Available online at www.sciencedirect.com
ScienceDirect
Computers & Graphics 32 (2008) 235–245
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Special Section: Point-Based Graphics

Tight and efficient surface bounds in meshless animation

Denis Steinemann*, Miguel A. Otaduy, Markus Gross

Computer Graphics Laboratory, ETH Zurich, Switzerland

Received 10 December 2007; received in revised form 25 January 2008; accepted 28 January 2008

Abstract

This paper presents a fast approach for computing tight surface bounds in meshless animation, and its application to collision detection. Given a high-resolution surface animated by a comparatively small number of simulation nodes, we are able to compute tight bounding volumes with a cost linear in the number of simulation nodes. Our approach extends concepts about bounds of convex sets to the meshless deformation setting, and we introduce an efficient algorithm for finding extrema of these convex sets. The extrema can be used for efficiently updating bounding volumes such as AABBs or k-DOPs, as we show in our results. The choice of particular bounding volume may depend on the complexity of the contact configurations, but in all cases we can compute surface bound orders of magnitude faster and/or tighter than with previous methods.

Keywords: Collision detection; Bounding volume hierarchies; Convex combinations; Meshless deformation

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 [1], topological changes in cutting or fracture [2–4], or state transitions [5]. Here we focus on the application of meshless methods to the simulation of elastic deformations derived from continuum mechanics, using moving least squares (MLSs) interpolation of shape functions [1], as reviewed in Section 3.

A meshless 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. As opposed to other work that employs point-based surface representations [2], we track object boundaries explicitly using triangle meshes, as they offer higher robustness for collision detection and topological changes [4]. 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). Such animations where deformation is separated from surface geometry can be implemented efficiently using graphics hardware. As noted by [6], the host CPU only has to compute deformation coordinates, without the need for access to the complete surface geometry. The unavoidable task of deforming the entire high-resolution object surface for rendering can be done on modern GPUs using matrix palette skinning, for example.

Collision detection is an essential component of the animation of deformable objects, and classical acceleration data structures include spatial partitioning [7] or bounding volume hierarchies (BVHs) [8]. Pruning of non-colliding regions requires the evaluation of the deformation on the boundary, with a cost that depends a priori on the complexity of the boundary surface, and not the number of simulation nodes. In the context of BVHs applied to reduced linear deformations [6], skinning [9,10], or low-resolution FEM deformations [11,12], several authors have exploited the existence of a small set of deformation degrees of freedom for efficiently computing surface bounds. Similarly, and as done by others as well [13,14],

*Corresponding author. Tel.: +41 44 6320782.
E-mail address: deniss@inf.ethz.ch (D. Steinemann).

0097-8493/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2008.01.013

we exploit the few degrees of freedom existing in meshless animations for efficiently computing bounds for BVHs. In Section 4 we outline the hybrid update of BVHs from a reduced set of deformation degrees of freedom. However, as we later elaborate in the paper, meshless deformations pose additional obstacles, making the application of previous methods highly inefficient.

The *main contribution* of this paper, described in Section 5, is a method for efficiently computing tight bounds for surfaces animated from meshless deformations. Specifically, we address the efficient computation of bounds along specific directions, which constitutes the building block for bounding volumes such as k-DOPs [15] or AABBs [16]. Given an object with ℓ vertices animated from n simulation nodes, we reduce the best-case $O(\ell)$ cost for computing a bound 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. [9,10]. They bound surfaces defined by skinning of articulated bodies, and we extend their method to surfaces defined by meshless deformation fields. This approach produces bounds that can be orders of magnitude tighter than the previous approaches based on accumulation of deformations [13]. 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 from convex combinations, with complexity linear in the number of simulation nodes.

In Section 6 we analyze the performance of our algorithm, in terms of tightness, efficiency, and scalability, and we compare it with respect to traditional bottom-up BVH updates and spatial partitioning. We also evaluate AABBs and more complex k-DOPs, and our results indicate that AABBs are better suited in situations with few collisions, while the use of more complex k-DOPs may pay off as the number of collisions increases (Fig. 1).

2. Related work

Several types of meshless deformation models are currently used in computer graphics [17–19], but we build our work on the one with MLS interpolation of shape functions by Müller et al. [1], due to its foundation on continuum mechanics. Refer to [20] for a survey on meshless methods, and to [21] for a recent survey on deformable models in computer graphics.

BVHs [8] 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 [16]. Using spheres [22], AABBs [16], or k-DOPs [15] as bounding volumes (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 [22], or even sublinear cost for exact collision detection.

Klug and Alexa [23] presented efficient BV computation for linearly interpolated shapes, whose degrees of freedom are the blending weights. James and Pai [6] 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 [11], geometric deformations through shape matching [14], or meshless animations [13], 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 6, our approach preserves tightness independently of the number of simulation nodes.

Kavan and Zara [9] computed efficient BVs for skinned articulated bodies, with each surface vertex defined by a convex combination of rigid transformations. A set of ℓ

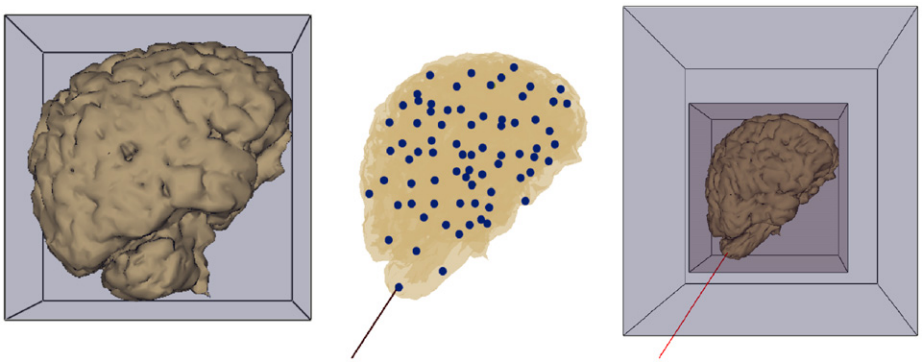

Fig. 1. Efficient bounds of the meshless deformation of a brain. Left: Undeformed brain model with an optimal AABB. Middle: Simulation nodes of the brain being deformed by a pulling force. Right: Under this large deformation, we can compute a tight AABB (outer box) that is only about twice as big as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

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 [10] and FEM deformations on coarse meshes [12]. However, their direct application to meshless animations would produce an explosion of the number of joint sets and corners.

3. Review of meshless animation

In this section, we review the meshless deformation model we use, describe the animation of the vertices of a triangle mesh using the meshless deformation field, and formulate this animation as a convex combination of affine transformations.

3.1. Meshless deformation and surface animation

According to the model of Müller et al. [1], 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 an MLSs 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

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

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.

3.2. Convex combination of transformations

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

$$W_n = \left\{ \mathbf{w} \in \mathbb{R}^n : 0 \leqslant w_j \leqslant 1, \sum_{j=1}^{n} w_j = 1 \right\}. \tag{2}$$

The vector of weights $\mathbf{w}_k = (w_{k1}, \ldots, w_{kn})$ that defines the animation of a surface vertex according to Eq. (1) is convex, i.e., $\mathbf{w}_k \in W_n$. Then, the transformed vertex can be written in a more general form (using homogeneous coordinates) as a convex combination of affine transformations $\mathbf{T}_j$:

$$\mathbf{v}_k = \sum_{j=1}^{n} w_{kj}\mathbf{T}_j\mathbf{v}_k^0, \quad \mathbf{T}_j = \begin{pmatrix} \mathbf{A}_j & \mathbf{t}_j \\ \mathbf{0} & 1 \end{pmatrix},$$
$$\mathbf{A}_j = \nabla\mathbf{u}_j^{\mathrm{T}} + \mathbf{I}, \quad \mathbf{t}_j = \mathbf{u}_j - \nabla\mathbf{u}_j^{\mathrm{T}}\mathbf{x}_j. \tag{3}$$

4. Construction and update of the BVH

Here we discuss the initialization of the BVH, and outline the update strategy of the complete BVH prior to collision detection queries.

4.1. BVH construction and initialization

We enclose each surface triangle in one leaf BV, and build the BVH as a binary tree. In practice, we construct the tree-structure of the BVH by successive top-down splitting of surface triangles at the median of the longest axis defined by the covariance matrix [8].

For each node of the BVH, we distinguish two types of BVs: a *rest-state box* B, and a *deformed-state k-DOP* D. The choices of rest-state box (i.e., AABB or OBB) and deformed-state k-DOP (e.g., AABB = 6-DOP, 14-DOP, 18-DOP, 26-DOP, etc.) are independent of each other. In Section 6 we discuss results with a few combinations. Note that in principle, the rest state BV could be any convex BV, but for simplicity we will only consider boxes in this paper.

We choose k-DOPs as deformed-state BVs because their update corresponds to finding maximum values along specific directions (i.e., the k directions). Such an operation can be efficiently carried out in the context of convex sets as we will show in Section 5.3.

A k-DOP D 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 Fig. 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 $w = 0$.

For every BV, we store the rest-state box, 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.

4.2. Run-time BVH update

Typically, BVHs for deformable bodies are updated in a bottom-up manner, by first refitting leaf BVs (with cost $O(1)$ for both AABBs and higher-order k-DOPs), and then refitting higher BVs by bounding their children. With our efficient bounds for meshless deformations, the preferred update strategy depends on the type of collision query to be carried out. For example, interruptible collision detection [22] suggests an on-demand top-down update of BVs.

In our simulations, we have carried out exact collision detection queries, and we have exploited temporal coherence in the update of the BVH. Instead of updating the

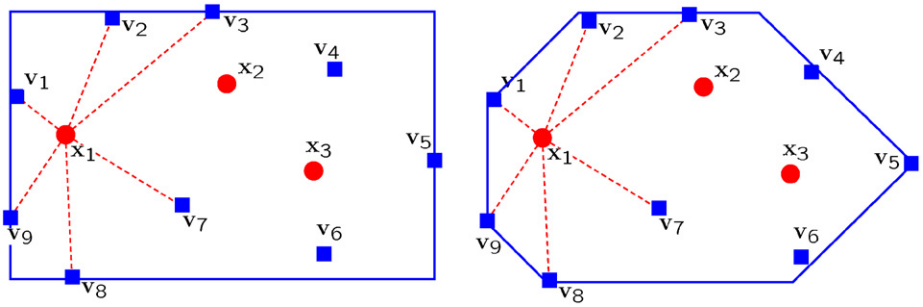

Fig. 2. Simulation nodes, vertices, and bounds. A set of nine vertices $\mathbf{v}_k$ (blue squares), and their three influencing simulation nodes $\mathbf{x}_j$ (red circles). Red dotted lines denote the vertices influenced by one particular node $\mathbf{x}_1$. On the left, the vertices are bounded by a 2D AABB, while on the right they are bounded by a 2D 8-DOP.

BVH top-down, we cache the front of the subtree of BVs visited in the previous query. We refit the BVs of this front with our novel algorithm, but we refit higher BVs by simply bounding their children. Below the front, we again update BVs with our algorithm on-demand.

For leaf BVs, we evaluate the positions of the vertices to be bounded, and we compute the optimal BV instead of following our novel method. At leaf BVs, 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.

5. Efficient refitting of bounding volumes

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-state boxes. Then, we show how to compute tight k-DOPs using limited convex combinations, and we present our algorithm for efficiently evaluating the extrema of the k-DOPs. We conclude with a summary of the algorithm for refitting one k-DOP.

5.1. Bounding volumes in deformed state

Given a box B that bounds 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, we can bound the deformed vertices by a convex combination of k-DOPs. Each k-DOP is computed as the bound of a transformed version of B. We first introduce the concepts of transformed box and convex combination of k-DOPs, as well as two associated lemmas.

Definition 1. Given a box B_0, we define the transformed box $B_j = \mathbf{T}_j B_0$ as the parallelepiped defined by the transformed corners of B_0. This parallelepiped can then be bounded by a k-DOP D_j.

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

Proof. 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

$$\mathbf{v}_k = \mathbf{T}_j \sum_i u_i \mathbf{c}_i^0 = \sum_i u_i (\mathbf{T}_j \mathbf{c}_i^0). \tag{4}$$

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 box. Again note that in principle this holds for any convex BV defined by its corners. □

Definition 2. Given a set of n k-DOPs $\{D_j\}$, we define their convex combination as the set of points obtained from convex combinations of their interior points.

$$\sum_{j=1}^{n} w_j D_j \equiv \left\{ \sum_{j=1}^{n} w_j \mathbf{p}_j : \mathbf{p}_j \in D_j \right\}. \tag{5}$$

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

Lemma 2. *A convex combination of* k-DOPs $\{D_j\}$ *is another* k-DOP *whose extrema are defined by the same convex combination of the extrema of* $\{D_j\}$.

Proof. Given k-DOPs $\{D_j\}$, with extrema $\{b_j^\gamma\}$ along the 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 interior points $\{\mathbf{p}_j\}$ of the k-DOPs yields a point bounded by b^γ along γ. This point can be expressed as $\mathbf{p} = \sum_j w_j \mathbf{p}_j$, and its projection onto the direction γ is $p^\gamma = \sum_j w_j \gamma^{\mathrm{T}} \mathbf{p}_j$. By definition of the k-DOPs, $\gamma^{\mathrm{T}}\mathbf{p}_j \leqslant b_j^\gamma$. Then, $p^\gamma \leqslant \sum_j w_j b_j^\gamma$. And, by definition of b^γ, $p^\gamma \leqslant b^\gamma$. □

5.1.1. k-DOP for a deformed vertex

Given a vertex $\mathbf{v}_k$ defined by convex combination of affine transformations as in Eq. (3), it is easy to see that the vertex can be bounded by a convex combination of k-DOPs:

$$\begin{aligned} &\text{Given } \mathbf{v}_k^0 \in B_0, \\ &\text{Applying Lemma 1:} \quad \mathbf{T}_j\mathbf{v}_k^0 \in \mathbf{T}_j B_0 = B_j \subseteq D_j, \\ &\text{Applying Lemma 2:} \quad \mathbf{v}_k = \sum_{j=1}^{n} w_{kj} \mathbf{T}_j \mathbf{v}_k^0 \in \sum_{j=1}^{n} w_{kj} D_j. \end{aligned} \tag{6}$$

Fig. 3 depicts a 2D box B_0 influenced by two nodes, the transformed boxes $\{B_1, B_2\}$ after deformation of the two nodes, the bounding 8-DOPs $\{D_1, D_2\}$, and the region defined by their convex combination.

5.1.2. k-DOP for a set of vertices

We will bound a set of ℓ vertices by bounding their convex hull $CH(\mathbf{v}_1,\ldots,\mathbf{v}_\ell) = \sum_{k=1}^{\ell} u_k \mathbf{v}_k$, where the vector of weights $\mathbf{u} \in W_\ell$. Given the n simulation nodes that define the deformation of all ℓ vertices, the box B_0 in rest configuration, the transformed boxes $B_j = \mathbf{T}_j B_0$, and their bounding k-DOPs D_j, we bound the convex hull by applying individual bounds (6) as

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

Swapping sums, we obtain

$$CH(\mathbf{v}_1,\ldots,\mathbf{v}_\ell) \subset \sum_{j=1}^{n} \left(\sum_{k=1}^{\ell} u_k w_{kj} \right) D_j = \sum_{j=1}^{n} \tilde{w}_j D_j. \tag{8}$$

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 k-DOPs. From Lemma 2, this is another k-DOP whose extrema are computed by convex combination of the extrema of the k-DOPs $\{D_j\}$. 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 k-DOPs.

A possible way to bound all vertices with cost $O(n)$ (i.e., linear in the number of simulation nodes) would be to compute all k-DOPs $\{D_j\}$ 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 k-DOP. Next, we will exploit the concept of limited convex combinations for designing tighter bounds.

5.2. Bounds from limited convex combinations

The term $\tilde{w}_j = \sum_{k=1}^{\ell} u_k w_{kj}$ in Eq. (8) represents all convex combinations of the weights with which the jth 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 jth node.

In the space $\mathbb{R}^n$ of weight vectors, the interval $[l_j, h_j]$ yields a region defined by two parallel halfspaces $w_j \geqslant l_j$ and $w_j \leqslant h_j$. Following Kavan and Zara [9], we define the *limited convex weight space* (see Fig. 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,

$$W'_n = \left\{ \mathbf{w} \in \mathbb{R}^n : 0 \leqslant l_j \leqslant w_j \leqslant h_j \leqslant 1, \sum_{j=1}^{n} w_j = 1 \right\}. \tag{9}$$

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

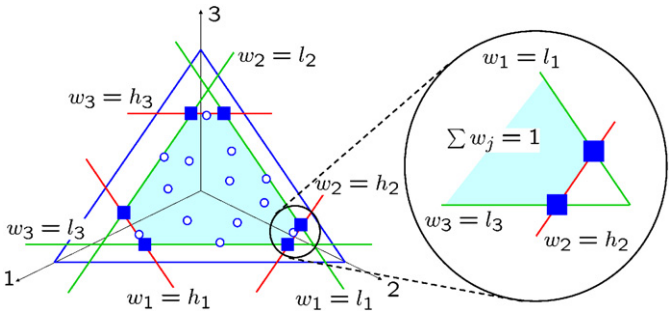

Fig. 4. Corners in the limited convex weight space. Left: The limited convex weight space $W'_n \in \mathbb{R}^n$ (i.e., with three simulation nodes), shaded in blue, is defined by hyperplanes of maximum (in red) and minimum weights (in green), and the hyperplane of convex weights. Blue circles represent 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.

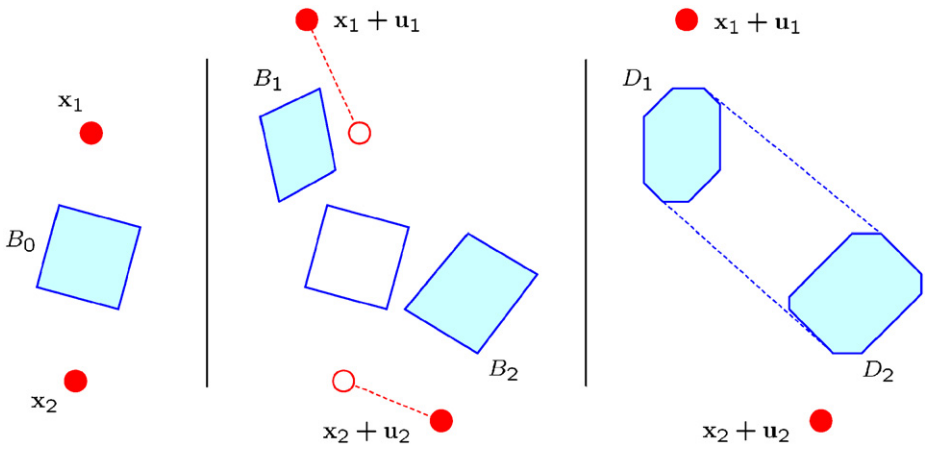

Fig. 3. Transformed boxes and bounding k-DOPs. Left: A 2D box B_0 in rest position, and the two nodes influencing it. Middle: Each deformed node defines an affine transformation on B_0, leading to the parallelepipeds B_1 and B_2. Right: The parallelepipeds are bounded to obtain the k-DOPs D_1 and D_2, and the space of their convex combinations is indicated with dotted blue lines.

The limited convex weight space W'_n may also be represented as the space spanned by convex combinations of its corners [10]. We first describe how the corners of W'_n can be used for refitting k-DOPs, and we then discuss the computation of the corners themselves.

5.2.1. 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 k-DOPs in Eq. (8),

$$D = \sum_{j=1}^{n} \tilde{w}_j D_j = \sum_{j=1}^{n} \sum_{i=1}^{m} u_i w'_{ij} D_j = \sum_{i=1}^{m} u_i D'_i. \tag{10}$$

From this expression, we can conclude that the deformed vertices can be bounded by first computing a combined k-DOP $D'_i = \sum_{j=1}^{n} w'_{ij} D_j$ for each corner of W'_n, and then bounding all the combined k-DOPs. This is, in essence, the algorithm proposed by Kavan and Zara [9] for sphere-trees in linear blend skinning, but in Section 5.3 we demonstrate its inefficiency for meshless animation.

5.2.2. Computation of corners

As noted by Kavan and Zara [9], 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 Fig. 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 $\mathbf{w}' = (l_1, \ldots, 1 - h_j - \sum_{k \notin \{i,j\}} l_k, \ldots, h_j, \ldots, l_n)$.

5.3. Efficient evaluation of extreme corners

As noted in Section 5.2.1, the deformed vertices can be bounded by computing a combination of k-DOPs for each corner of W'_n, and then bounding all the combined k-DOPs $\{D'_i\}$. Since there are $m = O(n^2)$ corners, and evaluating each combined k-DOP 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 k-DOP is defined simply by k extreme values along the k directions, and it would suffice to evaluate the corners that realize the k extreme values. In fact, with our definition of corners introduced in Section 5.2.2, selecting the corner that realizes each extreme value has a cost $O(n)$.

Let us pick a direction γ from the set of k directions (these directions are $\{x^+, x^-, y^+, y^-, z^+, z^-\}$ for an AABB). Given the k-DOPs $\{D_j\}$ associated with the n simulation nodes, we define as b_j^γ the extreme value of each k-DOP D_j along γ. Then, we identify the simulation node $j_1 = \operatorname{argmax}_j b_j^\gamma$ that realizes the largest extreme, as well as the second largest, $j_2 = \operatorname{argmax}_{j \neq j_1} b_j^\gamma$. As proved in Appendix A, the corner that realizes the extreme along γ is defined as

$$\mathbf{w}^\gamma = \left(l_1, \ldots, h_{j_1}, \ldots, 1 - h_{j_1} - \sum_{j \notin \{j_1, j_2\}} l_j, \ldots, l_n \right). \tag{11}$$

And the value of the extreme itself can be computed as

$$b^\gamma = h_{j_1} b_{j_1}^\gamma + (1 - h_{j_1} - \sum_{j \notin \{j_1, j_2\}} l_j) b_{j_2}^\gamma + \sum_{j \notin \{j_1, j_2\}} l_j b_j^\gamma. \tag{12}$$

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

5.4. Summary of k-DOP refitting

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

(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 5.1.
(2) Compute the transformed k-DOPs $\{D_j\}$ that bound $\{B_j\}$.
(3) For each orientation γ of the k-DOPs, identify the simulation nodes whose transformed k-DOPs realize the two largest extrema, and evaluate the bound b^γ based on Eq. (12).

6. Results

We have tested the tightness of BVs computed using our algorithm, the scalability of the approach, and its performance on several benchmark examples. We have also compared AABBs against higher-order k-DOPs (14-DOPs). All tests were carried out on a 3.4 GHz Pentium-4 PC with 1 GB of memory.

Fig. 1 shows a brain model deformed under pulling forces. In this scenario, we have evaluated the tightness of AABBs computed using 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. Fig. 5 shows the ratio between the radius of AABBs computed using 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. [13], 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 Fig. 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 4.2.

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 Fig. 6. Six *Santa Claus* models are attached to a ring. The ring is then rotated, producing deformations and collisions of the models. As listed in Table 2, we have tested models with ~3K and 45K vertices (18K and 270K in total in the scene), and with 115 and 550 simulation nodes (690 and 3300 in total in the scene). We have also considered two different ring motions, which produce different contact scenarios.

In Table 3 we report timings for collision queries with two state-of-the-art approaches for the benchmarks listed in Table 2: (i) AABB trees with full bottom-up update [16], and (ii) spatial hashing [7]. The collision query consists of finding all intersecting triangles, Note that, in both cases, timings are independent of the number of simulation nodes, and all bounding volumes are optimal, as they are directly evaluated from vertex positions. It is clear from the data that spatial hashing is not competitive in these benchmarks, but it would be better suited for detecting self-collisions.

In Table 4, we report timings (in ms) for the same benchmarks using our novel method (including front-tracking as discussed in Section 4.2). We also report the performance gain compared with full bottom-up update of AABB-trees, for which timings are given in Table 3. In the left part of Table 4, we have used AABBs both in rest state and in deformed state. With 45K vertices, the speed-up for refitting AABBs is between 69 and 126 times, and the total speed-up is between 15 and 27. The collision query 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, and one may extrapolate from the data that our method would provide even higher speed-up with more complex surfaces.

In the right part of Table 4, we have used OBBs in rest state and 14-DOPs in deformed state and on which the actual collision query is performed. Even though in both the rest state and the deformed state these BVs bound the vertices more tightly than AABBs, the speed-up of this approach compared to the AABB/AABB approach is not

Table 1
Scalability analysis

# 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

Time (in μs) for fitting an AABB to the brain model from Fig. 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.

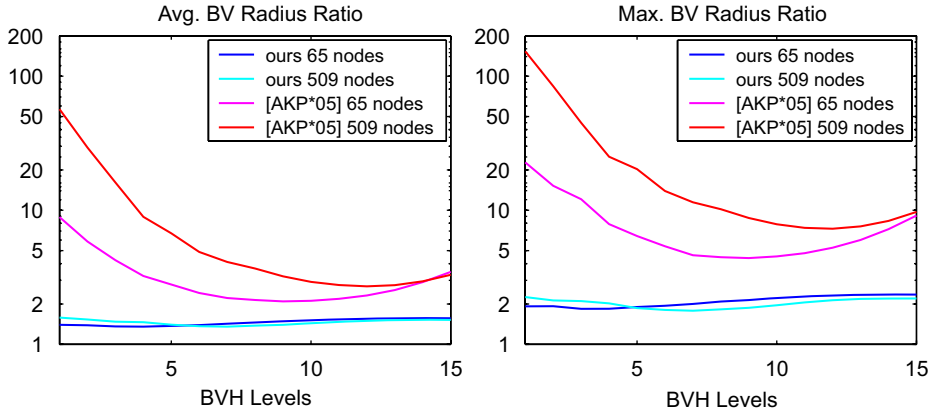

Fig. 5. BV tightness analysis. Ratios between the radii of AABBs computed using our method and the optimal AABBs, for all levels of the AABB-tree, on the brain model of Fig. 1. The left plot shows the average ratio over the course of a simulation, while the right plot shows the maximum ratio for each level. Our method largely improves the tightness of the sphere-tree of Adams et al. [13].

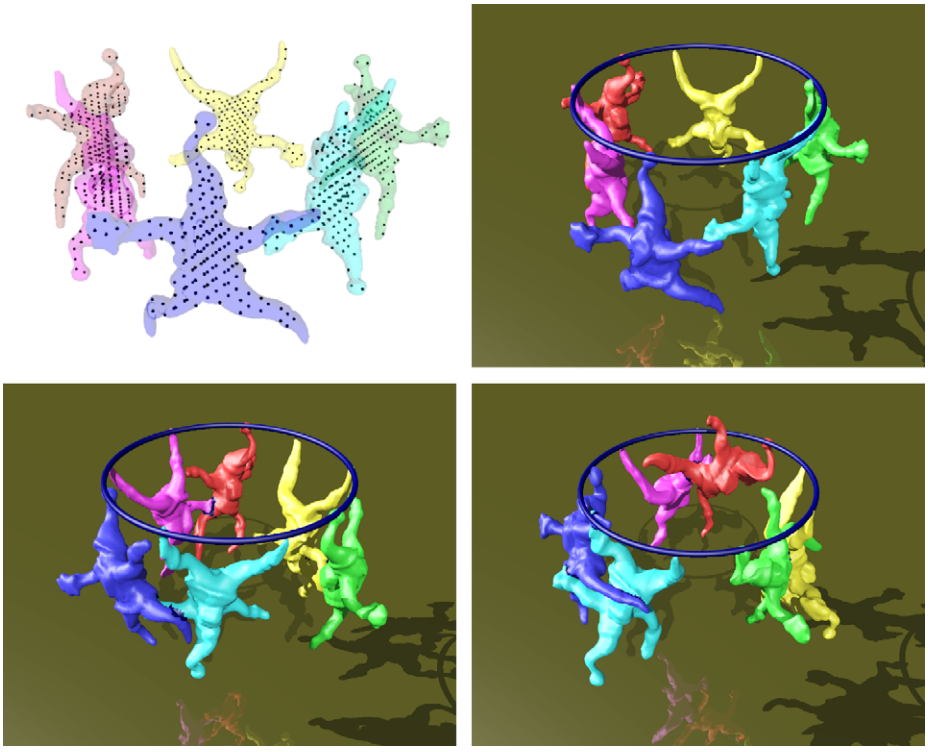

Fig. 6. Santa Claus models with meshless deformations. When the top ring moves, the models deform and collide with each other. The top-left image shows the sampling of nodes, while the bottom images highlight intersecting triangles.

Table 2
Scene types

Scene	1	2	3	4	5	6	7	8
# verts	2857	2857	45682	45682	2857	2857	45682	45682
# nodes	115	550	115	550	115	550	115	550
# contacts	Few	Few	Few	Few	Many	Many	Many	Many

Different settings (# vertices and # nodes per model, and contact scenario) for the 'Santa Claus' benchmark shown in Fig. 6.

Table 3
Performance analysis of state-of-the-art methods

Scene no.	AABBs full bottom-up			Spatial hashing		
	Refit	Query	Total	Load	Query	Total
1, 2	37.1	2.12	39.3	28.29	41.63	70.0
3, 4	575.7	5.89	581.6	479.6	1264.5	1744.1
5, 6	37.9	2.48	40.4	28.91	44.32	73.2
7, 8	569.7	8.84	578.6	484.2	1430.5	1914.7

Timings (in ms) for collision queries for the benchmarks of Table 2, using (i) AABB trees with full bottom-up update and (ii) spatial hashing.

Table 4
Performance analysis

Scene no.	AABB/AABB				OBB/14-DOP			
	Refit	Query	Total	Gain	Refit	Query	Total	Gain
1	1.83	5.72	7.55	5.2x	2.82	7.82	10.64	3.7x
2	2.36	5.60	7.97	4.9x	4.05	8.48	12.53	3.1x
3	4.54	16.6	21.1	27.5x	6.85	21.89	28.74	20.3x
4	5.77	18.7	24.5	23.7x	8.48	25.37	33.85	17.2x
5	3.38	10.0	13.4	3.0x	3.89	10.27	14.17	2.9x
6	3.26	9.28	12.5	3.2x	5.41	13.00	18.42	2.2x
7	6.87	24.5	31.4	18.4x	7.91	26.98	34.89	16.6x
8	8.23	29.3	37.5	15.4x	11.39	38.6	49.99	11.6x

Timings (in ms) for BVH update and collision queries for the benchmark of Fig. 6, with various vertex and simulation node resolutions, and under different contact scenarios. On the left, we use AABBs in both rest state and deformed state, while on the right, we use OBBs in rest state and 14-DOPs in deformed state.

quite as good, as can be seen in Table 4. The reason for this is that the transformation of an AABB can be implemented more efficiently than for an OBB. For both types of bounding boxes, one can exploit symmetry and instead of transforming all corners separately (as explained in Section 5.1), one may transform just the center and the three axes defining a box. For an AABB, an axis vector contains only one non-zero element, and therefore an axis-transformation amounts to a single scalar–vector multiplication. For an OBB one must compute a matrix–vector product for each axis and the center. Furthermore, overlap tests for 14-DOPs are more expensive than for AABBs, as more extrema must be computed. We have also tested the combination of AABBs in rest state and 14-DOPs in deformed state, and we found that it is about 5–10% slower than the OBB/14-DOPs combination. The reason for this is that the AABBs and k-DOPs are much less tight, resulting in more BV updates, while the

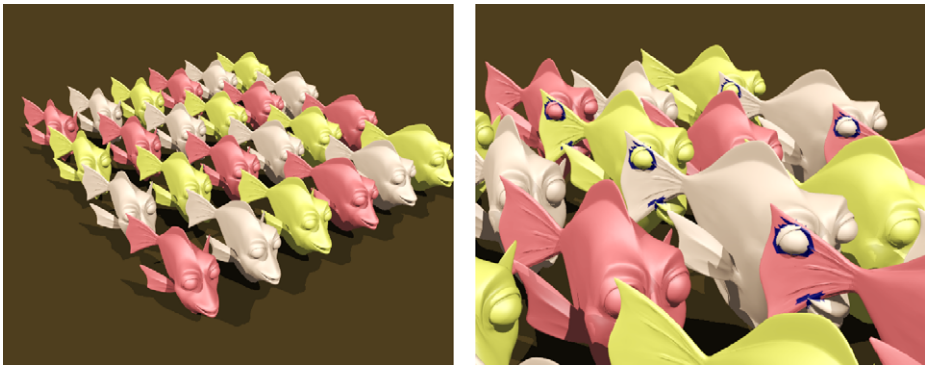

Fig. 7. Collisions between deforming fishes. Our method provides a speed-up of 12 times in this scenario. Intersecting triangles are highlighted on the right.

more efficient AABB transformation cannot quite make up for this.

We have only reported timings for 14-DOPs, but further experiments with 18-DOPs and 26-DOPs have shown that the better tightness of these BVs does not pay off due to the larger number of extrema that must be computed. The reason for using these three types of k-DOPs is that their direction vectors can be represented by integer values -1, 0, and 1, making it possible to project a point onto a direction without any multiplications [15]. Finally, one could also use tight k-DOPs in undeformed state. In this case, however, the BV transformation becomes too expensive, since BV symmetry cannot be exploited anymore.

Therefore, we conclude that using both AABBs in rest state and deformed state yield the highest performance gain compared to the two state-of-the-art methods shown in Table 3. In scenarios with few degrees of freedom and even more collisions, using tighter but more complex BVs may pay off, but more experiments would need to be done to confirm this.

We have also tested the performance on the scene of Fig. 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.

7. Conclusion

In this paper, we have presented a fast method for computing tight AABBs in the context of meshless 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.

As noted by an anonymous reviewer, our extrapolation of the deformation field to the surface might be replaced by a shape function formulation [20]. Our contributions for tight and efficient bound computation would still hold, but applied to combinations of displacements instead of linear transformations.

We are investigating extensions for topology changes (i.e., cutting and fracture) by restructuring the hierarchies [24], 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.

Acknowledgments

We would like to thank the anonymous reviewers, the editors, Ming C. Lin and Nico Galoppo for the fish model, and 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.

Appendix A. Extreme corner evaluation

As shown in Section 5.2.2, 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 (12) 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. Let us assume, w.l.o.g., that $j_1=1$ and $j_2=2$. In total, there are seven choices for the weighting schemes of the simulation nodes, as shown in Table A1.

Let us recall here Definition (12) of the extreme, renamed as b^1, and dropping the superindex γ for clarity

$$b^1 = h_1 b_1 + \left(1 - h_1 - \sum_{j \notin \{1,2\}} l_j\right) b_2 + \sum_{j \notin \{1,2\}} l_j b_j. \tag{A.1}$$

In order to prove that this extreme is a conservative bound, it suffices to prove that the difference $b^1 - b^i \geqslant 0$ for all

Table A1
Weighting schemes

Weights	w_1	w_2	w_3	w_4	Others
$\mathbf{w}_1$	h	$1-h-\sum l$	l	l	l
$\mathbf{w}_2$	h	l	$1-h-\sum l$	l	l
$\mathbf{w}_3$	$1-h-\sum l$	h	l	l	l
$\mathbf{w}_4$	$1-h-\sum l$	l	h	l	l
$\mathbf{w}_5$	l	h	$1-h-\sum l$	l	l
$\mathbf{w}_6$	l	$1-h-\sum l$	h	l	l
$\mathbf{w}_7$	l	l	h	$1-h-\sum l$	l

All possible convex weighting possibilities of corners in W'_n. Assuming that the first and second node realize the two largest bounds, b_1 and b_2, the weighting scheme $\mathbf{w}_1$ realizes the extreme corner.

other six possible weighting schemes, subject to weights $\mathbf{w} \in W'_n$:

Weighting scheme $\mathbf{w}_2$:

$$b^1 - b^2 = \left(1 - h_1 - \sum_{j \neq 1} l_j\right)(b_2 - b_3). \tag{A.2}$$

For the first factor, note that $w_3 = 1 - h_1 - \sum_{j \notin \{1,3\}} l_j$, as given in Table A1. Then, $(1 - h_1 - \sum_{j \neq 1} l_j) = w_3 - l_3$, and since $\mathbf{w}_2 \in W'_n$, then $w_3 \geqslant l_3$. For the second factor, by definition of $j = 2$ as the node realizing the second largest value, $b_2 - b_3 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^2$.

Weighting scheme $\mathbf{w}_3$:

$$b^1 - b^3 = \left(h_1 - \left(1 - h_2 - \sum_{j \notin \{1,2\}} l_j\right)\right)(b_1 - b_2). \tag{A.3}$$

For the first factor, $h_1 - (1 - h_2 - \sum_{j \notin \{1,2\}} l_j) = h_1 - w_1$, and since $\mathbf{w}_3 \in W'_n$, then $h_1 \geqslant w_1$. For the second factor, by definition of $j = \{1, 2\}$ as the nodes realizing the two largest values, $b_1 - b_2 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^3$.

Weighting scheme $\mathbf{w}_4$:

$$b^1 - b^4 = \left(h_1 - \left(1 - h_3 - \sum_{j \notin \{1,3\}} l_j\right)\right)(b_1 - b_2) + (h_3 - l_3)(b_2 - b_3). \tag{A.4}$$

For the first factor, $h_1 - (1 - h_3 - \sum_{j \notin \{1,3\}} l_j) = h_1 - w_1$, and since $\mathbf{w}_4 \in W'_n$, then $h_1 \geqslant w_1$. For the third factor, $h_3 \geqslant l_3$ by definition. For the second and fourth factors, by definition of $j = \{1, 2\}$ as the nodes realizing the two largest values, $b_1 - b_2 \geqslant 0$ and $b_2 - b_3 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^4$.

Weighting scheme $\mathbf{w}_5$:

$$b^1 - b^5 = \left(\left(1 - h_2 - \sum_{j \notin \{2,3\}} l_j\right) - l_3\right)(b_2 - b_3) + (h_1 - l_1)(b_1 - b_2). \tag{A.5}$$

For the first factor, $(1 - h_2 - \sum_{j \notin \{2,3\}} l_j) - l_3) = w_3 - l_3$, and since $\mathbf{w}_5 \in W'_n$, then $w_3 \geqslant l_3$. For the third factor, $h_1 \geqslant l_1$ by definition. For the second and fourth factors, by definition of $j = \{1, 2\}$ as the nodes realizing the two largest values, $b_2 - b_3 \geqslant 0$ and $b_1 - b_2 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^5$.

Weighting scheme $\mathbf{w}_6$:

$$b^1 - b^6 = (h_1 - l_1)(b_1 - b_2) + (h_3 - l_3)(b_2 - b_3). \tag{A.6}$$

For the first factor and third factors, $h_1 \geqslant l_1$ and $h_3 \geqslant l_3$ by definition. For the second and fourth factors, by definition of $j = \{1, 2\}$ as the nodes realizing the two largest values, $b_1 - b_2 \geqslant 0$ and $b_2 - b_3 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^6$.

Weighting scheme $\mathbf{w}_7$:

$$b^1 - b^7 = (h_1 - l_1)(b_1 - b_2) + (h_3 - l_3)(b_2 - b_3) + \left(\left(1 - h_3 - \sum_{j \notin \{3,4\}} l_j\right) - l_4\right)(b_2 - b_4). \tag{A.7}$$

The first four factors correspond to $b^1 - b^6$. For the fifth factor, $(1 - h_3 - \sum_{j \notin \{3,4\}} l_j) - l_4 = w_4 - l_4$, and since $\mathbf{w}_7 \in W'_n$, then $w_4 \geqslant l_4$. For the sixth factor, by definition of $j = 2$ as the node realizing the second largest value, $b_2 - b_4 \geqslant 0$. Hence, it follows that $b^1 \geqslant b^7$.

References

[1] Müller M, Keiser R, Nealen A, Pauly M, Gross M, Alexa M. Point based animation of elastic, plastic and melting objects. In: Proceedings of eurographics/ACM SIGGRAPH symposium on computer animation; 2004.

[2] Pauly M, Keiser R, Adams B, Dutré P, Gross M, Guibas LJ. Meshless animation of fracturing solids. In: Proceedings of ACM SIGGRAPH; 2005.

[3] Wicke M, Steinemann D, Gross M. Efficient animation of point-sampled thin shells. In: Proceedings of eurographics; 2005.

[4] Steinemann D, Otaduy MA, Gross M. Fast arbitrary splitting of deforming objects. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on computer animation; 2006.

[5] Keiser R, Adams B, Gasser D, Bazzi P, Dutre P, Gross M. A unified lagrangian approach to solid–fluid animation. In: Proceedings of eurographics symposium on point-based graphics; 2005.

[6] James DL, Pai DK. BD-tree: output-sensitive collision detection for reduced deformable models. In: Proceedings of ACM SIGGRAPH; 2004.

[7] Teschner M, Heidelberger B, Müller M, Pomeranets D, Gross M. Optimized spatial hashing for collision detection of deformable objects. In: Proceedings of VMV; 2003.

[8] Gottschalk S, Lin M, Manocha D. OBB-tree: a hierarchical structure for rapid interference detection. In: Proceedings of ACM SIGGRAPH; 1996. p. 171–80.

[9] Kavan L, Zara J. Fast collision detection for skeletally deformable models. In: Proceedings of eurographics; 2005.

[10] Kavan L, O'Sullivan C, Zara J. Efficient collision detection for spherical blend skinning. In: Proceedings of graphite; 2006.

[11] Mendoza C, O'Sullivan C. Interruptible collision detection for deformable objects. Computers & Graphics 2006;30(2).

[12] Otaduy MA, Germann D, Redon S, Gross M. Adaptive deformations with fast tight bounds. In: Proceedings of ACM SIGGRAPH/Eurographics symposium on computer animation; 2007.

[13] Adams B, Keiser R, Pauly M, Guibas LJ, Gross M, Dutre P. Efficient raytracing of deforming point-sampled surfaces. In: Proceedings of eurographics; 2005.

[14] Spillmann J, Becker M, Teschner M. Efficient updates of bounding sphere hierarchies for geometrically deformable models. In: Proceedings of VRIPHYS; 2006.

[15] Klosowski J, Held M, Mitchell JSB, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-DOPs. IEEE Transactions on Visualization and Computer Graphics 1998;4(1).
[16] Van den Bergen G. Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools 1997;2(4).
[17] Müller M, Heidelberger B, Teschner M, Gross M. Meshless deformations based on shape matching. In: Proceedings of ACM SIGGRAPH; 2005.
[18] Müller M, Heidelberger B, Hennix M, Ratcliff J. Position based dynamics. In: Proceedings of VRIPHYS; 2006.
[19] Rivers A, James DL. FastLSM: fast lattice shape matching for robust real-time deformation. In: Proceedings of ACM SIGGRAPH; 2007.
[20] Fries TP, Matthies HG. Classification and overview of meshfree methods. Technical Report, TU Brunswick, Germany; 2004.
[21] 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.
[22] Hubbard PM. Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics 1995; 15(3).
[23] Klug T, Alexa M. Bounding volumes for linearly interpolated shapes. In: Proceedings of computer graphics international; 2004.
[24] Otaduy MA, Chassot O, Steinemann D, Gross M. Balanced hierarchies for collision detection between fracturing objects. In: Proceedings of IEEE virtual reality conference; 2007.