessary, because it encloses all possible deformations of vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. Our algorithm uses an optimized bounding volume BV_2, which considers the actual vertex weights.*

which is a generalization of a well-known identity for convex hulls of points. Using this expression, we can take the weight intervals into account by defining $W_n' = \{\mathbf{x} \in W_n : l_i \le x_i \le h_i,\ i = 1,\ldots n\}$ and changing the bounding volume to

$$M' = \bigcup_{\mathbf{w} \in W_n'} \sum_{j=1}^{n} w_j S_j \tag{4}$$

This is correct, because only the weights that actually appear among the bounded vertices are important, and all these weights are within $\langle l_1, h_1\rangle \times \cdots \times \langle l_n, h_n\rangle$. Because of equation (3), we call the set M' as the *generalized convex hull* of spheres $S_1, \ldots, S_n$. It is obvious that the generalized convex hull is always a subset of the non-generalized one, and they are equal iff all weight intervals are $\langle 0, 1\rangle$.

The resulting refitted sphere computed by our algorithm is therefore the bounding sphere of M'. From the definition it is not straightforward to see how M' looks like, and how its enclosing sphere could be efficiently computed. This is an essential part of the proposed algorithm and therefore deserves our attention.

The main trick is that instead of working directly with spheres $S_1, \ldots, S_n$, we compute another set of spheres $R_1, \ldots, R_m$, whose ordinary (non-generalized) convex hull will be equivalent to the generalized convex hull of spheres $S_1, \ldots, S_n$. The spheres $R_1, \ldots, R_m$ reduce the problem to computing the minimal enclosing sphere of spheres.

Our task now is to transform the set of spheres $S_1, \ldots, S_n$ with weight intervals $\langle l_1, h_1\rangle, \ldots, \langle l_n, h_n\rangle$ into spheres $R_1, \ldots, R_m$, turning the generalized convex hull into the standard one. Let us examine the set W_n'. Assuming naturally $0 \le l_i \le h_i \le 1$, the set W_n' can be written explicitly as

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

We can interpret W_n' geometrically as an intersection of $2n$ half-spaces and one hyperplane. It means that W_n' is an $(n-1)$-dimensional convex set in R^n, and it can be expressed as a convex hull of some points in R^n. We call these points *corners* and denote them as $\mathbf{r}_1, \ldots, \mathbf{r}_m$. Because the set W_n' depends only on constant vertex weights, the corners can be precomputed and stored along with the model. We employed only a simple brute-force computation of corners: testing all intersections and discarding those outside $\langle l_1, h_1\rangle \times \ldots \times \langle l_n, h_n\rangle$. Once the corners are known, the spheres $R_1, \ldots, R_m$ can be computed as

$$R_i = \sum_{j=1}^{n} r_{ij} S_j,\quad i = 1, \ldots, m \tag{5}$$

In order to justify this formula, consider that W_n' is a convex hull of $\mathbf{r}_1, \ldots, \mathbf{r}_m$, and thus can be expressed as $W_n' = \{\sum_{i=1}^{m} u_i \mathbf{r}_i : \mathbf{u} \in W_m\}$. Therefore

$$\bigcup_{\mathbf{w} \in W_n'} \sum_{j=1}^{n} w_j S_j = \bigcup_{\mathbf{u} \in W_m} \sum_{j=1}^{n} \left(\sum_{i=1}^{m} u_i r_{ij} \right) S_j$$

because the j-th component of $\sum_{i=1}^{m} u_i \mathbf{r}_i$ is $\sum_{i=1}^{m} u_i r_{ij}$. Since the convex combination of spheres is nothing but a convex combination of their centers and radii, we can swap the sums

$$\bigcup_{\mathbf{u} \in W_m} \sum_{j=1}^{n} \left(\sum_{i=1}^{m} u_i r_{ij} \right) S_j = \bigcup_{\mathbf{u} \in W_m} \sum_{i=1}^{m} u_i \left(\sum_{j=1}^{n} r_{ij} S_j \right)$$

and $\left(\sum_{j=1}^{n} r_{ij} S_j\right)$ is the sphere R_i. Putting the equations together we see that

$$\bigcup_{\mathbf{w} \in W_n'} \sum_{i=1}^{n} w_i S_i = \bigcup_{\mathbf{u} \in W_m} \sum_{i=1}^{m} u_i R_i$$

It shows that the generalized convex hull of spheres $S_1, \ldots, S_n$ is really equal to the ordinary convex hull of spheres $R_1, \ldots, R_m$.

An example is presented in Figure 3. In the picture, the CH_1 is the convex hull of spheres S_1, S_2, S_3. The CH_2 is the generalized convex hull of spheres S_1, S_2, S_3 with respect to weight intervals $\langle 0, 0.6\rangle$, $\langle 0, 1\rangle$, $\langle 0, 1\rangle$. The right part

of the picture demonstrates that CH_2 can be also expressed as $CH(R_1,R_2,R_3,R_4)$ where spheres R_1,R_2,R_3,R_4 are computed according to equation (5).

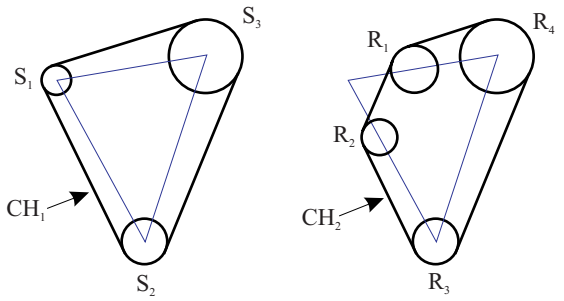

Figure 3: *The standard and generalized convex hull of spheres S_1,S_2,S_3. The generalized convex hull of S_1,S_2,S_3 is equivalent to the ordinary convex hull of spheres R_1,R_2,R_3,R_4.*

4.2. Algorithm Overview

In this section we summarize our CD algorithm for models deformed by LBS. First, we make some off-line preprocessing for every model. The preprocessing involves building the sphere tree, as discussed in section 3.1. For each bounding sphere S in the tree we perform further precomputations. Let us assume that the sphere S encloses vertices $\mathbf{v}_1,\ldots,\mathbf{v}_l$. Primarily we determine the joint-sets $J_1,\ldots,J_k$ that cover all the joints-sets within $\mathbf{v}_1,\ldots,\mathbf{v}_l$, i.e. such that $J_1\cup\ldots\cup J_k = J(\mathbf{v}_1)\cup\ldots\cup J(\mathbf{v}_l)$. We discard potential duplicates, ensuring that $J_i\neq J_j$ for each $i\neq j$.

For each joint set $J_i=\{j_{i1},\ldots,j_{ip_i}\}$, we compute the weight intervals as follows. We start with empty weight intervals and check all vertices associated with J_i. Their weights are included into $\langle l_{i1},h_{i1}\rangle,\ldots,\langle l_{ip_i},h_{ip_i}\rangle$, inflating the intervals if necessary. Then we compute the corners $\mathbf{r}_{i1},\ldots,\mathbf{r}_{im_i}$ of the resulting weight bound, as indicated in the previous section. We store the corners and the joint-sets in the final tree, while the vertices $\mathbf{v}_1,\ldots,\mathbf{v}_l$ can be discarded.

When processing a collision detection query for animated (deformed) models, we apply the standard algorithm: it first tests the parent spheres for intersection. If they intersect, it proceeds recursively to the children – possibly even to the individual triangles. If there is no intersection, then the routine reports no collisions. We modify this algorithm only by adding the refitting operation, which is inserted just prior to the sphere intersection test. This ensures that we work with correct bounding spheres even though the model has changed its shape.

In order to refit the bounding sphere S, we consider each of its joint-sets $J_1,\ldots,J_k$. For each $J_i=\{j_{i1},\ldots,j_{ip_i}\}$ we transform S by the transformations of joints $j_{i1},\ldots,j_{ip_i}$. It gives spheres $S_{i1}=C_{j_{i1}}S,\ldots,S_{ip_i}=C_{j_{ip_i}}S$, which are blended using the equation (5) and the corners $\mathbf{r}_{i1},\ldots,\mathbf{r}_{im_i}$. The result of blending is another set of spheres $R_{i1},\ldots,R_{im_i}$. We put together those spheres for all joint-sets. Finally, we enclose the spheres $R_{11},\ldots,R_{km_k}$ by a single bounding sphere, which is the result of the refitting operation. We use only a simple approximation of minimal enclosing sphere of spheres: the center of the resulting bounding sphere is set to the average of centers of $R_{11},\ldots,R_{km_k}$. The radius is then determined so that the resulting sphere encloses $R_{11},\ldots,R_{km_k}$. The same approximation is used in [JP04], because the speed is much more important for the CD than the accuracy of the smallest enclosing sphere. Note that according to the previous section, the $CH(R_{11},\ldots,R_{km_k})$ bounds the current deformation of the mesh.

The time complexity of the refitting operation can be derived as follows: the computation of spheres $S_{i1},\ldots,S_{ip_i}$ is only the transformation of the center of sphere S by the homogeneous matrix. Since the radius is unchanged, this requires only $18p_i$ floating point operations (the p_i is the number of joints in the i-th joint set). The blending of spheres according to equation (5) is also fast, because the corners are precomputed, and the blending of spheres is nothing but blending of their centers and radii. The computation of spheres $R_{i1},\ldots,R_{im_i}$ requires $8p_im_i-4m_i$ flops (the m_i is the number of corners of the i-th joint set). The approximate smallest enclosing sphere of spheres $R_{i1},\ldots,R_{im_i}$ can be computed using $15m_i-2$ flops. The total number of flops necessary for the refitting operation is thus $\sum_{i=1}^{k}(18p_i+11m_i+8p_im_i-2)$. The important fact is that the execution time of the refitting operation does not depend on the geometry that S actually encloses.

5. Results and Comparison

We tested our collision detection algorithm on three models, whose complexities are described in Table 1. First we examine the tightness of the bounding spheres. Before the tree optimization, the total number of spheres for the creature model is 27079 and the sphere-tree depth is 22. The average radii of spheres for each level are: 34.55 31.82 23.63 18.86 13.07 7.32 4.86 3.46 2.60 1.92 1.47 1.14 0.90 0.72 0.61 0.53 0.47 0.42 0.39 0.38 0.37 0.36. Note that the average radii on certain neighbouring levels are close to each other. This is improved by collapsing the tree according to section 3.1. After this step with C = 0.6 only 18679 spheres remain in the tree and its depth decreases to 9, while the number of children increases from 2 to 15 (the extremal case). The result

	Vertices	Triangles	Joints
Dwarf	859	1664	45
Man	4435	8270	27
Creature	6682	13590	56

Table 1: *Complexities of example models*

is presented in Figure 4. In the top row of the picture we see the reference position of the model, and in the bottom an animated one. The spheres on levels 4 and 6 of the tree, refitted by our algorithm, are also visualized in Figure 4.

Figure 4: *Top: reference posture of the creature with spheres on levels 4 and 6 of the tree (precomputed according to section 3.1), down: an animated posture with spheres refitted by our algorithm (during run-time).*

The average radii of the resulting spheres in the reference position are in the second column of Table 2 (Reference). The third column of Table 2 (On-demand) lists average radii of spheres refitted by our algorithm for the animated posture (Figure 4 bottom). Results for bottom-up refitted spheres are in the fourth column. The average radii of minimal enclosing spheres are reported in the last column of Table 2. They are computed directly from the deformed vertices and thus represent the best possible result of any refitting. From Table 2 as well as from Figure 4 it as apparent that our on-demand refitting operation produces tight bounding spheres. On most levels, the on-demand refitting overcomes the bottom-up refitting, and is not far from the optimal solution. Notice that the gap between on-demand and best possible results is smaller in higher levels. This is understandable, because the on-demand refitted spheres deep in the tree are typically only transformed reference spheres, which are optimal (section 3.1).

Level	Reference	On-demand	Bottom-up	Best
1	34.55	43.89	45.68	33.33
2	18.86	26.17	24.63	19.39
3	7.38	8.05	9.21	7.31
4	3.68	3.95	4.58	3.70
5	1.69	1.77	2.27	1.70
6	0.91	0.94	1.17	0.92
7	0.61	0.63	0.74	0.61
8	0.42	0.43	0.49	0.42
9	0.30	0.30	0.35	0.30

Table 2: *Average radii of spheres on each level of the sphere tree for the creature model. The column "Reference" considers spheres in the reference position, while the other columns refer to spheres in the animated posture.*

We tested the performance of the collision detection on a 2.5Ghz Athlon CPU under normal working conditions. In the first scenario, two men are walking towards each other. One frame of the animation is presented in Figure 5, together with spheres refitted by our algorithm. Our algorithm detects

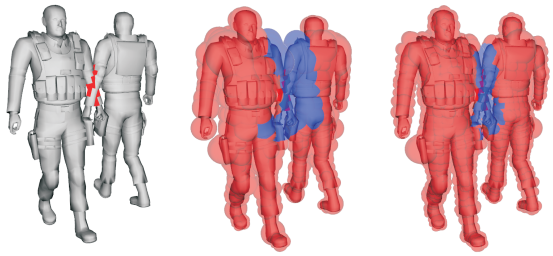

Figure 5: *A collision of two man models with spheres on level 5 and 6 of the tree. Our lazy algorithm does not refit the red spheres, in contrast to eager bottom-up refitting.*

all collisions in average 0.36ms per frame, while the algorithm using the complete bottom-up refitting needs 18.13ms. The almost 50 times faster execution is achieved by refitting only the spheres that are important for the CD query. Our algorithm refits only the blue spheres in Figure 5, while the bottom-up refitting has to refit all spheres, both blue and red. The time for the bottom-up refitting itself is 17.28ms, which reveals that this is the bottleneck of the CD indeed. The refitting of all the 15339 spheres by our on-demand refitting operation takes 20.21ms (1.3μs per sphere). This shows that our refitting operation is quite fast in practice, which complements the flops count derived in the end of section 4.2.

The previous scenario resulted in great speed-up because the number of intersections was not big. In order to test the algorithm in a more complex situation, we designed a "worst case" scenario: two creatures walking through each other, see Figure 6. Such a scenario is unlikely to occur in practice, because collision response algorithms usually prevent large interpenetrations automatically. In this animation the average time of our CD test is 6.97ms, and the CD using the bottom-up refitting takes 37.1ms. The speed-up of our approach is not as high as in the previous example, but shows that the on-demand refitting still performs much better than the bottom-up.

Some applications do not need to determine the whole set

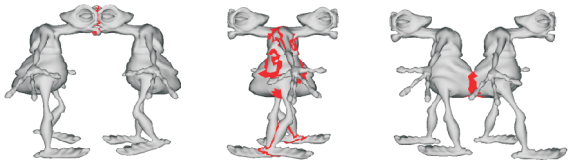

Figure 6: *The worst case scenario for our algorithm, involving a lot of collisions.*

of colliding triangles – the information whether the models are intersecting is sufficient, e.g. for backtracking to a collision-free state. In such a situation, the CD algorithm can stop after finding the first intersecting triangle pair. The CD algorithm with our on-demand refitting operation benefits greatly from this fact, unlike the complete bottom-up refitting. In this case, the average time of our CD test for the same scenario drops down to 0.82ms, while the CD with the bottom-up refitting time improves only slightly: to 30.18ms.

Another important aspect of a CD algorithm is its scalability, i.e. how its performance changes when the size of input data increases. We executed a test on a pair of dwarf models in an animated posture, both in collision free and interfering position, see Figure 7. Starting with a low-polygonal model, we subdivided each triangle to four smaller ones, which ensures the same shape for all CD queries. The results for the colliding situation are presented in Table 3. The speed-up of our algorithm increases with the size of the model. This could be expected because the proposed on-demand refitting does not depend on the actual geometry of the model, unlike the bottom-up refitting algorithm. On the posture from Figure 7 right we performed also a comparison with rigid body CD. A brand new sphere tree built for unsubdivided animated dwarves detects collisions in 0.7ms, whereas our algorithm needs 1.33ms. (However, the building of a tree for the rigid body CD lasts 1549ms, thus this is not a method of choice.) It shows that the performance of our deformable CD algorithm is comparable to rigid body CD.

Figure 7: *The meshes of dwarf models were subdivided several times in order to evaluate the performance of our algorithm while increasing the size of the input. The collision free position (left) gives great speed-ups: 32, 126, 502 and 1995 times for subdivision levels 0,1,2,3 respectively. The more decent results for the interfering situation (right) are reported in Table 3.*

Subdivisions	0	1	2	3
Triangles	1664	6656	26624	106496
Spheres	2046	8821	36506	147025
Collisions	46	90	178	350
On-demand time	1.33	2.17	3.8	8.04
On-demand S-S	4866	9309	17083	37682
Bottom-up time	5.24	18.1	66.37	260
Bottom-up S-S	9515	19833	40527	86437
Speed-up	3.9	8.34	17.47	32.34

Table 3: *The performance comparison for subdivided dwarf models, Figure 7 right. "Spheres" is the total number of spheres in the tree and "S-S" stands for the number of sphere-sphere overlap tests. The row "Speed-up" is the ratio between average time for CD using the bottom-up refitting and CD with our on-demand refitting.*

Finally, we tested our CD algorithm on a crowd simulation. The scenario consists of 50 creature models moving in a close proximity, see Figure 1. In this case, we obtained a 15-times faster collision detection query.

6. Conclusions

We demonstrate that an exact and fast collision detection is possible for models deformed by linear blend skinning. The key part of the new collision detection algorithm and our main contribution is an efficient operation that refits the bounding spheres in any order. This refitting operation speeds up the full collision detection considerably when compared to the bottom-up refitting. It outperforms the bottom-up refitting especially when testing only for the intersection (without reporting all colliding triangles).

The proposed algorithm suggests several directions of future work. The collision detection is nothing but an example of application of our new refitting method. It can speed-up many other algorithms as well, e.g. visibility culling. A possible improvement would be the support of interruptibility: our on-demand refitting operation could be useful even in time-critical collision detection, although we did not investigate this yet. We also did not address the problem of continuous collision detection, which can be especially interesting in deformable objects. To the best of our knowledge, no quantitative comparison of BVH-based collision detection methods with hashing and image-space techniques has been presented in the literature. A thorough comparison could help to determine the direction of the prospective research in collision detection.

7. Acknowledgments

This work has been partly supported by the Ministry of Education, Youth and Sports of the Czech Republic under research program MSM 6840770014 (Research in the Area of the Prospective Information and Navigation Technologies).

We would like to thank to the anonymous reviewers as well as to Daniel Sýkora and Ivana Kolingerová for numerous valuable comments. We thank also to Štěpán Prokop and Psionic for providing the example models and to Eliška Žárová and Ondřej Žára for help with the accompanying video.

Appendix A: Generalized Convex Combinations

We define the convex combination of spheres $S_1,\dots S_n$ (or any general convex sets) with weight vector $\mathbf{w}\in W_n$ as:

$$\sum_{i=1}^{n} w_i S_i \equiv \left\{ \sum_{i=1}^{n} w_i \mathbf{x}_i : \mathbf{x}_i \in S_i \right\}$$

i.e. the set of convex combinations of all points from $S_1,\dots,S_n$. This definition is a natural extension of standard convex combination of points. The following lemma claims that a convex combination of spheres is a sphere which can be computed as a convex combination of sphere centers and radii.

Lemma 1: Let $S_1,\dots S_n$ be spheres in R^d with centers $\mathbf{c}_i \in R^d$ and radii r_i. If $\mathbf{w}\in W_n$ then

$$\sum_{i=1}^{n} w_i S_i = S$$

where S is a sphere with center $\mathbf{c}=\sum_{i=1}^{n} w_i\mathbf{c}_i$ and radius $r=\sum_{i=1}^{n} w_i r_i$.

Proof: Let $\mathbf{x}\in\sum_{i=1}^{n} w_i S_i$. According to the definition of convex combination of convex sets, we have $\mathbf{x}_i\in S_i$ such that $\mathbf{x}=\sum_{i=1}^{n} w_i\mathbf{x}_i$. Obviously $\|\mathbf{x}_i-\mathbf{c}_i\|\le r_i$ for each $i=1,\dots n$. Multiplying the equations by w_i and summing them together yields

$$\sum_{i=1}^{n} \|w_i\mathbf{x}_i - w_i\mathbf{c}_i\| \le \sum_{i=1}^{n} w_i r_i$$

The triangle inequality says that

$$\left\| \sum_{i=1}^{n} w_i\mathbf{x}_i - \sum_{i=1}^{n} w_i\mathbf{c}_i \right\| \le \sum_{i=1}^{n} \|w_i\mathbf{x}_i - w_i\mathbf{c}_i\|$$

Putting both inequalities together gives

$$\|\mathbf{x}-\mathbf{c}\| = \left\| \sum_{i=1}^{n} w_i\mathbf{x}_i - \sum_{i=1}^{n} w_i\mathbf{c}_i \right\| \le \sum_{i=1}^{n} w_i r_i = r$$

which shows that $\mathbf{x}\in S$, and therefore $\sum_{i=1}^{n} w_i S_i \subseteq S$. In order to show the opposite inclusion, choose $\mathbf{x}\in S$, i.e. satisfying $\|\mathbf{x}-\mathbf{c}\|\le r$. Consider points

$$\mathbf{x}_i = \frac{(\mathbf{x}-\mathbf{c})r_i}{r} + \mathbf{c}_i$$

for $i=1,\dots n$. Note that that $\mathbf{x}_i\in S_i$, because

$$\|\mathbf{x}_i-\mathbf{c}_i\| = \left\| \frac{(\mathbf{x}-\mathbf{c})r_i}{r} \right\| \le \frac{r\,r_i}{r} = r_i$$

Moreover,

$$\sum_{i=1}^{n} w_i\mathbf{x}_i = \frac{\mathbf{x}-\mathbf{c}}{r}\sum_{i=1}^{n} w_i r_i + \sum_{i=1}^{n} w_i\mathbf{c}_i = \frac{(\mathbf{x}-\mathbf{c})r}{r} + \mathbf{c} = \mathbf{x}$$

which shows that $\mathbf{x}\in\sum_{i=1}^{n} w_i S_i$, and therefore also $S\subseteq\sum_{i=1}^{n} w_i S_i$. □

The lemma 1 is interesting by itself, but we use it also to proof that the bounding volume from equation (3) is convex indeed.

Lemma 2: Let $S_1,\dots S_n$ are spheres in R^d. Then the set

$$M = \bigcup_{\mathbf{w}\in W_n} \sum_{j=1}^{n} w_j S_j$$

is convex.

Proof: Let us assume that sphere S_i has center $\mathbf{c}_i\in R^d$ and radius r_i. We fix an arbitrary $\mathbf{x}\in M, \mathbf{x}'\in M$ and $\lambda\in\langle 0,1\rangle$, and we have to proof that $\mathbf{x}''=(1-\lambda)\mathbf{x}+\lambda\mathbf{x}'\in M$. By the definition of M, there exist weight vectors $\mathbf{w}\in W_n$ and $\mathbf{w}'\in W_n$ such that $\mathbf{x}\in\sum_{i=1}^{n} w_i S_i$ and $\mathbf{x}'\in\sum_{i=1}^{n} w'_i S_i$. According to lemma 1, it means that

$$\|\mathbf{x}-\sum_{i=1}^{n} w_i\mathbf{c}_i\| \le \sum_{i=1}^{n} w_i r_i,\quad \|\mathbf{x}'-\sum_{i=1}^{n} w'_i\mathbf{c}_i\| \le \sum_{i=1}^{n} w'_i r_i$$

Consider weights

$$w''_i = (1-\lambda)w_i + \lambda w'_i,\quad i=1,\dots,n$$

and observe that $\mathbf{w}''\in W_n$: obviously $w''_i\ge 0$ and $\sum_{i=1}^{n} w''_i = (1-\lambda)\sum_{i=1}^{n} w_i + \lambda\sum_{i=1}^{n} w'_i = 1$. Using the triangle inequality we derive

$$\begin{aligned}
&\|\mathbf{x}''-\sum w''_i\mathbf{c}_i\| = \\
&\|(1-\lambda)\mathbf{x}-\sum(1-\lambda)w_i\mathbf{c}_i+\lambda\mathbf{x}'-\sum\lambda w'_i\mathbf{c}_i\| \le \\
&(1-\lambda)\,\|\mathbf{x}-\sum w_i\mathbf{c}_i\| + \lambda\,\|\mathbf{x}'-\sum w'_i\mathbf{c}_i\| \le \\
&(1-\lambda)\sum w_i r_i + \lambda\sum w'_i r_i = \sum w''_i r_i
\end{aligned}$$

Again by lemma 1, this means that $\mathbf{x}''\in\sum_{i=1}^{n} w''_i S_i$ and thus also $\mathbf{x}''\in M$. □

The correctness of equation (3) is then straightforward:

Lemma 3 Let $S_1,\dots S_n$ are spheres in R^d. Then the set $M=\bigcup_{\mathbf{w}\in W_n}\sum_{j=1}^{n} w_j S_j$ is the convex hull of $S_1\cup\dots\cup S_n$.

Proof: By lemma 2 we know that the set M is convex, and M obviously contains $S_1,\dots,S_n$. Therefore it is sufficient to show that if any other convex set $C\subseteq R^d$ contains $S_1,\dots S_n$, then it contains also M. To show that C contains M, it is sufficient to show that $\sum_{i=1}^{n} w_i S_i\subseteq C$ for any $\mathbf{w}\in W_n$. Let us choose $\mathbf{x}\in\sum_{i=1}^{n} w_i S_i$. By definition, for any $i=1,\dots,n$ we have $\mathbf{x}_i\in S_i$ such that $\mathbf{x}=\sum_{i=1}^{n} w_i\mathbf{x}_i$. We assumed $S_i\subseteq C$ which means that $\mathbf{x}_i\in C$ for each $i=1,\dots,n$. The convexity of C assures $\mathbf{x}\in C$. This verifies that $\sum_{i=1}^{n} w_i S_i\subseteq C$. □

References

[BO02] BRADSHAW G., O'SULLIVAN C.: Sphere-tree construction using dynamic medial axis approximation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2002), ACM Press, pp. 33–40.

[BO04] BRADSHAW G., O'SULLIVAN C.: Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph. 23*, 1 (2004), 1–26.

[BSB*01] BROWN J., SORKIN S., BRUYNS C., LATOMBE K., STEPHANIDES M.: Real-time simulation of deformable objects: Tools and application. *Computer Animation* (2001).

[Eri04] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann Publishers Inc., 2004.

[Gae99] GAERTNER B.: Fast and robust smallest enclosing balls. In *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms* (1999), Springer-Verlag, pp. 325–338.

[GDO00] GANOVELLI F., DINGLIANA J., O'SULLIVAN C.: Buckettree: Improving collision detection between deformable objects. In *Proceedings of the 16th Spring Conference on Computer Graphics* (2000), Comenius University, Bratislava.

[GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics 30*, Annual Conference Series (1996), 171–180.

[GRLM03] GOVINDARAJU N. K., REDON S., LIN M. C., MANOCHA D.: 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* (Aire-la-Ville, Switzerland, 2003), Eurographics Association, pp. 25–32.

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. In *Proceedings of Computer Graphics, Visualization and Computer Vision WSCG'04* (2004), pp. 145–152.

[Hub96] HUBBARD P. M.: Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph. 15*, 3 (1996), 179–210.

[JP04] JAMES D. L., PAI D. K.: BD-Tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph. 23*, 3 (2004), 393–398.

[KA04] KLUG T., ALEXA M.: Bounding volumes for linearly interpolated shapes. In *Computer Graphics International* (2004), pp. 134–139.

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S. B., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (/1998), 21–36.

[LAM01] LARSSON T., AKENINE-MOLLER T.: Collision detection for continuously deforming bodies. In *Eurographics* (2001).

[LAM03] LARSSON T., AKENINE-MOLLER T.: Efficient collision detection for models deformed by morphing. *The Visual Computer 19*, 2–3 (2003), 164–174.

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (2003), 562–568.

[MTG03] MOHR A., TOKHEIM L., GLEICHER M.: Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), ACM Press, pp. 27–30.

[Qui94] QUINLAN S.: Efficient distance computation between non-convex objects. In *ICRA* (1994), pp. 3324–3329.

[RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies. *Comput. Graph. Forum 21*, 3 (2002).

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

[RKLM04] REDON S., KIM Y. J., LIN M. C., MANOCHA D.: Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications* (2004).

[THM*03] TESCHNER M., HEIDELBERGER B., MUELLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV'03* (Munich, Germany, 2003), pp. 47–54.

[TKZ*04] TESCHNER M., KIMMERLE S., ZACHMANN G., HEIDELBERGER B., RAGHUPATHI L., FUHRMANN A., CANI M.-P., FAURE F., MAGNETAT-THALMANN N., STRASSER W.: Collision detection for deformable objects. In *Proc. Eurographics, State-of-the-Art Report* (Grenoble, France, 2004), Eurographics Association, pp. 119–135.

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT 2*, 4 (1997), 1–14.