

# Bounded Normal Trees for Reduced Deformations of Triangulated Surfaces

Sara C. Schwartzman, Jorge Gascón & Miguel A. Otaduy

URJC Madrid, Spain<sup>†</sup>

---

## Abstract

*Several reduced deformation models in computer animation, such as linear blend skinning, point-based animation, embedding in finite element meshes, cage-based deformation, or subdivision surfaces, define surface vertex positions through convex combination of a rather small set of linear transformations. In this paper, we present an algorithm for computing tight normal bounds for a surface patch with an arbitrary number of triangles, with a cost linear in the number of governor linear transformations. This algorithm for normal bound computation constitutes the key element of the Bounded Normal Tree (BN-Tree), a novel culling data structure for hierarchical self-collision detection. In situations with sparse self-contact, normal-based culling can be performed with a small output-sensitive cost, regardless of the number of triangles in the surface.*

---

## 1. Introduction

Self-collision refers to the intersection of an object's surface with itself. Self-collision poses an important complexity challenge onto typical acceleration data structures for collision detection, because pairs of geodesically nearby surface primitives cannot be efficiently pruned away. Many applications in computer animation, such as cloth animation [BFA02], character skinning [LCF00], or soft-tissue cutting [SOG06], are likely to exhibit self-collision phenomena, hence the importance of efficient solutions.

In this work, we address self-collision detection of surfaces with a high triangle count that are deformed through their embedding in a reduced deformation field with relatively few degrees of freedom. In particular, we address reduced deformations where the position of each surface vertex is defined as a convex combination of linear transformations. This type of reduced deformation has multiple examples in computer animation: (i) character deformation through linear blend skinning or skeleton subspace deformation [LCF00], (ii) cage-based deformation using convex basis functions [JMD\*07], (iii) embedding in low-resolution finite-element meshes, (iv) subdivision surfaces applied, e.g., to post-processing cloth animation [BFA02], or (v) point-based animation [MKN\*04].

Our work complements hierarchical collision detection algorithms, in particular the approach for self-collision detection of Volino and Magnenat-Thalmann [VMT94]. Their algorithm computes in a bottom-up fashion a tree of bounding volumes and normal bounds, and executes self-collision detection by top-down traversal of the tree. In this paper, we present two main contributions:

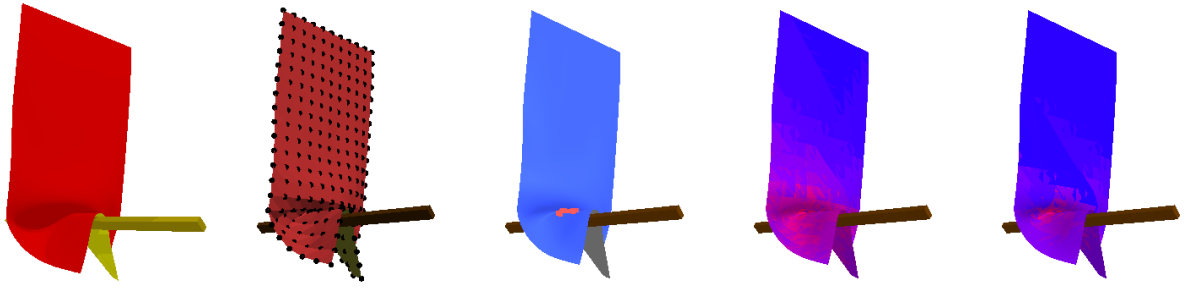
1. We have designed a novel algorithm for efficient computation of normal bounds (See Section 3). Given a patch with  $n$  triangles whose vertices are defined by convex combinations of  $m$  linear transformations, our algorithm computes a tight normal bound with cost  $O(m)$ . On dense surface patches with  $n \gg m$ , our algorithm produces large speed-ups. As a substep of the algorithm, we compute efficient bounds of the deformation gradient of a patch.
2. We have incorporated our efficient evaluation of normal bounds into an algorithm for self-collision detection (See Section 4). We refer to it as *Bounded Normal Tree* (BN-Tree). The algorithm employs on-demand update of normal bounds and exploits temporal coherence through front caching with conservative ascent.

## 2. Related Work

Bounding volume hierarchies (BVHs) [GLM96] have long been used as acceleration data structures for collision

---

<sup>†</sup> <http://www.gmr.v.es/Publications/2009/SGO09/>



**Figure 1:** From left to right: 13041-vertex cloth computed using Loop subdivision; 231-vertex control mesh; self-collisions shown in red; normal test front of the BN-Tree; and normal test front using state-of-the-art hierarchical self-collision detection [VMT94]. In the color-coded test front, blue means that self-collisions are pruned high in the tree, while red means that the tests reach the leaves. With bluish color, normal bounds are computed high in the BN-Tree, leading to high efficiency.

detection. When applied to general deformation models, BVHs are updated in a bottom-up manner every simulation frame [dB97]. If the choice of bounding volumes (BVs) is one of spheres, AABBs, or  $k$ -DOPs, the cost of the update is linear in the number of vertices, 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.

As mentioned in the introduction, self-collision detection poses an additional complexity, because geodesically nearby surface primitives cannot be easily pruned away. Most of the research on self-collision detection has targeted cloth animation. Volino and Magnenat-Thalmann [VMT94] presented a hierarchical algorithm that exploits normal and contour conditions for pruning large surface patches (See more details in Section 4.1). Their algorithm incurs an  $O(n)$  cost for updating the hierarchy. Their initial approach employed discrete normal cones, while Provot [Pro97] used actual cones. Baciú and Wong [BW02] adopted many of these ideas into a parallel algorithm and implemented it on graphics hardware. Mezger et al. [MKE03] used oriented inflated  $k$ -DOPs together with other heuristics. Govindaraju et al. [GKJ\*05] proposed a chromatic decomposition of a triangle mesh in order to circumvent adjacency-related problems. More recently, Tang et al. [TCYM08] have incorporated additional adjacency-related optimizations, as well as an extension of Volino’s normal criterion to the continuous collision detection setting. Self-collision detection has also been addressed with other data structures and algorithms, such as spatial partitioning data structures optimized through hashing [THM\*03] or visibility-based culling [GRLM03]. Hierarchies of normal cones have been used for culling in other problems such as general proximity queries [JC01], and half-space intersections (which are somewhat more involved but related to normal cones) have been used for hierarchical back-face culling [KMGL99]. The dynamic update of normal cones is easily computed as a rotation under rigid trans-

formations, but general deformations require visiting each and every triangle bounded by the cone.

Several researchers have designed collision detection algorithms with a potentially sublinear cost on the number of vertices, for deformation models governed by few degrees of freedom. Larsson and Akenine-Möller [LAM03] applied those ideas to morphing, while Klug and Alexa [KA04] later improved them for linearly interpolated shapes. 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 meshless animations [AKP\*05].

Kavan and Zara [KZ05] computed efficient BVs for skinned articulated bodies, with each surface vertex defined by a convex combination of rigid transformations. Given  $m$  rigid transformations, they found a set of *limited convex combinations* defined by a set of  $O(m^2)$  corners in  $\mathbb{R}^m$ . Then, finding the bound of a surface patch, regardless of the number of vertices, had a cost  $O(m^2)$ . The use of limited convex combinations has been extended to spherical blend skinning [KOZ06] and FEM deformations on coarse meshes [OGRG07]. Steinemann et al. [SOG08] found a way to compute bounds more efficiently, by identifying extreme corners with cost  $O(m)$ . Their algorithm was initially used on point-based animations, but it is applicable to any deformation defined through convex combination of linear transformations. Approaches for bound computation based on limited convex combinations are not directly applicable to the computation of normal bounds, as surface normals are quadratic in vertex positions. One may then think of augmenting the coordinate set to include quadratic position terms, but this would lead to an explosion into  $O(m^2)$  products of the original linear transformations.

Our work is perhaps closest in spirit to the one of Grinspun and Schröder [GS01], who proposed a computation of

interference of subdivision surfaces by efficient evaluation of normal bounds. Their work differs from ours, however, both in the method and its target application. Theirs exploits bounds of partial derivatives of the eigenvectors of subdivision matrices and is applicable to the limit surfaces produced by subdivision schemes. Ours, on the other hand, exploits bounds of a triangle's deformation gradient and is applicable to triangle meshes defined through convex combination of arbitrary linear transformations.

### 3. Efficient Normal Bounds

In this section, we describe our main contribution: An efficient method to compute a normal bound for a surface patch  $S$  with  $n$  triangles, whose vertices are defined through convex combinations of  $m$  linear transformations. The normal bound of  $S$  is computed with a cost  $O(m)$ , for a general case where the number of transformations,  $m$ , is much smaller than the number of triangles  $n$ .

The naïve approach to solve this problem would be to define transformed normals using the cross product of edge vectors, i.e.,  $\mathbf{n} = (\mathbf{a} - \mathbf{b}) \times (\mathbf{a} - \mathbf{c})$ , bound the various operands of this expression for all triangles in the patch, and then bound the cross product operation using interval arithmetic. However, edge vectors are not spatially coherent across triangles, hence bounding this expression yields typically a useless normal cone that spans the complete sphere.

Assuming spatially coherent triangle normals in the undeformed state, together with spatially coherent convex transformation weights, our approach for computing a bound of the normal will be the following: We will define the normal of each deformed triangle through an operation composed by spatially coherent operands; we will bound the normal of all triangles in the patch  $S$  by first bounding the spatially coherent operands, and then bounding their composition.

In the rest of this section, we describe first the expression we use for defining transformed normals, based on a triangle's deformation gradient. Hence, next we describe how to formulate a triangle's deformation gradient using spatially coherent operands. Finally, we discuss the run-time computation of bounds for the deformation gradient and the normals for a surface patch.

#### 3.1. Transformed Normals

Barr [Bar84] devised an expression for transforming the normal of a smooth surface given the deformation gradient  $\mathbf{J}$ . The deformation gradient is constant inside a triangle, hence the rest-state normal of a triangle,  $\mathbf{n}_0$ , can be transformed using Barr's expression as

$$\mathbf{n} = \mathbf{M}\mathbf{n}_0 = \det\mathbf{J} \cdot \mathbf{J}^{-T} \mathbf{n}_0. \quad (1)$$

With a column-wise expression of  $\mathbf{J} = (\mathbf{j}_1 \ \mathbf{j}_2 \ \mathbf{j}_3)$ , the transformation can be more easily computed as

$$\mathbf{M} = (\mathbf{j}_2 \times \mathbf{j}_3 \ \mathbf{j}_3 \times \mathbf{j}_1 \ \mathbf{j}_1 \times \mathbf{j}_2). \quad (2)$$

There are other alternatives for defining the deformed normal, such as using the cross product of triangle-edge vectors, or the cross product of surface partial derivatives. However, neither of them is well suited in our case. Triangle-edge vectors are not spatially coherent, as discussed above, and the partial derivatives cannot be obtained from an analytic expression for a triangulated surface. The use of Barr's expression, however, decomposes the computation of the normal into operands that are indeed spatially coherent: (i) the input normal and (ii) the deformation gradient.

The deformation gradient is not uniquely defined for a triangle. Botsch et al. [BSPG06] discuss extensively this issue, and they propose the following formula, which depends only on the initial and deformed positions of the triangle vertices. Given initial vertex positions  $\{\mathbf{a}_0, \mathbf{b}_0, \mathbf{c}_0\}$  and deformed positions  $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , they define the deformation gradient

$$\mathbf{J} = (\mathbf{a} \ \mathbf{b} \ \mathbf{c}) \mathbf{G}, \quad (3)$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} (\mathbf{a}_0 - \mathbf{c}_0) & (\mathbf{b}_0 - \mathbf{c}_0) & \mathbf{n}_0 \end{pmatrix}^{-1}.$$

The rows of the matrix  $\mathbf{G}$  represent the gradients of a triangle's basis functions.

We have considered other alternatives for computing the deformation gradient, such as using a fourth point per triangle, as done by Sumner et al. [SP04]. However, the choice of fourth point is not straightforward, and this alternative tends to suffer from amplification due to the inverse of badly conditioned matrices, which negatively affects the computation of bounds. We obtained the best results with our approach.

#### 3.2. Transformed Points

To calculate the new positions of the vertices we will assume they are defined as a convex combination of *governor* linear transformations:

$$\mathbf{p} = \sum w_k (\mathbf{A}_k \mathbf{p}_0 + \mathbf{t}_k). \quad (4)$$

This assumption applies to the deformation models discussed in the Introduction, such as subdivision surfaces (with control points as governors), linear blend skinning (with bones as governors), or point-based animation (with particles as governors). Convex combinations imply that  $0 \leq w_k \leq 1$  and  $\sum w_k = 1$ .

Instead of directly using the above expression for defining vertex positions, we employ their relative position w.r.t. the centroid of the patch. Specifically, we express the rest-state vertex positions as  $\mathbf{p}_0 = \Delta\mathbf{p} + \mathbf{x}_0$ , with  $\mathbf{x}_0$  the rest-state patch centroid. The transformation of the centroid can be extracted from Eq. (4), which yields modified translations  $\bar{\mathbf{t}}_k = \mathbf{A}_k \mathbf{x}_0 + \mathbf{t}_k$ . Altogether, the transformed vertex positions can be then defined as

$$\mathbf{p} = \sum w_k (\mathbf{A}_k \Delta\mathbf{p} + \bar{\mathbf{t}}_k). \quad (5)$$

Using relative vertex positions leads to tighter bounds for the deformation gradient, as the norm of the values under consideration becomes smaller. This is especially relevant for the term  $\mathbf{J}_k$  to be defined later in Eq. (15).

### 3.3. Decomposition of the Deformation Gradient

We decompose the deformed position of a vertex  $\mathbf{p}$  into three different terms: patch deformation  $\mathbf{p}_p$ , triangle deformation  $\mathbf{p}_t$ , and vertex deformation  $\mathbf{p}_v$ :

$$\mathbf{p} = \mathbf{p}_p + \mathbf{p}_t + \mathbf{p}_v. \quad (6)$$

These three terms, which we define in detail below, carry the following information. The patch-deformation term captures the per-patch average deformation of all triangles. Given the remaining deformation, the triangle-deformation term captures the average deformation for the three vertices in a triangle. And the vertex-deformation term captures the remaining in-triangle deformation. The decomposition of vertex positions leads to an analogous decomposition of the deformation gradient,

$$\mathbf{J} = \mathbf{J}_p + \mathbf{J}_t + \mathbf{J}_v. \quad (7)$$

This decomposition favors the computation of tight bounds.

#### 3.3.1. Patch Deformation

For each patch, we compute an average linear transformation  $\mathbf{A}_p$  and an average translation  $\mathbf{t}_p$ . Then, we express the governor transformations as  $\mathbf{A}_k = \mathbf{A}_p + \Delta\mathbf{A}_k$ ,  $\mathbf{t}_k = \mathbf{t}_p + \Delta\mathbf{t}_k$ . Thanks to the properties of convex weights, we can extract the patch-deformation term from Eq. (5):

$$\begin{aligned} \mathbf{p} &= \mathbf{p}_p + \sum w_k(\Delta\mathbf{A}_k\Delta\mathbf{p} + \Delta\mathbf{t}_k), \\ \text{with } \mathbf{p}_p &= \mathbf{A}_p\Delta\mathbf{p} + \mathbf{t}_p. \end{aligned} \quad (8)$$

From Eq. (3), and substituting the contribution to vertex positions due to patch deformation, we can now define the patch deformation gradient for a triangle,

$$\mathbf{J}_p = \begin{pmatrix} \mathbf{A}_p\Delta\mathbf{a} + \mathbf{t}_p & \mathbf{A}_p\Delta\mathbf{b} + \mathbf{t}_p & \mathbf{A}_p\Delta\mathbf{c} + \mathbf{t}_p \end{pmatrix} \mathbf{G}. \quad (9)$$

The contribution of the translations cancels out because the gradients of the basis functions (i.e., the rows of  $\mathbf{G}$ ) add up to zero. Then, the patch deformation gradient can be summarized as:

$$\mathbf{J}_p = \mathbf{A}_p\mathbf{J}_0, \quad \text{with } \mathbf{J}_0 = \begin{pmatrix} \Delta\mathbf{a} & \Delta\mathbf{b} & \Delta\mathbf{c} \end{pmatrix} \mathbf{G}. \quad (10)$$

#### 3.3.2. Triangle Deformation

For each triangle-governor pair, we define an average weight  $\bar{w}_k$  as the mean of the weights of that governor for the three vertices of the triangle. By separating the average weight in Eq. (8), we can also separate the triangle and vertex deformation terms:

$$\mathbf{p}_t = \sum \bar{w}_k(\Delta\mathbf{A}_k\Delta\mathbf{p} + \Delta\mathbf{t}_k), \quad (11)$$

$$\mathbf{p}_v = \sum (w_k - \bar{w}_k)(\Delta\mathbf{A}_k\Delta\mathbf{p} + \Delta\mathbf{t}_k). \quad (12)$$

By plugging the triangle-deformation terms into the definition of the gradient, Eq. (3), we obtain the triangle deformation gradient. The use of average weights simplifies this deformation gradient in two ways. First, since we use average weights, the transformations of the three triangle vertices are the same. Second, the use of average weights together with the fact that the basis-function gradients add up to zero, cancel out the contribution of the translation. Altogether, the triangle deformation gradient can be expressed as

$$\mathbf{J}_t = \mathbf{A}_t\mathbf{J}_0, \quad \text{with } \mathbf{A}_t = \sum \bar{w}_k\Delta\mathbf{A}_k. \quad (13)$$

#### 3.3.3. In-Triangle Vertex Deformation

The remaining in-triangle vertex deformation,  $\mathbf{p}_v$ , defined in Eq. (12), yields the following term of the deformation gradient by substitution into Eq. (3):

$$\begin{aligned} \mathbf{J}_v &= \sum (\Delta\mathbf{A}_k \ \Delta\mathbf{t}_k)\mathbf{J}_k, \\ \mathbf{J}_k &= \begin{pmatrix} (w_{ak} - \bar{w}_k)\Delta\mathbf{a} & (w_{bk} - \bar{w}_k)\Delta\mathbf{b} & (w_{ck} - \bar{w}_k)\Delta\mathbf{c} \\ w_{ak} - \bar{w}_k & w_{bk} - \bar{w}_k & w_{ck} - \bar{w}_k \end{pmatrix} \mathbf{G}. \end{aligned} \quad (14)$$

The in-triangle deformation gradient is not spatially coherent, hence it is crucial to minimize its magnitude. This is achieved by using vertex positions relative to the patch centroid, as discussed in Section 3.2, together with extracting the average transformations.

### 3.4. Bounding the Deformation Gradient

Substituting the various terms of the deformation gradient into Eq. (7), we obtain the complete decomposed expression for one triangle's deformation gradient:

$$\mathbf{J} = (\mathbf{A}_p + \mathbf{A}_t)\mathbf{J}_0 + \sum (\Delta\mathbf{A}_k \ \Delta\mathbf{t}_k)\mathbf{J}_k. \quad (15)$$

In this expression, the terms  $\mathbf{J}_0$ ,  $\mathbf{A}_t$ , and  $\mathbf{J}_k$  vary across the triangles in a patch. With spatially coherent weights, the term  $\mathbf{A}_t$  varies smoothly across triangles. With spatially coherent rest-state normals, the term  $\mathbf{J}_0$  varies smoothly across triangles as well. The term  $\mathbf{J}_k$  does not vary smoothly, but its magnitude is small compared to the other terms, as it depends only on in-triangle variations.

We bound the deformation gradient in a patch by bounding separately the various variable terms. We represent with  $[x]$  a variable that bounds each element of  $x$  with an interval. Then, applying interval arithmetic, we can compute the bound of the deformation gradient as

$$[\mathbf{J}] = (\mathbf{A}_p + [\mathbf{A}_t])[\mathbf{J}_0] + \sum (\Delta\mathbf{A}_k \ \Delta\mathbf{t}_k)[\mathbf{J}_k]. \quad (16)$$

We precompute the bounds  $[\mathbf{J}_0]$  and  $[\mathbf{J}_k]$  by simply bounding the per-triangle values as a pre-process. The bound of triangle average transformations,  $[\mathbf{A}_t]$ , needs to be computed at run-time. Since the average weights  $\bar{w}_k$  are convex, bounding  $\mathbf{A}_t$  reduces to a problem of bounding convex combinations of linear transformations. We apply the algorithm of Steinemann et al. [SOG08], which exploits limited convex combinations with efficient evaluation of extreme corners,

and bounds  $\mathbf{A}_t$  with cost  $O(m)$ , with  $m$  the number of transformations.

### 3.5. Bounding the Transformed Normals

For each patch, we precompute a bound of rest-state normals,  $[\mathbf{n}_0]$ . Then, following Eq. (1) and Eq. (2), and given the bound of the deformation gradient  $[\mathbf{J}]$ , we compute the normal bound of a patch using interval arithmetic as

$$[\mathbf{n}] = ([\mathbf{j}_2] \times [\mathbf{j}_3] \quad [\mathbf{j}_3] \times [\mathbf{j}_1] \quad [\mathbf{j}_1] \times [\mathbf{j}_2]) [\mathbf{n}_0]. \quad (17)$$

The normal bound  $[\mathbf{n}]$  described above can be regarded as an axis-aligned bounding box. Other possible representations include a normal cone defined by an axis-angle pair.

## 4. Self-Collision Detection Using BN-Trees

Normal bounds for hierarchical self-collision detection were introduced by Volino and Magnenat-Thalmann [VMT94]. In this section, we present a modified version of their algorithm, called the Bounded Normal Tree (BN-Tree), that computes normal bounds on-demand using the efficient algorithm presented in the previous section. We pay special attention to the elementary self-collision test based on normal bounds, and the traversal and update of the tree.

### 4.1. Elementary Self-Collision Test

The algorithm of Volino and Magnenat-Thalmann prunes surface patches  $S$  that do not self-collide if the following two conditions hold:

1. There exists a vector  $\mathbf{v}$  such that, for all triangles in  $S$ ,  $\mathbf{v}^T \mathbf{n}_i > 0$ , with  $\mathbf{n}_i$  the triangle normal.
2. The orthogonal projection of the contour of  $S$  onto a plane with normal  $\mathbf{v}$  does not self-intersect.

Following observations by Volino and Magnenat-Thalmann, we construct the BN-Tree by partitioning the surface into patches that maximize the area-perimeter ratio. In this way, the ‘contour test’ 2 above is extremely unlikely to fail. Hence, and also following Volino and Magnenat-Thalmann, we omitted the contour test in our implementation. Let us remark that the self-collision detection algorithm is not conservative without the contour test, and in Section 6 we discuss an extension to efficiently handle it.

The ‘normal test’ 1 above reduces to computing the intersection of the half-spaces defined by all triangle normals in the patch. Similar to Volino and Magnenat-Thalmann, we perform a slightly conservative version of the normal test by storing a *discrete normal cone* (DNC) consisting of a bit mask for 14 directions (6 for the directions of the Cartesian axes, and 8 for the corners of a cube aligned with the axes).

In the original algorithm by Volino and Magnenat-Thalmann, DNCs are computed bottom-up in a BVH, such that the DNC of a node is computed by intersecting the

DNCs of its children. The total cost for updating DNCs is then  $O(n)$ , with  $n$  the number of triangles in the surface. Instead, using our algorithm for computing normal bounds from the previous section, we can update a DNC on-demand with cost  $O(m)$ , with  $m$  the number of linear transformations that govern the deformation. Given a normal bound  $[\mathbf{n}]$ , a ‘true’ bit in the DNC denotes that all triangle normals in the patch have a positive dot product with the direction associated to that bit. The value of the bit is computed by checking whether the lower bound of the dot product between its associated direction and the bound  $[\mathbf{n}]$  is positive.

### 4.2. Hierarchical Self-Collision Detection

Given as input a connected triangle mesh, we construct as preprocessing a BN-Tree that successively partitions mesh patches into 2 connected components. If the triangle mesh is not connected, the various connected components may be treated separately. Note that we have used a branching factor of 2, but other branching factors are also possible.

At run-time, a self-collision detection test starts by issuing a self-collision query on the root node, and then proceeds hierarchically by performing three types of queries:

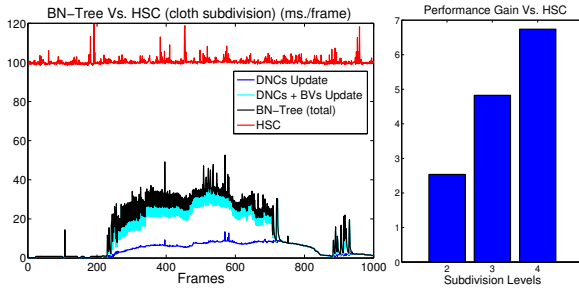
1. *self\_collide(a)* prunes the subtree rooted under node  $a$  if its corresponding surface patch fulfills the normal test described in Section 4.1. This test requires the computation of the DNC of  $a$ .
2. *self\_collide(a, b)* is a query on adjacent nodes  $a$  and  $b$ , and is executed similarly. It requires the computation of the intersection of the DNCs of  $a$  and  $b$ .
3. *collide(a, b)* is a query on disjoint nodes  $a$  and  $b$ , and it checks the collision between BVs of  $a$  and  $b$ .

Queries with a positive result descend recursively on children nodes, issuing *self\_collide()* or *collide()* queries based on whether the nodes are adjacent or disjoint. At the leaves of the tree, we perform primitive-level tests.

Before executing a *collide()* or *self\_collide()* test, we first check whether the BVs or DNCs have already been computed in the current simulation frame for the involved nodes. If they are not computed yet, we execute an on-demand update. For BVs, we use the efficient update algorithm for AABBs by Steinemann et al. [SOG08]. For DNCs, we use our novel normal bound computation described in Section 3. In order to test adjacency of nodes, we store adjacent pairs in a hash table, although other approaches are also possible [VMT94, GS01]. A hash table is efficient in our case because our queries descend simultaneously to all children, hence most nodes need only test for adjacency against nodes at the same BN-Tree level.

### 4.3. Front Caching

Instead of starting the update of DNCs at the root of the BN-Tree every animation frame, we exploit temporal coherence



**Figure 2:** Left: Performance comparison of BN-Tree and [VMT94] for cloth subdivision (Fig. 1); Right: performance gain as a function of the number of Loop subdivisions.

and store a front that determines where to apply our efficient computation of DNCs. At a new frame, we first compute the DNCs of front nodes, and then compute DNCs from the front up to the root by intersection of children’s DNCs. If a node below the front is visited during a self-collision query, we update its DNC on-demand. We store separate update fronts for DNCs and BVs.

In order to determine the update front, we mark nodes if they were tested in a self-collision query in the previous frame or if at least the DNC of one of their children was computed. In essence, the front for DNC computation separates the subtree of marked nodes from the unmarked ones. However, this showed to be insufficient, as the self-collision query may suffer a sudden descent on several levels due to degradation of DNCs. We avoid such costly query descents by performing a conservative ascent of the DNC update front. If a node whose DNC was computed in the previous frame is now a candidate for ascent (because it was not tested in a self-collision query and none of the DNCs of its children were computed), we perform a test computation of its parent’s DNC. If this DNC degrades and its bitmask has less than 3 active bits, we preserve the location of the update front. Front caching together with conservative ascent brought additional temporal coherence to the self-collision queries and, therefore, an important speed-up.

We add yet another optimization to the update front in order to avoid costly computation of DNCs low in the tree. As a preprocessing, we check for each node whether it is more efficient to compute the DNC in a bottom-up fashion on its subtree or using our algorithm for computing normal bounds. At run-time, if the update front reaches a node where bottom-up computation of the DNC is more efficient, we force the front all the way to the leaves on that subtree.

## 5. Results

We discuss now the application of the BN-Tree to cloth up-sampling using subdivision surfaces, linear blend skinning, and point-based animation. All experiments were executed on a 2.0-GHz dual-processor PC with 2-GB of RAM.

**Subdivision.** Cloth simulations are often post-processed with subdivision in order to obtain smoothed folds and wrinkles in final renders. Bridson et al. [BFA02] discuss a post-processing method that handles possible collisions produced by subdivision. With the BN-Tree, it is possible to test collisions on a highly subdivided surface with a cost possibly as low as linear in the number of vertices of the control mesh.

We have used Loop’s subdivision scheme in our examples. Then, the vertices in a surface patch are governed by a set of control points (up to 12 in our examples) with initial and deformed positions  $\{\mathbf{x}_i\}$  and  $\{\mathbf{y}_i\}$ . It would be possible to apply our algorithm for normal bound computation using as governor transformations  $\mathbf{A}_i = 0$ ,  $\mathbf{t}_i = \mathbf{y}_i$ . However, we obtained notably tighter bounds by extracting a best-fit average transformation for every patch. Given initial and deformed centroids  $\mathbf{x}_c$  and  $\mathbf{y}_c$ , we express the deformed control points that govern a patch as  $\mathbf{y}_i = \mathbf{A}\mathbf{x}_i + \mathbf{t}_i$ , with  $\mathbf{A} = \operatorname{argmin}_{\mathbf{A}} \sum \|\mathbf{y}_i - \mathbf{y}_c - \mathbf{A}(\mathbf{x}_i - \mathbf{x}_c)\|^2$ . The inverse matrix of the linear system for the least-squares fit can be precomputed for each patch.

Since we fit one single transformation per patch, the patch deformation defined in Section 3.3.1 is simply  $\mathbf{A}_p = \mathbf{A}$ , i.e., the best-fit average transformation. For the same reason, the triangle and differential deformations cancel out, i.e.,  $\mathbf{A}_t = 0$  and  $\Delta\mathbf{A}_k = 0$ .

Fig. 1 shows a cloth animation example where we tested the BN-Tree. The control mesh consists of 231 vertices, and we performed tests with 2, 3, and 4 subdivision levels. Fig. 2-left compares the per-frame update and query times across the whole simulation for 4 subdivision levels, using the BN-Tree and state-of-the-art hierarchical self-collision detection [VMT94] (denoted as HSC). We also evaluated the performance of spatial hashing [THM\*03], but we do not depict it because it was considerably slower (739 ms/frame on average). Note that spatial hashing is fully conservative, while the BN-Tree and our implementation of HSC do not perform the contour test (See Section 4.1). Fig. 2-right compares the average performance of the BN-Tree against HSC for various subdivision levels, and the speed-up increases, as expected, as the number of subdivisions increases. BN-Tree clearly outperforms when many regions of the cloth are close to planar, as the update front can remain high in the tree.

**Linear Blend Skinning.** Fig. 3 shows a comparison of timings on two different animations of a cat. The cat is animated with a skeleton with 40 bones, and the triangle mesh consists of 244 735 triangles. Each vertex is governed by up to 9 bones, using linear blend skinning. In one animation the cat walks slowly, producing sparse self-collisions, and the BN-Tree outperforms HSC consistently by a factor of 3. In the other animation the cat runs and jumps and suffers many self-collisions (including large pinching at joints). The amount of self-collision, together with the lack of temporal coherence, produce sudden changes in the update front of the BN-Tree, yet it still outperforms HSC, although by a



**Figure 3:** On the left, performance comparison between BN-Tree and [VMT94] for a cat animated with linear blend skinning, in two different situations: walking (middle) with sparse self-collisions, and running (right) with many self-collisions.

smaller factor. These examples show that the BN-Tree produces large speed-ups, as expected, with sparse contact, and it becomes comparable to previous methods under dense and/or non-temporally-coherent contact.

**Point-Based Animation.** In point-based animation, the positions of surface vertices are defined using a moving least squares interpolation of linear deformation fields [MKN\*04]. Our last example is a teddy bear model with 26402 vertices animated using point-based animation with 24 particles (shown in Fig. 4). The figure also shows the comparison of performance between our BN-Tree algorithm and HSC [VMT94].

## 6. Discussion and Future Work

We have presented an algorithm for efficiently computing tight normal bounds for triangulated surfaces governed by reduced deformations. This algorithm serves as the main building block for output-sensitive normal-based culling in hierarchical self-collision detection. Our algorithm relies on the computation of tight bounds of a triangle’s deformation gradient. Even though we have applied our algorithm in the context of self-collision detection, it would be interesting to evaluate other possible applications.

Our experiments demonstrate that the comparative efficiency of the BN-Tree increases with triangle density, hence it appears as a good solution for densely tessellated objects. The BN-Tree also works best in practice for objects with low curvature in the rest configuration, such as our cloth example, because there is higher potential for high-level culling. In models with relatively high local curvature, the DNC update front is too low, even at levels where the evaluation of bounds using our algorithm would become the bottleneck. For that reason, in such cases we simply force the front down to the leaves. By doing this, we locally treat the BN-Tree update in a way analogous to the traditional bottom-up update [VMT94]. In the near future, we plan to explore ways to locally transition to full bottom-up update when the animation does not exhibit temporal coherence, perhaps by evaluating the amount of inter-frame deformation.

The major limitation of our method is that it is not

fully conservative, as it does not support the contour test. This test, however, can be regarded as a lower-dimensional version of the normal test, as discussed by Grinspun and Schröder [GS01], and it might also benefit from our algorithm. Nevertheless, it remains to test the impact of efficient bound computations on the overall performance when the contour test is also considered.

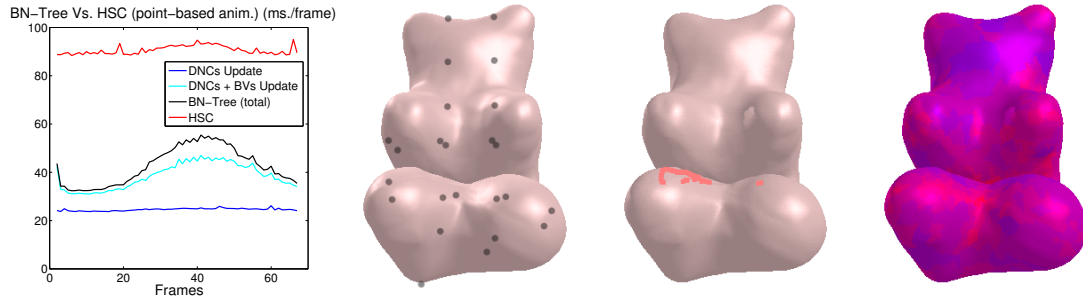
There are many possible avenues for future work, such as parallelization of the algorithm, addition of continuous collision detection, or handling of hierarchy updates under topological changes (i.e., cutting and fracture). It would also be interesting to explore algorithms for output-sensitive self-collision detection for other types of reduced deformation models not currently handled, such as modal analysis or pose-space deformation.

## Acknowledgements

We wish to thank the reviewers for their help in improving our paper, Caroline Larboulette for the animation of the running cat, Pablo Quesada and Juanpe Brito for further help with this demo, Denis Steinemann for point-based animation code, and Marcos García for proofreading and comments. This project was partially funded by URJC - Comunidad de Madrid (proj. CCG08-URJC/DPI-3647) and the Spanish Science and Innovation Dept. (proj. TIN2007-67188).

## 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).
- [Bar84] BARR A. H.: Global and local deformations of solid primitives. *Proc. of ACM SIGGRAPH* (1984).
- [BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH* (2002).
- [BSPG06] BOTSCH M., SUMNER R., PAULY M., GROSS M.: Deformation transfer for detail-preserving surface editing. *Proc. of Vision, Modeling & Visualization* (2006).
- [BW02] BACIU G., WONG W. S.-K.: Hardware-assisted self-collision for deformable surfaces. *Proc. of the ACM Symposium on Virtual Reality Software and Technology* (2002).



**Figure 4:** On the left, performance comparison between BN-Tree and [VMT94] for a teddy deformed with point-based animation. The other images show the simulated particles, self-collisions, and the DNC update front of the BN-Tree.

- [dB97] DEN BERGEN G. V.: Efficient collision detection of complex deformable models using aabb trees. *J. Graphics Tools* 2, 4 (1997), 1–14.
- [GKJ\*05] GOVINDARAJU N. K., KNOTT D., JAIN N., KABUL I., TAMSTORF R., GAYLE R., LIN M. C., MANOCHA D.: Interactive collision detection between deformable models using chromatic decomposition. *Proc. of ACM SIGGRAPH* (2005).
- [GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM SIGGRAPH* (1996), 171–180.
- [GRLM03] GOVINDARAJU N., REDON S., LIN M., MANOCHA D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), 25–32.
- [GS01] GRINSPUN E., SCHRÖDER P.: Normal bounds for subdivision-surface interference detection. *Proc. of IEEE Visualization Conference* (2001).
- [Hub95] HUBBARD P. M.: Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graphics* 15, 3 (1995).
- [JC01] JOHNSON D. E., COHEN E.: Spatialized normal cone hierarchies. *Proc. of the Symposium on Interactive 3D Graphics* (2001).
- [JMD\*07] JOSHI P., MEYER M., DE ROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. *Proc. of ACM SIGGRAPH* (2007).
- [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).
- [KMGL99] KUMAR S., MANOCHA D., GARRETT W., LIN M. C.: Hierarchical back-face computation. *Computers & Graphics* 23, 5 (1999).
- [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).
- [LAM03] LARSSON T., AKENINE-MÖLLER T.: Efficient collision detection for models deformed by morphing. *The Visual Computer* 19 (2003).
- [LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. *Proc. of ACM SIGGRAPH* (2000), 165–172.
- [MKE03] MEZGER J., KIMMERLE S., ETZMUß O.: Hierarchical techniques in collision detection for cloth animation. *Proc. of WSCG* (2003).
- [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).
- [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).
- [Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garment. *Proc. of 8th Eurographics Workshop on Computer Animation and Simulation* (1997), 177–189.
- [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).
- [SOG08] STEINEMANN D., OTADUY M. A., GROSS M.: Tight and efficient surface bounds in meshless animation. *Computers & Graphics* 32, 2 (2008).
- [SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *Proc. of ACM SIGGRAPH* (2004).
- [TCYM08] TANG M., CURTIS S., YOON S.-E., MANOCHA D.: Interactive continuous collision detection between deformable models using connectivity-based culling. *Proc. of ACM Symposium on Solid and Physical Modeling* (2008).
- [THM\*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. *Proc. of Vision, Modeling and Visualization* (2003).
- [VMT94] VOLINO P., MAGNENAT-THALMANN N.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Eurographics* (1994).