GAMES103: Intro to Physics-Based Animation

Collision Handling

Huamin Wang

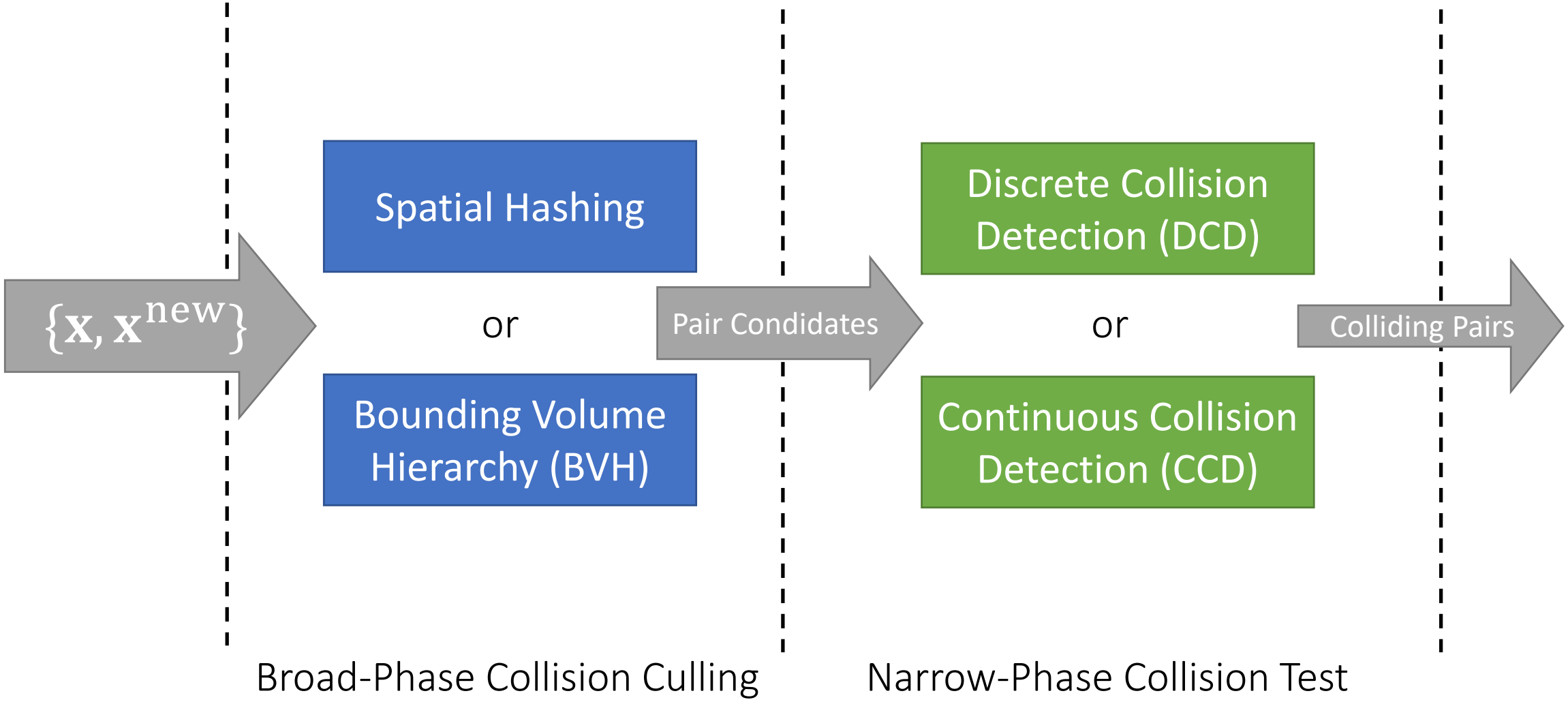
Dec 2021

Topics for the Day

- Collision Detection
- Interior Point Methods
- Impact Zone Optimization
- Untangling Cloth

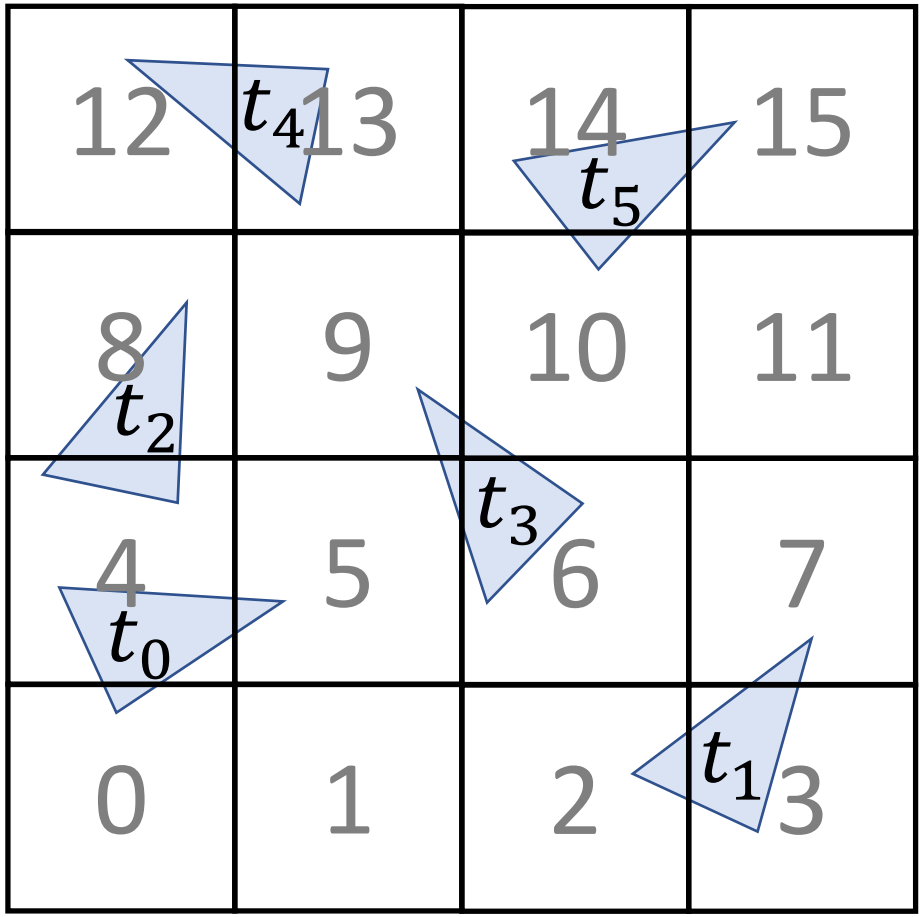
Collision Detection

Collision Detection Pipeline



Spatial Partitioning

Spatial partitioning divides the space by a grid and stores objects into grid cells.



Cell 12: t_4 Cell 13: t_4 Cell 14: t_5 Cell 15: t_5

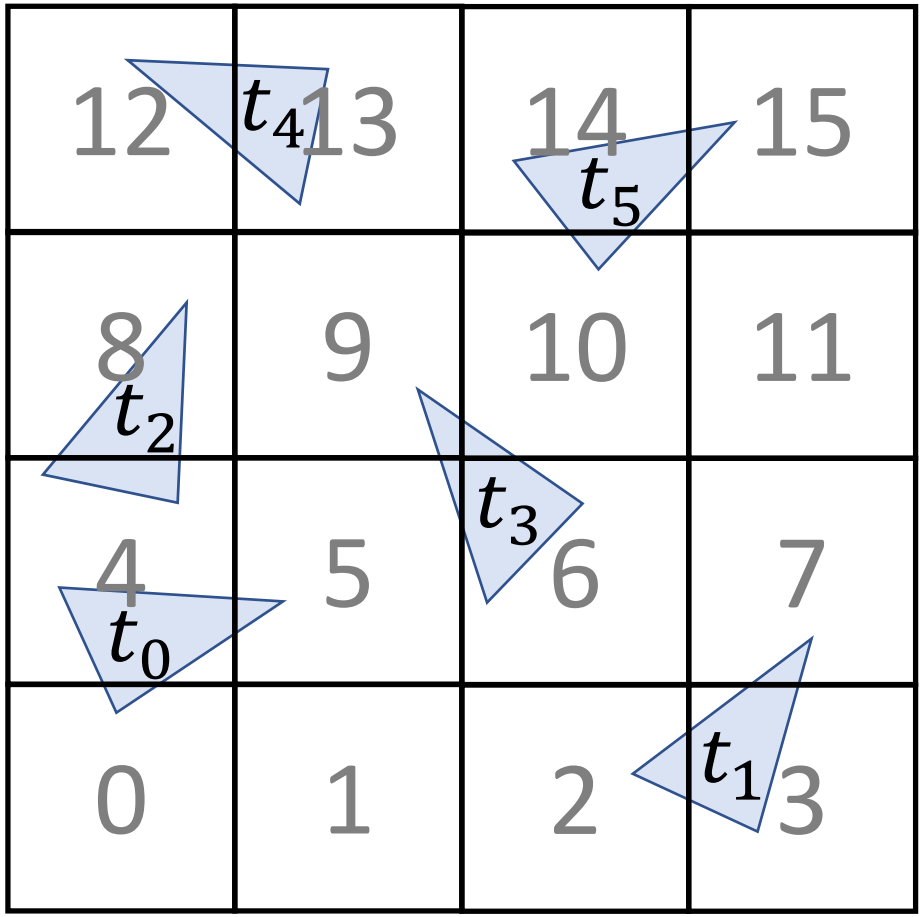
Cell 8: t_2 Cell 9: t_3 Cell 10: t_3, t_5 Cell 11:

Cell 4: t_0, t_2 Cell 5: t_0, t_3 Cell 6: t_3 Cell 7: t_1

Cell 0: t_0 Cell 1: Cell 2: t_1 Cell 3: t_1

Spatial Partitioning

To find pair candidates for collision test, we just have to check the grid cells.



Cell 12: t_4 Cell 13: t_4 Cell 14: t_5 Cell 15: t_5

t_3's pairs

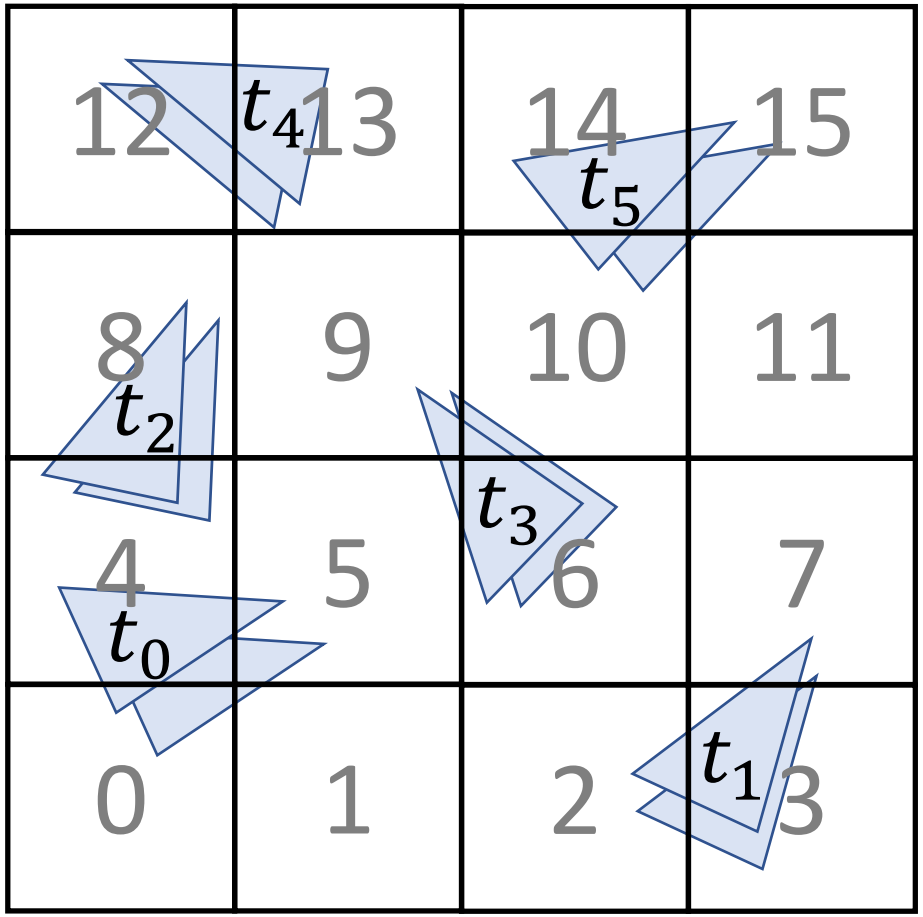
Cell 8: t_2 Cell 9: t_3 Cell 10: t_3, t_5 Cell 11:

Cell 4: t_0, t_2 Cell 5: t_0, t_3 Cell 6: t_3 Cell 7: t_1

Cell 0: t_0 Cell 1: Cell 2: t_1 Cell 3: t_1

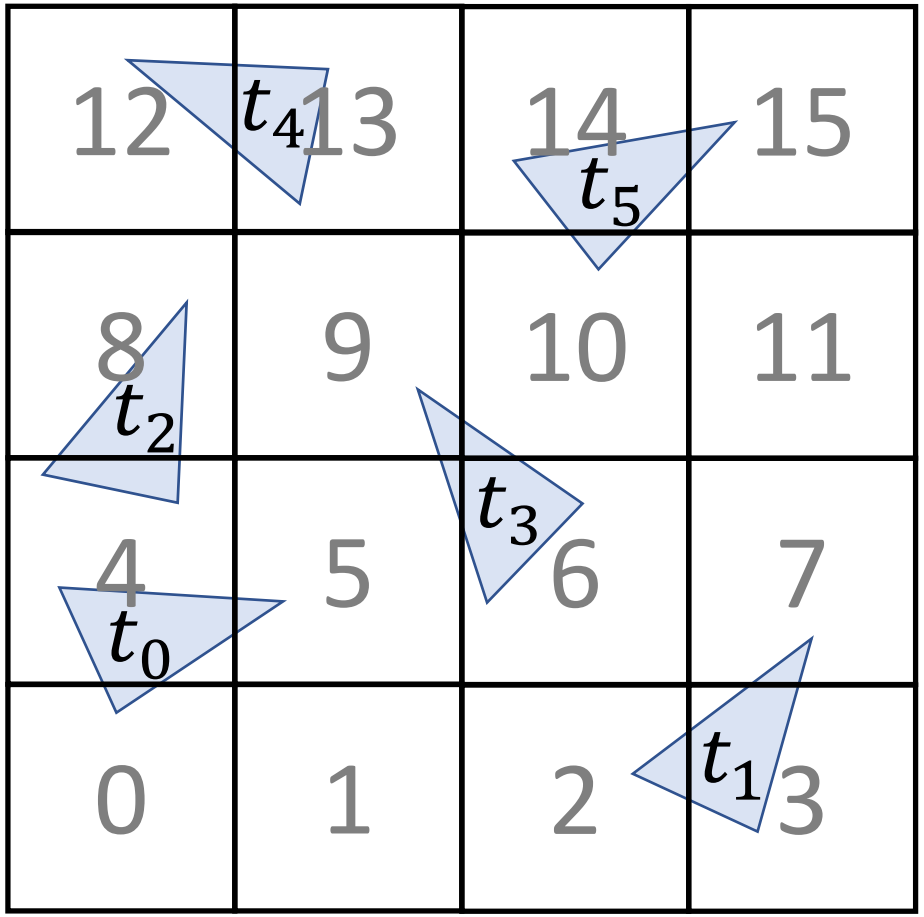
Spatial Partitioning

If we need to consider moving objects, we just expand the object region.



Spatial Partitioning

Instead of allocating memories to cells, we can build an object-cell list and then sort them. This avoids memories wasted in empty cells.



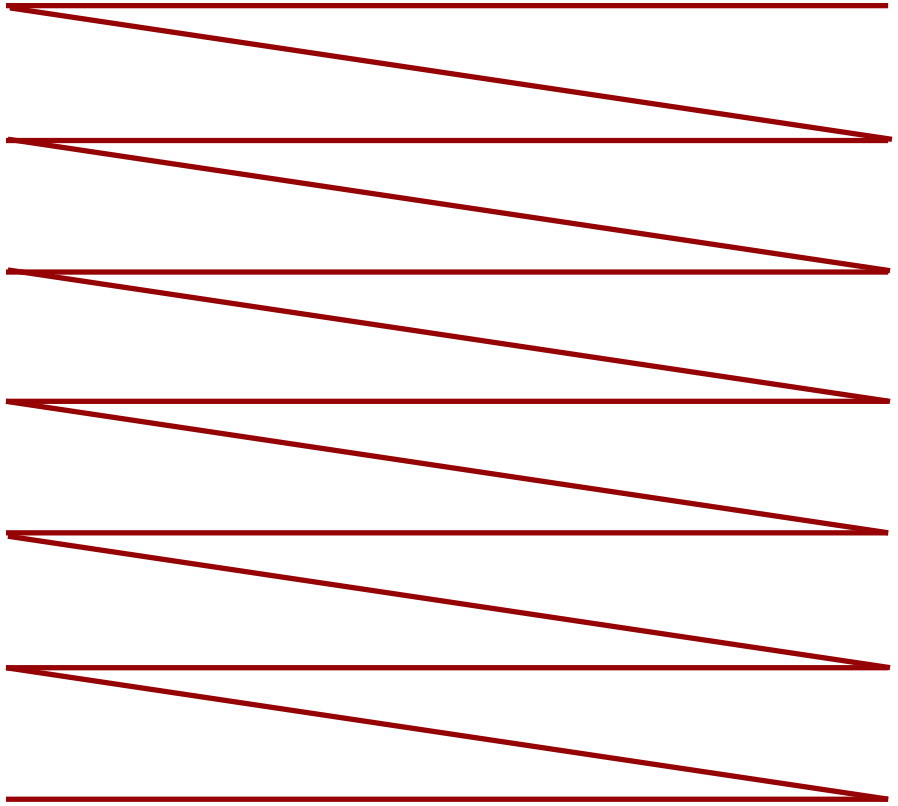
$\{0, t_0\}; \{4, t_0\}; \{5, t_0\}; \{2, t_1\}; \{3, t_1\}; \{7, t_1\}; \{4, t_2\};$
$\{8, t_2\}; \{5, t_3\}; \{6, t_3\}; \{9, t_3\}; \{10, t_3\}; \{12, t_4\};$
$\{13, t_4\}; \{10, t_5\}; \{14, t_5\}; \{15, t_5\}$

Sort by cell ID

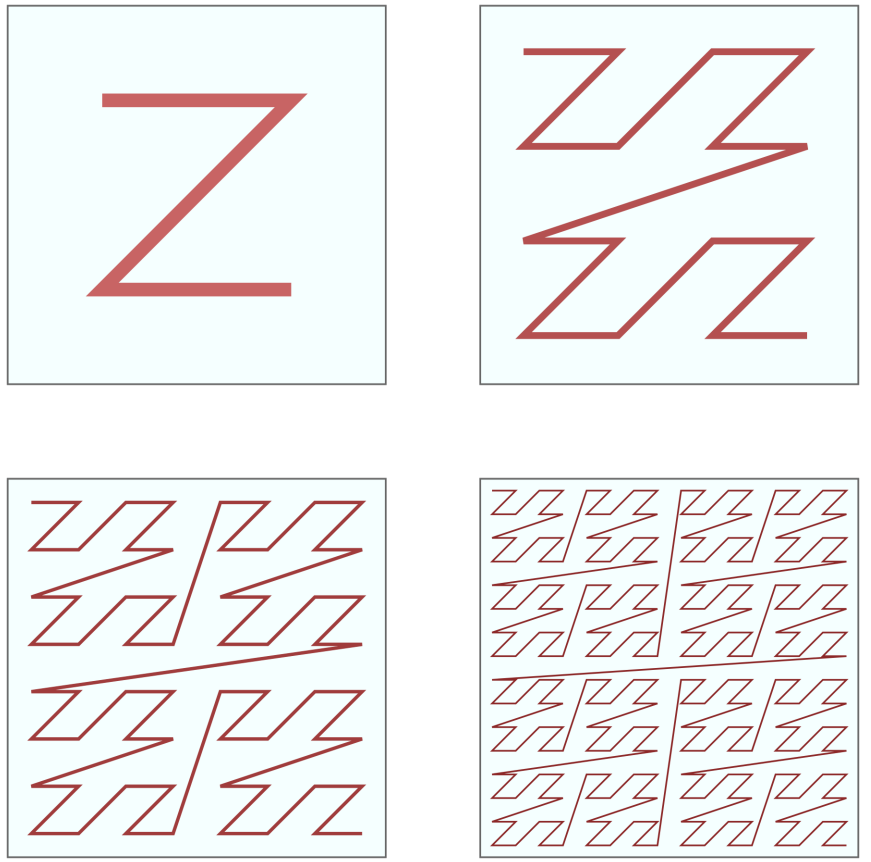
$\{0, t_0\}; \{2, t_1\}; \{3, t_1\}; \{4, t_0\}; \{4, t_2\}; \{5, t_0\}; \{5, t_3\};$
$\{6, t_3\}; \{7, t_1\}; \{8, t_2\}; \{9, t_3\}; \{10, t_3\}; \{10, t_5\};$
$\{12, t_4\}; \{13, t_4\}; \{14, t_5\}; \{15, t_5\}$

Morton Code

One question is how to define the cell ID. Using the grid order is not optimal, since it cannot be easily extended and it is lack of locality. Morton code uses a Z-pattern instead.



Grid order



Morton code order

After-Class Reading

GPU Gems 3

GPU Gems 3 is now available for free online!

The CD content, including demos and content, is available on the web and for download.

You can also subscribe to our Developer News Feed to get notifications of new material on the site.

Chapter 32. Broad-Phase Collision Detection with CUDA

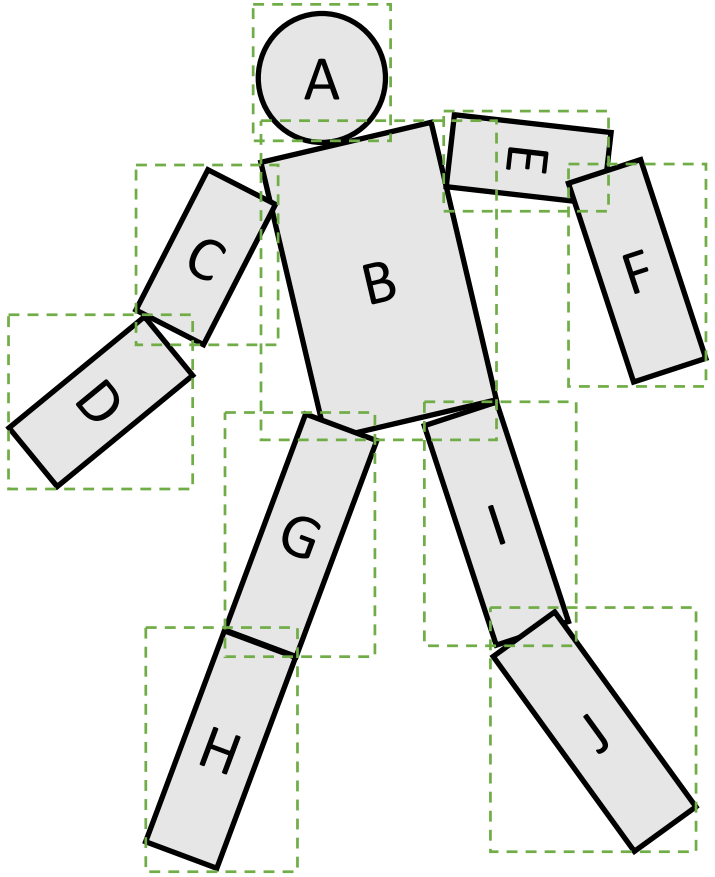
Scott Le Grand
NVIDIA Corporation

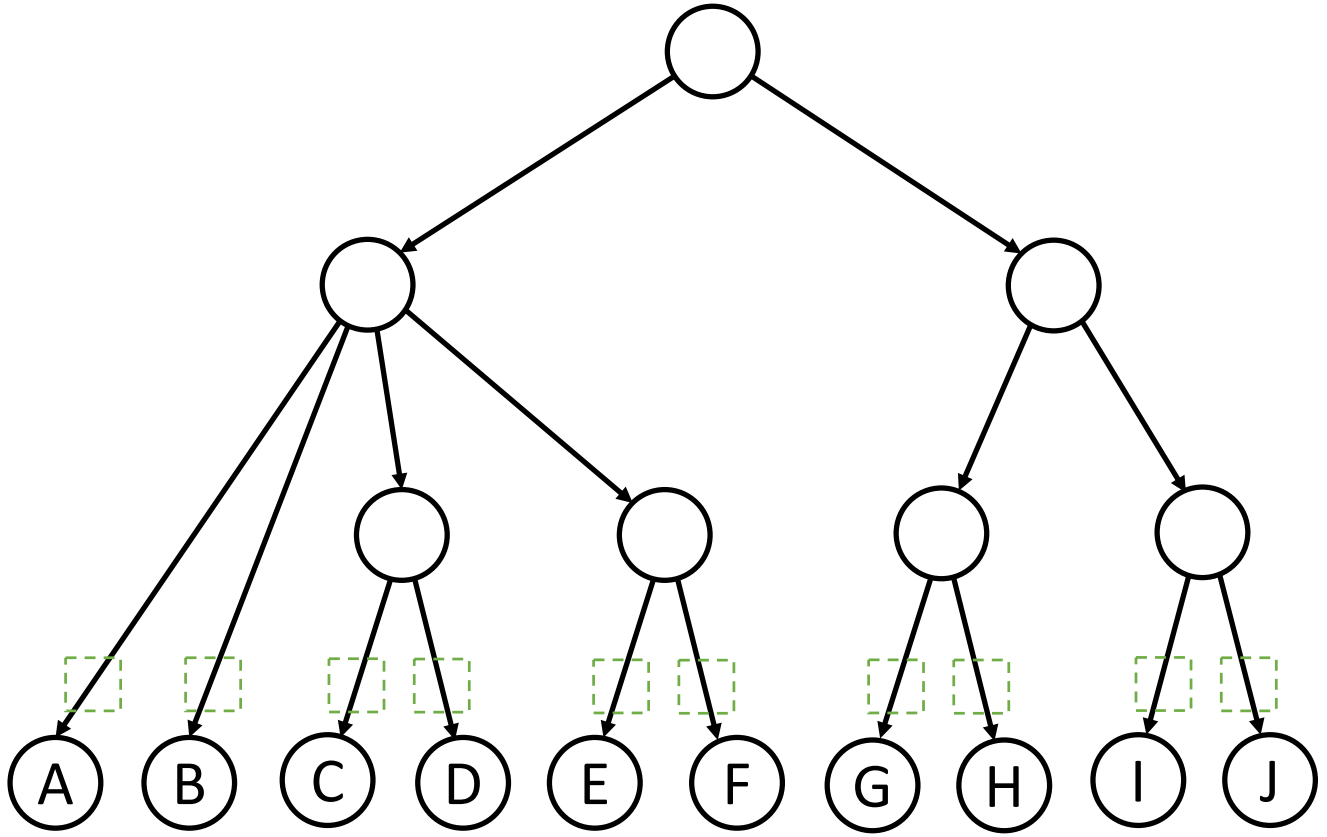
Collision detection among many 3D objects is an important component of physics simulation, computer-aided design, molecular modeling, and other applications. Most efficient implementations use a two-phase approach: a *broad phase* followed by a *narrow phase*. In the broad phase, collision tests are conservative—usually based on bounding volumes only—but fast in order to quickly prune away pairs of objects that do not collide with each other. The output of the broad phase is the potentially colliding set of pairs of objects. In the narrow phase, collision tests are exact—they usually compute exact contact points and normals—thus much more costly, but performed for only the pairs of the potentially colliding set. Such an approach is very suitable when a lot of the objects are actually not colliding—as is the case in practice—so that many pairs of objects are rejected during the broad phase.

In this chapter, we present a GPU implementation of broad-phase collision detection ("broad phase," for short) based on CUDA that is an order of magnitude faster than the CPU implementation.

Bounding Volume Hierarchy

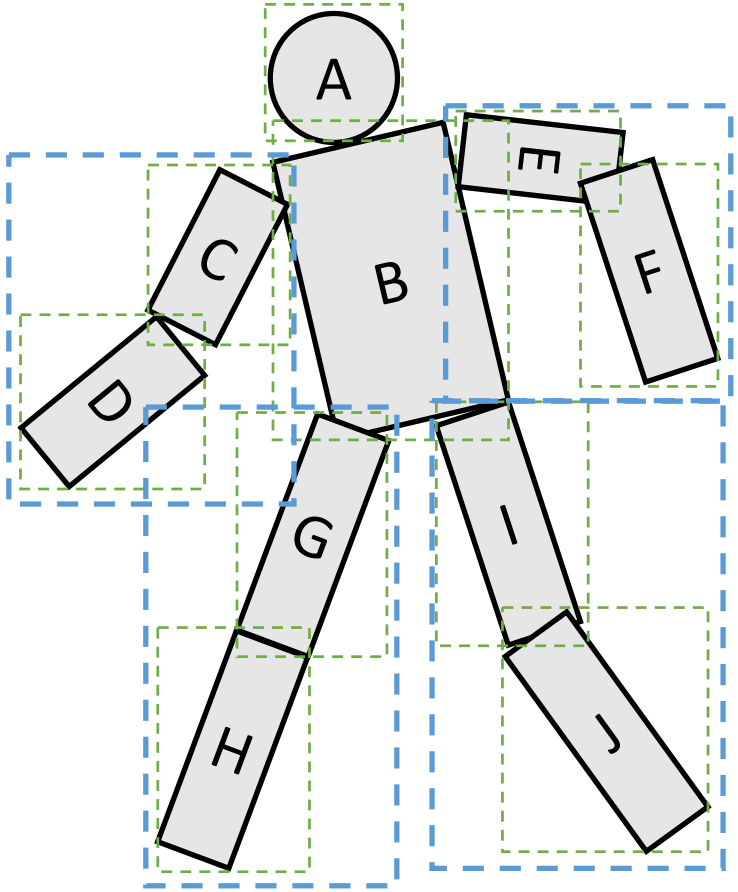
Bounding volume hierarchy is built on geometric/topological proximity of objects.

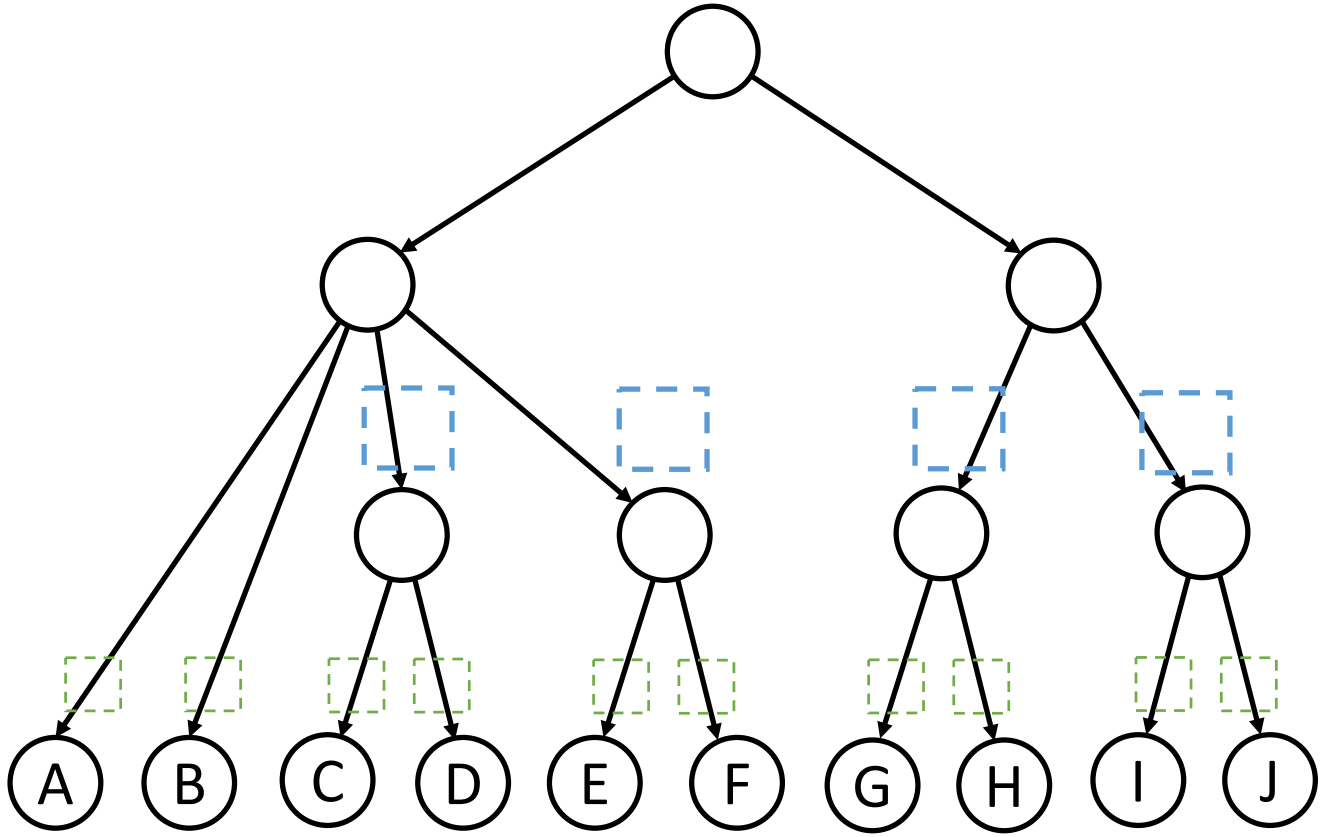




Bounding Volume Hierarchy

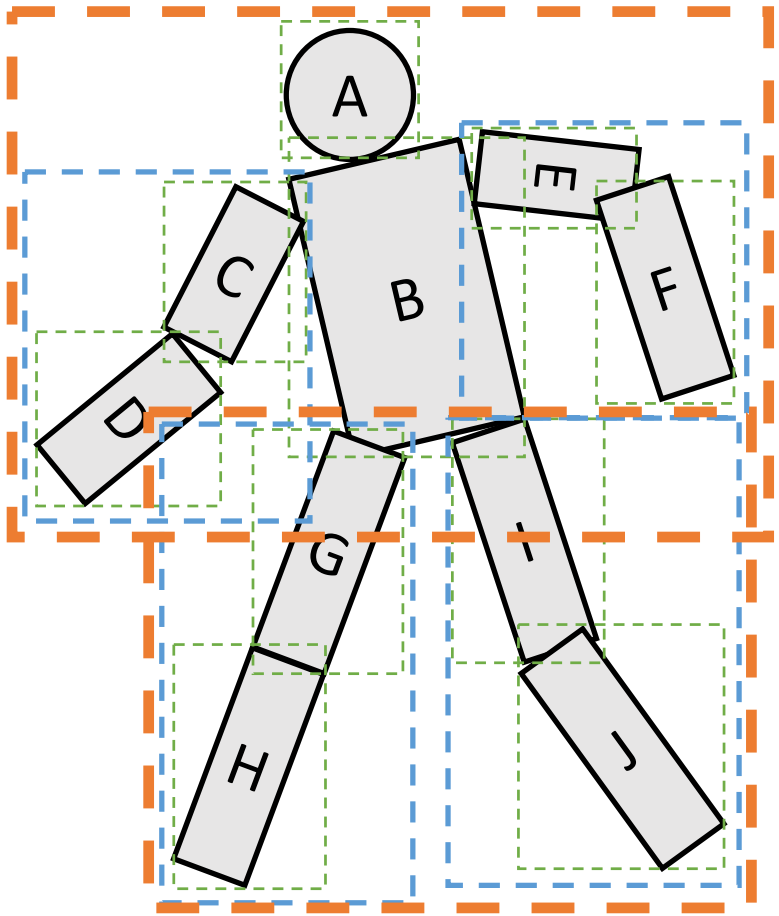
Bounding volume hierarchy is built on geometric/topological proximity of objects.

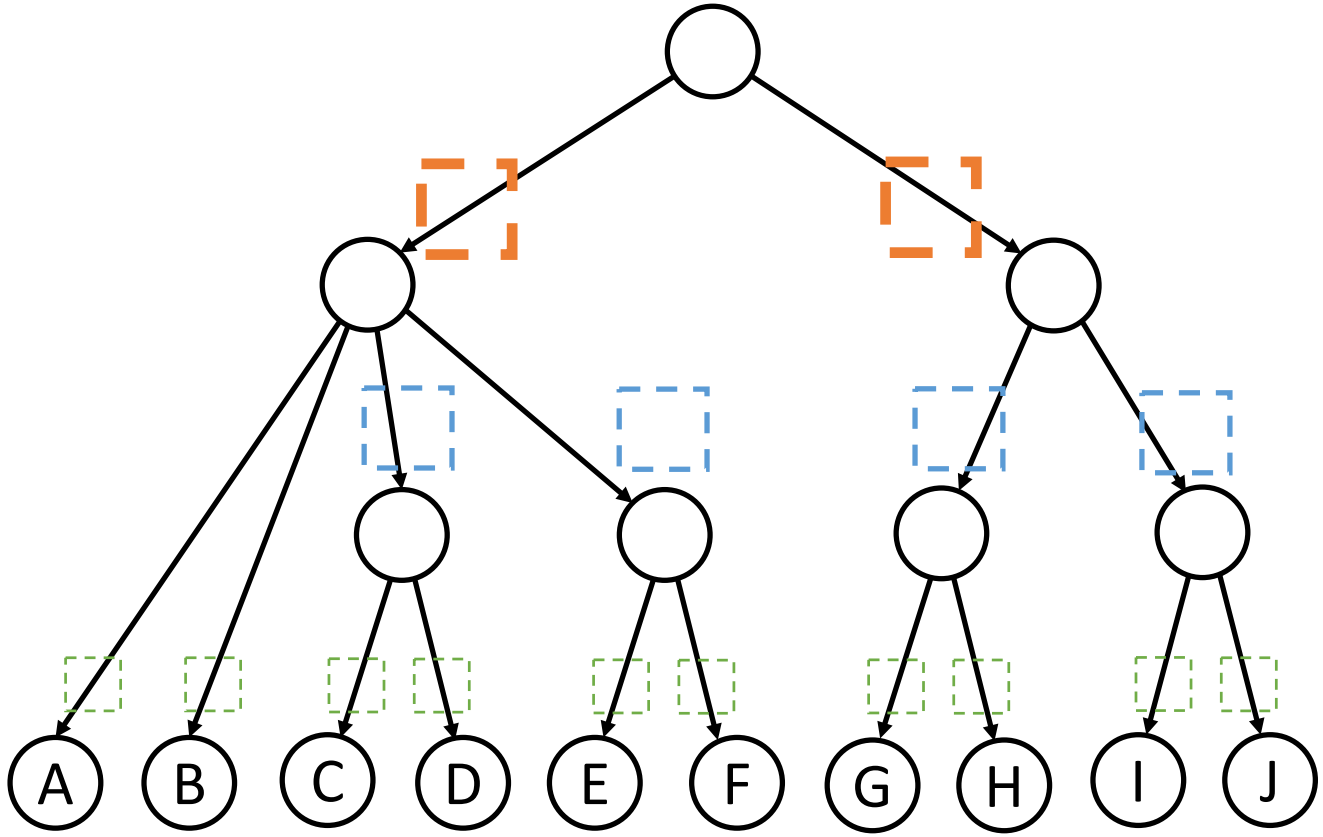




Bounding Volume Hierarchy

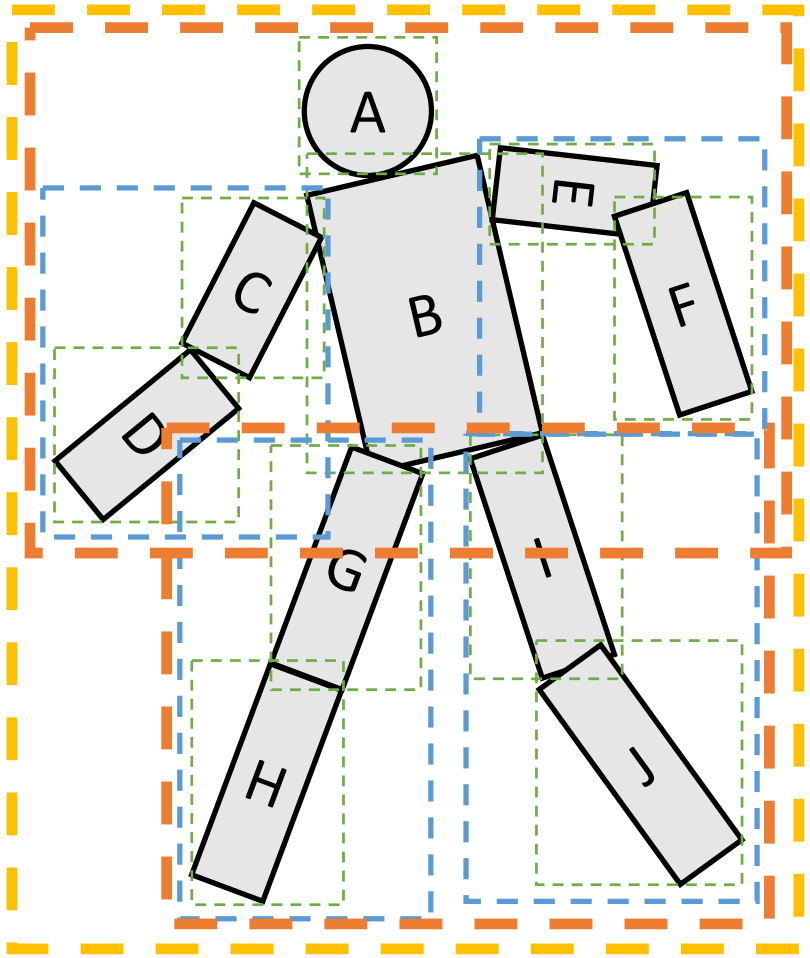
Bounding volume hierarchy is built on geometric/topological proximity of objects.

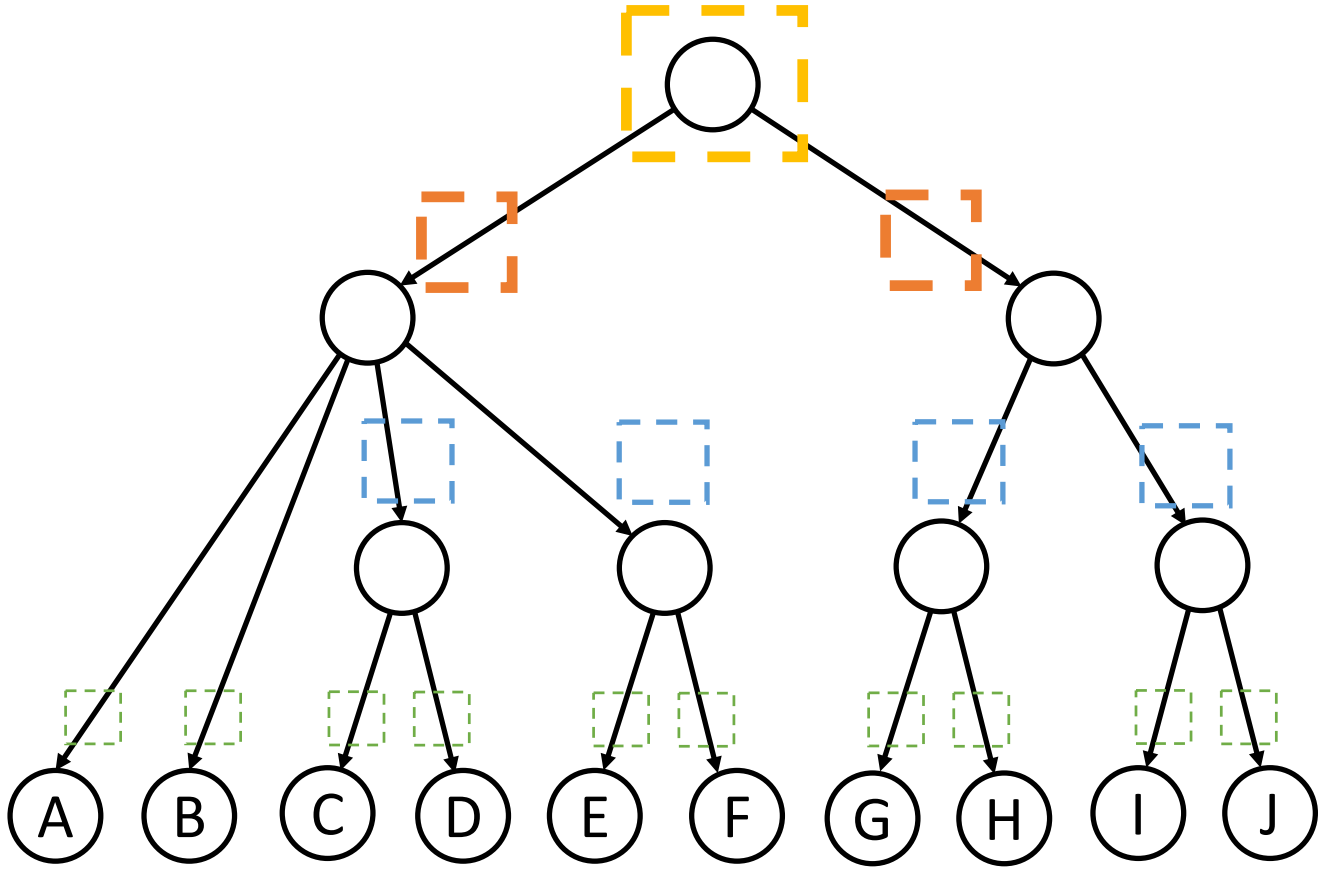




Bounding Volume Hierarchy

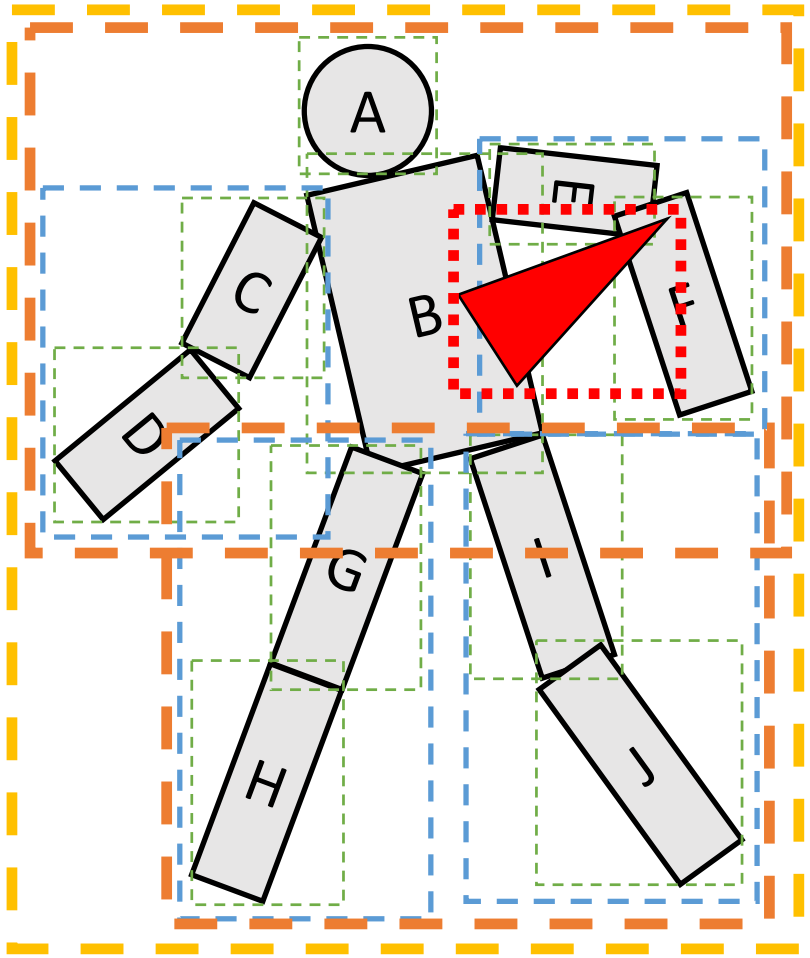
Bounding volume hierarchy is built on geometric/topological proximity of objects.

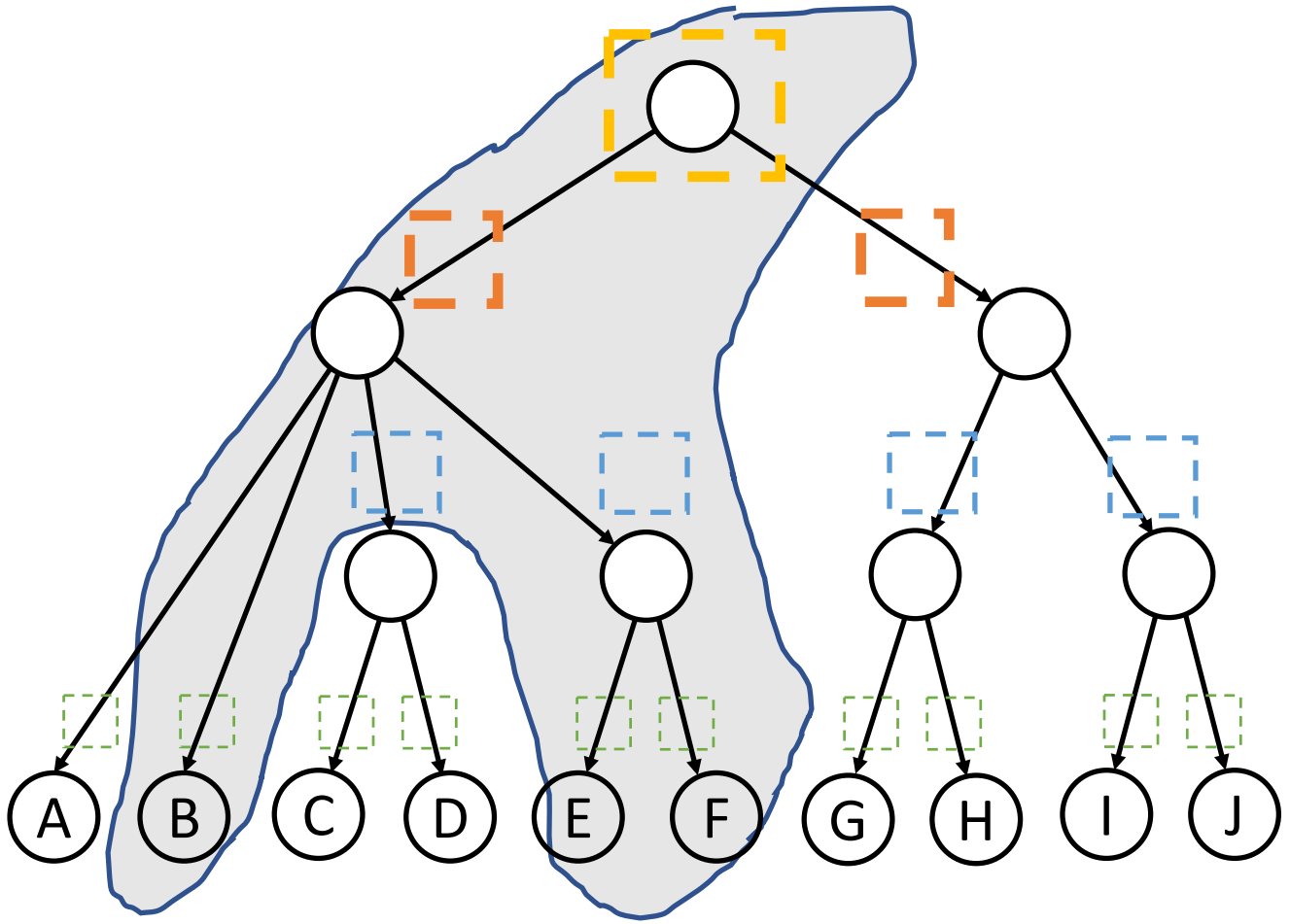




Bounding Volume Hierarchy

To find elements potentially in collision with an object, we just traverse the tree.

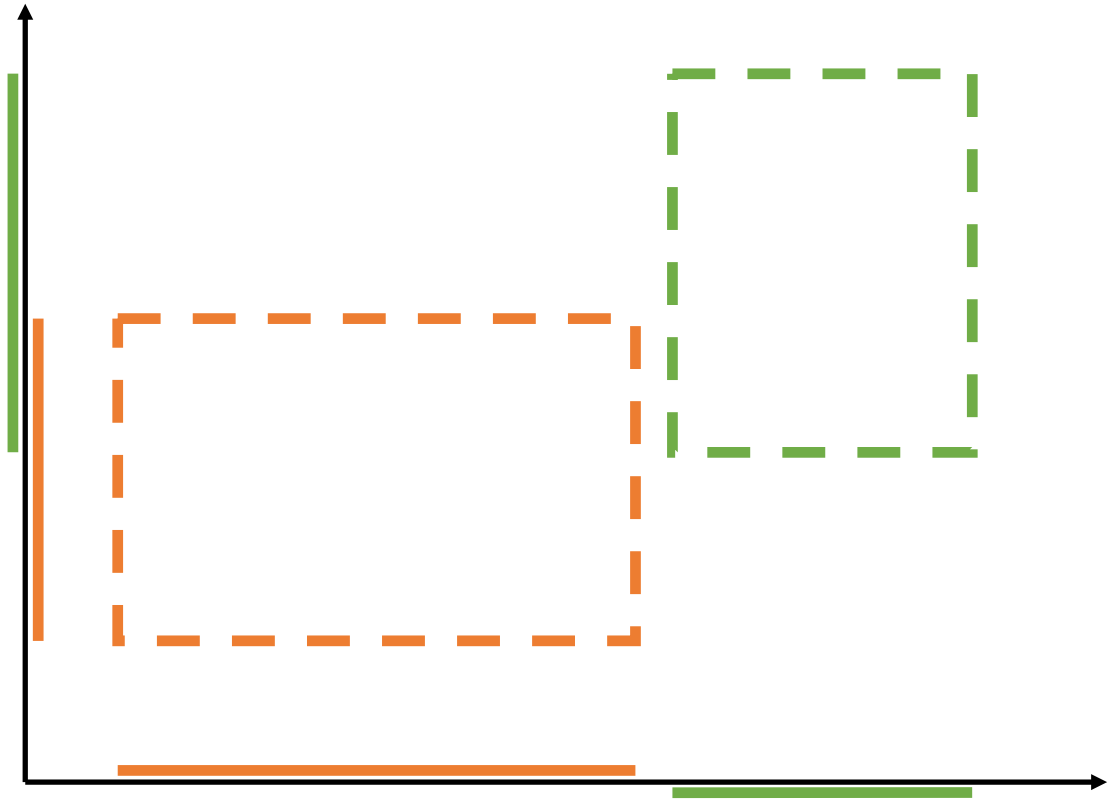


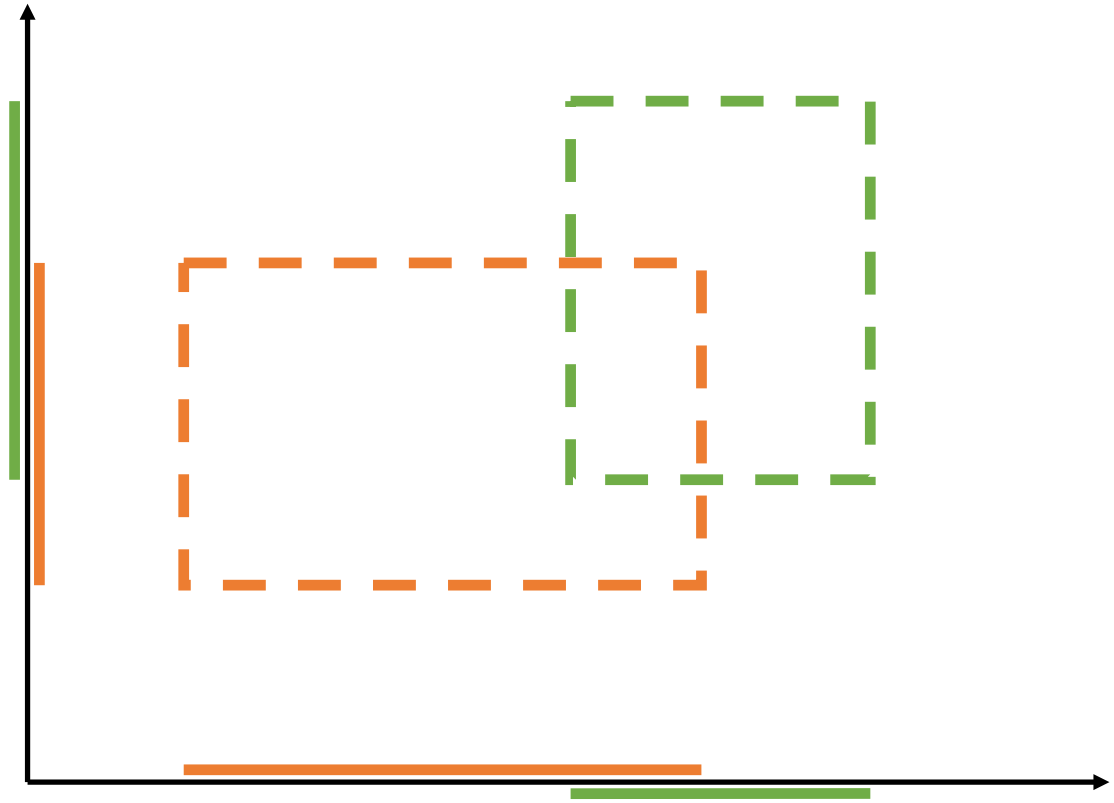


Bounding Volume Hierarchy

Axis-aligned bounding box (AABB) is the most popular bounding volume. Besides that, there are also spheres and oriented bounding box (OBB).

Two AABBs intersect if and only if they intersect in every axis.



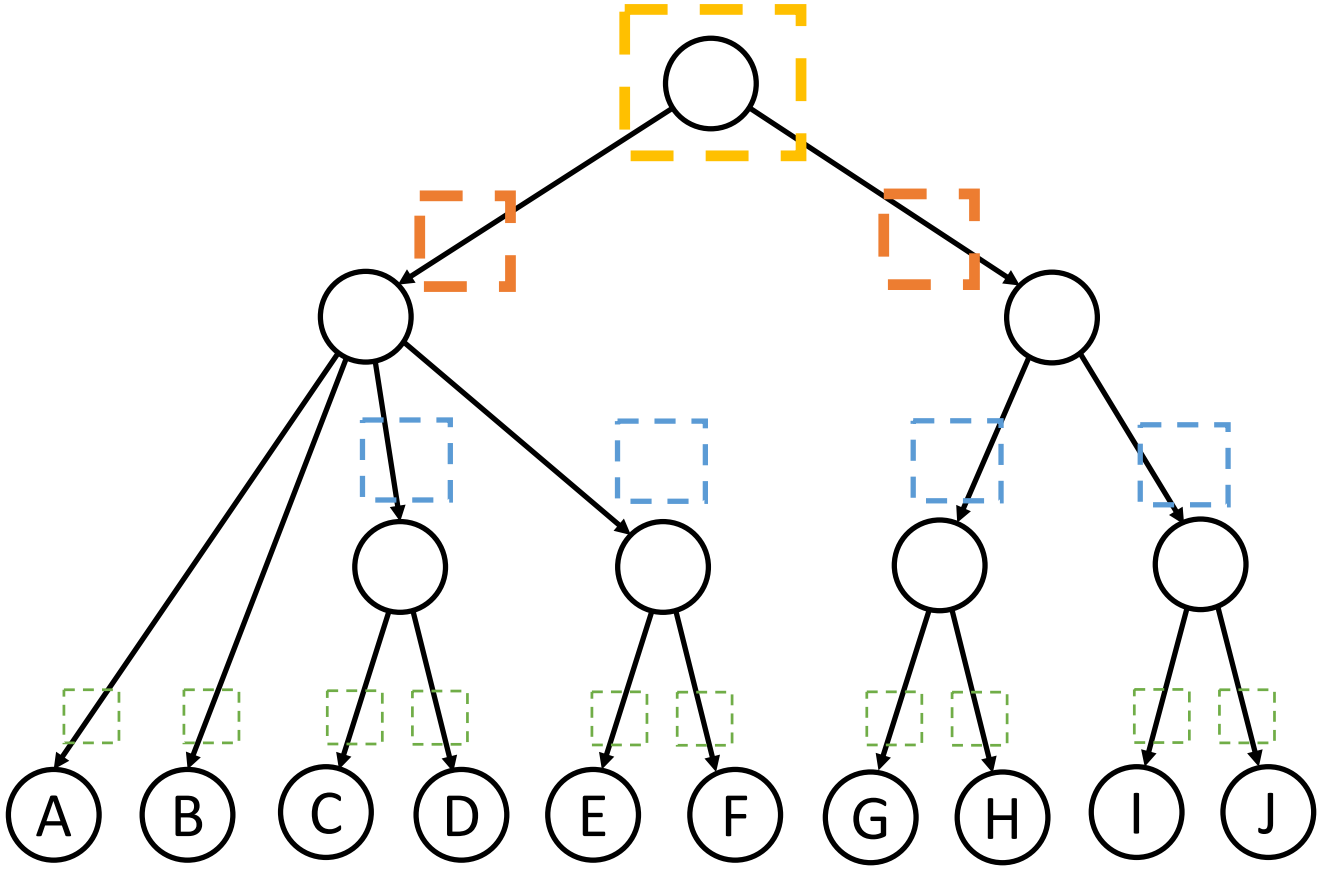


Bounding Volume Hierarchy

To process self collisions by BVH, we define two procedures.

```
Process_Node(A)
{
        For every A's child: B
                Process_Node(B)
        For every A's children pair <B, C>
                if B and C intersect
                        Process_Pair(B, C)
}
```

```
Process_Pair(B, C)
{
        For every B's child: B'
        For every C's child: C'
                if B' and C' intersect
                        Process_Pair(B', C')
}
```



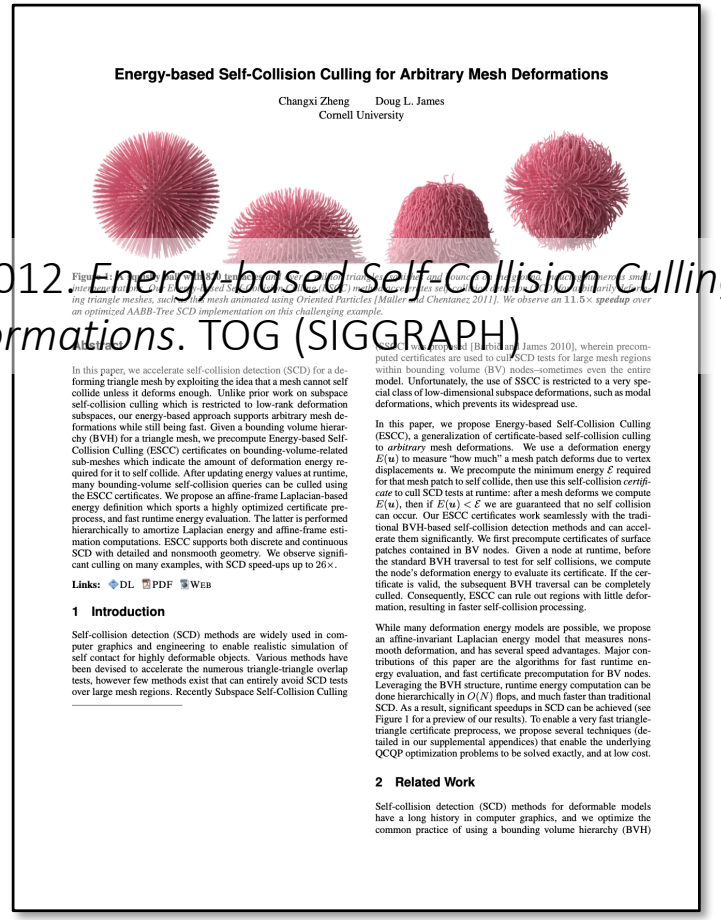
Bounding Volume Hierarchy

The performance depends on the effectiveness of culling.

```
Process_Node(A)
{
    For every A's child: B
        Process_Node(B)
    For every A's children pair <B, C>
        if B and C intersect
            Process_Pair(B, C)
}
```

```
Process_Pair(B C)
{
    For every B's child: B'
    For every C's child: C'
        if B' and C' intersect
            Process_Pair(B', C')
}
```

Zheng and James. 2012. *Energy-based Self-Collision Culling for Arbitrary Mesh Deformations*. TOG (SIGGRAPH)



Energy-based Self-Collision Culling for Arbitrary Mesh Deformations

Changxi Zheng Doug L. James
Cornell University

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 [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 $E(\mathbf{u})$ to measure "how much" a mesh patch deforms due to vertex displacements $\mathbf{u}$. 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 $E(\mathbf{u})$, then if $E(\mathbf{u}) < \mathcal{E}$ 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.

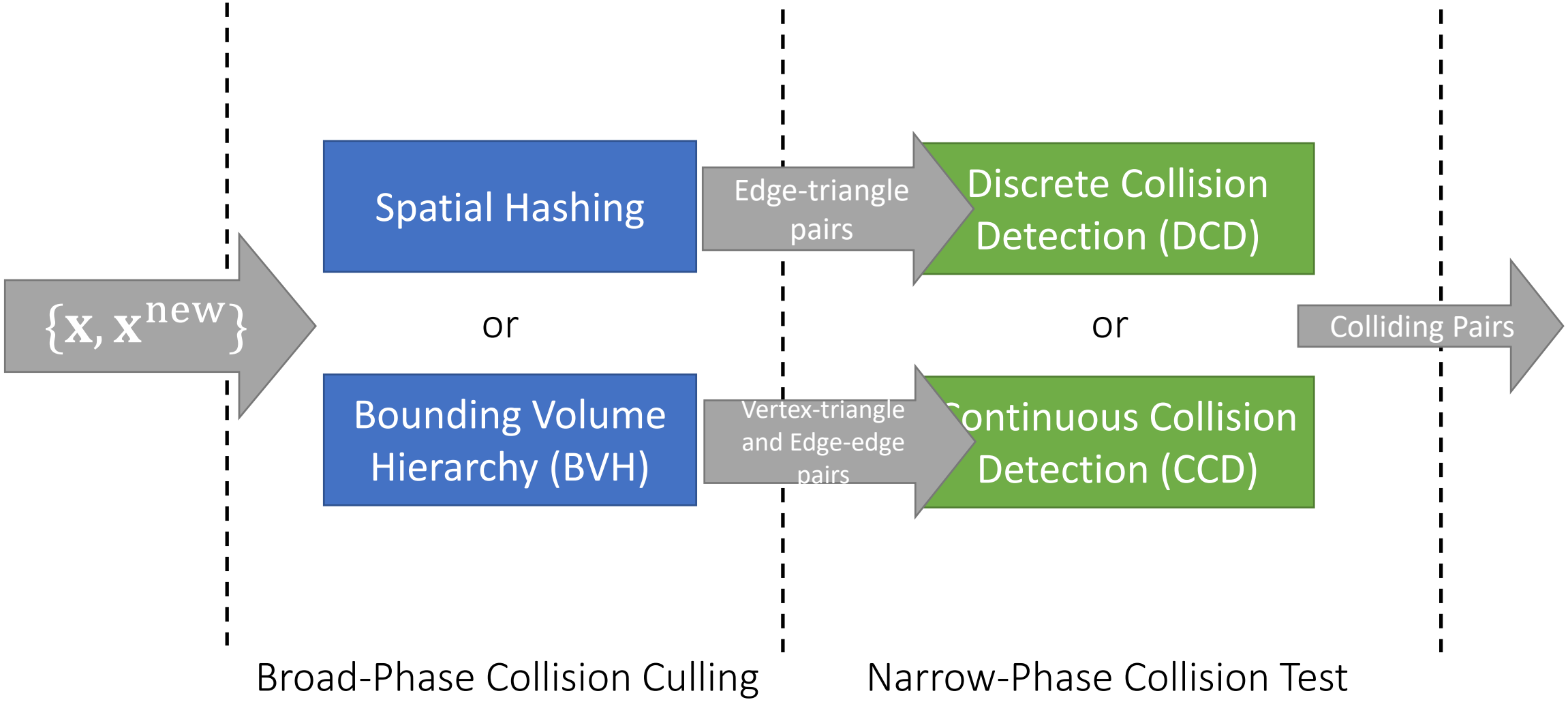
2 Related Work

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)

Comparison between SH and BVH

- Spatial Hashing
  - Easy to implement
  - GPU friendly
  - Needs to recompute after updating objects
- Bounding Volume Hierarchy
  - More involved
  - Not GPU friendly
  - To update BVH, just update bounding volumes

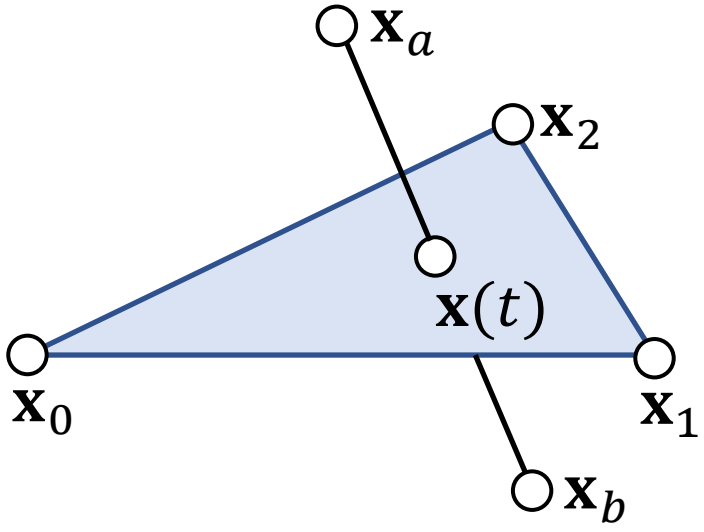
Collision Detection Pipeline



Discrete Collision Detection (DCD)

DCD tests if any intersection exists in each state at discrete time instant: $\mathbf{x}^{[0]}$, $\mathbf{x}^{[1]}$, …

To a triangle mesh, the basic test is edge-triangle intersection test.



$$\left((1-t)\mathbf{x}_a + t\mathbf{x}_b - \mathbf{x}_0\right) \cdot \mathbf{x}_{10} \times \mathbf{x}_{20} = 0$$

$$t = \frac{\mathbf{x}_{0a} \cdot \mathbf{x}_{10} \times \mathbf{x}_{20}}{\mathbf{x}_{ba} \cdot \mathbf{x}_{10} \times \mathbf{x}_{20}}$$

Edge-plane intersection

If $t \in [0, 1]$

$$\mathbf{x}(t) = (1-t)\mathbf{x}_a + t\mathbf{x}_b$$

Test if $\mathbf{x}(t)$ is inside (Lecture 2, page 18)

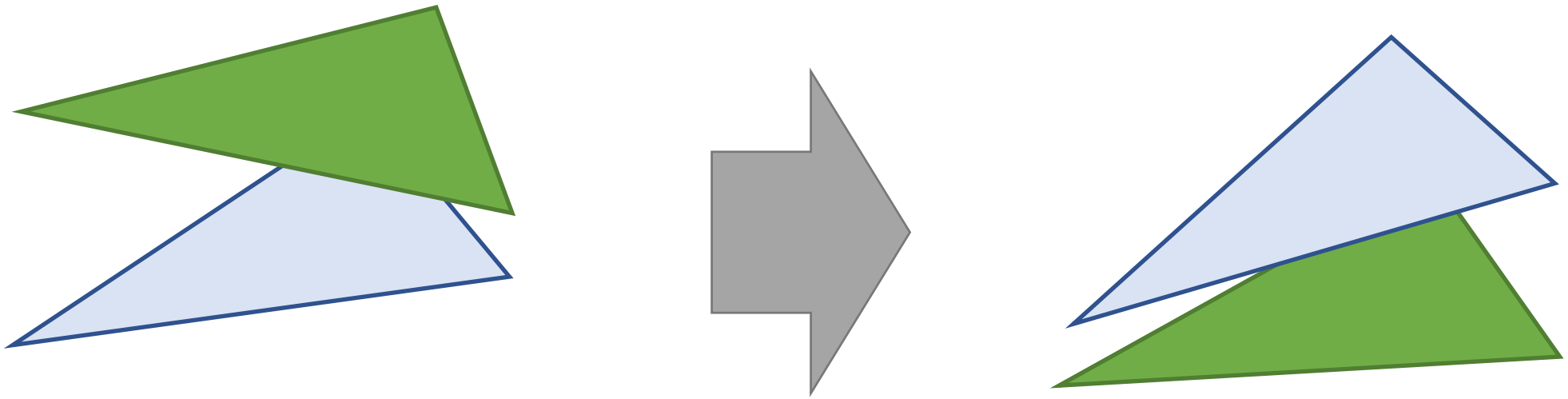
Inside?

If Yes

Intersection

Tunneling

DCD is simple and robust, but it suffers from the tunneling problem: objects penetrating through each other without being detected.



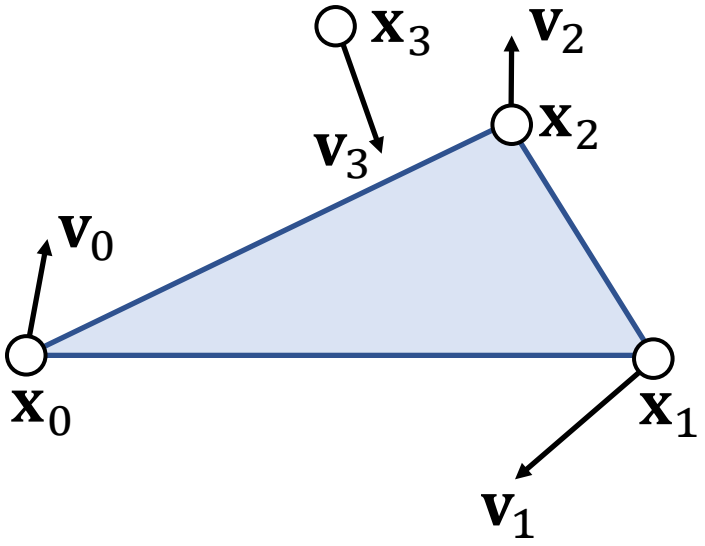
State $\mathbf{x}^{[0]}$

State $\mathbf{x}^{[1]}$

Continuous Collision Detection (CCD)

CCD tests if any intersection exists between two states: $\mathbf{x}^{[0]}$ and $\mathbf{x}^{[1]}$.

To a triangle mesh, there two basic tests: vertex-triangle and edge-edge tests.



$$\mathbf{x}_{30}(t) \cdot \mathbf{x}_{10}(t) \times \mathbf{x}_{20}(t) = 0$$
$$at^3 + bt^2 + ct + d = 0$$

Co-planar test

If $t \in [0, 1]$

Test if $\mathbf{x}_3(t)$ is now inside of triangle made by $\mathbf{x}_0(t)$, $\mathbf{x}_1(t)$ and $\mathbf{x}_2(t)$

Inside?

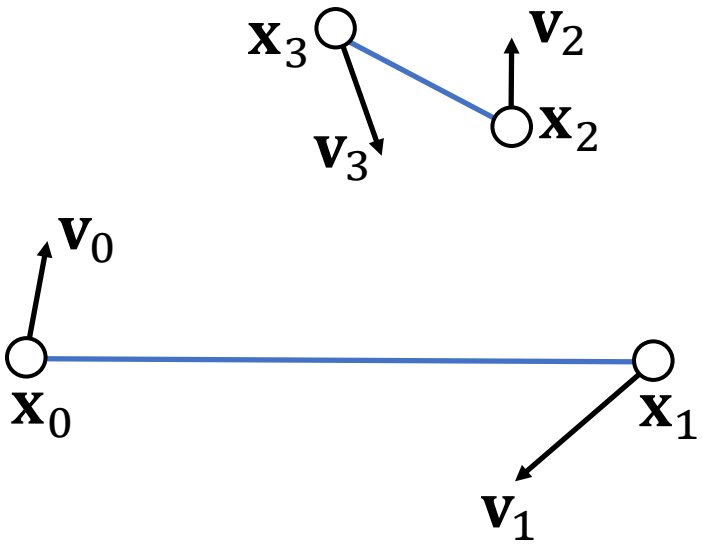
If Yes

Intersection

Continuous Collision Detection (CCD)

CCD tests if any intersection exists between two states: $\mathbf{x}^{[0]}$ and $\mathbf{x}^{[1]}$.

To a triangle mesh, there two basic tests: vertex-triangle and edge-edge tests.



$$\mathbf{x}_{30}(t) \cdot \mathbf{x}_{10}(t) \times \mathbf{x}_{20}(t) = 0$$
$$at^3 + bt^2 + ct + d = 0$$

Co-planar test

If $t \in [0, 1]$

Test if the two edges $\mathbf{x}_2(t)\mathbf{x}_3(t)$ and $\mathbf{x}_0(t)\mathbf{x}_1(t)$ intersect inside

Inside?

If Yes

Intersection

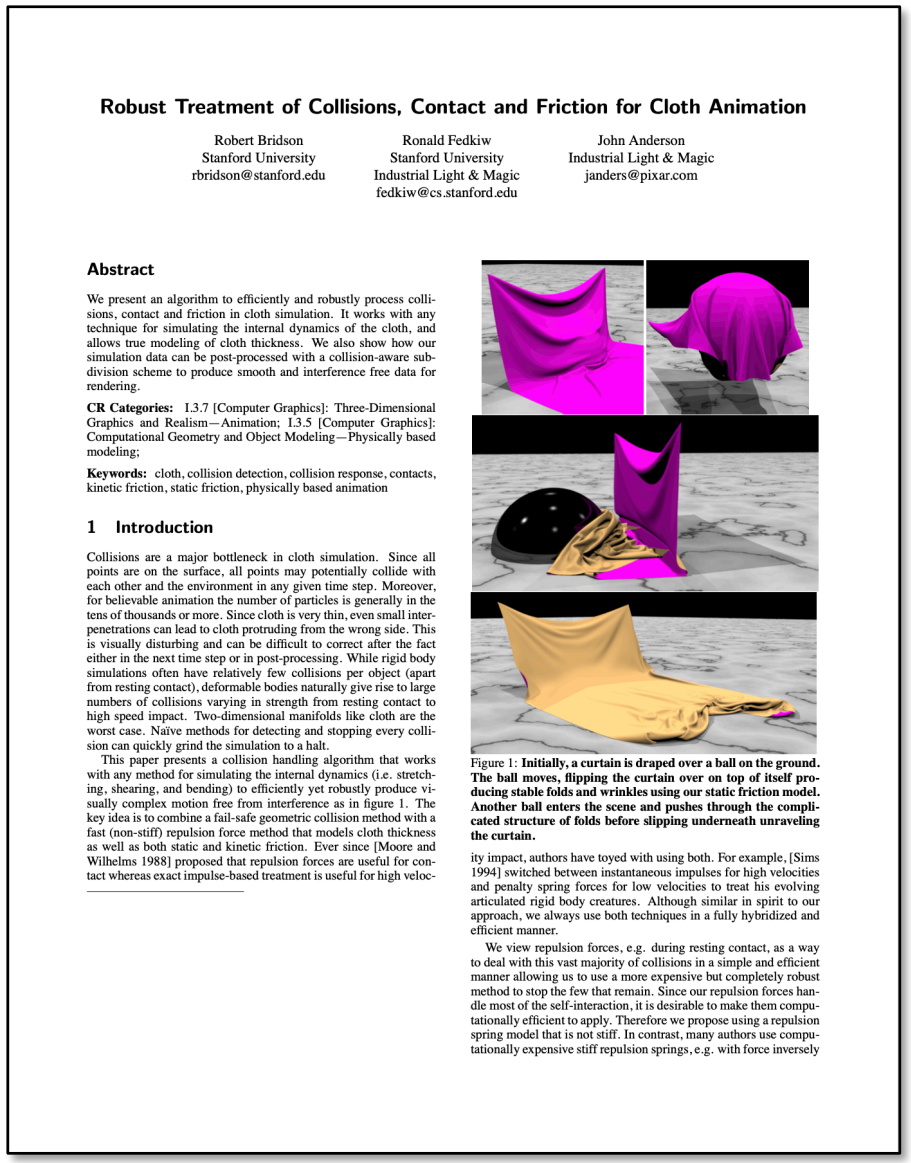
Issues with CCD

- Floating-point errors, especially due to root finding of a cubic equation
  - Buffering epsilons, but that causes false positives.
  - Gaming GPUs often use single floating-point precision.
- Computational costs: more expensive than DCD.
  - Some argue that broad-phase collision culling is the bottleneck.
- Difficulty in implementation.

After-Class Reading

Bridson et al. 2002. *Robust Treatment of Collisions, Contact and Friction for Cloth Animation. TOG (SIGGRAPH)*.

Relative simple explicit integration of cloth dynamics



Robust Treatment of Collisions, Contact and Friction for Cloth Animation

Robert Bridson
Stanford University
rbridson@stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

John Anderson
Industrial Light & Magic
janders@pixar.com

Abstract

We present an algorithm to efficiently and robustly process collisions, contact and friction in cloth simulation. It works with any technique for simulating the internal dynamics of the cloth, and allows true modeling of cloth thickness. We also show how our simulation data can be post-processed with a collision-aware subdivision scheme to produce smooth and interference free data for rendering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: cloth, collision detection, collision response, contacts, kinetic friction, static friction, physically based animation

1 Introduction

Collisions are a major bottleneck in cloth simulation. Since all points are on the surface, all points may potentially collide with each other and the environment in any given time step. Moreover, for believable animation the number of particles is generally in the tens of thousands or more. Since cloth is very thin, even small interpenetrations can lead to cloth protruding from the wrong side. This is visually disturbing and can be difficult to correct after the fact either in the next time step or in post-processing. While rigid body simulations often have relatively few collisions per object (apart from resting contact), deformable bodies naturally give rise to large numbers of collisions varying in strength from resting contact to high speed impact. Two-dimensional manifolds like cloth are the worst case. Naïve methods for detecting and stopping every collision can quickly grind the simulation to a halt.

This paper presents a collision handling algorithm that works with any method for simulating the internal dynamics (i.e. stretching, shearing, and bending) to efficiently yet robustly produce visually complex motion free from interference as in figure 1. The key idea is to combine a fail-safe geometric collision method with a fast (non-stiff) repulsion force method that models cloth thickness as well as both static and kinetic friction. Ever since [Moore and Wilhelms 1988] proposed that repulsion forces are useful for contact whereas exact impulse-based treatment is useful for high velocity impact, authors have toyed with using both. For example, [Sims 1994] switched between instantaneous impulses for high velocities and penalty spring forces for low velocities to treat his evolving articulated rigid body creatures. Although similar in spirit to our approach, we always use both techniques in a fully hybridized and efficient manner.

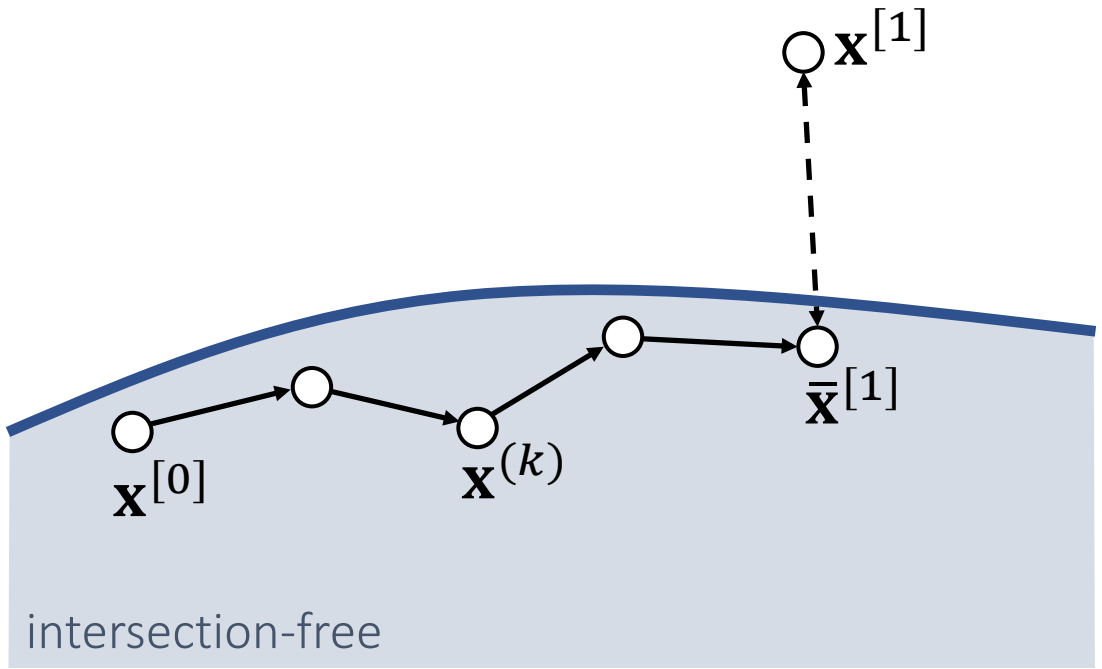
Figure 1: **Initially, a curtain is draped over a ball on the ground. The ball moves, flipping the curtain over on top of itself producing stable folds and wrinkles using our static friction model. Another ball enters the scene and pushes through the complicated structure of folds before slipping underneath unraveling the curtain.**

We view repulsion forces, e.g. during resting contact, as a way to deal with this vast majority of collisions in a simple and efficient manner allowing us to use a more expensive but completely robust method to stop the few that remain. Since our repulsion forces handle most of the self-interaction, it is desirable to make them computationally efficient to apply. Therefore we propose using a repulsion spring model that is not stiff. In contrast, many authors use computationally expensive stiff repulsion springs, e.g. with force inversely

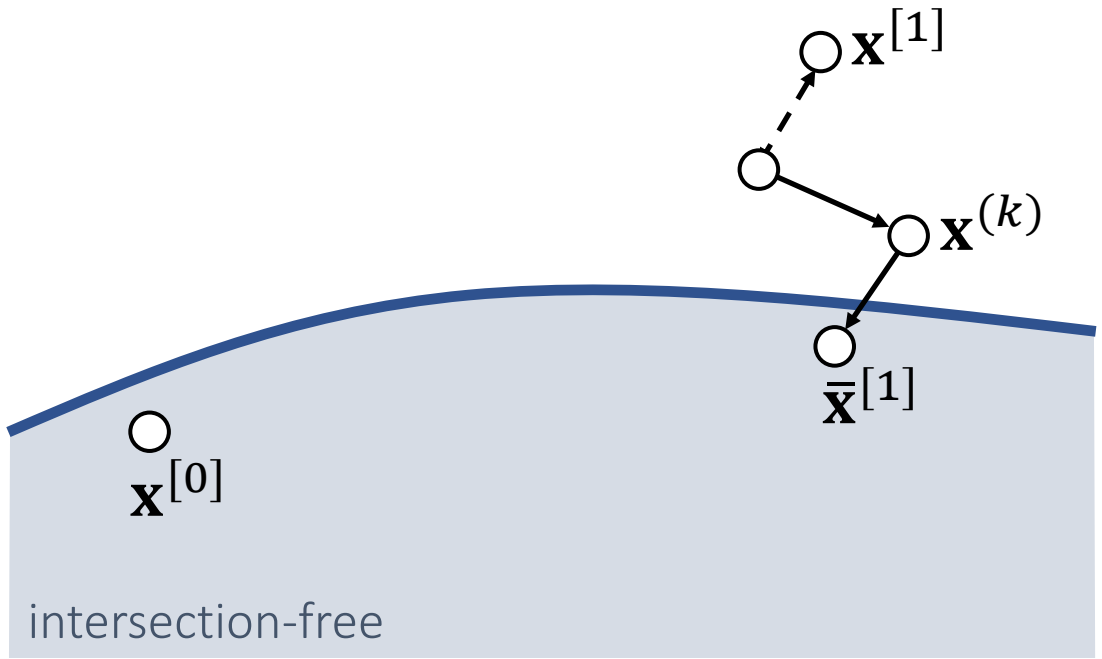
Interior Point Methods
and
Impact Zone Optimization

Two Continuous Collision Response Approaches

Given the calculated next state $\mathbf{x}^{[1]}$, we want to update it into $\bar{\mathbf{x}}^{[1]}$, such that the path from $\mathbf{x}^{[0]}$ to $\bar{\mathbf{x}}^{[1]}$ is intersection-free.



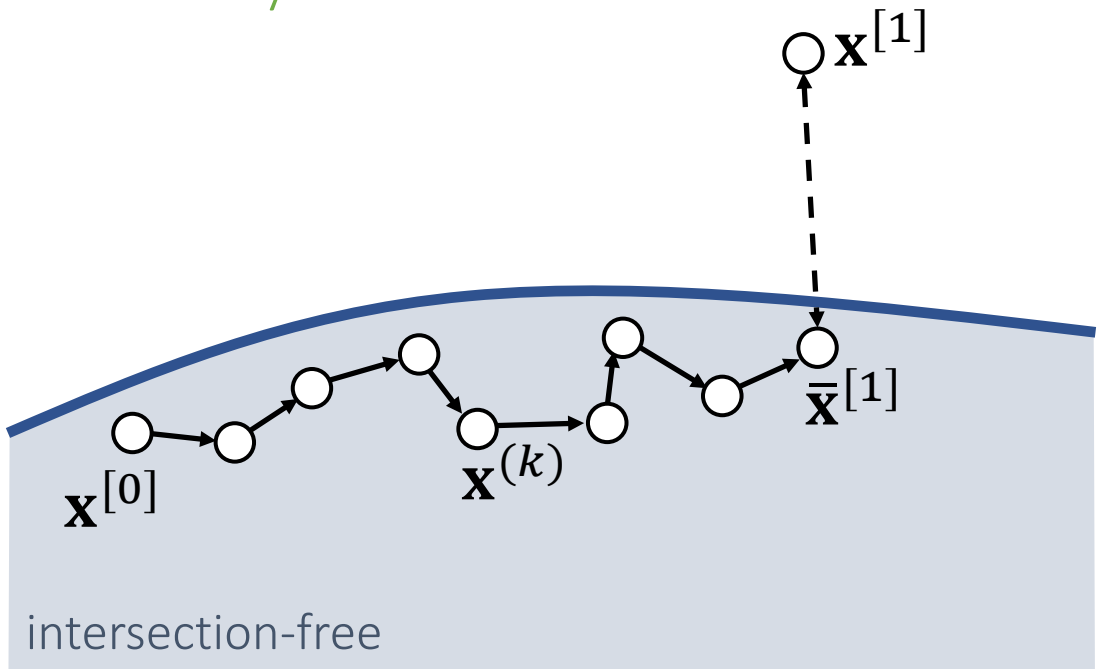
Interior Point Methods



Impact Zone Optimization

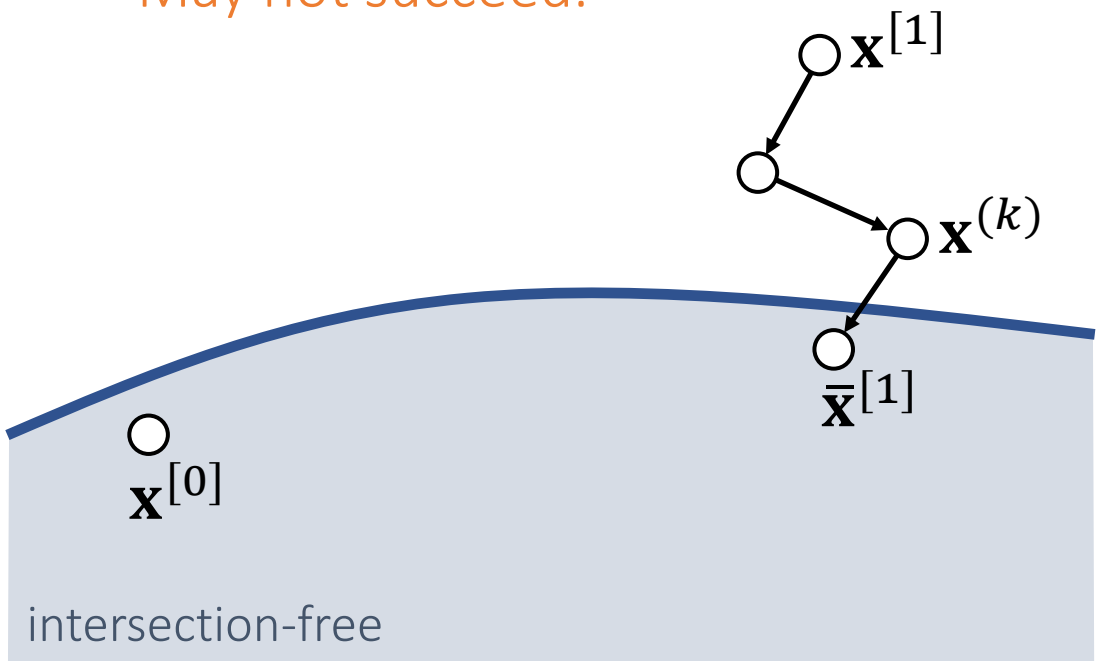
Pros and Cons

- Slow.
  - Far from solution.
  - All of the vertices.
  - Cautiously by small step sizes.
- Always succeed.



Interior Point Methods

- Fast.
  - Close to solution.
  - Only vertices in collision (impact zones).
  - Can take large step sizes.
- May not succeed.



Impact Zone Optimization

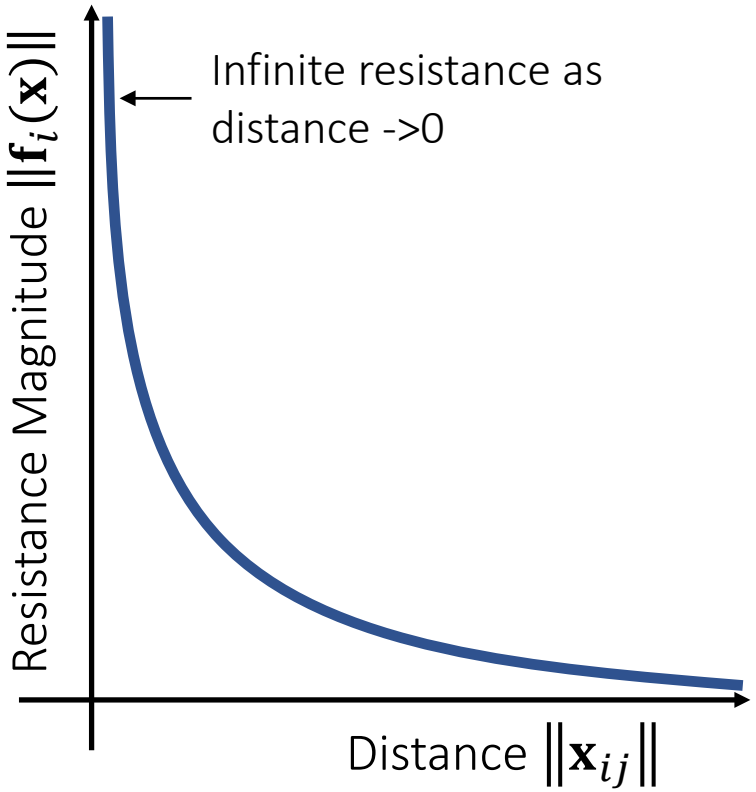
Log-Barrier Interior Point Methods

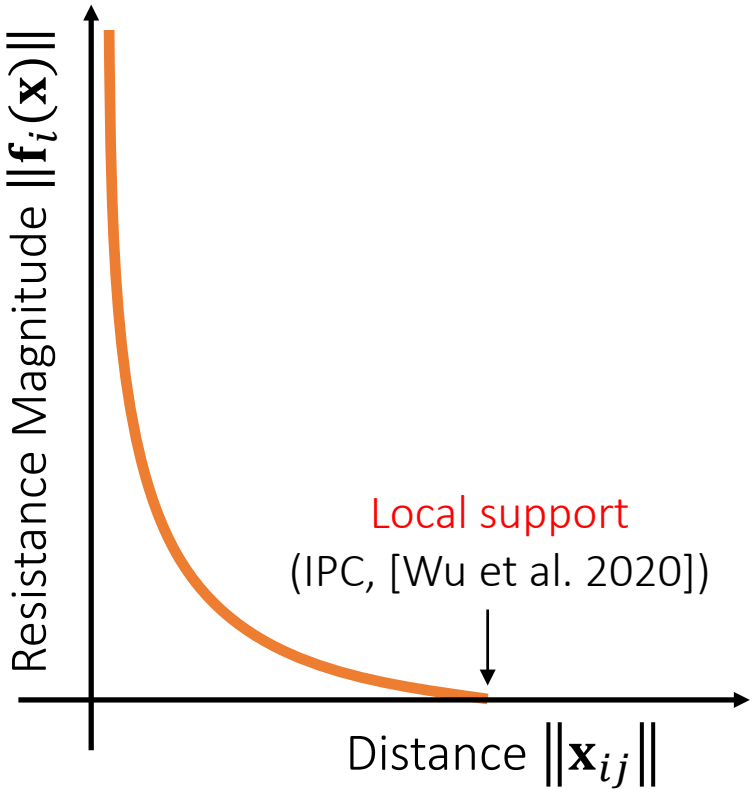
For simplicity, let's consider the Log-barrier repulsion between two vertices.

$$E(\mathbf{x}) = -\rho \log \|\mathbf{x}_{ij}\|$$

$$\mathbf{f}_i(\mathbf{x}) = -\nabla_i E = \rho \frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|^2}$$

$$\mathbf{f}_j(\mathbf{x}) = -\nabla_j E = -\rho \frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|^2}$$





Interior Point Methods – Implementation

We can then formulate the problem as:

$$\bar{\mathbf{x}}^{[1]} \leftarrow \operatorname{argmin}_{\mathbf{x}} \left(\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{[1]} \right\|^2 - \rho \sum \log \left\| \mathbf{x}_{ij} \right\| \right)$$

Gradient Descent:

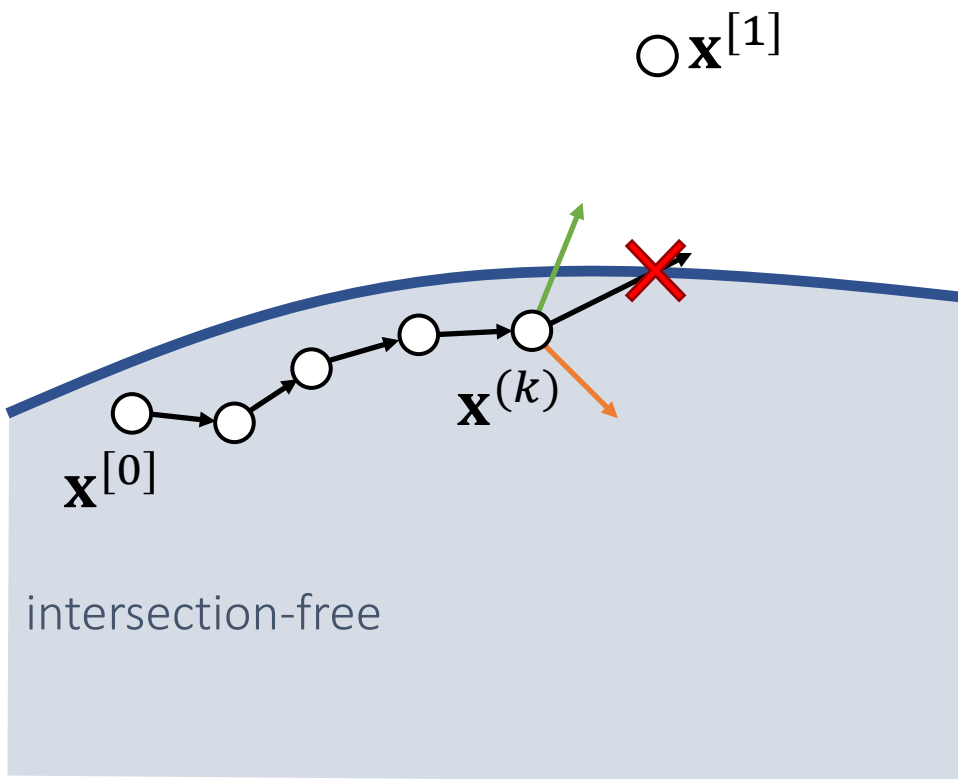
$$\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{[0]}$$

For $k = 0 \ldots K$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha \left(\mathbf{x}^{[1]} - \mathbf{x}^{(k)} + \rho \sum \frac{\mathbf{x}_{ij}}{\left\| \mathbf{x}_{ij} \right\|^2} \right)$$

$$\bar{\mathbf{x}}^{[1]} \leftarrow \mathbf{x}^{(k+1)}$$

The step size α must be adjusted to ensure that no collision happens on the way. To find α, we need collision tests.



Impact Zone Optimization

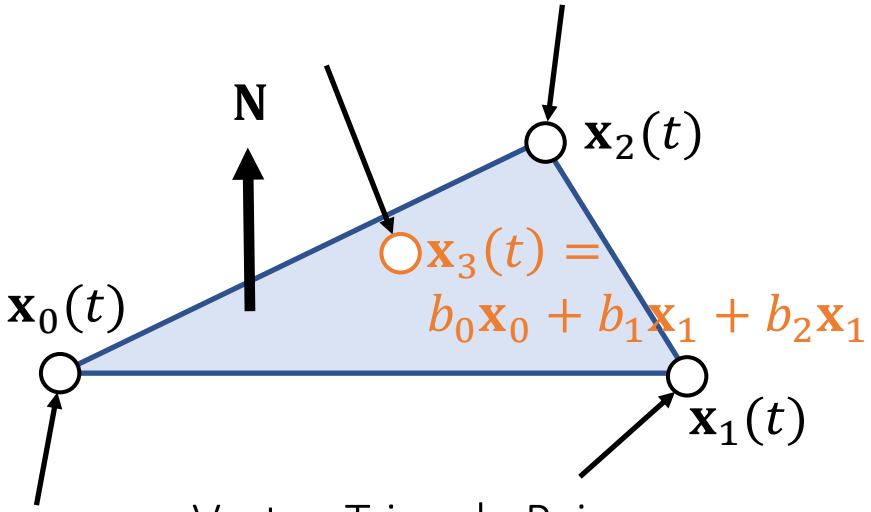
The goal of impact zone optimization is to optimize $\mathbf{x}^{[1]}$ until it becomes intersection-free. (This potentially suffers from the tunneling issue, but it's uncommon.)

$$\bar{\mathbf{x}}^{[1]} \leftarrow \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\left\|\bar{\mathbf{x}} - \mathbf{x}^{[1]}\right\|^2$$

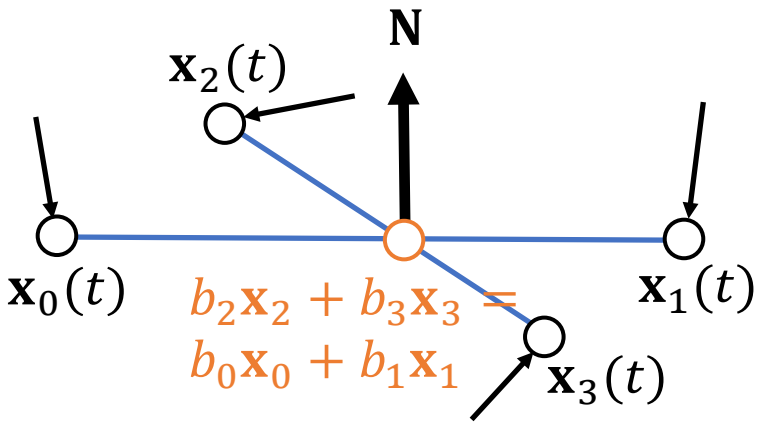
such that

$$C(\mathbf{x}) = -(\mathbf{x}_3 - b_0\mathbf{x}_0 - b_1\mathbf{x}_1 - b_2\mathbf{x}_1) \cdot \mathbf{N} \leq 0 \quad \text{For each detected vertex-triangle pair}$$

$$C(\mathbf{x}) = -(b_2\mathbf{x}_2 + b_3\mathbf{x}_3 - b_0\mathbf{x}_0 - b_1\mathbf{x}_1) \cdot \mathbf{N} \leq 0 \quad \text{For each detected edge-edge pair}$$



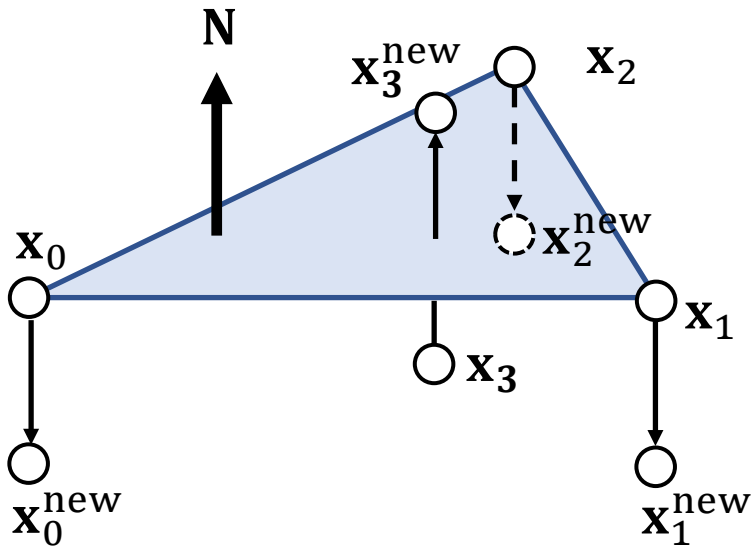
Vertex-Triangle Pair



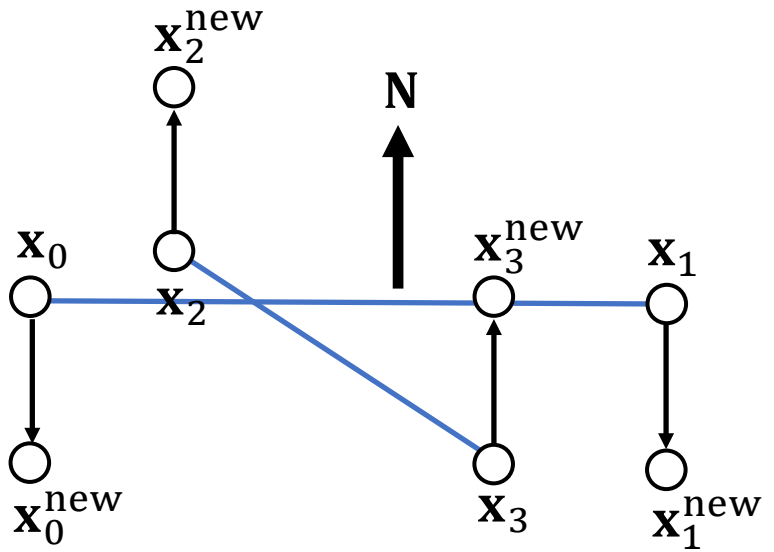
Edge-Edge Pair

Geometric Impulse

The goal of impact zone optimization is to optimize $\mathbf{x}^{[1]}$ until it becomes intersection-free. (This potentially suffers from the tunneling issue, but it's uncommon.)



Vertex-Triangle Pair



Edge-Edge Pair

Every pair gives new positions to the involved vertices. We can combine them together in a Jacobi, or Gauss-Seidel fashion, just like position-based dynamics.

After-Class Reading (Cont.)

Bridson et al. 2002. *Robust Treatment of Collisions, Contact and Friction for Cloth Animation. TOG (SIGGRAPH)*.

Relative simple explicit integration of cloth dynamics

Robust Treatment of Collisions, Contact and Friction for Cloth Animation

Robert Bridson
Stanford University
rbridson@stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

John Anderson
Industrial Light & Magic
janders@pixar.com

Abstract

We present an algorithm to efficiently and robustly process collisions, contact and friction in cloth simulation. It works with any technique for simulating the internal dynamics of the cloth, and allows true modeling of cloth thickness. We also show how our simulation data can be post-processed with a collision-aware subdivision scheme to produce smooth and interference free data for rendering.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling;

Keywords: cloth, collision detection, collision response, contacts, kinetic friction, static friction, physically based animation

1 Introduction

Collisions are a major bottleneck in cloth simulation. Since all points are on the surface, all points may potentially collide with each other and the environment in any given time step. Moreover, for believable animation the number of particles is generally in the tens of thousands or more. Since cloth is very thin, even small interpenetrations can lead to cloth protruding from the wrong side. This is visually disturbing and can be difficult to correct after the fact either in the next time step or in post-processing. While rigid body simulations often have relatively few collisions per object (apart from resting contact), deformable bodies naturally give rise to large numbers of collisions varying in strength from resting contact to high speed impact. Two-dimensional manifolds like cloth are the worst case. Naïve methods for detecting and stopping every collision can quickly grind the simulation to a halt.

This paper presents a collision handling algorithm that works with any method for simulating the internal dynamics (i.e. stretching, shearing, and bending) to efficiently yet robustly produce visually complex motion free from interference as in figure 1. The key idea is to combine a fail-safe geometric collision method with a fast (non-stiff) repulsion force method that models cloth thickness as well as both static and kinetic friction. Ever since [Moore and Wilhelms 1988] proposed that repulsion forces are useful for contact whereas exact impulse-based treatment is useful for high velocity impact, authors have toyed with using both. For example, [Sims 1994] switched between instantaneous impulses for high velocities and penalty spring forces for low velocities to treat his evolving articulated rigid body creatures. Although similar in spirit to our approach, we always use both techniques in a fully hybridized and efficient manner.

Figure 1: **Initially, a curtain is draped over a ball on the ground. The ball moves, flipping the curtain over on top of itself producing stable folds and wrinkles using our static friction model. Another ball enters the scene and pushes through the complicated structure of folds before slipping underneath unraveling the curtain.**

We view repulsion forces, e.g. during resting contact, as a way to deal with this vast majority of collisions in a simple and efficient manner allowing us to use a more expensive but completely robust method to stop the few that remain. Since our repulsion forces handle most of the self-interaction, it is desirable to make them computationally efficient to apply. Therefore we propose using a repulsion spring model that is not stiff. In contrast, many authors use computationally expensive stiff repulsion springs, e.g. with force inversely

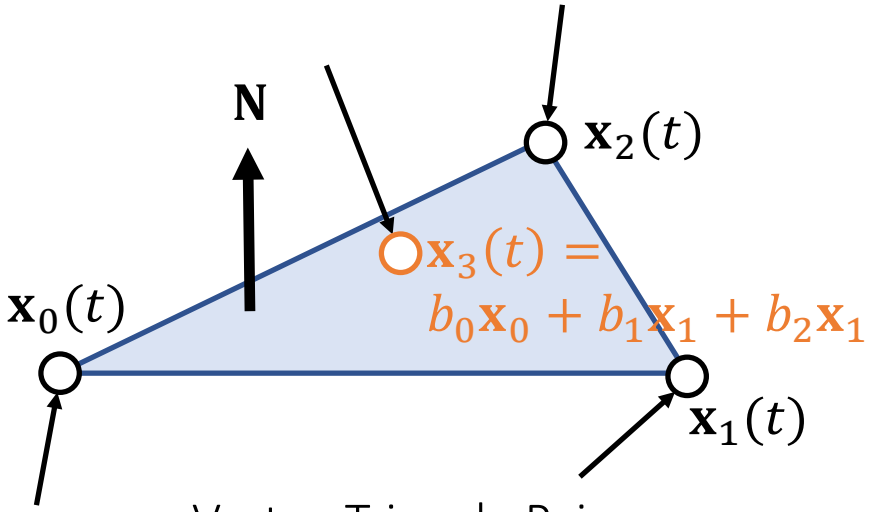
Augmented Lagrangian

We can also formulate this into a constrained optimization problem:

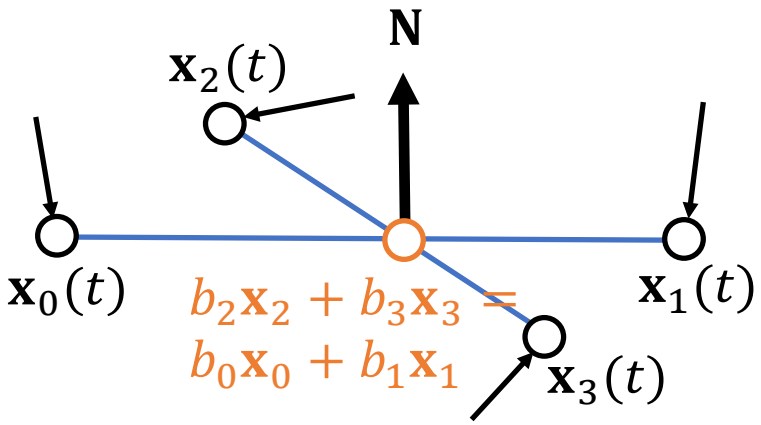
$$\bar{\mathbf{x}}^{[1]} \leftarrow \operatorname{argmin}_{\mathbf{x}} \frac{1}{2}\left\|\mathbf{x}-\mathbf{x}^{[1]}\right\|^2$$

such that

$$\begin{cases} C(\mathbf{x}) = -(\mathbf{x}_3 - b_0\mathbf{x}_0 - b_1\mathbf{x}_1 - b_2\mathbf{x}_1)\cdot\mathbf{N} \le 0 & \text{For each detected vertex-triangle pair} \\ C(\mathbf{x}) = -(b_2\mathbf{x}_2 + b_3\mathbf{x}_3 - b_0\mathbf{x}_0 - b_1\mathbf{x}_1)\cdot\mathbf{N} \le 0 & \text{For each detected edge-edge pair} \end{cases}$$



Vertex-Triangle Pair



Edge-Edge Pair

Augmented Lagrangian

We can then convert it into an unconstrained form:

$$\bar{\mathbf{x}}^{[1]} \leftarrow \operatorname{argmin}_{\mathbf{x},\boldsymbol{\lambda}} \left(\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{[1]} \right\|^2 + \frac{\rho}{2} \left\| \max\left(\tilde{\mathbf{C}}(\mathbf{x}) \right) \right\|^2 - \frac{1}{2\rho} \|\boldsymbol{\lambda}\|^2 \right)$$

$$\tilde{\mathbf{C}}(\mathbf{x}) = \max(\mathbf{C}(\mathbf{x}) + \boldsymbol{\lambda}/\rho)$$

Augmented Lagrangian:

$\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{[0]}$

$\boldsymbol{\lambda} \leftarrow \mathbf{0}$

For $k = 0 \ldots K$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha \nabla \left(\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}^{[1]} \right\|^2 + \frac{\rho}{2} \left\| \max\left(\tilde{\mathbf{C}}(\mathbf{x}) \right) \right\|^2 - \frac{1}{2\rho} \|\boldsymbol{\lambda}\|^2 \right)$$

$\lambda \leftarrow \rho \tilde{\mathbf{C}}(\mathbf{x})$

$\bar{\mathbf{x}}^{[1]} \leftarrow \mathbf{x}^{(k+1)}$

Tang et al. 2018. *I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation*. TOG. (SIGGRAPH Asia)

About Impact Zone Optimization

- Fast per iteration
  - Only have to deal with vertices in collision.
- Convergence sensitive to $\|\mathbf{x}^{[0]} - \mathbf{x}^{[1]}\|^2$, or the time step Δt
  - Can take many iterations to, or never achieve *intersection-free*.
  - Easy solution is to reduce Δt, but that increases total costs.

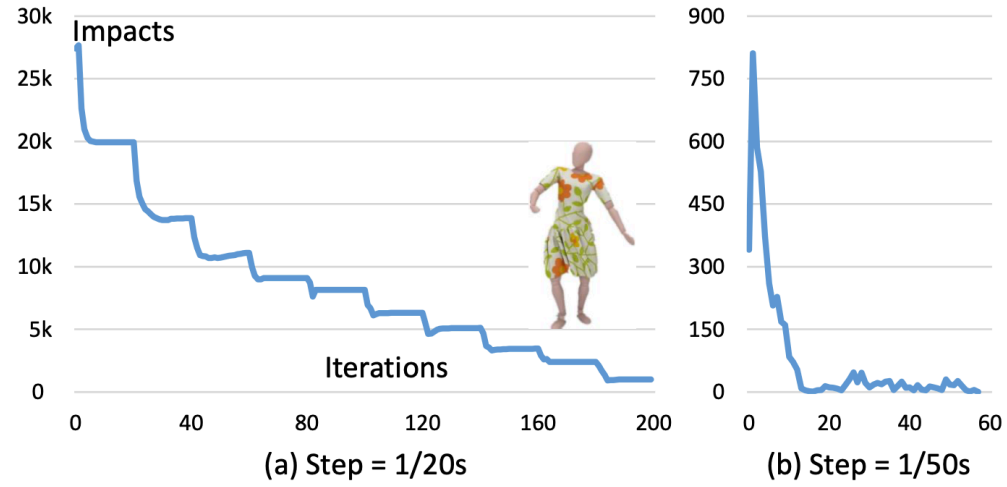
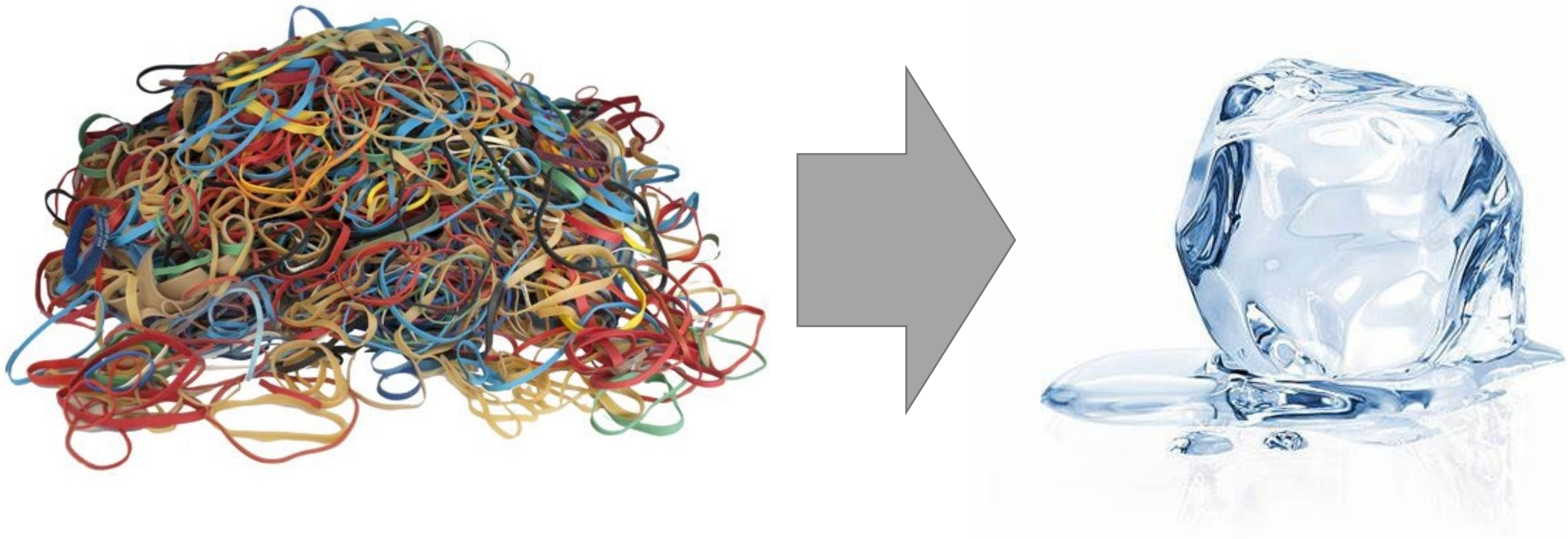


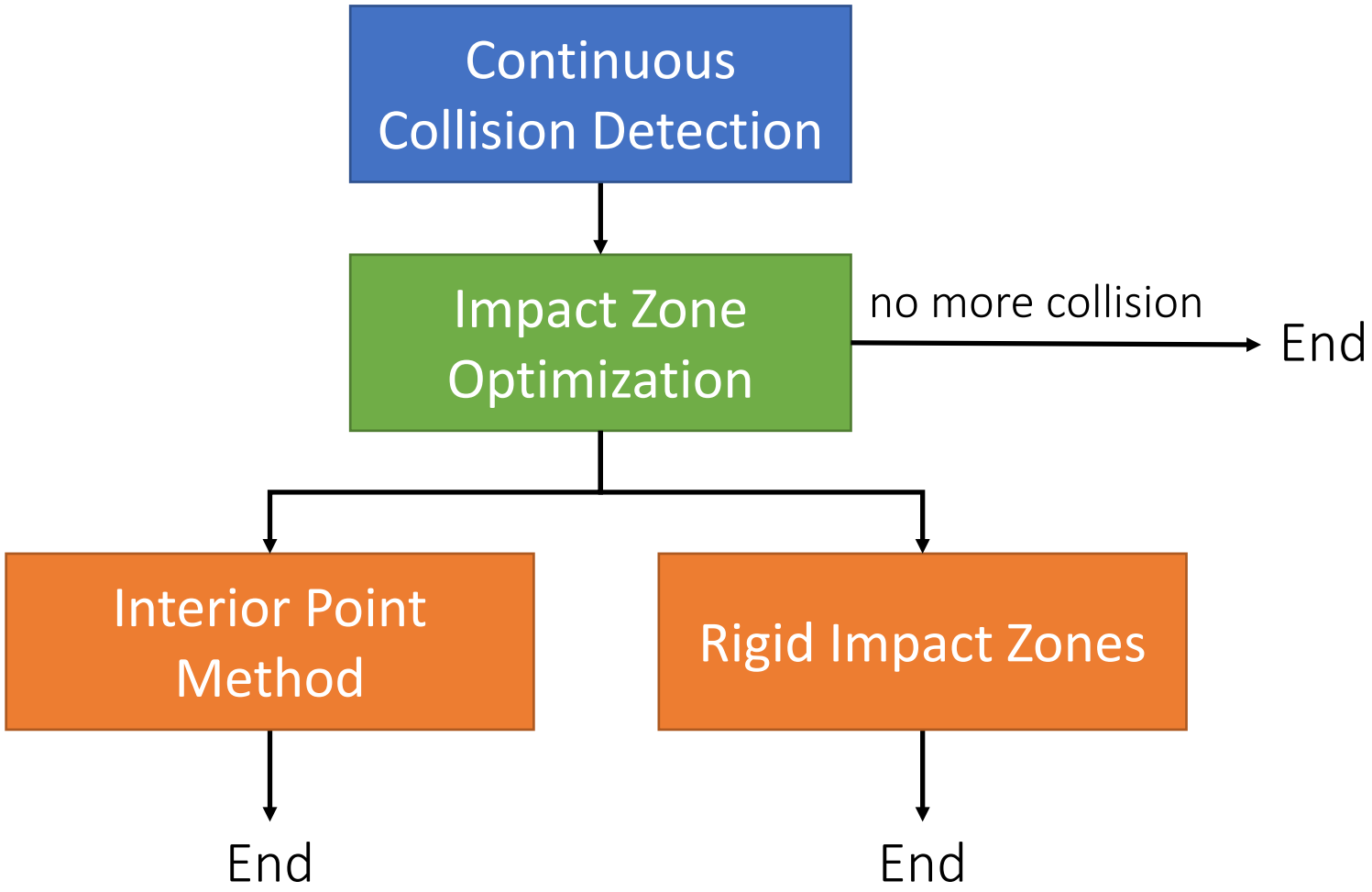
Fig. 7. **Failure Case Analysis:** (a) highlights a failure case where our GPU solver cannot converge to a non-penetration state when simulating the Dress benchmark with a large time step, i.e., 1/20s. Even after 200 iterations, there are still 1002 impacts unresolved. However, with a small time step, i.e., 1/50s, our method can resolve all the impacts within 57 iterations (b).

Rigid Impact Zones

The rigid impact zone method simply freezes vertices in collision from moving in their pre-collision state. It's simple and safe, but has noticeable artifacts.



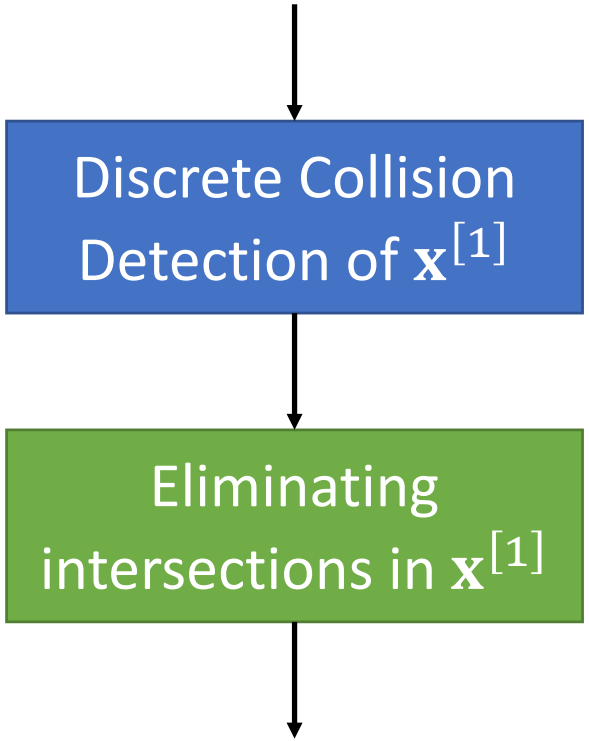
A Practical System



Untangling Cloth

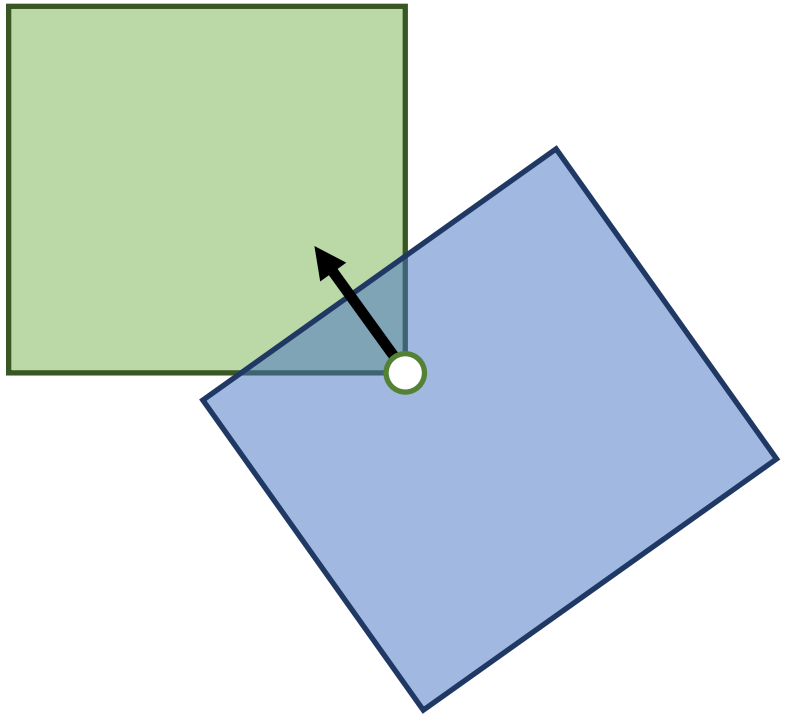
Intersection Elimination

- Let's consider how to eliminate existing intersections, but without using any collision history.
- Such a method is useful when there are already intersections in simulation, due to:
  - Past collision handling failures
  - Intense user interaction
- In this case, we don't require the simulation is to always intersection-free.

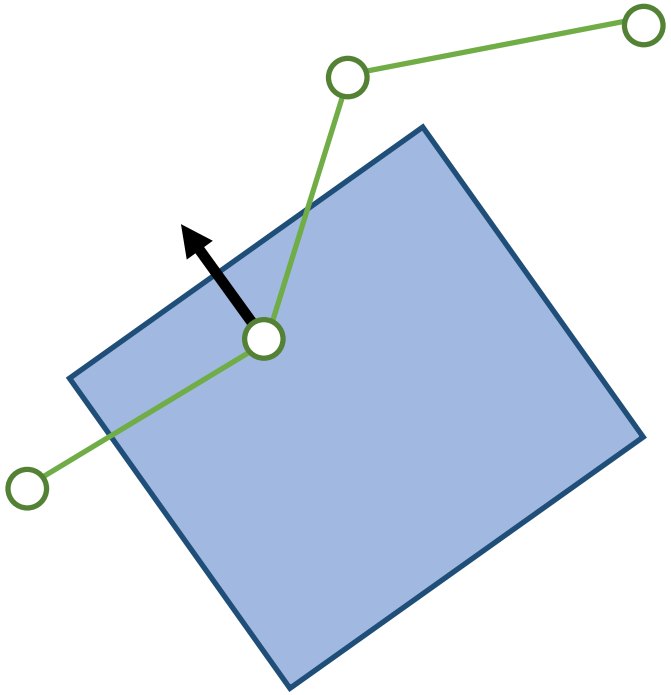


Intersection Elimination

Eliminating cloth-volume and volume-volume intersections is straightforward: simply pushing vertices/edges in the volume out.



Cloth-volume intersection

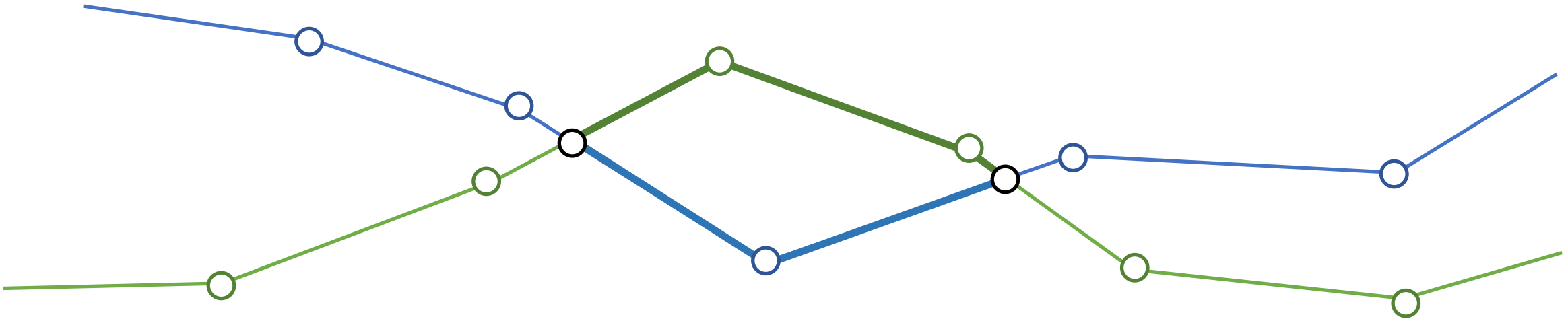


Volume-volume intersection

Untangling Cloth

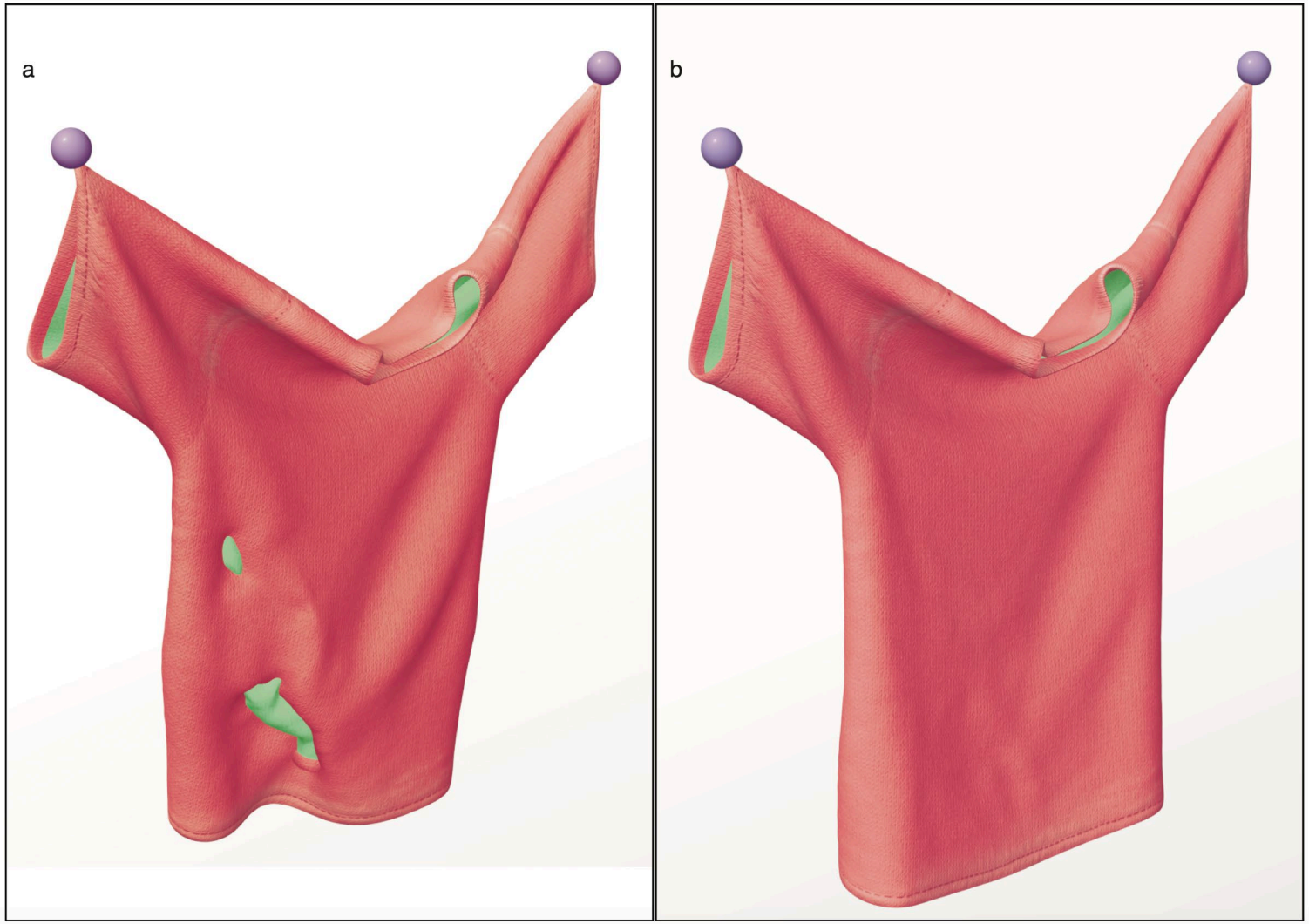
The situation is complicated in cloth-cloth intersection, since we don't have a clear definition of inside and outside.

Baraff et al. used flood-fill to segment cloth into regions and decided which region is in intersection. (Cannot handle boundary well.)



Baraff et al. 2003. Untangling Cloth. TOG (SIGGRAPH)

Untangling Cloth

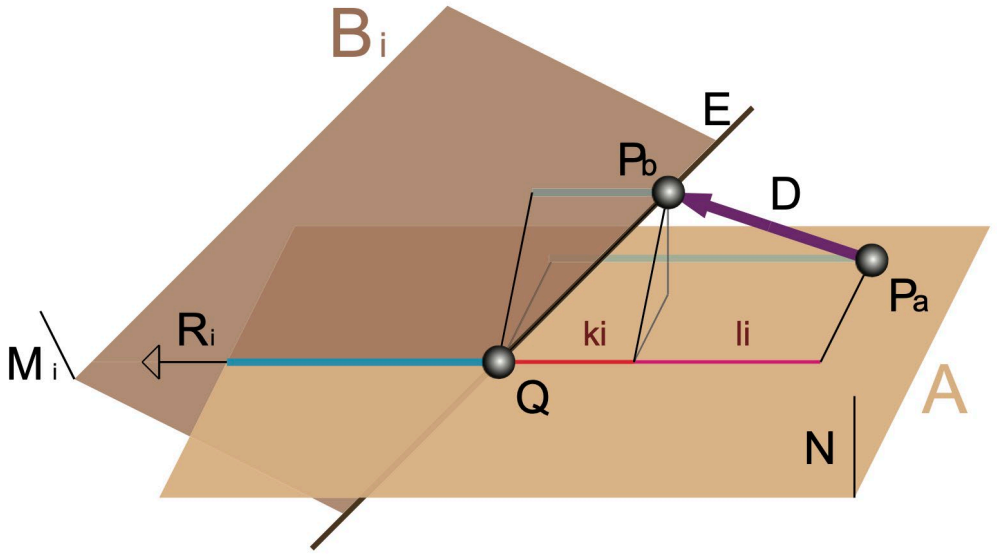


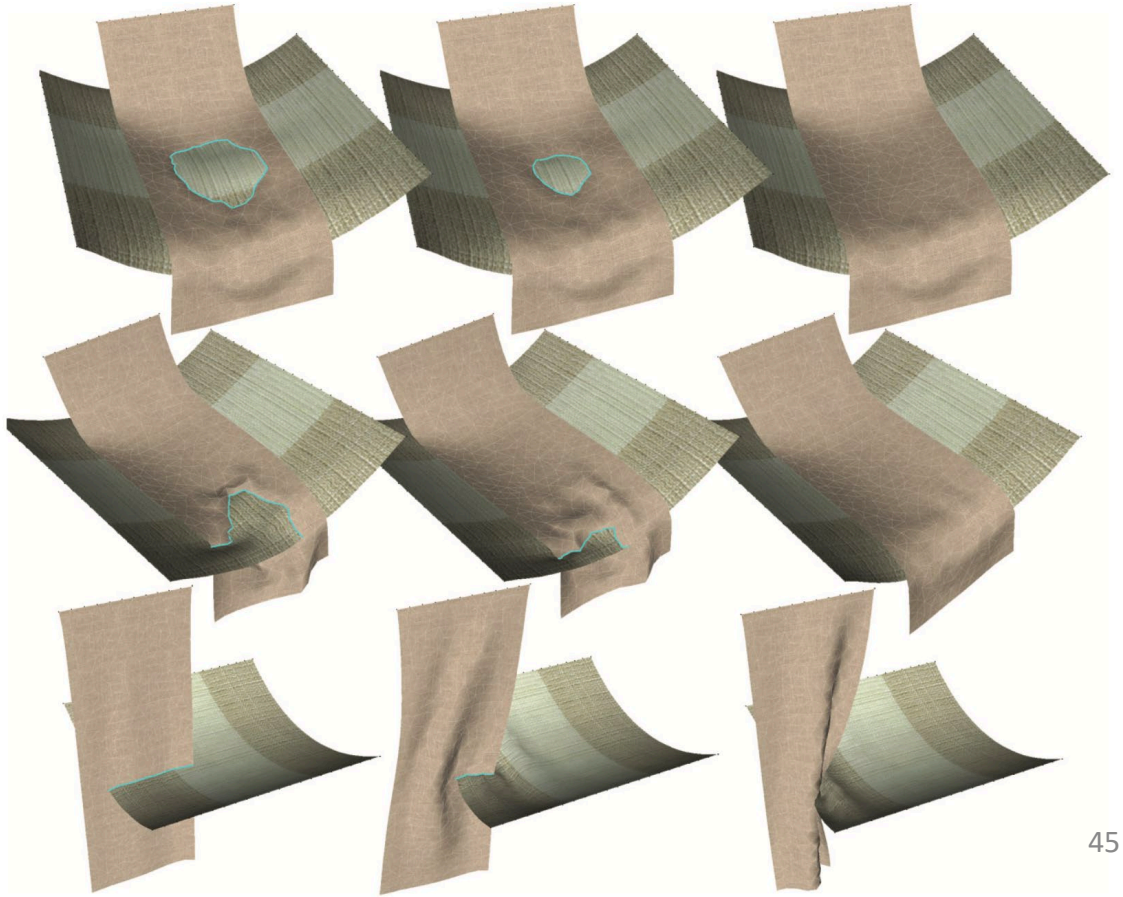
Baraff et al. 2003. Untangling Cloth. TOG (SIGGRAPH)

Untangling Cloth

Volino and Magnenat-Thalmann proposed to untangle cloth by reducing the intersection contour.

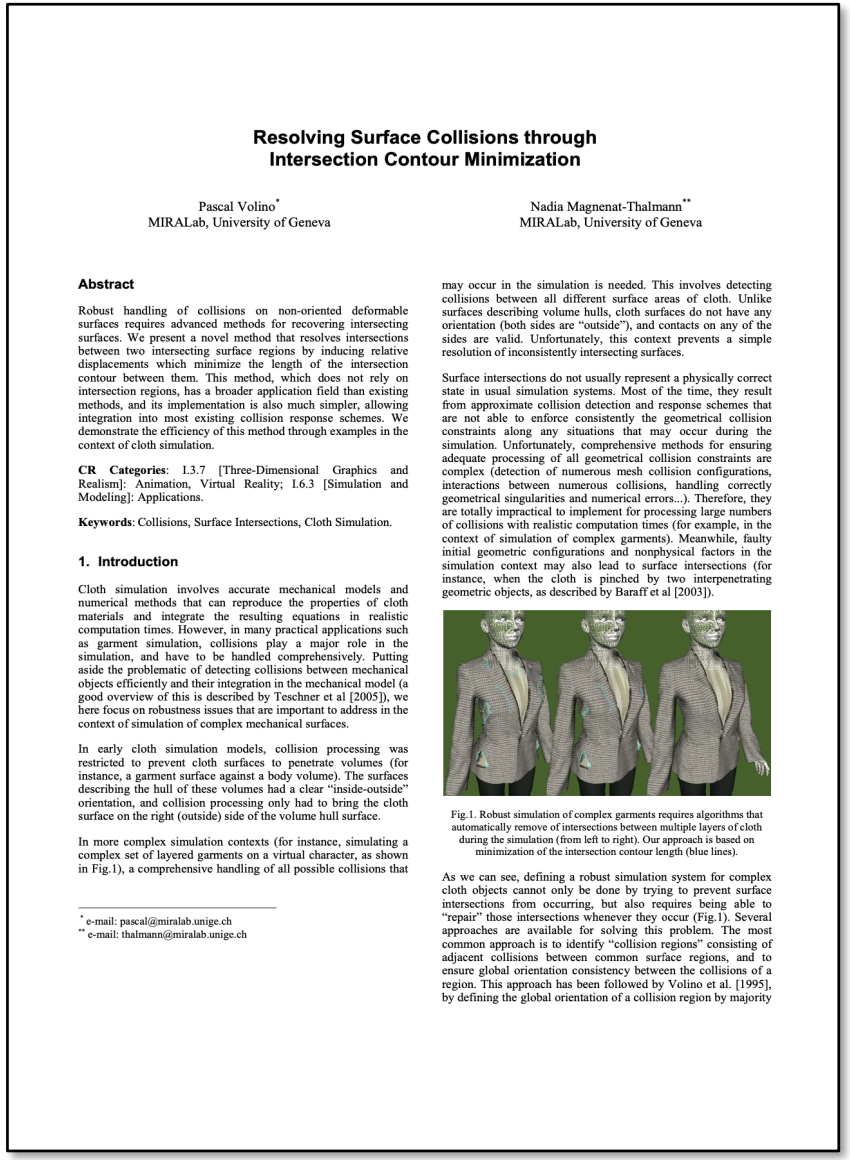
Their method can handle boundaries, but it doesn't always work.





After-Class Reading

Volino and Magnenat-Thalmann et al. 2006. *Resolving Surface Collisions through Intersection Contour Minimization. TOG (SIGGRAPH)*.



Resolving Surface Collisions through Intersection Contour Minimization

Pascal Volino*
MIRALab, University of Geneva

Nadia Magnenat-Thalmann**
MIRALab, University of Geneva

Abstract

Robust handling of collisions on non-oriented deformable surfaces requires advanced methods for recovering intersecting surfaces. We present a novel method that resolves intersections between two intersecting surface regions by inducing relative displacements which minimize the length of the intersection contour between them. This method, which does not rely on intersection regions, has a broader application field than existing methods, and its implementation is also much simpler, allowing integration into most existing collision response schemes. We demonstrate the efficiency of this method through examples in the context of cloth simulation.

CR Categories: I.3.7 [Three-Dimensional Graphics and Realism]: Animation, Virtual Reality; I.6.3 [Simulation and Modeling]: Applications.

Keywords: Collisions, Surface Intersections, Cloth Simulation.

1. Introduction

Cloth simulation involves accurate mechanical models and numerical methods that can reproduce the properties of cloth materials and integrate the resulting equations in realistic computation times. However, in many practical applications such as garment simulation, collisions play a major role in the simulation, and have to be handled comprehensively. Putting aside the problematic of detecting collisions between mechanical objects efficiently and their integration in the mechanical model (a good overview of this is described by Teschner et al [2005]), we here focus on robustness issues that are important to address in the context of simulation of complex mechanical surfaces.

In early cloth simulation models, collision processing was restricted to prevent cloth surfaces to penetrate volumes (for instance, a garment surface against a body volume). The surfaces describing the hull of these volumes had a clear "inside-outside" orientation, and collision processing only had to bring the cloth surface on the right (outside) side of the volume hull surface.

In more complex simulation contexts (for instance, simulating a complex set of layered garments on a virtual character, as shown in Fig.1), a comprehensive handling of all possible collisions that may occur in the simulation is needed. This involves detecting collisions between all different surface areas of cloth. Unlike surfaces describing volume hulls, cloth surfaces do not have any orientation (both sides are "outside"), and contacts on any of the sides are valid. Unfortunately, this context prevents a simple resolution of inconsistently intersecting surfaces.

Surface intersections do not usually represent a physically correct state in usual simulation systems. Most of the time, they result from approximate collision detection and response schemes that are not able to enforce consistently the geometrical collision constraints along any situations that may occur during the simulation. Unfortunately, comprehensive methods for ensuring adequate processing of all geometrical collision constraints are complex (detection of numerous mesh collision configurations, interactions between numerous collisions, handling correctly geometrical singularities and numerical errors...). Therefore, they are totally impractical to implement for processing large numbers of collisions with realistic computation times (for example, in the context of simulation of complex garments). Meanwhile, faulty initial geometric configurations and nonphysical factors in the simulation context may also lead to surface intersections (for instance, when the cloth is pinched by two interpenetrating geometric objects, as described by Baraff et al [2003]).

Fig.1. Robust simulation of complex garments requires algorithms that automatically remove of intersections between multiple layers of cloth during the simulation (from left to right). Our approach is based on minimization of the intersection contour length (blue lines).

As we can see, defining a robust simulation system for complex cloth objects cannot only be done by trying to prevent surface intersections from occurring, but also requires being able to "repair" those intersections whenever they occur (Fig.1). Several approaches are available for solving this problem. The most common approach is to identify "collision regions" consisting of adjacent collisions between common surface regions, and to ensure global orientation consistency between the collisions of a region. This approach has been followed by Volino et al. [1995], by defining the global orientation of a collision region by majority

* e-mail: pascal@miralab.unige.ch
** e-mail: thalmann@miralab.unige.ch

A Summary For the Day

- Collision handling involves two steps: *collision detection* and *collision response*.
- Collision detection contains two phases: *broad-phase culling* and *narrow-phase test*.
- There are two types of collision detection tests: *discrete* and *continuous*.
- Similarly, there are discrete and continuous collision responses.
- For continuous collision responses, we must update the state to become collision-free state. There are two approaches: *interior point method* and *impact zone optimization*. Rigid impact zone is also a method, but it's problematic.
- For discrete collision responses, we allow intersections to stay and hope to remove them in long turn. Cloth-cloth intersections are difficult to handle.