Energy-based Self-Collision Culling for Arbitrary Mesh Deformations

Changxi Zheng   Doug L. James
Cornell University

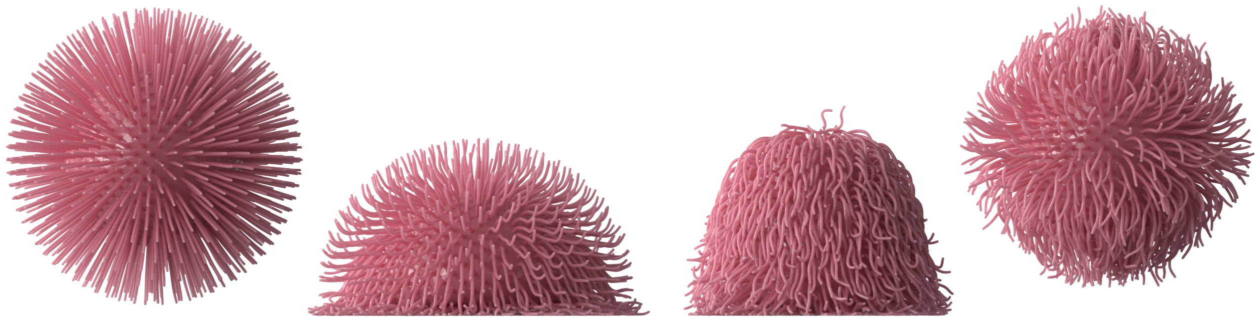


Figure 1: A squishy ball with 820 tentacles *and over 1 million triangles, squishes and bounces on the ground, inducing numerous small interpenetrations. Our Energy-based Self-Collision Culling (ESCC) method accelerates self-collision detection (SCD) for arbitrarily deforming triangle meshes, such as this mesh animated using Oriented Particles [Müller and Chentanez 2011]. We observe an* $\mathbf{11.5\times}$ ***speedup*** *over an optimized AABB-Tree SCD implementation on this challenging example.*

Abstract

In this paper, we accelerate self-collision detection (SCD) for a deforming triangle mesh by exploiting the idea that a mesh cannot self collide unless it deforms enough. Unlike prior work on subspace self-collision culling which is restricted to low-rank deformation subspaces, our energy-based approach supports arbitrary mesh deformations while still being fast. Given a bounding volume hierarchy (BVH) for a triangle mesh, we precompute Energy-based Self-Collision Culling (ESCC) certificates on bounding-volume-related sub-meshes which indicate the amount of deformation energy required for it to self collide. After updating energy values at runtime, many bounding-volume self-collision queries can be culled using the ESCC certificates. We propose an affine-frame Laplacian-based energy definition which sports a highly optimized certificate preprocess, and fast runtime energy evaluation. The latter is performed hierarchically to amortize Laplacian energy and affine-frame estimation computations. ESCC supports both discrete and continuous SCD with detailed and nonsmooth geometry. We observe significant culling on many examples, with SCD speed-ups up to 26×.

Links: DL PDF WEB

1 Introduction

Self-collision detection (SCD) methods are widely used in computer graphics and engineering to enable realistic simulation of self contact for highly deformable objects. Various methods have been devised to accelerate the numerous triangle-triangle overlap tests, however few methods exist that can entirely avoid SCD tests over large mesh regions. Recently Subspace Self-Collision Culling (SSCC) was proposed [Barbič and James 2010], wherein precomputed certificates are used to cull SCD tests for large mesh regions within bounding volume (BV) nodes–sometimes even the entire model. Unfortunately, the use of SSCC is restricted to a very special class of low-dimensional subspace deformations, such as modal deformations, which prevents its widespread use.

In this paper, we propose Energy-based Self-Collision Culling (ESCC), a generalization of certificate-based self-collision culling to *arbitrary* mesh deformations. We use a deformation energy $E(\boldsymbol{u})$ to measure "how much" a mesh patch deforms due to vertex displacements $\boldsymbol{u}$. We precompute the minimum energy $\mathcal{E}$ required for that mesh patch to self collide, then use this self-collision *certificate* to cull SCD tests at runtime: after a mesh deforms we compute $E(\boldsymbol{u})$, then if $E(\boldsymbol{u}) < \mathcal{E}$ we are guaranteed that no self collision can occur. Our ESCC certificates work seamlessly with the traditional BVH-based self-collision detection methods and can accelerate them significantly. We first precompute certificates of surface patches contained in BV nodes. Given a node at runtime, before the standard BVH traversal to test for self collisions, we compute the node's deformation energy to evaluate its certificate. If the certificate is valid, the subsequent BVH traversal can be completely culled. Consequently, ESCC can rule out regions with little deformation, resulting in faster self-collision processing.

While many deformation energy models are possible, we propose an affine-invariant Laplacian energy model that measures nonsmooth deformation, and has several speed advantages. Major contributions of this paper are the algorithms for fast runtime energy evaluation, and fast certificate precomputation for BV nodes. Leveraging the BVH structure, runtime energy computation can be done hierarchically in $O(N)$ flops, and much faster than traditional SCD. As a result, significant speedups in SCD can be achieved (see Figure 1 for a preview of our results). To enable a very fast triangle-triangle certificate preprocess, we propose several techniques (detailed in our supplemental appendices) that enable the underlying QCQP optimization problems to be solved exactly, and at low cost.

2 Related Work

Self-collision detection (SCD) methods for deformable models have a long history in computer graphics, and we optimize the common practice of using a bounding volume hierarchy (BVH) for triangle-based SCD [Teschner et al. 2005]. Our ESCC certificates represent a geometric property of the sub-mesh inside any BV node (or nodes), and can therefore be used with any flavor of bounding volume, including spheres [Hubbard 1995], axis-aligned bounding boxes (AABBs) [van den Bergen 1997], oriented bounding boxes (OBBs) [Gottschalk et al. 1996], and discrete oriented polytopes (k-DOPs) [Klosowski et al. 1998]. In our implementation, we use AABB-based binary trees [van den Bergen 1997]. Our ESCC certificates are complementary to many existing SCD acceleration approaches for BVHs, and can provide yet another way to cull triangle-overlap tests. Our ESCC certificates support continuous SCD needed for simulating thin objects [Bridson et al. 2002].

The most closely related work to ours is subspace self-collision culling (SSCC) [Barbič and James 2010]. It assumes object-frame subspace deformations of the form $\boldsymbol{u} = \mathsf{U}\boldsymbol{q}$, and uses certificates that measure minimal deformation essentially using $\|\boldsymbol{q}\|_2 = \|\boldsymbol{u}\|_2$. The effectiveness of the certificates relied upon the ability of the modal basis U to produce smooth deformations which tend to avoid localized self collisions. For the arbitrary deformations we consider, one cannot use $\|\boldsymbol{u}\|_2$ as a certificate measure since its certificates degenerate into measuring individual vertex displacements. In contrast, we use an energy-based measure $E(\boldsymbol{u})$ of arbitrary patch deformations, and exploit the sparse structure of the arbitrary-deformation problem to derive an exact ESCC certificate preprocess and runtime that is mathematically different and highly efficient.

Barbič and James [2010] accelerated BVH-based SCD by reducing both (i) the cost of post-deformation BVH updates, $\mathsf{C}_{\mathsf{Update}}$, and (ii) the cost of recursively testing the BVH against itself to identify overlapping triangle pairs, $\mathsf{C}_{\mathsf{Query}}$. For arbitrary deformations, our method is stuck with inherently $\Omega(N)$ $\mathsf{C}_{\mathsf{Update}}$ and $\mathsf{C}_{\mathsf{Query}}$ costs for an N triangle mesh patch. Our method adds additional $O(N)$ overhead to update patch-specific certificate energies $E(\boldsymbol{u})$, and cannot exploit subspace certificates or BV updates [James and Pai 2004] to avoid looking at the triangles entirely. Nevertheless, the BVH-based SCD bottleneck for arbitrary deformations remains the $\mathsf{C}_{\mathsf{Query}}$ cost, and our ESCC certificates measure localized patch deformations, as opposed to global deformation amplitude, and can thus cull localized deformations more effectively—ESCC can even be faster than SSCC on subspace deformations (see §8).

For arbitrary deformations, many prior works have attempted to reduce the $\mathsf{C}_{\mathsf{Query}}$ cost bottleneck. For example, methods based on chromatic decompositions [Govindaraju et al. 2005a] and representative triangles [Curtis et al. 2008] can effectively reduce the number of redundant low-level triangle overlap tests. Curvature tests [Volino and Magnenat-Thalmann 1994; Schvartzman et al. 2010] and normal bounds [Provot 1997; Grinspun and Schröder 2001; Tang et al. 2009; Schvartzman et al. 2009] can also cull overlap tests in potentially large regions, however, unlike ESCC, these methods can only cull effectively in regions with smooth geometry and smooth deformations. In contrast, our certificates can efficiently cull self-collision tests even in areas of non-smooth geometry (like [Barbič and James 2010]), and, furthermore, we can cull patches with nonsmooth deformations. Curvature-based culling also requires a potentially expensive "contour test" for correctness, but recently it was shown how this could be performed efficiently using "Star Contours" [Schvartzman et al. 2010]. In comparisons provided later (§8), we observe larger speedups using ESCC on the same examples. Nevertheless, many of these narrow-phase triangle-triangle optimizations are complementary to patch-based ESSC, and could be used simultaneously.

Our use of ESCC certificates for BVH-based SCD is an instance of a kinetic data structure (KDS) [Guibas 2004] for reasoning about self collisions of dynamic meshes. In contrast to other geometric KDS techniques for SCD [Guibas et al. 2002; Gao et al. 2006], we propose energy-based certificates for triangle meshes.

Algorithm 1: **Energy-based self-collision detection**

```
procedure: BVH_Self_Traversal (r)
input    : BVH root node r
begin
    stack.put (r, r);
    while not stack.empty () do
        n,n′ ← stack.pop ();
        if n = n′ then
            if has_certificate (n) and
              deformation_energy (n) < certificate (n)
            then
                continue;
            if is_leaf(n) then
                triangle-triangle intersection tests for all
                non-neighboring triangle pairs of n;
            else
L1              foreach distinct children pairs c, c′ of n do
L2                  stack.put(c, c′);
L3              foreach child c of n do stack.put(c, c);
        else
            if has_certificate ($n \cup n'$) and
              deformation_energy ($n \cup n'$) <
              certificate ($n \cup n'$)
            then continue;
            if not is_BV_intersected(n, n′) then continue;
            if is_leaf(n) and is_leaf(n′) then
                triangle-triangle intersection tests for all
                non-neighboring triangle pairs from n and n′;
            else
                $t_1$ ←higher non-leaf node in n and n′ ;
                $t_2$ ←lower or leaf node in n and n′ ;
                foreach child c of $t_1$ do stack.put(c, $t_2$);
end
```

Our energy-based certificates are complementary to GPU-based methods that use brute force to accelerate triangle-level SCD computations, but less so for rasterization-based methods [Heidelberger et al. 2004]. In addition to improving the speed of triangle overlap tests, GPU-based methods also try to reduce the number of needed overlap tests [Govindaraju et al. 2005b; Sud et al. 2006]. Our current implementation exploits multi-core optimizations, but GPU parallelization is a logical extension.

3 Self-Collision Detection Using Certificates

The most common approach to detect self-collisions of a triangle mesh is to build a bounding volume hierarchy (BVH) for the mesh and query the BVH against itself (see Algorithm 1 without the blue-colored code blocks). Each BV node contains a subset of the triangle mesh. To find collisions between mesh regions contained by two BV nodes, the algorithm first checks if their bounding boxes are intersected. If they are separated, it is guaranteed to have no collisions between the two nodes, and it is sufficient to go no deeper on the BVH for collision detection. If the two BV nodes overlap, then the algorithm has to check collisions over all pairs of their children. Fast collision detection algorithms rely on culling expensive triangle-triangle intersection tests at high levels of BVH traversal. However, this is usually hopeless for self-collision detection, because, even for undeformed meshes, the algorithm always has to test triangle-triangle intersections for non-adjacent but geometrically close triangle pairs, which are contained in either the same BV nodes or overlapping BV nodes.

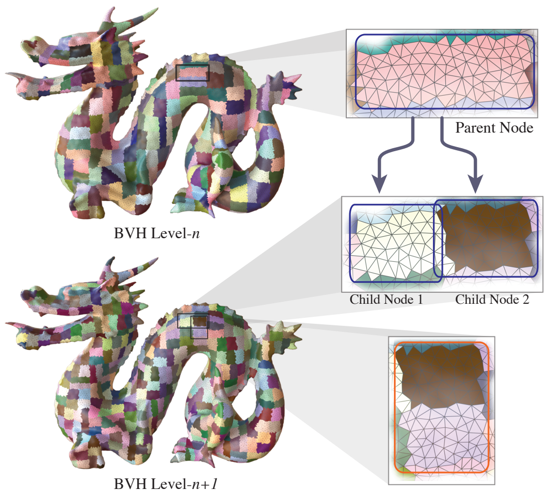


Figure 2: Intra-node and inter-node certificates: *(Top) We compute intra-node certificates for the submesh associated with a BV node, here shown as a level-n node in a binary AABB-Tree. (Bottom) We also compute inter-node certificates between sibling nodes on level $n+1$, as well as (and unlike [Barbič and James 2010]) all same-level nodes with adjacent submeshes.*

Our algorithm integrates certificate validation into the BVH traversal (see Algorithm 1) to achieve more efficient self-collision culling. It works for certificates defined by any surface deformation energy model, $E(\boldsymbol{u})$. Given an energy model, we precompute certificate values $\mathcal{E}$ for surface patches of individual BV nodes as well as the combined patches of some pairs of connected BV nodes (see Figure 2). We call these *intra-node* and *inter-node* certificates, respectively. Certificate computation is described in §6. At runtime, to detect self-collisions of a node n, we first compute its deformation energy E_n. If E_n is less than the intra-node certificate $\mathcal{E}_n$, there is no need to traverse its subtree because it is guaranteed to be self-collision free. Similarly, to detect collisions between node i and j, if the inter-node certificate $\mathcal{E}_{ij}$ has been precomputed, we evaluate the current deformation energy E_{ij} of the joint surface patch of nodes i and j, and compare it to $\mathcal{E}_{ij}$. If $E_{ij} < \mathcal{E}_{ij}$, the traversal of the two subtrees from i and j can be culled, otherwise subtree traversal is needed. In practice, we observe that certificates increase culling performance as one traverses deeper on the BVH, so that even for a mesh undergoing large deformation, many BV nodes at fine geometric scales can still be culled (see Figure 3). Also, the use of inter-node *and* intra-node certificates ensures that for SCD on a nearly undeformed model, ESCC can cull essentially all triangle-triangle overlap tests.

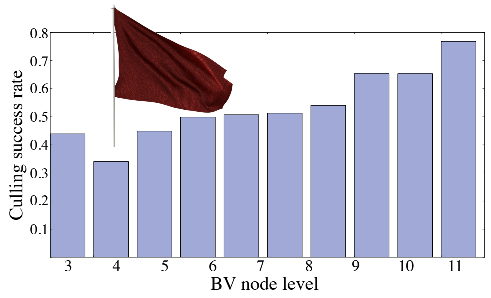


Figure 3: Culling performance increases at finer scales: *Certificates tend to become stronger for smaller nodes as more deformation energy (per triangle) is required to deform smaller submeshes to self collide. (Data for* `flag` *example.)*

Algorithm 2: Precompute certificates for BV nodes

```
procedure: precompute_intranode_certificate (n)
begin
    if number_of_triangles (n) < $T_c$ and is_connected(n)
    then
        return intranode_certificate (n)        // see §6
    else return null
end

procedure: precompute_internode_certificate (n, n′)
begin
    if number_of_triangles(n)+
      number_of_triangles(n′) < $T_c$ and
      is_connected(n) and is_connected(n′) and
      is_connected(n, n′) and
      BVH_level(n) = BVH_level(n′) then
        return internode_certificate (n, n′)    // see §6
    else return null
end
```

Selective use of certificates on BV nodes: Computing all possible certificates on a BVH requires quite expensive precomputation effort, and storing them for runtime validation consumes a large amount of memory since there are $O(N^2)$ possible inter-node certificates for N BV nodes. For inter-node self-collision detection, if the sub-meshes of two nodes are not directly connected, we can rely on the traditional BV-intersection tests to cull collision-free nodes because these tests can often offer efficient culling between two distinct tree nodes. However, if two nodes are geometrically connected on the mesh, their BVs always intersect, and we can use inter-node certificates for collision culling. In practice, we only compute inter-node certificates for connected nodes if they are on the same BVH level and their joint sub-mesh is connected and has less than T_c triangles (see Algorithm 2). Consequently, both the computation and storage of certificates require $O(N)$ complexity. For the nodes without certificates, the algorithm simply returns to the traditional BVH traversal scheme.

The order of BV node traversal: While our certificate-accelerated self-collision detection simply adds certificate comparison before subtree traversal and requires no change on the other parts, further performance improvement can be achieved by adjusting the order of node traversal. Notice that the satisfaction of inter-node certificate $\mathcal{E}_{ij}$ guarantees the absence of self-collisions on the *entire* joint sub-mesh of nodes i and j. In other words, the certificates $\mathcal{E}_i$ and $\mathcal{E}_j$ are both guaranteed to be satisfied as well. Therefore, when pushing node pairs into the stack to traverse, we push the cross-node test on first (see line L1 and L2 of Algorithm 1) and the self-node tests on last (see line L3 of Algorithm 1). If the inter-node certificate $\mathcal{E}_{ij}$ is satisfied, the subsequent self-node traversals on both node i and j can be immediately skipped. In our implementation, we notice about 2% to 6% performance improvement over the BVH traversal without this optimization.

4 Geometrically Based Deformation Energy

Laplacian Energy: Our energy-based certificate framework can be used with arbitrary surface deformation energy models. However, for practical reasons, we propose a parameter-free geometrically based energy model which is fast for both certificate precomputation and runtime energy evaluation. We use a Laplacian-based mesh deformation energy based on the simple quadratic form,

$$E = \|\mathsf{L}\boldsymbol{u}\|_2^2 = \boldsymbol{u}^T\mathsf{L}^T\mathsf{L}\boldsymbol{u} = \boldsymbol{u}^T\mathsf{K}\boldsymbol{u} \tag{1}$$

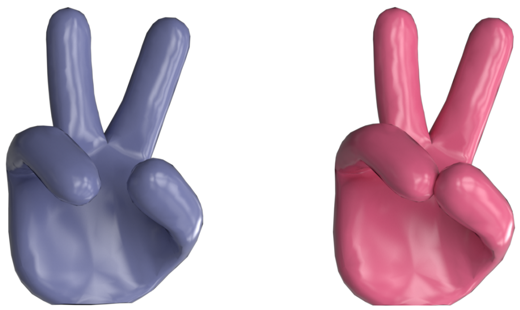


Figure 4: Minimum-energy deformation for self contact: *(Left) A hand model is globally and smoothly deformed (Right) to bring two fingertips into contact as per the minimum displacement Laplacian energy defined in* (1). *The smoothness of minimum-energy deformation enables meaningful certificates on BV nodes.*

where L is the discrete *Laplace-Beltrami* operator on a surface mesh patch [Meyer et al. 2002], $\boldsymbol{u}$ are its vertex displacements, and $\mathsf{K} = \mathsf{L}^T\mathsf{L}$ is the effective stiffness matrix of this potential energy functional. This E will essentially minimize the squared mean curvature of the displacement field (See Figure 4).

An important property of K is its particular 3×3 block matrix structure: each (i,j) block is a scalar k_{ij} times a 3×3 identity matrix, such that $E = \boldsymbol{u}^T\mathsf{K}\boldsymbol{u} = \sum_{ij} k_{ij}\boldsymbol{u}_i^T\boldsymbol{u}_j$. We will exploit this structure to devise a faster certificate preprocess, and to accelerate runtime energy computation, by exploiting various properties, e.g., L commutes with affine transforms.

Affine Pull-Back: The Laplace-Beltrami operator L is rank-3 deficient, leading to a translation invariant energy measure (1). Unfortunately, this measure is not rotation invariant, and so mere rigid-body displacements will add fictitious deformation energies that reduce culling. It turns out that if we approximate the vertex displacement field with a smooth spatial deformer $\phi : X \rightarrow x$ we can use ϕ^{-1} to "pull back" the deformed geometry to a less deformed state while still preserving any intersections that exist. For affine deformers, the pulled-back triangle geometry will also preserve triangle planarity, thereby leading to efficient algorithms. Therefore we estimate displacements in a local affine frame via

$$\boldsymbol{x} = \mathsf{F}(\boldsymbol{X}+\boldsymbol{u}) + \boldsymbol{t} \quad \Leftrightarrow \quad \boldsymbol{u} = \mathsf{F}^{-1}(\boldsymbol{x}-\boldsymbol{t}) - \boldsymbol{X}, \qquad (2)$$

where $\boldsymbol{x}$ are current vertex positions and $\boldsymbol{X}$ are their rest positions, F and $\boldsymbol{t}$ denote linear and translation components, and $\boldsymbol{u}$ is the displacement field deforming the sub-mesh. Note that while affine deformation cannot induce self-collisions, how we estimate F will affect $E(\boldsymbol{u})$. Estimating an F which minimizes E is preferable since it will improve culling, but it is not necessary. In practice, we estimate $(\mathsf{F}^{-1}, \boldsymbol{t})$ using an hierarchical least-squares computation (§5). Using this affine pull-back we obtain an affine-invariant measure, and can significantly reduce deformation energy (see Figure 5).

Sub-mesh Energy: Given a BV node, we detach its sub-mesh from the rest of the entire mesh and compute the L matrix based on the isolated sub-mesh. Similarly, for a pair of connected BV nodes, we combine their sub-meshes as a single triangle patch isolated from the rest of the original mesh, and compute the L matrix. All these L matrices can be precomputed and stored with the BVH together. Later, the related certificates and energy values are evaluated based on their corresponding detached sub-meshes.

5 Hierarchical Energy Computation

We now present a fast hierarchical method for runtime evaluation of deformation energy on BVH sub-meshes. We have two steps for the hierarchical energy evaluation: (i) estimate the affine transform; and (ii) compute the quadratic energy using (1).

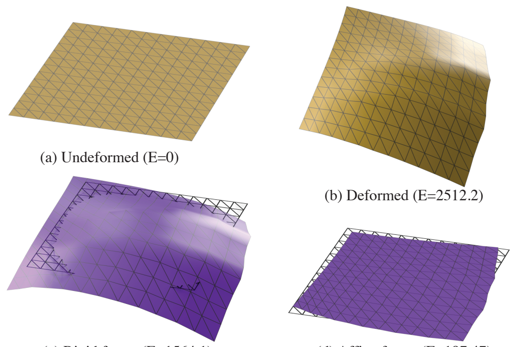


Figure 5: Smaller deformation energies *can be obtained by using a suitable affine transformation to reduce mesh deformation. Shown are (a) the undeformed mesh, and (b) the deformed mesh. While deformation energies in a tracked rigidbody frame (c) are smaller, they cannot undo stretching. In contrast, pulling back to an affine frame (d) can further reduce deformation energy while still preserving intersection properties. Affine frames are also cheaper to estimate than rigidbody frames.*

5.1 Hierarchical estimation of sub-mesh transforms

We use least squares to estimate sub-mesh affine transforms similar to "shape matching" [Müller et al. 2005], and exploit the hierarchical and overlapping nature of sub-meshes for fast summation of intermediate quantities analogous to [Rivers and James 2007; Steinemann et al. 2008]. Given a set of BV-node vertices deforming from the rest positions $\boldsymbol{X}_i$ to the current positions $\boldsymbol{x}_i$, we estimate the affine matrix $\mathsf{A} = \mathsf{F}^{-1}$ and rigid translation vector $\boldsymbol{t}$ to minimize

$$\sum_i m_i \|\mathsf{A}(\boldsymbol{x}_i - \boldsymbol{t}) - (\boldsymbol{X}_i - \boldsymbol{t}_0)\|^2 \qquad (3)$$

where m_i is the vertex weights, and $\boldsymbol{t}_0$ is the center of mass of the initial shape. Setting its derivative with respect to A and $\boldsymbol{t}$ to zero yields the solution: $\boldsymbol{t}$ is the center of mass of the deformed shape, i.e., $\boldsymbol{t} = \sum_i m_i\boldsymbol{x}_i / \sum_i m_i$, and A can be computed by

$$\mathsf{A} = \left(\sum_i m_i(\boldsymbol{X}_i - \boldsymbol{t}_0)(\boldsymbol{x}_i - \boldsymbol{t})^T\right)\left(\sum_i m_i(\boldsymbol{x}_i - \boldsymbol{t})(\boldsymbol{x}_i - \boldsymbol{t})^T\right)^{-1}$$
$$\equiv \mathsf{A}_1\mathsf{A}_2^{-1}$$

Both A_1 and A_2 can be computed quickly. Computing A_1 of a BV node n requires two summations,

$$\boldsymbol{s}_x = \sum_{\text{vertex } i \in \mathsf{n}} m_i\boldsymbol{x}_i \quad \text{and} \quad \boldsymbol{S}_{\mathsf{A}_1} = \sum_{\text{vertex } i \in \mathsf{n}} m_i\boldsymbol{X}_i\boldsymbol{x}_i^T, \qquad (4)$$

and then $\mathsf{A}_1 = \boldsymbol{S}_{\mathsf{A}_1} - \boldsymbol{t}_0\boldsymbol{s}_x^T$. Computing A_2 requires two sums,

$$m_c = \sum_{\text{vertex } i \in \mathsf{n}} m_i \quad \text{and} \quad \boldsymbol{S}_{\mathsf{A}_2} = \sum_{\text{vertex } i \in \mathsf{n}} m_i\boldsymbol{x}_i\boldsymbol{x}_i^T, \qquad (5)$$

and $\mathsf{A}_2 = \boldsymbol{S}_{\mathsf{A}_2} - \boldsymbol{s}_x\boldsymbol{s}_x^T/m_c$. These four summations are computed hierarchically on the BVH. For example, to compute $\boldsymbol{S}_{\mathsf{A}_1}$ of node n, we sum up the $\boldsymbol{S}_{\mathsf{A}_1}$ matrices of all its children, and remove the repeated counting on the children's shared boundary vertices. Proceeding recursively, the summations on node n and all its descendants can be computed by going through the subtree of n from bottom up. Shape matching of joint sub-meshes of node i and j can be computed similarly by computing the summations on i and j individually and adding them together without repeating boundary vertices. Furthermore, on multi-core processors, the summations on different subtrees can be computed in parallel, and hence are even faster. In practice, we implement the recursive parallel computation using Intel's *Thread Building Block*. Note that by using affine instead of rigid transforms, we avoid the need for polar decompositions to estimate rotations, achieving both cheaper computation and better culling efficiency (see Figure 5).

5.2 Hierarchical evaluation of deformation energy

Given a sub-mesh's $\boldsymbol{x}$ and A at runtime, its energy is given by

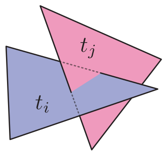
$$\begin{aligned} E = \|\mathsf{L}\boldsymbol{u}\|_2^2 &= \|\mathsf{L}(\mathsf{A}(\boldsymbol{x}-\boldsymbol{t}) - (\boldsymbol{X}-\boldsymbol{t}_0))\|_2^2 \\ &= \|\mathsf{LA}\boldsymbol{x} - \mathsf{LA}\boldsymbol{t} - \mathsf{L}\boldsymbol{X} + \mathsf{L}\boldsymbol{t}_0\|_2^2 \\ &= \|\mathsf{LA}\boldsymbol{x} - \mathsf{L}\boldsymbol{X}\|_2^2 = \|\mathsf{AL}\boldsymbol{x} - \mathsf{L}\boldsymbol{X}\|_2^2 \end{aligned} \tag{6}$$

where $\mathsf{L}\boldsymbol{X}$ is the mean curvature normal vector of the vertices on the undeformed mesh (which is precomputed), $\mathsf{L}\boldsymbol{x}$ evaluates the mean curvature normals of the vertices at the current configuration, and $\mathsf{AL}\boldsymbol{x}$ are node-transformed versions. Here we have used the fact that A and L commute. Note that the energy is independent of the rigid-body translation, $\boldsymbol{t}$. The mean-curvature normal $(\mathsf{L}\boldsymbol{x})_i$ of vertex v_i is completely determined by the positions of v_i and its direct neighbors, as seen from the sparsity structure of L, which enables us to reuse $\mathsf{L}\boldsymbol{x}$ hierarchically. In particular, consider a vertex v_i contained in a BV node n and its child node n_c. If v_i is not on the boundary of the sub-mesh contained by n_c, then v_i has the same local connectivity on both n and n_c. Therefore, if the mean-curvature normal of v_i on node n has been computed, there is no need to recompute the normal of v_i on n_c. This observation is particularly helpful for BVH traversal. If the certificate checking fails on node n, we need to traverse to its child nodes to evaluate the deformation energy. For most of the vertices on the children, we can reuse the normals $\mathsf{L}\boldsymbol{x}$ computed at node n and only need to recompute the normals for boundary vertices. Finally, normals are then multiplied by A to construct $\mathsf{AL}\boldsymbol{x}$ for a specific BV node.

6 Certificate Precomputation

We now show how to efficiently precompute certificates for a triangle sub-mesh $\mathcal{T}$, such as one associated with a BV node. The precomputation for intra-node and inter-node certificates is identical, since both minimize the energy in (1) for a self-colliding sub-mesh $\mathcal{T}$ detached from the original mesh. For an intra-node certificate, $\mathcal{T}$ is the sub-mesh contained by the BV node. For an inter-node certificate, $\mathcal{T}$ is the joint sub-mesh of both nodes.

6.1 Certificate for triangle-triangle collision



Here we show how to compute the minimum energy required to intersect two nonadjacent triangles, t^i and t^j. Let $\boldsymbol{X}_k^i \in \mathbb{R}^3, k = 1,2,3$ denote the rest positions of the three vertices of t^i, and, similarly, use $\boldsymbol{X}_k^j, k = 1,2,3$ for t^j. Let $\boldsymbol{u}_k^i, \boldsymbol{u}_k^j \in \mathbb{R}^3, k = 1,2,3$ indicate the vertex displacements of t^i and t^j, respectively. If a point on t^i is in contact with a point on t^j, denote the barycentric coordinates of the two points by α_k and $\beta_k, k = 1,2,3$, respectively. Then the minimum energy E_{ij} required to intersect t^i with t^j is given by the following non-convex quadratically constrained quadratic program (QCQP),

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{u}^T \mathsf{K} \boldsymbol{u} \\ \text{subject to} \quad & \alpha_1 + \alpha_2 + \alpha_3 = 1,\ \alpha_i \geq 0,\ i = 1,2,3 \\ & \beta_1 + \beta_2 + \beta_3 = 1,\ \beta_i \geq 0,\ i = 1,2,3 \\ & \alpha_1(\boldsymbol{u}_1^i + \boldsymbol{X}_1^i) + \alpha_2(\boldsymbol{u}_2^i + \boldsymbol{X}_2^i) + \alpha_3(\boldsymbol{u}_3^i + \boldsymbol{X}_3^i) = \\ & \beta_1(\boldsymbol{u}_1^j + \boldsymbol{X}_1^j) + \beta_2(\boldsymbol{u}_2^j + \boldsymbol{X}_2^j) + \alpha_3(\boldsymbol{u}_3^j + \boldsymbol{X}_3^j), \end{aligned} \tag{7}$$

where $\mathsf{K} = \mathsf{L}^T\mathsf{L}$ is a positive semidefinite matrix. While non-convex QCQP is considered to be NP-hard in general, we notice that this problem can be solved exactly.

Reduction to Rayleigh quotient form: If we assume the two points in collision (α and β) are known a priori, then the QCQP (7) simplifies to a linearly constrained quadratic program (LCQP),

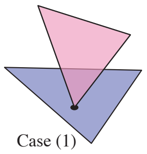
$$\begin{aligned} \text{minimize} \quad & \boldsymbol{u}^T \mathsf{K} \boldsymbol{u} \\ \text{subject to} \quad & \alpha_1(\boldsymbol{u}_1^i + \boldsymbol{X}_1^i) + \alpha_2(\boldsymbol{u}_2^i + \boldsymbol{X}_2^i) + \alpha_3(\boldsymbol{u}_3^i + \boldsymbol{X}_3^i) = \\ & \beta_1(\boldsymbol{u}_1^j + \boldsymbol{X}_1^j) + \beta_2(\boldsymbol{u}_2^j + \boldsymbol{X}_2^j) + \alpha_3(\boldsymbol{u}_3^j + \boldsymbol{X}_3^j). \end{aligned} \tag{8}$$

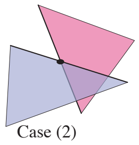
This problem can be easily solved analytically using Lagrange multipliers. We defer the details of the LCQP solver to Appendix A, and only present the result here. In particular, the achieved minimum energy value $\boldsymbol{u}^T\mathsf{K}\boldsymbol{u}$ can be expressed as a generalized Rayleigh quotient,

$$\hat{E}_{ij} = \frac{\boldsymbol{a}^T \mathsf{M} \boldsymbol{a}}{\boldsymbol{a}^T \mathcal{G} \boldsymbol{a}}, \tag{9}$$

where $\boldsymbol{a}$ is a 6×1 vector, $\boldsymbol{a} = [\,\alpha_1\ \alpha_2\ \alpha_3\ -\beta_1\ -\beta_2\ -\beta_3\,]^T$; $\mathsf{M} = \mathsf{B}^T\mathsf{B} \in \mathbb{R}^{6\times6}$ is a rank-3 symmetric positive semidefinite matrix, where $\mathsf{B} = [\,\boldsymbol{x}_1^i\ \boldsymbol{x}_2^i\ \boldsymbol{x}_3^i\ \boldsymbol{x}_1^j\ \boldsymbol{x}_2^j\ \boldsymbol{x}_3^j\,]$; and $\mathcal{G} = \tilde{\mathsf{K}}_s^\dagger \in \mathbb{R}^{6\times6}$ is a submatrix of the pseudo-inverse of a related matrix $\tilde{\mathsf{K}}$, where the elements of $\tilde{\mathsf{K}}_s^\dagger$ correspond to the 6 vertices of t^i and t^j. Following the convention of elastostatic mechanics, we call this K pseudo-inverse the *Green's function* matrix. Appendix B details a fast computation of the Green's function matrix that exploits the block structure and null space of K. Now (7) is equivalent to the following problem:

$$\begin{aligned} \text{minimize} \quad & \frac{\boldsymbol{a}^T \mathsf{M} \boldsymbol{a}}{\boldsymbol{a}^T \mathcal{G} \boldsymbol{a}} \\ \text{subject to} \quad & \alpha_1 + \alpha_2 + \alpha_3 = 1,\ \alpha_i \geq 0,\ i = 1,2,3 \\ & \beta_1 + \beta_2 + \beta_3 = 1,\ \beta_i \geq 0,\ i = 1,2,3. \end{aligned} \tag{10}$$


Case (1)


Case (2)

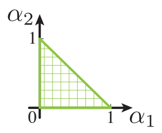
Its optimum is achieved only at one of the two cases: (i) a vertex of one triangle touches the other triangle; or (ii) an edge of one triangle touches an edge of the other triangle. Intuitively, if a collision does not satisfy either case, one can relax the vertex displacements to shrink the deformation energy while still keeping the two triangles in collision until either case (i) or (ii) is achieved. In the remainder of this subsection, we consider both cases.

Vertex-triangle collision: There are 6 vertex-triangle collision cases. Without loss of generality, we consider the first vertex $\boldsymbol{X}_1^j$ of t^j in collision with t^i. Then, for problem (10), $a_3 = 1 - a_1 - a_2$, $\beta_1 = 1$, $\beta_2 = \beta_3 = 0$, and we can simplify the problem using reduced parameters. Let $\boldsymbol{a} = \mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}$, where $\tilde{\boldsymbol{a}} = \begin{bmatrix}\alpha_1 & \alpha_2\end{bmatrix}^T$,

$$\mathsf{D} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}^T \text{ and } \boldsymbol{b} = \begin{bmatrix} 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix}^T.$$

Substitution into (10), yields the reduced problem,

$$\begin{aligned} \text{minimize} \quad & \frac{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathsf{M} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathsf{M} \boldsymbol{b}}{\tilde{\boldsymbol{a}}^T \mathsf{D}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T \mathcal{G} \mathsf{D} \tilde{\boldsymbol{a}} + \boldsymbol{b}^T \mathcal{G} \boldsymbol{b}} \\ \text{subject to} \quad & \alpha_1 \geq 0,\ \alpha_2 \geq 0,\ \alpha_1 + \alpha_2 \leq 1. \end{aligned} \tag{11}$$



In Appendix C, we show that the optimum of this problem is achieved in the interior of the domain only at certain situations which can be verified by solving 3 1-D quadratic equations.

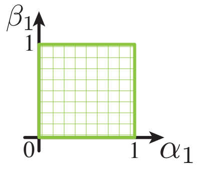
If the optimum happens in the interior of the domain, computing the optimum value requires solving two cubic equations, one of which has only a single real root. For most cases in practice, the optimum is on the boundary defined by three segments. A boundary point corresponds to a case where a vertex touches an edge. The optimum on each boundary segment can be computed by solving a 1-D quadratic equation. Please see the detailed derivation of these equations in Appendix C. For all 6 vertex-triangle collisions, we need to solve 36 quadratic equations and in very rare cases solve cubic equations.

Edge-edge collision: There are 9 edge-edge collision cases. Again, without loss of generality, we consider the edge connecting $\boldsymbol{X}_1^i$ and $\boldsymbol{X}_2^i$ on t^i in collision with the edge connecting $\boldsymbol{X}_1^j$ and $\boldsymbol{X}_2^j$ on t^j. This corresponds to $a_2 = 1 - a_1$, $a_3 = 0$, $\beta_2 = 1 - \beta_1$ and $\beta_3 = 0$ in (10), and we can construct a reduced problem using $\boldsymbol{a} = \mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}$, where $\tilde{\boldsymbol{a}} = \begin{bmatrix}\alpha_1 & \beta_1\end{bmatrix}^T$,

$$\mathsf{D} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}^T \text{ and } \boldsymbol{b} = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 0 \end{bmatrix}^T.$$

Problem (10) now becomes

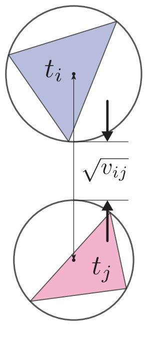
$$\begin{aligned} &\text{minimize} \quad \frac{\tilde{\boldsymbol{a}}^T\mathsf{D}^T\mathsf{M}\mathsf{D}\tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T\mathsf{M}\mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}^T\mathsf{M}\boldsymbol{b}}{\tilde{\boldsymbol{a}}^T\mathsf{D}^T\mathcal{G}\mathsf{D}\tilde{\boldsymbol{a}} + 2\boldsymbol{b}^T\mathcal{G}\mathsf{D}\tilde{\boldsymbol{a}} + \boldsymbol{b}^T\mathcal{G}\boldsymbol{b}} \\ &\text{subject to} \quad 0 \le \alpha_1 \le 1,\ 0 \le \beta_1 \le 1. \end{aligned} \tag{12}$$



Similar to problem (11), the optimum occurs in the interior of the domain very rarely. Checking such situations involves 4 quadratic equation solves. For most cases, the optimum is on the boundary where a vertex touches an edge. Note that all these boundary cases were considered when we solved the boundary optimum values for vertex-triangle collisions (11), therefore we can safely ignore all of them. For all 9 edge-edge collisions, we need to solve 36 1-D quadratic equations and very infrequently solve cubic equations. See Appendix C for derivations.

6.2 Certificate for an entire sub-mesh

The certificate $\mathcal{E}$ for an entire triangle sub-mesh $\mathcal{T}$ simply takes the minimum of $\mathcal{E}_{ij}$ over all possible pairs of triangles, i.e., $\mathcal{E} = \min_{t^i,t^j\in\mathcal{T}} \mathcal{E}_{ij}$. However, processing through all triangle pairs on a large sub-mesh can be impractical. Instead, we use a two-pass algorithm. In the first pass, we quickly compute a *cheap* lower bound $\tilde{E}_{ij}$ of $\mathcal{E}_{ij}$ for all triangle pairs, i.e., $\tilde{E}_{ij} \le \mathcal{E}_{ij}$. In the second pass, we process the triangle pairs in order of ascending $\tilde{E}_{ij}$. The computation of $\mathcal{E}_{ij}$ can be immediately skipped if the so-far encountered smallest $\mathcal{E}$ value is less than $\tilde{E}_{ij}$ of the current triangle pair. Similar ideas have been used in [Barbič and James 2010].



We first compute a lower bound v_{ij} and an upper bound w_{ij} for the numerator and denominator of the generalized Rayleigh quotient (9). Then we have $\tilde{E}_{ij} = v_{ij}/w_{ij} \le \mathcal{E}_{ij}$. Note that the numerator of (9) measures the squared Euclidean distance between the two touching points in undeformed mesh configuration. We compute the circumcenter $\boldsymbol{p}_i$ and the circumradius R_i of each triangle. The Euclidean distance of any two points on triangle pair i, j must be no less than $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2 - R_i - R_j$. An upper bound of the denominator of (9) can be derived by noticing that $\mathcal{G}$ is symmetric positive semidefinite and $\boldsymbol{a}$ satisfies the constraints in (10):

$$\boldsymbol{a}^T\mathcal{G}\boldsymbol{a} \le \max_{k=1..3}\mathcal{G}_{kk} + \max_{k=4..6}\mathcal{G}_{kk} - \sum_{k\ne t,\ \mathcal{G}_{kt}<0}\mathcal{G}_{kt}.$$

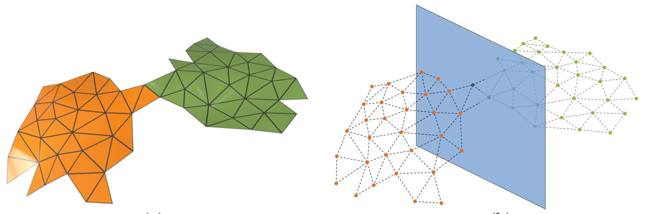


Figure 6: Weakly coupled sub-meshes with separation plane

and so a lower bound for $\mathcal{E}_{ij}$ is

$$\tilde{E}_{ij} = \frac{(\|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2 - R_i - R_j)^2}{\max_{k=1..3}\mathcal{G}_{kk} + \max_{k=4..6}\mathcal{G}_{kk} - \sum_{k\ne t,\ \mathcal{G}_{kt}<0}\mathcal{G}_{kt}}.$$

Discussion: Note that if $\mathcal{T}$ has disconnected components, one of the mesh components can collide with the other component under rigid translation. Consequently, certificates on such sub-meshes are always zero and provide no culling capability at runtime. Therefore, we ignore disconnected BV nodes or node pairs in Algorithm 2 and leave them to be handled by traditional BVH intersection tests. A pathological case comes with a mesh whose BV nodes are almost all disconnected, e.g., the squishy ball in Figure 1, preventing meaningful certificates at most of its BV nodes. We discuss BVH optimizations to improve connectivity in §7.

Certificates for weakly connected BV nodes: To achieve further performance, we take special care of pairs of weakly connected BV nodes (see Figure 6a). Due to the weak connectivity, these node pairs can yield very low certificate values which can easily fail with small deformations at runtime. However, we note that a weakly connected pair of nodes can often be easily separated by a separation plane $ax + by + cz + 1 = 0$ (see Figure 6b). Formally, let us define the vertex sets of the weakly connected nodes n_1 and n_2 as V_1 and V_2 respectively, and a plane equation $f(\boldsymbol{x}) = ax + by + cz + 1$. Then following the separating plane theorem, the absence of intersections between n_1 and n_2 is guaranteed if the following condition is satisfied (for a suitably oriented plane),

$$\begin{cases} f(\boldsymbol{v}) < 0, & \forall \boldsymbol{v} \in V_1 - V_1 \cap V_2 \\ f(\boldsymbol{v}) > 0, & \forall \boldsymbol{v} \in V_2 - V_1 \cap V_2 \end{cases} \tag{13}$$

Finding the optimal separation plane $f(\boldsymbol{x}) = 0$ is a classical problem of finding the *maximum-margin hyperplane* in linear Support Vector Machines [Burges 1998]. Essentially we are classifying two sets of nodes, $V_1 - V_1 \cap V_2$ and $V_2 - V_1 \cap V_2$, with a separation plane in 3D space. For each pair of weakly connected nodes, we compute a maximum-margin separation plane based on their undeformed positions. At runtime, for each vertex in $V_1 - V_1 \cap V_2$ and $V_2 - V_1 \cap V_2$, we first transform it to the node's initial frame using the already-computed affine shape matching (recall §5.1), and then check whether (13) is satisfied if the corresponding certificates fail. In practice, we only apply separation-plane checks at low-level BV nodes (leaf nodes and their parents), and observe up to 11% speedup over the implementation without separation planes.

7 Extensions

Optimizing mesh connectivity of BV nodes: While ESCC certificates can be applied to any BVH, sub-meshes with disconnected or weakly connected triangles will degrade culling performance. We therefore propose a method to construct an AABB-based BVH whose nodes have well-connected sub-meshes. To split a BV node into child nodes, n_1 and n_2, we randomly sample two triangles, and

Example	**Complexity**		**Precomputation**							**Runtime** (in ms)			**Culling difficulty measure**
	Tri	Vtx	Intra-node certificates	Inter-node certificates	Cover ratio, r	G cost (s)	Cert cost (s)	sep. plane num	sep. plane cost (s)	AABB	ESCC	Speedup	
squishy ball	1064216	532110	260059	467162	0.92	5469.7	548.7	297002	1064	3100	270	11.5×	0.72
monkeys	108928	54468	30246	61371	0.96	966.8	201.2	40867	144.3	379.4	49.7	7.6×	0.41
flowing cloth	51200	25921	10164	10605	0.98	467.0	183.6	11109	77.6	124.6	6.5	19.1×	0.10
16 bunnies‡	14964	7484	2324	4652	0.99	112.3	37.6	2332	14.7	448.2	29.3	15.3×	0.21
dragon‡	77203	38602	2434	4870	0.78	140.9	36.7	6323	48.3	320.8	12.3	26.1×	0.22
spring wire‡	16000	8000	10163	3820	0.99	129.0	17.4	2002	10.3	33.2	1.2	25.6×	0.10
snake⋆	18354	9179	5275	9433	0.99	230.2	76.0	6667	45.7	44.0	4.9	9.0×	0.71
dance⋆	14118	7061	3896	7174	0.98	223.5	36.6	3207	34.5	33.1	5.8	5.7×	0.55
flag⋆	13436	6906	2524	3300	0.92	136.1	27.7	3324	24.2	41.5	5.0	8.3×	0.29
bunny (low res.)†	13632	6818	1712	3278	0.99	123.7	25.4	1424	10.4	41.7	7.6	5.5×	0.42
bunny (high res.)†	54528	27266	6534	13443	0.99	453.9	256.6	6238	44.5	169.7	41.1	4.2×	0.36
bunny (noisy)†	54528	27271	4068	8526	0.62	403.4	158.5	2865	18.9	185.7	55.9	3.3×	0.78
bunny (spiky)†	54528	27271	1993	4237	0.54	238.4	147.2	1126	8.0	194.9	95.5	2.0×	1.30
cloth sphere†	20000	10201	4080	6251	0.99	216.3	59.6	7867	50.1	54.4	12.0	4.5×	0.40
cat†	41184	20631	4919	9142	0.99	344.6	73.4	3900	28.1	130.1	31.9	4.1×	0.66
collapsing horse†⋆	25344	12674	2999	5529	0.94	138.7	45.0	2608	17.7	108.7	78.2	1.4×	4.12
cloth-sphere 2°	91470	46216	9198	16217	0.99	516.8	176.2	10685	62.8	52.4	15.5	3.4×	0.98

Table 1: Example statistics *for triangle/vertex counts, the number of precomputed certificates, intra-node coverage, precomputation timings for Green's function* G *matrices and certificates, separating planes for weakly connected inter-nodes. Runtime performance timings (per frame) for our optimized AABB SCD test, and the same code with ESCC culling enabled, demonstrate significant ESCC speedups on most examples. The lowest speedups are for examples with significant interpenetration or dynamic close-proximity scenarios as indicated by the* culling difficulty measure *defined as the ratio of the number of inter-node overlap tests between disconnected leaf nodes to the total number of leaf nodes. Examples from prior work are indicated using † for [Schvartzman et al. 2010], ‡ for [Barbič and James 2010], ⋆ for [Briceño et al. 2003; James and Twigg 2005], and ∘ for [Curtis et al. 2008].*

perform region growing using a cost based on the Euclidean distance between the centroids of two triangles [Cohen-Steiner et al. 2004]. We compute a cost $C(\mathsf{n}_1, \mathsf{n}_2)$ for the split, and repeat the sampling T times and use the one with least cost to create child nodes; we use $T = 60000$ in our examples. Our cost function tries to minimize both the difference in submesh areas, A_1 and A_2, and the bounding-box surface areas, B_1 and B_2:

$$C(\mathsf{n}_1, \mathsf{n}_2) = |A_1 - A_2| + 0.1\,(B_1 + B_2). \tag{14}$$

This metric is similar to the "surface area heuristics" used in previous BV optimization methods [Goldsmith and Salmon 1987].

Continuous Self-Collision Culling: Our method can also be used in continuous self-collision detection, such as for cloth simulation [Bridson et al. 2002]. For example, consider a piecewise-linear deformation of a sub-mesh from displacement $\boldsymbol{u}^{(0)}$ at $t = 0$ to $\boldsymbol{u}^{(1)}$ at $t = 1$, such that $\boldsymbol{u}^{(t)} = (1-t)\boldsymbol{u}^{(0)} + t\boldsymbol{u}^{(1)}, t \in [0, 1]$. Now the energy function $E^{(t)} = E(u(t)) = \|\mathsf{L}\boldsymbol{u}^{(t)}\|_2^2$ might not be convex in $\boldsymbol{u}$, therefore we cannot cull SCD tests by simply validating certificates at $t = 0$ and $t = 1$. The problem comes from a continuously changing affine pull-back, however, we can circumvent it by fixing the affine pull-back. For example, we estimate the affine pull-back based on $\boldsymbol{u}^{(0.5)}$ computed by interpolating $\boldsymbol{u}^{(0)}$ and $\boldsymbol{u}^{(1)}$. We then use the resulting affine transformation to pull back the mesh at both $t = 0$ and $t = 1$, and evaluate corresponding energy values $\bar{E}^{(0)}$ and $\bar{E}^{(1)}$. With this fixed affine pull-back, the deformation energy change from $\bar{E}^{(0)}$ to $\bar{E}^{(1)}$ is convex in t. It follows that $\bar{E}^{(t)} \le \max(\bar{E}^{(0)}, \bar{E}^{(1)})$, $t \in [0, 1]$. Thus if the certificate indicates a collision-free state at both endpoints, then no collision can occur along the piecewise linear trajectory. If at least one endpoint's certificate fails, then we must recurse and perform additional checks. In this way, the ESCC certificates can be used to augment traditional BVH-based continuous SCD queries as in [Barbič and James 2010]. We implemented such continuous SCD, and found that ESCC offers a slightly larger speedup than that in discrete SCD (see Table 2).

Example	AABB	ESCC	Speedup
monkey	2.1s	0.22s	9.2×
dragon	2.3s	0.078s	29.3×
cloth sphere	0.3s	0.061s	4.9×

Table 2: Speedup of CCD

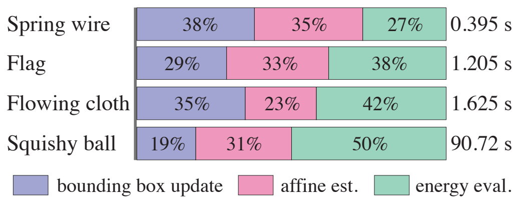


Figure 7: Timing breakdown of post-deformation update

8 Results

Numerous results and statistics are shown in Table 1 for a range of animated mesh examples, including ones made by us and from prior works [Briceño et al. 2003; James and Twigg 2005; Barbič and James 2010; Schvartzman et al. 2010; Curtis et al. 2008]. Please see the supplemental video for animations and detailed performance information. All reported timings were measured on a dual Intel Xeon X5570 (2.93 GHz) processor machine with 8 physical cores. Due diligence has been taken to exploit multi-core parallelization for accelerating both the AABB-Tree updates and queries, and ESCC certificate precomputation and runtime energy evaluation. We also profiled the performance of single-core computation and observed that our parallel ESCC implementation using Intel's TBB achieved 2.1x-3.4x speedups over the single-core computation, depending on the mesh size. Code and data will be made available online. Culling effectiveness versus BVH levels is illustrated in Figure 3. Detailed runtime timing breakdowns for post-deformation tree updates are shown in Figure 7. The impact of various optimizations on runtime SCD are shown in Figure 8.

Squishy Ball: Our most challenging example is the million-triangle squishy ball, simulated using 22176 Oriented Particles [Müller and Chentanez 2011] (recall Figure 1). Although it has many hair-like "tentacles" in close proximity, ESCC still achieves significant culling and an 11.5× speedup (see Figure 10).

Comparison to SSCC: We evaluated ESCC on many SSCC datasets from [Barbič and James 2010], and observe that ESCC

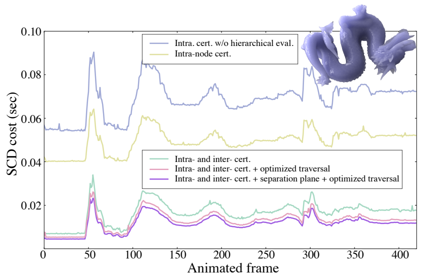
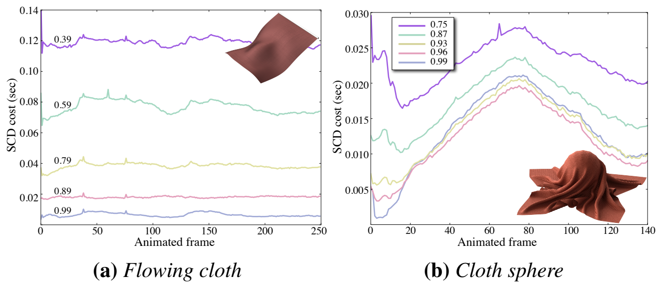


Figure 8: Benefits of ESCC optimizations for runtime SCD



(a) *Flowing cloth* **(b)** *Cloth sphere*

Figure 9: SCD performance vs cover ratio, r

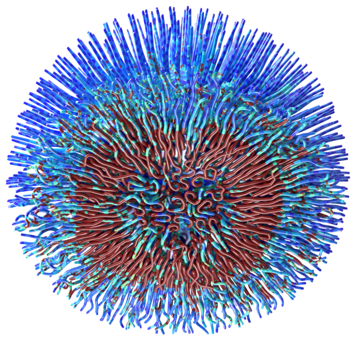


Figure 10: ESCC Culling: *Bottom view of the squishy ball during impact. Colors indicate ESCC culling level, with nodes traversed to the leaf level in red. Despite many mashed tentacles, widespread high-level culling is observed as indicated by many blue nodes.*

also achieves significant speedups but without needing to exploit any subspace information (see Table 1). We were also able to perform comparisons of SSCC to ESCC on the same computer; the speedups compared as (SSCC|ESCC) are: bunny (29.1x|15.3x), dragon (15.1x|23.2x), and spring wire (127.1x|25.6x). See the video for more comparisons. Our method outperforms SSCC on dragon due to a relatively large impact deformation at one point where SSCC is much slower than ESCC. However, when there is no or little deformations, SSCC is faster on subspace motions.

Comparison to Star Contours: We evaluated ESCC on challenging datasets from "Star Contours" [Schvartzman et al. 2010]. In comparison, we always observe larger speedups than "Star Contours" method; the speedups compared as (Star-Contours|ESCC) are: bunny$_{lowres}$ (1.8x|5.5x), bunny$_{highres}$ (2.2x|4.2x), bunny$_{noisy}$ (1.1x|3.3x), bunny$_{spiky}$ (0.83x|2.0x), cloth (1.5x|4.5x) horse (1.05x|1.4x), and cat (1.2x|4.1x). Combining these methods might provide additional speedups.

Performance versus certificate coverage: As presented in §3 and Algorithm 2, we use certificate-based culling only when the number of triangles on a node is less than a threshold T_c; in our implementation, we use $T_c = 2500$ for all examples, except the giant squishy ball where we only used $T_c = 128$. One natural question is how T_c will affect the culling performance. To indicate the fraction of nodes where certificates are applied, we use a **cover ratio**, r, defined as the ratio of the number of computed intra-node certificates to the total number of BV nodes. Table 1 shows the r values used for all the examples. Furthermore, Figure 9 shows the SCD cost with different cover ratios. For smooth small deformations (e.g., Figure 9a), the SCD cost decreases as we use more certificates. However, for fairly large deformations (e.g., Figure 9b), the certificates on high-level BV nodes fail more frequently. Consequently, the payoff of using certificates on high-level BV nodes is smaller than the overhead of evaluating deformation energy, since higher nodes usually have many triangles. Thus too many high-level certificates can hurt the performance slightly (see Figure 9b).

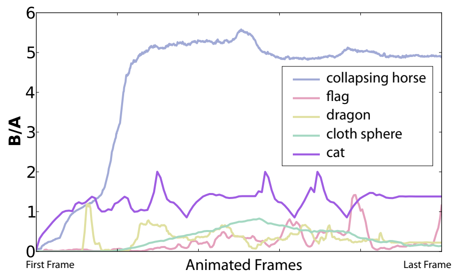


Figure 11: Conservativeness

Conservativeness: We designed a metric to reflect the conservativeness of the certificates. Validating certificates at each node can be a waste of computation if it can not cull SCD tests for collision-free nodes. Therefore, we count the number, A, of succeed certificate validations which cull the SCD tests as expected, as well as the number, B, of false positive leaf-node intersection tests for which there is no self collision, but certificate validations cannot cull the tests. We use the ratio B/A as a conservativeness metric. Figure 11 shows how it changes for different examples. This metric varies a lot for different test cases. In the examples with lots of self collisions (e.g., the collapsing horse), many triangles that are geodesically far on the mesh are close to each other under the mesh deformation. On the BVH, those triangles are in a single BV node or connected BV nodes only when the nodes are at high levels of the BVH. Certificates on those high-level nodes are less conservative, and are more easily violated with small deformations. This metric is also generally consistent with our culling difficulty measure in Table 1 to reflect the effectiveness of self-collision culling.

Memory overhead: At runtime, we load the precomputed certificates, the sparse Laplace-Beltrami matrices and the coefficients of separating planes into memory. In practice, all the certificates, the Laplace-Beltrami matrices, and separating plane coefficients are stored in single precision (32-bit floats). The memory requirements of this ESCC data for our examples were: squishy ball (354.8 MB), monkeys (83.2 MB), flowing cloth (24.9 MB), dragon (37.5 MB), dance (8.3 MB), low-resolution bunny (7.9 MB), high-resolution bunny (34.8 MB), and cloth sphere (9.5 MB),

Difficult Cases: As reported in Table 1, there are certain cases where ESCC produces little speedup. Those are the examples involving large deformations and self-collisions in many parts of the mesh, e.g., the "collapsing horse" in [Schvartzman et al. 2010]. For those examples, our conservative certificate-based culling is ineffective, and the method has to traverse the BVH to the leaf level to detect triangle-triangle collisions. In the last column of Table 1, we use a metric defined as the ratio of the number of inter-node overlap tests between disconnected leaf nodes to the total number of leaf nodes to indicate the culling difficulty. For the examples where this ratio is large, the ESCC method could not offer good speedups.

The techniques proposed in this paper focus on fast *self-collision* detection. In practice, inter-object collision detection is another expensive part for resolving collisions. For simulations where many objects close to each other are involved (such as the "flamenco dancer" in [Curtis et al. 2008]), inter-object collision tests can dominate collision processing timings, especially as self-collision tests are reduced greatly. In such cases, the overall speedup would be low. However, our method is complementary to other types of intra-object and inter-object collision detection techniques, and combining ESCC with other methods may produce higher performance.

9 Conclusion

We have introduced energy-based self-collision culling (ESCC), a new technique for accelerating self-collision detection (SCD) for triangle meshes undergoing arbitrary deformations. Using bounding volume hierarchies augmented with ESCC certificates enables significant speedups, as demonstrated on numerous examples. Our certificate preprocess enables rapid computation of certificates by exploiting numerous algorithmic and mathematical insights.

Limitations and Future Work: ESCC certificates are complementary to many existing SCD approaches, and future work should investigate combining ESCC culling with other methods for narrow-phase culling, such as [Curtis et al. 2008; Schvartzman et al. 2010]. Our precomputed sub-mesh certificates require a fixed BVH topology, and so one cannot adapt the BVH at runtime, which may reduce speedups for highly deformable models such as cloth. Our ESCC method can be further accelerated if some runtime energy computations can be shared with other code, e.g., a physics simulator which already computes, $\mathsf{L}\boldsymbol{x}$. Future work should investigate the possibility of other energy models. Our affine-invariant Laplacian-based energy model enables fast runtime evaluation, material parameter independence, and a fast certificate preprocess; however, one could use more sophisticated deformation energy models provided that the runtime and precomputation costs were fast enough. For example, a user simulating cloth with a hyperelastic potential energy model might also use that model for ESCC, thereby benefitting from embedded energy computations. Finally, while we believe that BVHs are ideal for exploiting ESCC certificates, it may be possible to use ESCC certificates with other SCD approaches, e.g., spatial partitionings [Teschner et al. 2005].

A Analytical Solution of LCQP Problem (8)

Here we present the analytical solution to the LCQP problem (8) which evaluates the optimal deformation energy for specified contact locations given by the barycentric coordinates α and β. Suppose there are n vertices on the mesh, then the displacement vector $\boldsymbol{u}$ is of length $3n$, and K is a $3n \times 3n$ matrix. First, we introduce some notations: let $\boldsymbol{a} = \left[\alpha_1\ \alpha_2\ \alpha_3\ -\beta_1\ -\beta_2\ -\beta_3\right]^T$, $\mathsf{B} = \left[\boldsymbol{x}_1^i\ \boldsymbol{x}_2^i\ \boldsymbol{x}_3^i\ \boldsymbol{x}_1^j\ \boldsymbol{x}_2^j\ \boldsymbol{x}_3^j\right]^T$, and

$$\mathsf{A} = \left[0\ \dots\ \alpha_1\boldsymbol{I}\ \alpha_2\boldsymbol{I}\ \alpha_3\boldsymbol{I}\ \dots\ 0\ \dots\ -\beta_1\boldsymbol{I}\ -\beta_2\boldsymbol{I}\ -\beta_3\boldsymbol{I}\ \dots\ 0\right],$$

where $\boldsymbol{I}$ is a 3×3 identity matrix, A is a $3 \times 3n$ sparse matrix, and the positions of the 3×3 block matrices $\alpha_t\boldsymbol{I}, -\beta_t\boldsymbol{I}, t = 1, 2, 3$ in A correspond to the positions of vertices $\boldsymbol{X}_t^i, \boldsymbol{X}_t^j, t = 1, 2, 3$ in the vector $\boldsymbol{u}$. Consequently, the equality constraint of (8) becomes

$$\mathsf{A}\boldsymbol{u} + \mathsf{B}^T\boldsymbol{a} = 0. \tag{15}$$

Using Lagrange multipliers, the displacement $\boldsymbol{u}$ should satisfy the following equation at the optimum,

$$\mathsf{K}\boldsymbol{u} = \mathsf{A}^T\boldsymbol{\lambda}, \tag{16}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^3$ are Lagrange multipliers. Note that although K is rank-3 deficient (recall §4), equation (16) can be exactly satisfied with an infinite number of $\boldsymbol{u}$. This is because the null space of K spans the displacements of rigid translation, and the matrix A, whose sum of each row is always zero, has vanished projection on the null space of K. All the solutions of (16) yield the same energy value $\boldsymbol{u}^T\mathsf{K}\boldsymbol{u}$, and can be expressed as $\boldsymbol{u} = \mathsf{K}^\dagger\mathsf{A}^T\boldsymbol{\lambda}$, where $\mathsf{K}^\dagger$ is the Moore-Penrose pseudo-inverse of K. Then $\boldsymbol{\lambda}$ can be determined using the constraint (15). Namely,

$$\mathsf{A}\boldsymbol{u} = \mathsf{A}\mathsf{K}^\dagger\mathsf{A}^T\boldsymbol{\lambda} = -\mathsf{B}^T\boldsymbol{a}. \tag{17}$$

Note that the K matrix can be seen as an $n \times n$ block matrix, in which each block is a scaled 3×3 identity matrix, $a\boldsymbol{I}_{3\times3}$, and similarly A can be seen as a $1 \times n$ block matrix. We first condense K into a $n \times n$ matrix $\tilde{\mathsf{K}}$, where each element $\tilde{\mathsf{K}}_{ij}$ is the scalar of the corresponding 3×3 block in K. It can be shown that $\tilde{\mathsf{K}}$ is a rank-1 deficient matrix, and its pseudo-inverse $\tilde{\mathsf{K}}^\dagger$ is the condensed version of $\mathsf{K}^\dagger$. Moreover, A is sparse, having only non-zero blocks corresponding to the involved 6 vertices. The left-hand side of (17) can be written as a 6×6 quadratic form,

$$\mathsf{A}\mathsf{K}^\dagger\mathsf{A}^T = \boldsymbol{a}^T\tilde{\mathsf{K}}_s^\dagger\boldsymbol{a}\boldsymbol{I}_{3\times3},$$

where $\tilde{\mathsf{K}}_s^\dagger$ is a 6×6 sub-matrix of $\tilde{\mathsf{K}}^\dagger$, consisting of the elements of $\tilde{\mathsf{K}}^\dagger$ at the positions corresponding the 6 involved vertices. Therefore, $\boldsymbol{\lambda}$ has the quotient form,

$$\boldsymbol{\lambda} = \frac{-\mathsf{B}^T\boldsymbol{a}}{\boldsymbol{a}^T\tilde{\mathsf{K}}_s^\dagger\boldsymbol{a}}.$$

And hence the optimum energy value is

$$\hat{E}_{ij} = \boldsymbol{u}^T\mathsf{K}\boldsymbol{u} = \boldsymbol{u}^T\mathsf{A}^T\boldsymbol{\lambda} = -(\mathsf{B}\boldsymbol{a})^T\boldsymbol{\lambda} = \frac{\boldsymbol{a}^T\mathsf{B}\mathsf{B}^T\boldsymbol{a}}{\boldsymbol{a}^T\tilde{\mathsf{K}}_s^\dagger\boldsymbol{a}}. \tag{18}$$

B Fast Precomputation of Green's function

As shown in appendix A, the certificate precomputation involves computing the Green's function $\mathsf{G} = \tilde{\mathsf{K}}^\dagger$ of the $n \times n$ matrix $\tilde{\mathsf{K}}$ condensed from K. Direct computation requires the SVD or eigen-decomposition of $\tilde{\mathsf{K}}$ (recall that $\tilde{\mathsf{K}}$ is a rank-1 deficient symmetric positive semi-definite matrix). Let the thin SVD of $\tilde{\mathsf{K}}$ be $\tilde{\mathsf{K}} = \mathsf{V}\mathsf{S}\mathsf{V}^T$, where V is $n \times (n-1)$ orthonormal matrix, and S is an $(n-1) \times (n-1)$ diagonal matrix, then the pseudo-inverse is $\tilde{\mathsf{K}}^\dagger = \mathsf{V}\mathsf{S}^{-1}\mathsf{V}^T$. This computation requires $O(n^3)$ work for SVD or eigen-decomposition.

In this section, instead of computing $\tilde{\mathsf{K}}^\dagger$ directly, we present a fast computation of a matrix G equivalent to $\tilde{\mathsf{K}}^\dagger$ in the sense that

$$\boldsymbol{a}^T\mathsf{G}\boldsymbol{a} = \boldsymbol{a}^T\tilde{\mathsf{K}}^\dagger\boldsymbol{a} \tag{19}$$

for all possible $\boldsymbol{a}$, where $\boldsymbol{a}$ corresponds to barycentric coordinates of two points on the triangle, and has the form $\boldsymbol{a} = \left[0\ \dots\ \alpha_1\ \alpha_2\ \alpha_3\ \dots\ 0\ \dots\ -\beta_1\ -\beta_2\ -\beta_3\ \dots\ 0\right]$, with $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\beta_1 + \beta_2 + \beta_3 = 1$. The equivalence (19) is sufficient for the certificate computation using (18).

First, note that the null space of $\tilde{\mathsf{K}}^\dagger$ is spanned by the vector $\boldsymbol{v} = \left[1\ 1\ \dots\ 1\right]_{1\times n}^T$, and $\boldsymbol{v}^T\boldsymbol{a} = 0$. Then the vector $\boldsymbol{a}$ can be written using the eigen-matrix of $\tilde{\mathsf{K}}$ as $\boldsymbol{a} = \mathsf{V}\boldsymbol{k}$. Now given an $n \times (n-1)$ matrix U such that $\mathsf{V}^T\mathsf{U}$ is a $(n-1) \times (n-1)$ full-rank matrix, then $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ is invertible, and $\mathsf{G} = \mathsf{U}(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}\mathsf{U}^T$ is equivalent to $\tilde{\mathsf{K}}^\dagger$ in the sense of (19). This is because

$$\begin{aligned}\boldsymbol{a}^T\mathsf{U}(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}\mathsf{U}^T\boldsymbol{a} &= \boldsymbol{k}^T\mathsf{V}^T\mathsf{U}(\mathsf{U}^T\mathsf{V}\mathsf{S}\mathsf{V}^T\mathsf{U})^{-1}\mathsf{U}^T\mathsf{V}\boldsymbol{k}\\ &= \boldsymbol{k}^T\mathsf{S}^{-1}\boldsymbol{k} = \mathsf{k}^T\mathsf{V}^T\mathsf{V}\mathsf{S}^{-1}\mathsf{V}^T\mathsf{V}\boldsymbol{k}\\ &= \boldsymbol{a}^T\tilde{\mathsf{K}}^\dagger\boldsymbol{a}.\end{aligned}$$

In practice, we use a sparse matrix $\mathsf{U} = \begin{bmatrix} \boldsymbol{I}_{n\times n} & \mathbf{0}_{n\times 1} \end{bmatrix}^T$, and then $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ is just the $(n-1)\times(n-1)$ upper-left sub-matrix of $\tilde{\mathsf{K}}$. We compute $\tilde{\mathsf{G}} = (\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})^{-1}$ using the sparse Cholesky factorization of $\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U}$ (with complexity $O(n^{\frac{3}{2}})$) and a solve $(\mathsf{U}^T\tilde{\mathsf{K}}\mathsf{U})\tilde{\mathsf{G}} = \boldsymbol{I}_{(n-1)\times(n-1)}$ (with complexity $O(n^2 \lg n)$). Both the factorization and back-substitution for the solve can be efficiently performed in parallel using the PARDISO direct solver [Schenk and Gärtner 2004]. Finally, the $n \times n$ G matrix is a simple expansion of $\tilde{\mathsf{G}}$,

$$\mathsf{G} = \mathsf{U}\tilde{\mathsf{G}}\mathsf{U}^T = \begin{bmatrix} \tilde{\mathsf{G}} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}.$$

For complexity estimates, we used the fact that a 2D grid Laplacian on n vertices suggests an $O(n^{\frac{3}{2}})$ cost for sparse Cholesky factorization, and $O(n^2 \lg n)$ for back-substitution (or less if done in parallel) to get G assuming the Cholesky factors require $O(n \lg n)$ space [Demmel 1997].

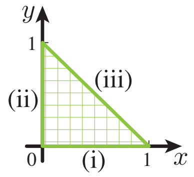
C Exact Evaluation of Tri-Tri Certificates

In this section, we present the details of computing the triangle-triangle certificate by solving optimization problems (11) and (12). Both problems are instances of the more general problem,

$$\begin{aligned} &\text{minimize} && \frac{\boldsymbol{x}^T\mathsf{A}\boldsymbol{x} + \boldsymbol{a}^T\boldsymbol{x} + c_u}{\boldsymbol{x}^T\mathsf{B}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c_d} \\ &\text{subject to} && \boldsymbol{x} \in \mathcal{C}, \end{aligned} \tag{20}$$

where both A and B are symmetric positive-definite matrices, $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ is a 2D vector, and $\mathcal{C}$ is a convex set on 2D space. Recall that $\mathcal{C}$ is a triangle for problem (11), and a square for problem (12).

C.1 Optimum value on the boundary



First we consider the optimum value on the boundary of $\mathcal{C}$. As presented in §6.1, a boundary point of problem (11) corresponds to a case where a vertex touches an edge, and so does a boundary point of problem (12). Therefore, we only need to compute the boundary optimum for (11). There are 3 segments on the boundary: (i) $\boldsymbol{x} = \begin{bmatrix} a & 0 \end{bmatrix}^T, a \in [0,1]$, (ii) $\boldsymbol{x} = \begin{bmatrix} 0 & a \end{bmatrix}^T, a \in [0,1]$, and (iii) $\boldsymbol{x} = \begin{bmatrix} a & 1-a \end{bmatrix}^T, a \in [0,1]$. Substituting them respectively into (20), we obtain a 1D objective function,

$$f(a) = \frac{A_u a^2 + B_u a + C_u}{A_d a^2 + B_d a + C_d}.$$

For the segment (i),

$$\begin{aligned} A_u &= \mathsf{A}_{11},\ B_u = \boldsymbol{a}_1,\ C_u = c_u \\ A_d &= \mathsf{B}_{11},\ B_d = \boldsymbol{b}_1,\ C_d = c_d. \end{aligned}$$

For the segment (ii),

$$\begin{aligned} A_u &= \mathsf{A}_{22},\ B_u = \boldsymbol{a}_2,\ C_u = c_u \\ A_d &= \mathsf{B}_{22},\ B_d = \boldsymbol{b}_2,\ C_d = c_d \end{aligned}$$

For the segment (iii),

$$\begin{aligned} A_u &= \mathsf{A}_{11} - 2\mathsf{A}_{12} + \mathsf{A}_{22}, \\ B_u &= 2(\mathsf{A}_{12} - \mathsf{A}_{22}) + \boldsymbol{a}_1 - \boldsymbol{a}_2, \\ C_u &= \mathsf{A}_{22} + \boldsymbol{a}_2 + c_u \\ A_d &= \mathsf{B}_{11} - 2\mathsf{B}_{12} + \mathsf{B}_{22}, \\ B_d &= 2(\mathsf{B}_{12} - \mathsf{B}_{22}) + \boldsymbol{b}_1 - \boldsymbol{b}_2, \\ C_d &= \mathsf{B}_{22} + \boldsymbol{b}_2 + c_d. \end{aligned}$$

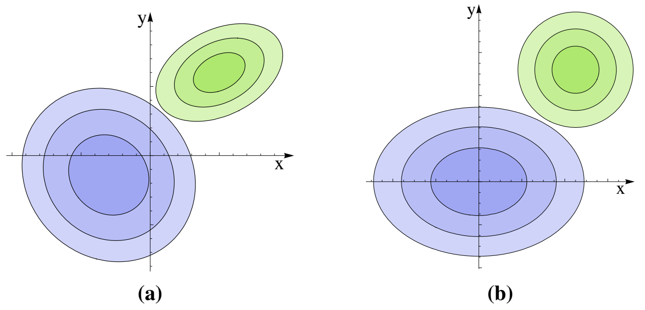


Figure 12: Simplification of the objective function: *(a) The numerator and denominator of the objective function in* (20) *are plotted as two elliptical contours. (b) We apply an affine transformation to regularize the objective function into a simpler form* (21)*, in which the numerator has a circular contour and the denominator has an axis-aligned elliptical contour centered at the origin.*

To compute the optimum on a segment, taking $f'(a) = 0$, we get a cubic equation. Fortunately, we observe that for all cases the 3rd-order coefficient always vanishes, and therefore we only need to solve the quadratic equation $A_q a^2 + B_q a + C_q = 0$, where $A_q = A_u B_d - B_u A_d$, $B_q = 2(C_d A_u - C_u A_d)$, and $C_q = C_d B_u - C_u B_d$. If the solution of these quadratic equations is in $[0,1]$, we compute its corresponding optimum value $f(a)$. Otherwise, the optimum occurs at the segment end-points, i.e., $f(0)$ or $f(1)$.

C.2 Optimum value in the interior of domain

Next we check if an optimum appears in the interior of $\mathcal{C}$. For each interior optimum, we compute the optimum value, compare it with the boundary optimum values, and take the minimum. This case is more involved. To ease the derivation, we first regularize the problem (20) into a simpler form. Both the numerator and denominator of (20) are 2-D quadratic forms, representing two sets of ellipse contours on the 2-D plane (see Figure 12a). Using an affine transformation $\psi : \boldsymbol{x} \mapsto \mathsf{F}_a\boldsymbol{x} + \boldsymbol{t}_a$, we simplify (20) into the following form (see Figure 12b),

$$\begin{aligned} &\text{minimize} && \frac{(u - c_x)^2 + (v - c_y)^2 + \tilde{c}_u}{au^2 + bv^2 + \tilde{c}_d} \\ &\text{subject to} && \begin{bmatrix} u & v \end{bmatrix}^T \in \psi[\mathcal{C}]. \end{aligned} \tag{21}$$

In practice, this simplification is performed in three steps: (i) we compute the eigen-decomposition, $\mathsf{A} = \mathsf{V}\Sigma\mathsf{V}^T$, to simplify the numerator of (20), and using $\boldsymbol{x} = \mathsf{V}\Sigma^{-1/2}\mathsf{V}^T\boldsymbol{y}$, the objective function of (20) becomes into

$$\frac{\boldsymbol{y}^T\boldsymbol{y} + \boldsymbol{a}^T\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + c_u}{\boldsymbol{y}^T\Sigma^{-\frac{1}{2}}\mathsf{V}^T\mathsf{B}\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + \boldsymbol{b}^T\mathsf{V}\Sigma^{-\frac{1}{2}}\boldsymbol{y} + c_d} \equiv \frac{\boldsymbol{y}^T\boldsymbol{y} + \tilde{\boldsymbol{a}}^T\boldsymbol{y} + c_u}{\boldsymbol{y}^T\tilde{\mathsf{B}}\boldsymbol{y} + \tilde{\boldsymbol{b}}^T\boldsymbol{y} + c_d},$$

(ii) next we compute the eigen-decomposition, $\tilde{\mathsf{B}} = \mathsf{U}\mathsf{D}\mathsf{U}^T$, to diagonalize the denominator: using $\boldsymbol{z} = \mathsf{U}\boldsymbol{y}$ further transforms the above objective function into

$$\frac{\boldsymbol{z}^T\boldsymbol{z} + \tilde{\boldsymbol{a}}^T\mathsf{U}\boldsymbol{z} + c_u}{\boldsymbol{z}^T\mathsf{D}\boldsymbol{z} + \tilde{\boldsymbol{b}}^T\mathsf{U}\boldsymbol{z} + c_d} \equiv \frac{\boldsymbol{z}^T\boldsymbol{z} + \hat{\boldsymbol{a}}^T\boldsymbol{z} + c_u}{\boldsymbol{z}^T\mathsf{D}\boldsymbol{z} + \hat{\boldsymbol{b}}^T\boldsymbol{z} + c_d},$$

and (iii) we take the translation $\boldsymbol{z} = \boldsymbol{v} - \frac{1}{2}\mathsf{D}^{-1}\hat{\boldsymbol{b}}$, and finally simplify the objective function into

$$\frac{\boldsymbol{v}^T\boldsymbol{v} + (\hat{\boldsymbol{a}} - \mathsf{D}^{-1}\hat{\boldsymbol{b}})^T\boldsymbol{v} + c_u + \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-2}\hat{\boldsymbol{b}} - \frac{1}{2}\hat{\boldsymbol{a}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}}{\boldsymbol{v}^T\mathsf{D}\boldsymbol{v} + c_d - \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}}.$$

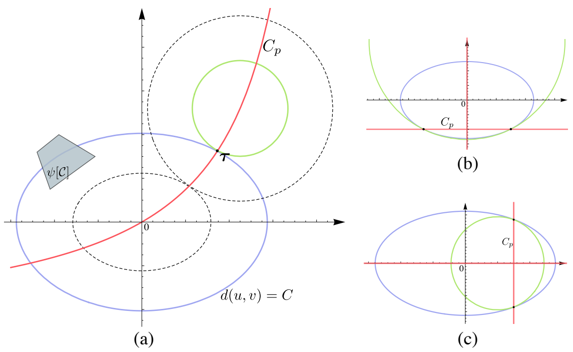


Figure 13: Contour optimum: *The optimum of objective function* (21) *on elliptical contours form the red curve which can be analytically determined. (a) shows the general case, and (b) and (c) illustrate the special cases where the center of the circular contours of the numerator is on x- or y- axes.*

This form of objective function agrees with (21): let $\boldsymbol{p} \equiv \hat{\boldsymbol{a}} - \mathsf{D}^{-1}\hat{\boldsymbol{b}}$, then $c_x = -\boldsymbol{p}_1/2$, $c_y = -\boldsymbol{p}_2/2$, $a = \mathsf{D}_{11}$, $b = \mathsf{D}_{22}$,

$$\tilde{c}_u = c_u + \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-2}\hat{\boldsymbol{b}} - \frac{1}{2}\hat{\boldsymbol{a}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}} - c_x^2 - c_y^2, \text{ and}$$
$$\tilde{c}_d = c_d - \frac{1}{4}\hat{\boldsymbol{b}}^T\mathsf{D}^{-1}\hat{\boldsymbol{b}}.$$

Affine transformation preserves convexity of a domain, therefore $\psi[\mathcal{C}]$ is still a convex set. Since the simplified problem (21) is equivalent to (20), from now on, we check if the optimum of (21) can be achieved in the interior of $\psi[\mathcal{C}]$, and then compute the optimum value if it is inside of $\psi[\mathcal{C}]$. Let $n(u,v)$ and $d(u,v)$ denote the numerator and denominator respectively in the objective function of (21). Consider the objective value on a single contour $\Omega_C : d(u,v) = C$ (see blue ellipse in Figure 13). The method of Lagrange multipliers shows that the local optimum of the objective function $\frac{n(u,v)}{d(u,v)}$ on the contour Ω_C occurs when a contour of $n(u,v)$ meets Ω_C tangentially (see green circle in Figure 13a). Let $\boldsymbol{\tau}$ denote the optimum point on $d(u,v) = C$. If $\boldsymbol{\tau}$ is outside of $\psi[\mathcal{C}]$, then the optimum on the set $\Omega_C \cap \psi[\mathcal{C}]$ (see the gray polygon in Figure 13a) is on the boundary of $\psi[\mathcal{C}]$, in which case the optimum has been computed as presented in Appendix C.1. Otherwise, we need to compute the interior-point optimum.

Now put together the optimum points of all contours of $d(u,v)$. We observe that they form a curve C_p (see red curve in Figure 13a) which is analytically determined by the following function,

$$y = \frac{1}{C_A x + C_B} - \frac{1}{C_B}, \tag{22}$$

where $C_A = \frac{-(b-a)^2}{abc_xc_y}$ and $C_B = \frac{b-a}{ac_y}$. Then we determine if the curve C_p intersects with the domain $\psi[\mathcal{C}]$ by checking the intersection of C_p and the piecewise boundary segments of $\psi[\mathcal{C}]$. In particular, to check the intersection of C_p and a boundary segment defined by its two end points (x_1, y_1) and (x_2, y_2), we solve a quadratic equation $A_b a^2 + B_b a + C_b = 0$, where

$$A_b = C_A C_B (x_1 - x_2)(y_1 - y_2),$$
$$B_b = C_B^2 (y_1 - y_2) + C_A[(x_1 - x_2) + C_B(x_1y_2 + x_2y_1 - 2x_2y_2)],$$
$$C_b = C_B^2 y_2 + C_A(x_2 + C_B x_2 y_2).$$

The curve C_p intersects with the boundary segment if a root a of this quadratic equation is in $[0,1]$. For problem (11), its domain has 3 boundary segments, hence it requires 3 quadratic solves, and for problem (12), it needs 4 quadratic solves to check intersections. If C_p is separated from $\psi[\mathcal{C}]$, no further computation is needed, because the optimum will occur on the boundary of $\psi[\mathcal{C}]$, and the optimum values have been computed in Appendix C.1. Otherwise, we compute the local optimum $\boldsymbol{q}$ of the objective function $\frac{n(u,v)}{d(u,v)}$ on the curve C_p. If $\boldsymbol{q}$ is outside of $\psi[\mathcal{C}]$, then again the optimum on the set $C_p \cap \psi[\mathcal{C}]$ is on the boundary of $\psi[\mathcal{C}]$, and hence the optimum of the entire domain $\psi[\mathcal{C}]$ occurs on the boundary. If $\boldsymbol{q}$ is in the interior of $\psi[\mathcal{C}]$, we compare its objective value with the minimum value from the boundary (computed in Appendix C.1) and take the minimum. $\boldsymbol{q}$ is computed as follows. Substituting (22) into the objective function of (21), we get a single variable rational function $r(x)$. The optimum occurs when $r'(x) = 0$. This equation is a 6th-order polynomial equation, which fortunately can be factorized into two cubic equations, i.e. $r'(x) = p(x)q(x) = 0$. The first cubic equation $p(x) = P_a x^3 + P_b x^2 + P_c x + P_d$ has the coefficients

$$P_a = (a-b)^3,$$
$$P_b = 3(a-b)^2 bc_x,$$
$$P_c = 3(a-b)b^2c_x^2, \text{ and}$$
$$P_d = b^2 c_x(bc_x^2 + ac_y^2).$$

The discriminant of this cubic equation is always negative, indicating this equation has only one real root. The second cubic equation $q(x) = Q_a x^3 + Q_b x^2 + Q_c x + Q_d$ has the coefficients

$$Q_a = a(a-b)c_x,$$
$$Q_b = (b-a)(a\tilde{c}_u - \tilde{c}_d) + ac_x^2(2b-a) + abc_y^2,$$
$$Q_c = -c_x(\tilde{c}_d(a-2b) + ab(\tilde{c}_u + c_x^2 + c_y^2)), \text{ and}$$
$$Q_d = -b\tilde{c}_d c_x^2.$$

We solve these cubic equations using the method of Nickalls [1993]. In practice, the curve C_p is separated from the domain $\psi[\mathcal{C}]$ in most of the cases. In practice, we only need to solve cubic equations for less than 0.2% of the triangle-triangle certificates.

Care needs to be taken for two special cases where C_p produces lines parallel to the X or Y axes: (i) $c_x = 0$ and (ii) $c_y = 0$. Both cases largely simplify the computation. When $c_x = 0$, C_p are the lines $x = 0$ and $y = \frac{a}{a-b}c_y$ (See Figure 13b); when $c_y = 0$, C_p are the lines $y = 0$ and $x = \frac{b}{b-a}c_x$ (see Figure 13c). Determining the intersection of these straight lines with the convex domain is trivial. When $c_x = 0$, the optimum on the curve C_p always occurs on Y-axis, and is determined by a quadratic equation $A_y y^2 + B_y y + C_y = 0$ (instead of a cubic equation), where

$$A_y = 2bc_y,\ B_y = 2[\tilde{c}_d - b(\tilde{c}_u + c_y^2)], \text{ and } C_y = -2\tilde{c}_d c_y.$$

Symmetrically, when $c_y = 0$, the optimum on the curve C_p appears on X-axis, and is computed by the quadratic equation $A_x x^2 + B_x x + C_x = 0$, where

$$A_x = 2ac_x,\ B_x = 2[\tilde{c}_d - a(\tilde{c}_u + c_x^2)], \text{ and } C_x = -2\tilde{c}_d c_x.$$

Acknowledgments: We thank the anonymous reviewers for their constructive feedback, and Jernej Barbic for early discussions. This work was supported in part by the National Science Foundation (HCC-0905506), fellowships from the Alfred P. Sloan Foundation and the Guggenheim Foundation, and donations from Pixar and Autodesk. This research was conducted in conjunction with the Intel Science and Technology Center – Visual Computing. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or others.

References

BARBIČ, J., AND JAMES, D. L. 2010. Subspace Self-Collision Culling. *ACM Transactions on Graphics 29*, 4 (July), 81:1–81:9.

BRICEÑO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3D animations. In *Symposium on Computer Animation*, 136–146.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. *ACM Transactions on Graphics 21*, 3, 594–603.

BURGES, C. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery 2*, 2, 121–167.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Transactions on Graphics 23*, 3 (Aug.), 905–914.

CURTIS, S., TAMSTORF, R., AND MANOCHA, D. 2008. Fast collision detection for deformable models using representative-triangles. In *Proc. ACM Symp. Interactive 3D Graphics and Games*, 61–69.

DEMMEL, J. 1997. *Applied numerical linear algebra*. SIAM, Philadelphia, PA.

GAO, J., GUIBAS, L., AND NGUYEN, A. 2006. Deformable spanners and applications. *Computational Geometry: Theory and Appl. 35*, 1-2, 2–19.

GOLDSMITH, J., AND SALMON, J. 1987. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5, 14–20.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 171–180.

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

GOVINDARAJU, N., LIN, M., AND MANOCHA, D. 2005. Quick-CULLIDE: Fast inter-and intra-object collision culling using graphics hardware. In *Proc. IEEE Virtual Reality*, 59–66.

GRINSPUN, E., AND SCHRÖDER, P. 2001. Normal bounds for subdivision-surface interference detection. In *IEEE Visualization 2001*, 333–340.

GUIBAS, L., NGUYEN, A., RUSSEL, D., AND ZHANG, L. 2002. Collision Detection for Deforming Necklaces. In *Proc. of the ACM Symposium on Computational Geometry*, 33–42.

GUIBAS, L. 2004. Kinetic Data Structures. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3, 145–152.

HUBBARD, P. M. 1995. *Collision Detection for Interactive Graphics Applications*. PhD thesis, Department of Computer Science, Brown University.

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics 23*, 3 (Aug.), 393–398.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Transactions on Graphics 24*, 3 (Aug.), 399–407.

KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S. B., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Trans. Vis. & Comp. Graphics 4*, 1, 21–36.

MEYER, M., DESBRUN, M., SCHRÖEDER, P., AND BARR, A. H. 2002. Discrete differential geometry operators for triangulated 2-manifolds. In *Proceedings of VisMath*.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. *ACM Transactions on Graphics 30* (Aug.), 92:1–92:10.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. on Graphics 24*, 3, 471–478.

NICKALLS, R. 1993. A new approach to solving the cubic: Cardan's solution revealed. *The Mathematical Gazette 77*, 480, 354–359.

PROVOT, X. 1997. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface*, 177–189.

RIVERS, A. R., AND JAMES, D. L. 2007. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. *ACM Transactions on Graphics 26*, 3, 82:1–82:6.

SCHENK, O., AND GÄRTNER, K. 2004. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems 20*, 3, 475–487.

SCHVARTZMAN, S. C., GASCÓN, J., AND OTADUY, M. A. 2009. Bounded normal trees for reduced deformations of triangulated surfaces. In *Symp. on Computer Animation (SCA)*, 75–82.

SCHVARTZMAN, S. C., LVARO G. PÉREZ, AND OTADUY, M. A. 2010. Star-contours for efficient hierarchical self-collision detection. *ACM Transactions on Graphics 29*, 4 (July), 80:1–80:8.

STEINEMANN, D., OTADUY, M. A., AND GROSS, M. 2008. Fast adaptive shape matching deformations. In *Symposium on Computer Animation*, 87–94.

SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., AND MANOCHA, D. 2006. Fast Proximity Computation Among Deformable Models using Discrete Voronoi Diagrams. *ACM Transactions on Graphics 25*, 3, 1144–1153.

TANG, M., CURTIS, S., YOON, S.-E., AND MANOCHA, D. 2009. Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Trans. on Visualization and Computer Graphics 15*, 544–557.

TESCHNER, M., ET AL. 2005. Collision Detection for Deformable Objects. *Computer Graphics Forum 24*, 1, 61–81.

VAN DEN BERGEN, G. 1997. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools 2*, 4, 1–14.

VOLINO, P., AND MAGNENAT-THALMANN, N. 1994. Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity. *Computer Graphics Forum 13*, 3, 155–166.