Eurographics Symposium on Point-Based Graphics (2007)
M. Botsch, R. Pajarola (Editors)

Efficient Bounds for Point-Based Animations

Denis Steinemann Miguel A. Otaduy Markus Gross†

Computer Graphics Laboratory, ETH Zurich

Abstract

We introduce a new and efficient approach for collision detection in point-based animations, based on the fast computation of tight surface bounds. Our approach is able to tightly bound a high-resolution surface with a cost linear in the number of simulation nodes, which is typically small. We extend concepts about bounds of convex sets to the point-based deformation setting, and we introduce an efficient algorithm for finding extrema of these convex sets. We can compute surface bounds orders of magnitude faster and/or tighter than with previous methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

1. Introduction

Point-based or meshless discretization methods have gained rapid popularity for performing physically-based simulations in computer graphics, due to the versatility of the discretization and the capability of handling large deformations [MKN*04], topological changes in cutting or fracture [PKA*05, WSG05, SOG06], or state transitions [KAG*05]. Here we focus on the application of meshless methods to the simulation of elastic deformations derived from continuum mechanics, using moving least squares (MLS) interpolation of shape functions [MKN*04].

Point-based discretization defines a deformation field in the continuum, but in computer graphics we are particularly interested with the deformation of object boundaries, which are *animated* along with the deformation field. We track object boundaries explicitly using triangle meshes, as they offer higher robustness for collision detection and topological changes [SOG06]. In visually interesting animations, object boundaries have high complexity (e.g., tens of thousands of vertices), while the deformation field may be well captured by many fewer simulation nodes (e.g., several hundreds).

Collision detection is an essential component of the animation of deformable objects, and classical acceleration data structures include spatial partitioning [THM*03] or bounding volume hierarchies (BVH) [GLM96]. Pruning of non-colliding regions requires the evaluation of the deformation on the boundary, whose cost depends a priori on the complexity of the boundary surface, not the number of simulation nodes. In the context of BVHs applied to reduced linear deformations [JP04], skinning [KZ05, KOZ06], or low-resolution FEM deformations [MO06, OGRG07], several authors have exploited the existence of a small set of deformation degrees of freedom for efficiently computing surface bounds. Similarly, one could exploit the few degrees of freedom existing in point-based animations for efficiently computing bounds for BVHs [AKP*05, SBT06]. However, as we later elaborate in the paper, point-based deformations pose additional obstacles, making the application of previous methods highly inefficient.

Our Contribution

In this paper, we present a method for efficiently computing tight surface bounds in the context of BVHs applied to point-based animations. Given an object with ℓ vertices animated from n simulation nodes, we reduce the best-case $O(\ell)$ cost with classical BVH-based approaches to a much more efficient $O(n)$ cost. In practice, we obtain more than one-order-of-magnitude speed-up w.r.t. bottom-up update of BVHs.

Our approach builds on the concept of *limited convex combinations* designed by Kavan et al. [KZ05, KOZ06]. They bound surfaces defined by skinning of articulated bodies, and we extend their method to surfaces defined by point-based deformation fields. This approach produces bounds that can be orders of magnitude tighter than previous approaches based on accumulation of deformations [AKP*05].

† {deniss,otaduy,grossm}@inf.ethz.ch

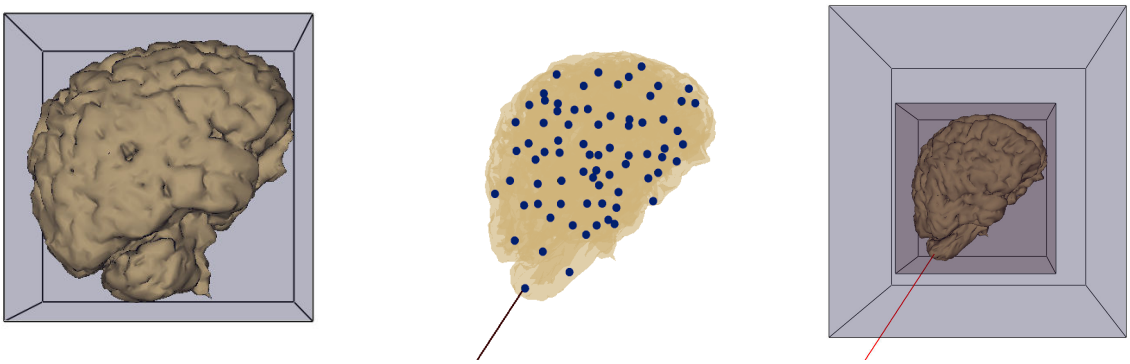

Figure 1: ***Efficient Bounds of the Point-Based Deformation of a Brain.*** *Left: Undeformed brain model with an optimal AABB. Middle: Simulation nodes of the brain being deformed by a pulling force. Right: Under this large deformation, we can compute a tight AABB (outer box) that is only about twice as big as the optimal one (inner box), with cost linear in the number of simulation nodes, independent of the number of surface vertices.*

However, a direct extension of the method of Kavan et al. yields a complexity quadratic in the number of simulation nodes. We introduce a novel evaluation of surface bounds (specifically, an AABB) from convex combinations, with complexity linear in the number of simulation nodes.

The rest of the paper is organized as follows. In Section 3 we review the point-based deformation model and outline the algorithm for efficiently updating the associated BVH. We then present in Section 4 our contribution for efficiently refitting BVs in point-based deformations. We discuss results in Section 5 and conclude with ideas for future work.

2. Related Work

Several types of point-based deformation models are currently used in computer graphics [MHTG05, MHHR06, RJ07], but we build our work on the one with MLS interpolation of shape functions by Müller et al. [MKN*04], due to its foundation on continuum mechanics. Please refer to [FM04] for a survey on meshless methods, and to [NMK*05] for a recent survey on deformable models in computer graphics.

BVHs [GLM96] constitute the most popular acceleration data structure for collision detection, in particular with rigid bodies. When applied to general deformation models, updating a BVH suffers from a cost linear in the number of vertices [Van97]. Using spheres [Hub95], AABBs [Van97], or k-DOPs [KHM*98] as BVs, the BVH can easily be updated in a bottom-up manner with constant cost per BV. However, deformation models with far fewer degrees of freedom than the number of vertices potentially allow for sublinear update of BVs high in the hierarchy, and thereby efficient interruptible collision detection [Hub95], or even sublinear cost for exact collision detection.

Klug and Alexa [KA04] presented efficient BV computation for linearly interpolated shapes, whose degrees of freedom are the blending weights. James and Pai [JP04] introduced the BD-Tree, an efficient sphere-tree for bounding surfaces described by linear combination of a few degrees of freedom. The BD-Tree was originally applied to reduced deformable models, and other extensions of sphere-trees have been applied to FEM deformations on coarse meshes [MO06], geometric deformations through shape matching [SBT06], or point-based animations [AKP*05], exploiting knowledge about the deformation model. All these approaches compute bounds by accumulating deformations from all degrees of freedom, and therefore they suffer tightness degradation with increasing number of simulation nodes. As we show in Section 5, our approach preserves tightness independently of the number of simulation nodes.

Kavan and Zara [KZ05] computed efficient BVs for skinned articulated bodies, with each surface vertex defined by a convex combination of rigid transformations. A set of ℓ vertices animated from a common set of n joints ($\ell \gg n$) can be represented as a set of ℓ points in the $\mathbb{R}^n$ space of possible convex combinations. Kavan and Zara found a bounding set of *limited convex combinations* defined by a simpler set of $m = O(n^2)$ *corners* in $\mathbb{R}^n$. Then, finding the contribution of a joint set to a BV reduces to bounding the m corners instead of the original ℓ vertices. The use of limited convex combinations has been extended to spherical blend skinning [KOZ06] and FEM deformations on coarse meshes [OGRG07]. However, their direct application to point-based animations would produce an explosion of the number of joint sets and corners.

3. Overview

In this section, we review the point-based deformation model we use, and describe the animation of the vertices of a triangle mesh using the point-based deformation field. Then, we discuss the initialization of the BVH (specifically, an AABB-

tree), and outline the update strategy of the complete BVH prior to collision detection queries. But, first, we introduce some useful conventions.

3.1. Conventions

We use convex combinations extensively throughout this paper, hence we define the set of convex weights in $\mathbb{R}^n$ as

$$W_n = \{\mathbf{w} \in \mathbb{R}^n : 0 \leq w_j \leq 1, \sum_{j=1}^{n} w_j = 1\}. \qquad (1)$$

3.2. Point-Based Animation

According to the model of Müller et al. [MKN*04], the deformation field $\mathbf{u}(\mathbf{x})$ of an object is defined at a discrete set of simulation nodes. The gradient $\nabla\mathbf{u}$ of this vector field (which is needed for calculating material stress and strain) is computed using a moving least-squares approximation. A common approach to deform the surface of the object (a triangle mesh in our case) is to carry it along with the simulation nodes, which requires an extrapolation of the deformation field to the surface. The position $\mathbf{v}_k$ of a surface vertex is then defined by

$$\mathbf{v}_k = \mathbf{v}_k^0 + \sum_{j=1}^{n} w_{kj} \left(\mathbf{u}_j + \nabla\mathbf{u}_j^T (\mathbf{v}_k^0 - \mathbf{x}_j) \right), \qquad (2)$$

where $\mathbf{u}_j$, $\mathbf{x}_j$ and $\nabla\mathbf{u}_j$ are, respectively, the displacement, reference position and deformation gradient of a simulation node, $\mathbf{v}_k^0$ is the reference position of the vertex, and w_{kj} is the constant weight with which a node influences the vertex.

Since the vector of weights $\mathbf{w}_k = (w_{k1}, .., w_{kn})$ is convex, i.e., $\mathbf{w}_k \in W_n$, the transformed vertex can be written in a more general form (using homogeneous coordinates) as a convex combination of affine transformations $\mathbf{T}_j$:

$$\mathbf{v}_k = \sum_{j=1}^{n} w_{kj} \mathbf{T}_j \mathbf{v}_k^0, \qquad \mathbf{T}_j = \begin{pmatrix} \mathbf{A}_j & \mathbf{t}_j \\ \mathbf{0} & 1 \end{pmatrix}, \qquad (3)$$

$$\mathbf{A}_j = \nabla\mathbf{u}_j^T + \mathbf{I}, \qquad \mathbf{t}_j = \mathbf{u}_j - \nabla\mathbf{u}_j^T \mathbf{x}_j.$$

3.3. AABB-Tree Construction and Initialization

We have chosen AABBs as the BVs because their update corresponds to finding extreme values along specific directions (i.e., the coordinate axes), which can be efficiently carried out in the context of convex sets as we will show in Section 4.3. We construct the initial AABB-tree by successive top-down splitting of surface triangles at the median of the longest axis defined by the covariance matrix [GLM96].

A given AABB B must bound a set of ℓ vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_\ell\}$, which are animated from n simulation nodes (i.e., those nodes that influence at least one of the ℓ vertices). As shown in Figure 2, each of the vertices may effectively be animated from a subset of the n nodes, but we handle all nodes at once by considering weights $w = 0$.

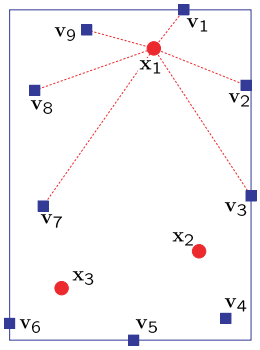

Figure 2: ***Simulation Nodes, Vertices, and AABB.*** *A set of 9 vertices $\mathbf{v}_k$ (blue squares) in rest configuration, bounded by an AABB B_0, and the 3 influencing simulation nodes $\mathbf{x}_j$ (red circles). Red dotted lines denote the vertices influenced by one particular node $\mathbf{x}_1$.*

For every AABB B, we store its center and extensions in the rest position, the set of influencing nodes, and, for every node, the maximum and minimum weights, h and l, with which it influences the vertices to be bounded.

3.4. Run-Time AABB-Tree Update

Typically, AABB-trees for deformable bodies are updated in a bottom-up manner, by first refitting leaf AABBs with cost $O(1)$, and then refitting higher AABBs by bounding their children. With our efficient bounds for point-based deformations, the preferred update strategy depends on the type of collision query to be carried out. For example, interruptible collision detection [Hub95] suggests an on-demand top-down update of AABBs.

In our simulations, we have carried out exact collision detection queries, and we have exploited temporal coherence in the update of the AABB-tree. Instead of updating the AABB-tree top-down, we cache the front of the subtree of AABBs visited in the previous query. We refit the AABBs of this front with our novel algorithm, but we refit higher AABBs by simply bounding their children. Below the front, we again update AABBs with our algorithm on-demand.

For leaf AABBs, we evaluate the positions of the vertices to be bounded, and we compute the optimal AABB instead of following our method. At leaf AABBs, evaluating vertices incurs little penalty, as they are likely to be evaluated for primitive-level queries anyway, and the bounds turn out tighter, thereby saving primitive-level queries as well.

4. Efficient Refitting of AABBs

In this section we present our main contribution: bounding ℓ vertices animated from n simulation nodes with cost $O(n)$. We first show that the deformed vertices can be bounded by combining transformed versions of the rest-position AABB. Then, we show how to bound the resulting AABB using limited convex combinations, and we present our algorithm for efficiently evaluating the extrema of the AABB. We conclude with a summary of the algorithm for refitting AABBs.

4.1. AABBs in Deformed Space

Given an AABB B_0 for a set of vertices $\{\mathbf{v}_k^0\}$ in rest configuration, here we show that, if the vertices are deformed by convex combinations of affine transformations, the deformed vertices can be bounded by convex combinations of transformed AABBs. We first introduce the concepts of transformed AABB and convex combination of AABBs, as well as two associated lemmas (proved in Appendix A).

Definition 1: Given an AABB B_0, we define the transformed AABB $\mathbf{T}_j B_0$ as the parallelepiped defined by the transformed corners of B_0. This parallelepiped is bounded by another AABB $\tilde{B}$.

Lemma 1: Based on Definition 1, given a vertex $\mathbf{v}_k^0$ bounded by an AABB B_0, the transformed vertex $\mathbf{T}_j \mathbf{v}_k^0$ is also bounded by the transformed AABB $\mathbf{T}_j B_0$.

Definition 2: Given a set of n AABBs $\{B_j\}$, we define their convex combination as the set of points obtained from convex combinations of their interior points.

$$\sum_{j=1}^{n} w_j B_j \equiv \left\{ \sum_{j=1}^{n} w_j \mathbf{p}_j : \mathbf{p}_j \in B_j \right\} \tag{4}$$

This is a natural application of the standard definition of convex combination of sets of points.

Lemma 2: A convex combination of AABBs $\{B_j\}$ is another AABB whose extrema are defined by the same convex combination of the extrema of $\{B_j\}$.

4.1.1. AABB for a Deformed Vertex

Given a vertex $\mathbf{v}_k$ defined by convex combination of affine transformations as in Eqn. (3), and applying Lemmas 1 and 2, it is easy to see that the vertex can be bounded by the same convex combination of transformed AABBs:

$$\mathbf{v}_k = \sum_{j=1}^{n} w_{kj} \mathbf{T}_j \mathbf{v}_k^0 \in \sum_{j=1}^{n} w_{kj} \mathbf{T}_j B_0 \subseteq \sum_{j=1}^{n} w_{kj} \tilde{B}_j, \tag{5}$$

where $\mathbf{v}_k^0 \in B_0$, and $\tilde{B}_j$ is the AABB that tightly bounds the parallelepiped $\mathbf{T}_j B_0$.

Figure 3 depicts a 2D AABB B_0 influenced by two nodes, the transformed parallelepipeds $\{B_1, B_2\}$ after deformation of the two nodes, the bounding AABBs $\{\tilde{B}_1, \tilde{B}_2\}$, and the region defined by their convex combination.

4.1.2. AABB for a Set of Vertices

We will bound a set of ℓ vertices by bounding their convex hull $CH(\mathbf{v}_1, ..., \mathbf{v}_\ell) = \sum_{k=1}^{\ell} u_k \mathbf{v}_k$, where the vector of weights $\mathbf{u} \in W_\ell$. Given the n simulation nodes that define the deformation of all ℓ vertices, the AABB B_0 in rest configuration, and the transformed AABBs $\mathbf{T}_j B_0 \in \tilde{B}_j$, we bound the convex hull by applying individual bounds (5) as:

$$CH(\mathbf{v}_1, ..., \mathbf{v}_\ell) = \sum_{k=1}^{\ell} u_k \mathbf{v}_k \subset \sum_{k=1}^{\ell} u_k \left(\sum_{j=1}^{n} w_{kj} \tilde{B}_j \right). \tag{6}$$

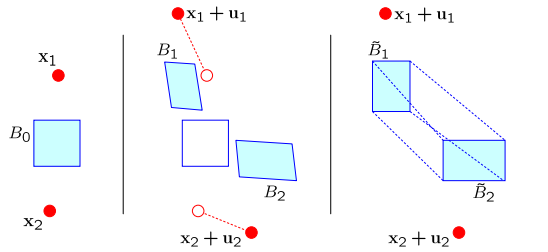

Figure 3: ***Transformed AABBs.*** *Left: A 2D AABB B_0 in rest position, and the two nodes influencing it. Middle: Each deformed node defines an affine transformation on B_0, leading to the parallelepipeds B_1 and B_2. Right: The parallelepipeds are bounded to obtain the transformed AABBs $\tilde{B}_1$ and $\tilde{B}_2$, and the space of their convex combinations is indicated with dotted blue lines.*

Swapping sums, we obtain:

$$CH(\mathbf{v}_1, ..., \mathbf{v}_\ell) \subset \sum_{j=1}^{n} \left(\sum_{k=1}^{\ell} u_k w_{kj} \right) \tilde{B}_j = \sum_{j=1}^{n} \tilde{w}_j \tilde{B}_j. \tag{7}$$

The weights $\{\tilde{w}_j\}$ represent a convex combination of convex weights, which yield another convex combination, i.e., $\tilde{\mathbf{w}} = (\tilde{w}_1, \ldots, \tilde{w}_n) \in W_n$. In other words, every point in the convex hull of the deformed vertices can be bounded by a convex combination of transformed AABBs. From Lemma 2, this is another AABB whose extrema are computed by convex combination of the extrema of the transformed AABBs. However, each point in the convex hull of the deformed vertices is defined by one convex combination, and is therefore bounded by one different convex combination of AABBs.

A possible way to bound all vertices with cost $O(n)$ (i.e., linear in the number of simulation nodes) would be to compute the transformed AABBs $\tilde{B}_j$ and bound them all. This amounts to replacing the set of possible convex combinations $\tilde{\mathbf{w}}$ with a more conservative set W_n, which would yield a loose AABB. Next, we will exploit the concept of limited convex combinations for designing tighter bounds.

4.2. Bounds from Limited Convex Combinations

The term $\tilde{w}_j = \sum_{k=1}^{\ell} u_k w_{kj}$ in Eqn. (7) represents all convex combinations of the weights with which the j^{th} simulation node influences the vertices. This term is bounded by an interval of weights, i.e., $\tilde{w}_j \in [l_j, h_j]$, where l_j and h_j are the minimum and maximum weight of the j^{th} node.

In the space $\mathbb{R}^n$ of weight vectors, the interval $[l_j, h_j]$ yields a region defined by two parallel halfspaces $w_j \geq l_j$ and $w_j \leq h_j$. Following Kavan and Zara [KZ05], we define the *limited convex weight space* (See Figure 4) as the region $W_n' \subset W_n \subset \mathbb{R}^n$ bounded by pairs of parallel hyperplanes and

intersecting the hyperplane of convex weights. Formally,

$$W'_n = \left\{ \mathbf{w} \in \mathbb{R}^n : 0 \le l_j \le w_j \le h_j \le 1, \sum_{j=1}^{n} w_j = 1 \right\}. \quad (8)$$

It is important to highlight that W'_n is a conservative bound of all possible weight vectors $\tilde{\mathbf{w}}$.

The limited convex weight space W'_n may also be represented as the space spanned by convex combinations of its corners [KOZ06]. We first describe how the corners of W'_n can be used for refitting AABBs, and we then discuss the computation of the corners themselves.

4.2.1. Bounds from Corners

Let us assume for now that W'_n has m corners $\{\mathbf{w}'_i\}$. Then, a weight vector $\tilde{\mathbf{w}}$ can be represented as $\tilde{\mathbf{w}} = \sum_{i=1}^{m} u_i \mathbf{w}'_i$, $\mathbf{u} \in W_m$, or for each component, $\tilde{w}_j = \sum_{i=1}^{m} u_i w'_{ij}$. Applying this definition to the convex combination of AABBs in Eqn. (7),

$$B = \sum_{j=1}^{n} \tilde{w}_j \tilde{B}_j = \sum_{j=1}^{n} \sum_{i=1}^{m} u_i w'_{ij} \tilde{B}_j = \sum_{i=1}^{m} u_i B'_i \,. \quad (9)$$

From this expression, we can conclude that the deformed vertices can be bounded by first computing a combined AABB $B'_i = \sum_{j=1}^{n} w'_{ij} \tilde{B}_j$ for each corner of W'_n, and then bounding all the combined AABBs. This is, in essence, the algorithm proposed by Kavan and Zara [KZ05] for sphere trees in linear blend skinning, but in Section 4.3 we demonstrate its inefficiency for point-based animation.

4.2.2. Computation of Corners

As noted by Kavan and Zara [KZ05], the corners of W'_n are defined by intersections of hyperplanes $w_j = l_j$, $w_j = h_j$, and $\sum_{j=1}^{n} w_j = 1$. Finding the exact corners in $\mathbb{R}^n$ is a hard geometric problem, but here we define easy-to-compute alternative corners that conservatively bound W'_n.

We first identify the region of the hyperplane of convex weights, $\sum_{j=1}^{n} w_j = 1$, bounded by the hyperplanes of minimum weights $w_j = l_j$. This region constitutes an $n-1$ dimensional simplex in $\mathbb{R}^n$, and has, therefore, n corners. Each of the corners can be truncated by one of the hyperplanes of maximum weight, $w_j = h_j$, thus cutting the $n-1$ lines meeting at the corner, as shown in Figure 4-right for a case with three simulation nodes. In total, the truncated simplex yields $m = n(n-1) = O(n^2)$ corners. For example, the corner obtained by truncating with $w_j = h_j$ the line resulting from hyperplanes $\{w_k = l_k : k \notin \{i, j\}\}$ is trivially defined as $\mathbf{w}' = (l_1, \ldots, 1 - h_j - \sum_{k \notin \{i,j\}} l_k, \ldots, h_j, \ldots, l_n)$.

4.3. Efficient Evaluation of Extreme Corners

As noted in Section 4.2.1, the deformed vertices can be bounded by computing a combination of transformed AABBs for each corner of W'_n, and then bounding all the combined AABBs $\{B'_i\}$. Since there are $m = O(n^2)$ corners, and evaluating each combined AABB has an $O(n)$ cost, the total cost of this procedure would be $O(n^3)$, although a coherence-aware $O(n^2)$ implementation is also possible. However, note that the resulting AABB is defined simply by six extreme values along the three coordinate axes, and it would suffice to evaluate the corners that realize the six extreme values. In fact, with our definition of corners introduced in Section 4.2.2, selecting the corner that realizes each extreme value has a cost $O(n)$.

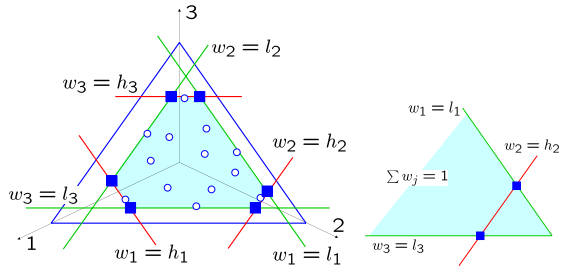

Figure 4: ***Corners in the Limited Convex Weight Space.*** *Left: The limited convex weight space $W'_n \in \mathbb{R}^n$ (i.e., with 3 simulation nodes), shaded in blue, is defined by hyperplanes of maximum (in red) and minimum weights (in green), and the hyperplane of convex weights. Blue circles represent the weight vectors for the vertices to be bounded, and blue squares represent the corners of W'_n. Right: close-up on one corner of the simplex defined by minimum-weight hyperplanes, being truncated by a maximum-weight hyperplane.*

Let us pick a direction $\gamma \in \{x^+, x^-, y^+, y^-, z^+, z^-\}$. Given the transformed AABBs $\{\tilde{B}_j\}$ associated with the n simulation nodes, we define as b^γ_j the extreme value of each AABB $\tilde{B}_j$ along γ. Then, we identify the simulation node $j_1 = \text{argmax}_j b^\gamma_j$ that realizes the largest extreme, as well as the second largest, $j_2 = \text{argmax}_{j \neq j_1} b^\gamma_j$. As proved in Appendix B, the corner that realizes the extreme along γ is defined as:

$$\mathbf{w}^\gamma = (l_1, \ldots, h_{j_1}, \ldots, 1 - h_{j_1} - \sum_{j \notin \{j_1, j_2\}} l_j, \ldots, l_n). \quad (10)$$

And the value of the extreme itself can be computed as:

$$b^\gamma = h_{j_1} b^\gamma_{j_1} + (1 - h_{j_1} - \sum_{j \notin \{j_1, j_2\}} l_j) b^\gamma_{j_2} + \sum_{j \notin \{j_1, j_2\}} l_j b^\gamma_j. \quad (11)$$

For negative directions $\{x^-, y^-, z^-\}$, we flip the sign of extrema before searching for the largest values.

It can easily be deduced that computing each of the six extrema requires an $O(n)$ search for the two largest values, plus an $O(n)$ evaluation of the extreme corner.

4.4. Summary of AABB Refitting

After explaining the principles of our AABB refitting algorithm, we can now list the steps for its implementation. Given a rest-position AABB B_0:

# verts	# nodes			Optimal
	65	225	509	AABB
3.5K	15	35	90	2.0e3
30K	18	48	100	17.2e3
145K	18	46	100	82.5e3

Table 1: ***Scalability Analysis.*** *Time (in μs) for fitting an AABB to the brain model from Figure 1, with varying numbers of vertices and simulation nodes. The trend matches the expected linear cost in the number of nodes. For comparison, the last column shows the time to compute the optimal AABB, which is linear in the number of vertices.*

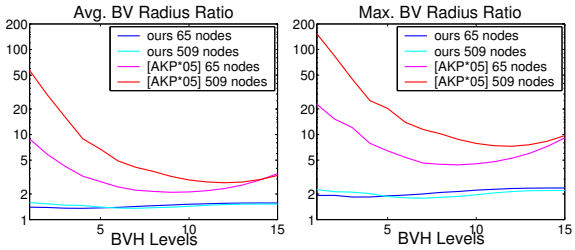

Figure 5: ***BV Tightness Analysis.*** *Ratios between the radii of AABBs computed with our method and the optimal AABBs, for all levels of the AABB-tree, on the brain model of Figure 1. The left plot shows the average ratio over the course of a simulation, while the right plot shows the maximum ratio for each level. Our method largely improves the tightness of the sphere-tree of Adams et al. [AKP*05].*

1. For every influencing simulation node j, transform B_0 by the affine transformation $\mathbf{T}_j$ to obtain a parallelepiped B_j, according to Definition 1 in Section 4.1.
2. Compute transformed AABBs $\{\tilde{B}_j\}$ that bound $\{B_j\}$.
3. For each Cartesian orientation γ, identify the simulation nodes whose transformed AABBs realize the two largest extrema, and evaluate the bound b^γ based on Eqn. (11).

5. Results

We have tested the tightness of AABBs computed with our algorithm, the scalability of the approach, and its performance on several benchmark examples. All tests were carried out on a 3.4 GHz Pentium-4 PC with 1 GB of memory.

Figure 1 shows a brain model deformed under pulling forces. In this scenario, we have evaluated the tightness of AABBs computed with our method, for a surface mesh consisting of 29 966 vertices, with two different simulation node sets: 65 and 509. For such a dense surface, AABB tightness is practically independent of the number of vertices, as the weights of simulation nodes vary very little between adjacent vertices. Figure 5 shows the ratio between the radius of AABBs computed with our method and optimal AABBs, across all levels of the BVH (1 stands for the root, 15 stands for the leaves). The left plot shows the average ratio in the course of the simulation, while the right plot shows the maximum ratio. We measure the radius of an AABB as half of its diagonal. We have also compared the tightness with the approach of Adams et al. [AKP*05], and with our approach we obtain a root BV up to 68 times tighter with 509 simulation nodes. Our root AABB is at most 2.5 times larger than the optimum, and only 1.5 times larger on average, as highlighted in Figure 1. For completeness, in this test we have used our refitting method even on leaf AABBs, although it would be more efficient to evaluate vertex positions and compute optimal bounds, as discussed in Section 3.4.

Using the same brain model, we have tested the scalability of our method as a function of the number of vertices and simulation nodes. Table 1 shows the time (in μs) for fitting an AABB to the brain model. As expected, with ℓ vertices and n simulation nodes, the cost is $O(n)$, i.e., linear in the number of simulation nodes and invariant in the number of vertices. The last column shows the time for computing the optimal AABB, which is $O(\ell)$, i.e., linear in the number of vertices, and up to almost three orders of magnitude larger than with our method.

We have evaluated the performance of our approach on the scene of Figure 6. Six *Santa Claus* models are attached to a ring. The ring is then rotated, producing deformations and collisions of the models. In Table 2, we report timings (in *ms*) for updating the AABB-trees and performing collision detection queries, with different model complexities, and two different ring motions, which produce different contact scenarios. We have tested models with $\sim$ 3K and 45K vertices (18K and 270K in total in the scene), and with 115 and 550 simulation nodes (690 and 3 300 in total in the scene). We compare our AABB-tree update method (using the front-tracking approach discussed in Section 3.4), with a full bottom-up update. Note that in this case, bounding boxes are optimal and the timings are independent of the number of simulation nodes. With 45K vertices, the speed-up for updating AABBs is between 69 and 126 times, and the total speed-up is between 15 and 27. The collision query consists of finding all intersecting triangles, and it is up to four times slower with our method, as it includes on-demand AABB updates and does not use optimal bounding boxes. However, the bottleneck of the entire collision detection process is refitting the BVH-hierarchy, and one may extrapolate from the data that our method would provide even higher speed-up with more complex surfaces.

We have also tested the performance on the scene of Figure 7, with 24 fishes with 8K vertices and 96 simulation nodes each. With our method, the refitting of AABB-trees takes 5.3 *ms* on average, and collision queries take 26.3 *ms*. With full bottom-up update, the refitting takes 401.7 *ms*. on average, and collision queries take 16.6 *ms*. In total, our method provides a speed-up of about 12 times.

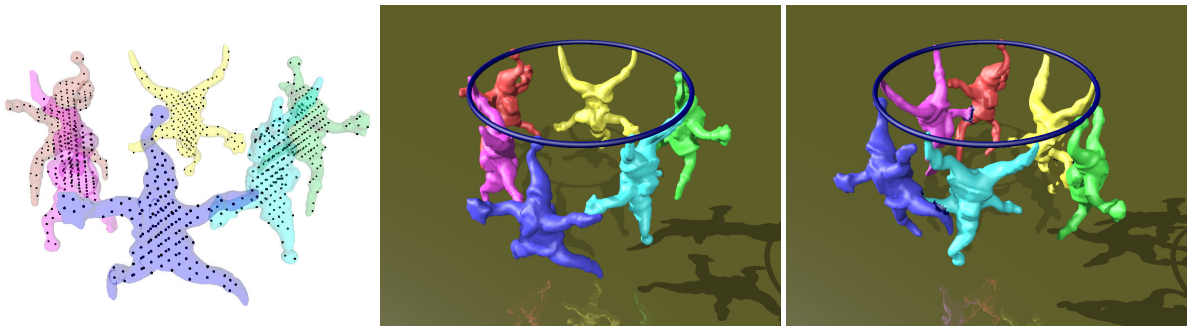

Figure 6: ***Santa Claus Models with Point-Based Deformations.*** *When the top ring moves, the models deform and collide with each other. The left image shows the sampling of simulation nodes, while the right image highlights intersecting triangles.*

Scene Description			Full Bottom-Up			Our Method			
# verts (×6)	# nodes (×6)	Contacts	Refit	Query	Total	Refit	Query	Total	Speed-Up
2 857	115	small	37.1	2.12	39.3	1.83	5.72	7.55	5.2x
2 857	550	small	37.1	2.12	39.3	2.36	5.60	7.97	4.9x
45 682	115	small	575.7	5.89	581.6	4.54	16.6	21.1	27.5x
45 682	550	small	575.7	5.89	581.6	5.77	18.7	24.5	23.7x
2 857	115	large	37.9	2.48	40.4	3.38	10.0	13.4	3.0x
2 857	550	large	37.9	2.48	40.4	3.26	9.28	12.5	3.2x
45 682	115	large	569.7	8.84	578.6	6.87	24.5	31.4	18.4x
45 682	550	large	569.7	8.84	578.6	8.23	29.3	37.5	15.4x

Table 2: ***Performance Analysis.*** *Timings (in ms) for AABB-tree update and collision queries for the benchmark of Figure 6, with various vertex and simulation node resolutions, and under different contact scenarios.*

6. Conclusion

In this paper, we have presented a fast method for computing tight AABBs in the context of point-based deformations, with a cost linear in the number of simulation nodes, and independent of surface complexity. As demonstrated in the experiments, our method achieves both tighter bounds and faster culling than previous methods. It is best suited in situations with intermittent contact, and it will not pay off if objects undergo continuously very large-area contacts.

We are investigating extensions for topology changes (i.e., cutting and fracture) and local resampling of the discretization, which incur modifications of the simulation nodes and surface vertices associated with each AABB. Similarly, our method cannot handle self-colliding situations, but the inherent difficulties for pruning adjacent primitives in a hierarchical manner do not suggest the existence of trivial extensions.

Acknowledgement

We would like to thank the anonymous reviewers, members of the Computer Graphics Lab in Zurich and Ladislav Kavan for their helpful comments. This research was supported in part by the NCCR Co-Me of the Swiss National Science Foundation.

References

[AKP*05] ADAMS B., KEISER R., PAULY M., GUIBAS L. J., GROSS M., DUTRE P.: Efficient raytracing of deforming point-sampled surfaces. *Proc. of Eurographics* (2005).

[FM04] FRIES T. P., MATTHIES H. G.: *Classification and Overview of Meshfree Methods*. Tech. rep., TU Brunswick, Germany, 2004.

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM SIGGRAPH* (1996), 171–180.

[Hub95] HUBBARD P. M.: Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graphics 15*, 3 (1995).

[JP04] JAMES D. L., PAI D. K.: BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH* (2004).

[KA04] KLUG T., ALEXA M.: Bounding volumes for linearly interpolated shapes. *Proc. of Computer Graphics International* (2004).

[KAG*05] KEISER R., ADAMS B., GASSER D., BAZZI P., DUTRE P., GROSS M.: A unified lagrangian approach to solid-fluid animation. *Proc. of Eurographics Symposium on Point-Based Graphics* (2005).

[KHM*98] KLOSOWSKI J., HELD M., MITCHELL J., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding

volume hierarchies of k-DOPs. *IEEE Trans. on Visualization and Computer Graphics 4*, 1 (1998).

[KOZ06] KAVAN L., O'SULLIVAN C., ZARA J.: Efficient collision detection for spherical blend skinning. *Proc. of Graphite* (2006).

[KZ05] KAVAN L., ZARA J.: Fast collision detection for skeletally deformable models. *Proc. of Eurographics* (2005).

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *Proc. of VRIPHYS* (2006).

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *Proc. of ACM SIGGRAPH* (2005).

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point based animation of elastic, plastic and melting objects. *Proc. of Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2004).

[MO06] MENDOZA C., O'SULLIVAN C.: Interruptible collision detection for deformable objects. *Computers & Graphics 30*, 2 (2006).

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMANN E., CARLSON M.: Physically based deformable models in computer graphics (state of the art report). *Eurographics STAR* (2005).

[OGRG07] OTADUY M. A., GERMANN D., REDON S., GROSS M.: Adaptive deformations with fast tight bounds. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007).

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRÉ P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. *Proc. of ACM SIGGRAPH* (2005).

[RJ07] RIVERS A., JAMES D. L.: FastLSM: fast lattice shape matching for robust real-time deformation. *Proc. of ACM SIGGRAPH* (2007).

[SBT06] SPILLMANN J., BECKER M., TESCHNER M.: Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Proc. of VRIPHYS* (2006).

[SOG06] STEINEMANN D., OTADUY M. A., GROSS M.: Fast arbitrary splitting of deforming objects. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006).

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. *Proc. of VMV* (2003).

[Van97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4 (1997).

[WSG05] WICKE M., STEINEMANN D., GROSS M.: Efficient animation of point-sampled thin shells. *Proc. of Eurographics* (2005).

Appendix A: Transformed and Combined AABBs

Proof of Lemma 1: A vertex $\mathbf{v}_k^0 \in B_0$ can be defined as a convex combination of the corners $\mathbf{c}^0$ of B_0. Then, the transformed vertex can be expressed as

$$\mathbf{v}_k = \mathbf{T}_j \sum_i u_i \mathbf{c}_i^0 = \sum_i u_i (\mathbf{T}_j \mathbf{c}_i^0). \tag{12}$$

We observe that, due to linearity of the affine transformation, the transformed vertex can be represented as the same convex combination of the transformed corners, therefore it is bounded by the transformed AABB.

Proof of Lemma 2: Given AABBs $\{B_j\}$, with extrema b_j^γ along the Cartesian direction γ, the convex combination of the extrema yields a bound $b^\gamma = \sum_j w_j b_j^\gamma$. We aim to proof that the same convex combination applied to points in the AABBs yields a point bounded by b^γ along γ. Convex combinations on points are applied independently on each coordinate, therefore we have $p^\gamma = \sum_j w_j p_j^\gamma$. By definition of bounds, $b_j^\gamma \geq p_j^\gamma$. From convex combinations, $w_j \leq 1$. Then, it is obvious that $p^\gamma \leq b^\gamma$. The proof applies also to negative directions by flipping the sign of point coordinates.

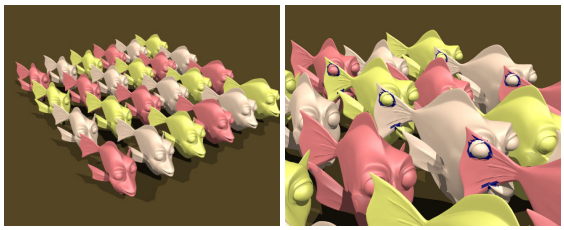

Figure 7: ***Collisions between Deforming Fishes.*** *Our method provides a speed-up of* 12 *times in this scenario. Intersecting triangles are highlighted on the right.*

Appendix B: Extreme Corner Evaluation

As shown in section 4.2.2, a corner is defined by one maximum weight, one based on the convex constraint $1-h-\sum l$, and $n-2$ minimum weights. In the closed-form definition (11) of the extreme b^γ, the simulation node with largest associated value, j_1, the one with second largest value, j_2, and the $n-2$ remaining nodes are weighted with this set of weights. This gives us seven choices for the weighting schemes of the simulation nodes. For one possible scheme,

$$b^* = h_{j_2} b_{j_2}^\gamma + (1 - h_{j_2} - \sum_{j \notin \{j_1, j_2\}} l_j) b_{j_1}^\gamma + \sum_{j \notin \{j_1, j_2\}} l_j b_j^\gamma, \tag{13}$$

we prove that $b^\gamma \geq b^*$, and the same applies to the remaining combinations.

Subtracting terms, we have

$$b^\gamma - b^* = (h_{j_1} - (1 - h_{j_2} - \sum_{j \notin \{j_1, j_2\}} l_j))(b_{j_1}^\gamma - b_{j_2}^\gamma). \tag{14}$$

By definition of j_1 as the node realizing the largest value, $b_{j_1}^\gamma \geq b_{j_2}^\gamma$. By definition of h_{j_1} as the maximum weight for its node, $h_{j_1} \geq (1 - h_{j_2} - \sum_{j \notin \{j_1, j_2\}} l_j)$. Therefore, $b^\gamma \geq b^*$.