Interactive Collision Detection between Deformable Models using Chromatic Decomposition

Naga K. Govindaraju * David Knott * Nitin Jain * Ilknur Kabul * Rasmus Tamstorf †
Russell Gayle * Ming C. Lin * Dinesh Manocha *
http://gamma.cs.unc.edu/CDCD

Abstract

We present a novel algorithm for accurately detecting all contacts, including self-collisions, between deformable models. We precompute a chromatic decomposition of a mesh into non-adjacent primitives using graph coloring algorithms. The chromatic decomposition enables us to check for collisions between non-adjacent primitives using a linear-time culling algorithm. As a result, we achieve higher culling efficiency and significantly reduce the number of false positives. We use our algorithm to check for collisions among complex deformable models consisting of tens of thousands of triangles for cloth modeling and medical simulation. Our algorithm accurately computes all contacts at interactive rates. We observed up to an order of magnitude speedup over prior methods.

CR Categories: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Visible surface algorithms, animation, virtual reality;

Keywords: Deformable collision detection, self-collision, graph coloring, cloth simulation

1 Introduction

Dynamic simulation of deformable models is essential to many applications, including character animation, path planning, computer gaming, haptic rendering, medical simulation and human-computer interaction. Fast and accurate collision detection is critical for generating realistic deformation and achieving real-time performance. In this paper, we address the problem of computing all contacts between deformable objects for interactive simulation. We restrict our inputs to polygonal meshes and assume that mesh connectivity does not change throughout the simulation. The mesh primitives may undergo any motion, but no vertices or edges can be inserted or deleted in the mesh. We make no other assumptions with respect to model topology, vertex connectivity, or the underlying simulation model. Many commonly used models for interactive deformation, such as mass-spring systems, free-form deformation, boundary element methods, and linearized finite element methods, satisfy these constraints.

One of the driving motivations for this work is cloth simulation. Efficient and robust handling of contacts remains a major challenge in simulating cloth dynamics [Baraff et al. 2003; Bridson et al. 2002; Volino and Thalmann 2000]. High-quality animation often requires modeling cloth with tens of thousands of mass particles. This combinatorial complexity leads to many collisions and numerous primitive pairs in close proximity. It is important to accurately detect all interferences, including self-collisions and collisions between the cloth and other objects. Even a single missed collision can result in an invalid simulation and noticeable visual artifacts, such as cloth passing through itself.

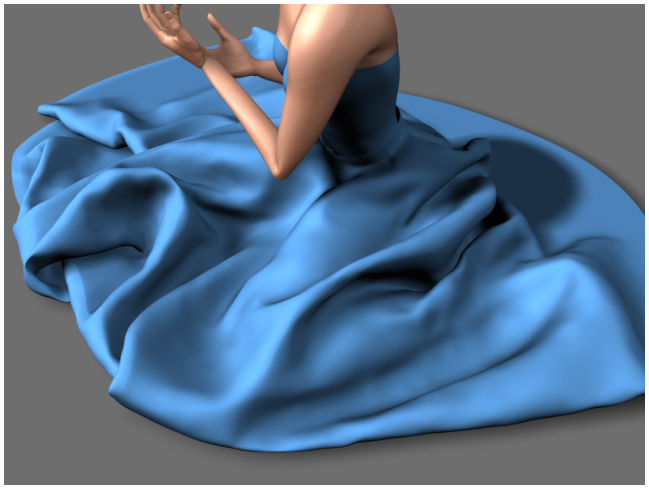


Figure 1: **Benchmark I***: This simulation generates complex folds and wrinkles on the skirt. The cloth is modeled as a mesh with 13K triangles. Our collision detection algorithm accurately checks for all self-collisions within 400-500ms during each step of the simulation. We significantly reduce the number of false positives and perform relatively few exact intersection tests between the primitives.*

Several techniques have been proposed to accelerate collision detection. Most existing methods, like hierarchical representations, work well for rigid bodies. Some of these methods extend to deformable meshes, but may not be able to offer interactive performance on complex models. Only a limited number of techniques can handle self-collisions for deformable models – often at non-interactive rates even for relatively simple meshes. Accurate self-collision detection is challenging to perform in real time because many adjacent or nearby primitives of a deforming mesh are always in close proximity. Current algorithms are unable to achieve significant culling in such cases and often result in a high number of false positives. In practice, collision detection, especially self-collisions, can account for 50-90% of the total simulation time and may take tens of seconds for a single time step. Therefore, many interactive simulators for deformable models either use approximate algorithms or do not check for self-collisions.

*University of North Carolina at Chapel Hill
†Walt Disney Feature Animation

Main Results: We present a novel algorithm for fast and reliable collision detection between deformable meshes with fixed connectivity. We assume that the simulator proceeds in discrete time steps. Continuous motion is modeled by linearly interpolating the vertex positions. In order to efficiently handle self-collisions and perform significant culling, we employ two new techniques:

1. Chromatic Mesh Decomposition: As part of a preprocess, we partition the mesh of each deforming object into *independent sets*. We ensure that no two primitives within each set are adjacent and impose some additional constraints. We compute the chromatic decomposition by performing graph coloring on the *extended-dual* graph of the mesh. Our chromatic decomposition transforms the self-collision detection problem into pairwise N-body collision detection between non-adjacent primitives.

2. Set-based Self-Collision Detection: We use a set-based formulation to compute potentially colliding primitives within each independent set. Our algorithm culls away primitives within each set using a bounding volume hierarchy. We describe a linear-time culling algorithm that performs 1D overlap tests on the CPU and 2.5D overlap tests on the GPU. We exploit spatial relationships between the primitives and also frame-to-frame coherence to reduce the number of exact overlap tests.

Our algorithm is robust and can handle various types of meshes with arbitrary topology and connectivity. The overall time complexity of the algorithm is nearly linear in the size of the input and the number of primitive pairs in close proximity. We have implemented our algorithm on a PC with a 3.4 GHz Pentium IV CPU and an NVIDIA GeForce 6800 GPU and applied it to many complex deformable meshes, including cloth simulation with 10K - 40K polygons and a medical simulation with over 90K polygons. We can compute all the contacts within $60 - 550$ msec per frame in IEEE 64-bit floating-point arithmetic. We also compare the performance of our algorithm with earlier techniques based on bounding volume hierarchies. Our set-based culling algorithm reduces the number of false positives by more than an order of magnitude for close proximity configurations. And, we have observed up to an order of magnitude speedup.

Organization: The rest of the paper is organized in the following manner. We survey some related work on collision detection in Section 2. Section 3 gives an overview of our approach and we present our algorithm to compute a chromatic decomposition in Section 4. Section 5 describes the set-based collision detection algorithm and we highlight its performance on different benchmarks in Section 6. We analyze the runtime complexity of our algorithm in Section 7 and discuss some of its limitations.

2 Related Work

In this section, we give a brief overview of related work in collision detection and cloth simulation. The problem of collision detection has been extensively studied and some recent surveys are available in [Lin and Manocha 2004; Teschner et al. 2004]. We limit our discussion to collision detection between deformable models, including cloth.

2.1 Hierarchical Approaches

Many collision detection algorithms use spatial partitioning or bounding volumes hierarchies to accelerate interference computations. Different bounding volumes including axis-aligned bounding boxes (AABBs) [Bridson et al. 2002; Baraff et al. 2003; DeRose et al. 1998] and k-DOPs [Mezger et al. 2003; Volino and Thalmann 2000] have been used to accelerate collision detection between deformable models. The cost of updating the hierarchies can be high and it is relatively inexpensive to use spheres or AABBs as bounding volumes. Recently, algorithms have been proposed to lower the overhead of updating the hierarchy during each time step of the deformable simulation: including top-down and bottom-up techniques [Larsson and Akenine-Möller 2001; van den Bergen 1997], models deformed by morphing [Larsson and Akenine-Möller 2003], and a sub-linear-time algorithm for deformable models expressed as linear superposition of precomputed displacements [James and Pai 2004]. Kimmerle et al. [Kimmerle et al. 2004] present an improved stochastic collision detection algorithm using hierarchies of k-DOPs and permit a trade-off between accuracy and speed. However, these hierarchies may not be able to perform significant culling. The overlap tests performed using bounding volumes can be rather conservative especially when some triangles have high aspect ratios or the primitives are in close proximity configurations (e.g. wrinkles, folds, self-collisions). In practice, these algorithms can result in a high number of false positives.

Figure 2: **Benchmark II**: *We use our collision detection to compute all contacts, including self-collisions, during cloth simulation. The cloth mesh consists of more than 23K triangles and our chromatic decomposition partitions the mesh into 19 independent sets. Our algorithm accurately computes all the collisions within 400-550 msec during each step of the simulation.*

2.2 GPU-based Collision Detection

GPUs have been used to perform interference and proximity computations [Heidelberger et al. 2003; Knott and Pai 2003; Govindaraju et al. 2003; Govindaraju et al. 2004; Rossignac et al. 1992]. These algorithms involve no preprocessing and are directly applicable to deformable models. Some specialized GPU-based algorithms have also been presented for self-collision and cloth collision detection [Baciu and Wong 2002; Heidelberger et al. 2004; Govindaraju et al. 2005; Vassilev et al. 2001]. However, these algorithms may be limited in terms of handling the type of input models (e.g. closed objects) and contact configurations (e.g. tangential contacts). A major limitation of the GPU-based algorithms is that the interference computations are performed at image-space resolution. As a result, these algorithms can miss collisions between small triangles due to sampling errors. Some enhancements based on computing offsets of primitives have been proposed to overcome image-precision errors and check for overlaps among disjoint objects at object-space resolution [Govindaraju et al. 2004]. However, these approaches cannot handle self-collisions.

2.3 Cloth Collision Detection

Many other algorithms have been proposed for collision detection and avoidance. Some earlier methods used repulsion forces between the two potentially colliding primitives [Baraff and Witkin 1998; Breen et al. 1994; Huh et al. 2001] and actual collisions are tested by interference computations between the primitives. Volino and Thalmann [1994] presented a sufficient condition for detecting self-collisions in highly tessellated surfaces using curvature and convexity properties [Mezger et al. 2003; Provot 1997; Volino and Thalmann 2000]. This test can be applied in a hierarchical manner

on large models, though it can be expensive for interactive applications [Volino and Thalmann 2000]. Many algorithms treat each polygonal primitive as a separate object and apply N-body collision detection algorithms based on uniform grids or AABB based sorting [Baraff 1992; Cohen et al. 1995]. However, prior N-body approaches have two major limitations in terms of using them for self-collision detection. First, the level of culling based on AABBs or rectangular cells of a grid may be low. Second, the storage requirements of coherence based sorting algorithms can grow as a quadratic function of the number of primitives.

Approximate Techniques: Given the complexity of self-collision detection, many interactive algorithms either do not check for self-collisions [Cordier and Magnenat-Thalmann 2002; Fuhrmann et al. 2003] or perform approximate collision detection using multiple layers [Kang and Cho 2002] or voxelized grids [Meyer et al. 2000]. It may be difficult to give bounds on the accuracy of a simulation with approximate collision detection.

3 Overview

In this section, we introduce our notation and give an overview of our approach.

3.1 Notation and Background

We focus on fast collision detection between deformable models represented using polygonal meshes. The mesh representing a deformable object can have arbitrary topology or vertex connectivity. We assume that the mesh connectivity or the polygon neighborhood information does not change during the simulation. Such meshes can be used to model stretching, shearing and bending. The simulator proceeds in discrete time steps, and during each time step we use a piecewise linear motion of the vertices to model the continuous motion of the triangles in the mesh.

We use the following notation in the rest of the paper. The mesh is represented using the symbol $\mathbf{M}$, and the polygonal primitives are represented as p_i, $i = 1, \ldots, n$, where n is the number of polygons in $\mathbf{M}$. The symbols v and e are used to denote vertices and edges, respectively, of a polygon. We use Δt to represent the maximum time step between successive instances of the simulator. We use the symbol P_i to refer the swept volume of a primitive p_i during the interval $[0, \Delta t]$ and represent the volume with a prism. The function $Adj(p_i, p_j)$ tests if two primitives p_i and p_j are adjacent, i.e., share a common edge or a vertex.

Exact elementary tests between features: Checking whether the swept volumes of two primitives overlap reduces to performing elementary tests based on co-planarity conditions. These include vertex-face (VF) and to edge-edge (EE) elementary tests. Each elementary test requires the computation of roots of a cubic equation that lie in the interval $[0, \Delta t]$ [Provot 1997; Bridson et al. 2002]. In case of non-adjacent triangles, we perform 9 EE tests and 6 VF tests. For adjacent polygons, we do not test the shared vertices and edges for co-planarity. We also add distance threshold constraints in the vertex-face or edge-edge co-planarity conditions for collision response computations.

3.2 Our Approach

Given a mesh $\mathbf{M}$, the goal of self-collision detection is to compute the first time of contact, $t \in [0, \Delta t]$, such that no two primitives in the mesh overlap at any time in the interval $[0, t]$. Otherwise, we report there are no collisions between the two discrete time steps. We cull primitives that do not overlap, reducing the number of false positives in terms of VF or EE elementary tests. In the rest of this paper, we describe our algorithm to detect self-collisions in a deformable mesh. Our algorithm can be directly applied to collision detection between two different objects using CULLIDE [Govindaraju et al. 2003].

We decompose the self-collision detection problem into two subproblems:

- **Non-Adjacent Collision Detection (NACD)** between non-adjacent primitives (e.g. the pairs $\{p_1^5, p_2^5\}$, $\{p_2^1, p_1^2\}$ in Fig. 3(a)).

- **Adjacent Collision Detection (ACD)** between adjacent primitives (e.g. the pairs $\{p_1^1, p_1^2\}$, $\{p_1^3, p_2^6\}$ in Fig. 3(a)).

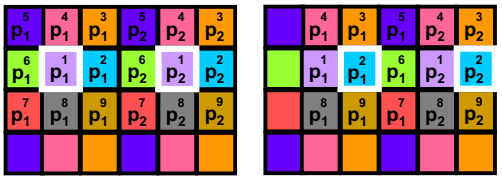


Figure 3: **Chromatic decomposition and reordering:** *We decompose a rectangular mesh with polygons p_i into nine independent sets, each shown in a different color (e.g. in Fig. 3(a) $\{p_1^1, p_2^1\}$, $\{p_1^6, p_2^6\}$). We use a linear-time algorithm to compute the chromatic decomposition of rectangular meshes. For each pair of sets (e.g. $\mathbf{S}^i$, $\mathbf{S}^j$), we reorder the primitives such that $p_l^i \in \mathbf{S}^i$ and $p_l^j \in \mathbf{S}^j$ are adjacent. Fig. 3(a) and 3(b) show the reordering of primitives in the remaining sets based on the primitive order in the sets $\mathbf{S}^1$ and $\mathbf{S}^2$, respectively.*

The ACD amounts to checking $O(n)$ pairs of adjacent primitives for overlap by performing elementary tests. On the other hand, NACD involves checking up to $O(n^2)$ pairs for overlap. Therefore, the performance of a self-collision detection algorithm is governed mainly by the performance of NACD, in terms of reducing the number of false positives between non-adjacent primitives. We use the problem decomposition and check for self-collisions as:

1. Compute all pairs of overlapping, non-adjacent primitives using a linear-time culling algorithm. We obtain high culling efficiency because we are only considering overlaps between non-adjacent primitives.

2. Perform exact VF and EE elementary tests between adjacent primitives. The results of non-adjacent primitive pairwise tests are used to eliminate a high fraction of false positives in terms of elementary tests between the adjacent primitives.

In order to compute pairs of overlapping non-adjacent primitives, we partition the mesh into disjoint sets $\mathbf{M} = \{\mathbf{S}^1, \ldots, \mathbf{S}^k\}$, called *independent sets*. **NACD** performs exact intersection tests between non-adjacent primitives (p^i, p^j) within every pair of sets $(\mathbf{S}^i, \mathbf{S}^j)$, $p^i \in \mathbf{S}^i, p^j \in \mathbf{S}^j$. In order to ignore collision tests between adjacent primitives, we introduce constraints and ensure that each primitive in one independent set is adjacent to at most one primitive in another independent set, as shown in Fig. 3(b). We compute a extended-dual graph of the mesh and use graph coloring algorithms to decompose the mesh into independent sets, i.e. chromatic decomposition.

We present an efficient self-collision culling algorithm to compute the overlapping primitives within each pair of sets. We extend the idea of *potentially colliding sets* (PCS) presented by Govindaraju et al. [2004] to handle swept volumes (i.e. the prisms). In particular, a primitive in a set $\mathbf{S}^i$ is potentially colliding if its swept volume during the time interval $[0, \Delta t]$ overlaps with the swept volume of some non-adjacent primitive in a set $\mathbf{S}^j$ during $[0, \Delta t]$. We compute the PCS by making linear passes over the primitives in the independent sets and using reliable image-based 2.5D overlap tests to cull non-overlapping primitives. We also present techniques based on temporal coherence and depth relationships to reduce the number of 2.5D overlap tests.

Collision Detection: We use a set-based collision detection algorithm to compute all pairs of non-adjacent overlapping primitives. In order to improve the culling performance, we precompute a AABB hierarchy in a bottom-up manner for each mesh. The leaf

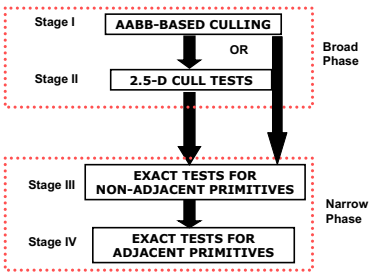


Figure 4: **Self-collision detection algorithm:** *Our algorithm proceeds in four stages. Stages I and II perform broad phase collision detection and cull many of the primitives that do not overlap with other non-adjacent primitives. Stage II is optional and performs 2.5D overlap tests when the PCS computed after stage I is large. Stages III and IV perform elementary tests on the primitives for exact collision detection.*

nodes of the hierarchy enclose the swept volume of a single polygon of the mesh. We add an offset to each box in the hierarchy to account for distance tolerances used in collision resolution computation. Our overall collision detection algorithm proceeds in four stages (as shown in Fig. 4):

- **Stage I:** Update the AABB hierarchy to account for changing vertex positions. Use the hierarchy to cull primitives in independent sets that do not overlap with other non-adjacent primitives (Section 5.1).
- **Stage II:** Use a set-based culling algorithm and perform 1D and 2.5D overlap tests to compute the PCS for each independent set (Section 5.2).
- **Stage III:** Compute pairs of overlapping, non-adjacent primitives by performing exact VF and EE elementary tests on the primitives within each PCS (Section 5.3).
- **Stage IV:** Check for overlaps between adjacent primitives by performing elementary tests and eliminate false positives by using the results of Stage III (Section 5.4).

4 Chromatic Mesh Decomposition

In this section, we present our mesh decomposition algorithm for computing independent sets. Given a polygonal mesh, we transform the mesh decomposition problem into a graph coloring problem and compute a chromatic decomposition of the graph. We first define the independent sets and then present an algorithm to compute them.

4.1 Independent Sets

The goal of mesh decomposition is to partition a mesh $\mathbf{M}$ into k disjoint independent sets $\mathbf{S}^i$ such that $\mathbf{M} = \mathbf{S}^1 \cup \mathbf{S}^2 \cup \ldots \cup \mathbf{S}^k$, and $\forall i, j, \mathbf{S}^i \cap \mathbf{S}^j = \emptyset$. Each independent set $\mathbf{S}^i$ is composed of a set of primitives $p^i_m, 1 \leq m \leq \|\mathbf{S}^i\|$. We impose two constraints on the primitives within each $\mathbf{S}^i$:

1. No two primitives in $\mathbf{S}^i$ are adjacent, and
2. For every pair of independent sets, $(\mathbf{S}^i, \mathbf{S}^j)$, we ensure that each primitive $p^i \in \mathbf{S}^i$ has at most one primitive $p^j \in \mathbf{S}^j$ such that $Adj(p^i, p^j)$ is true.

The second constraint is required by our self-collision culling algorithm, which is described in Section 5. We illustrate the decomposition for a rectangular mesh in Fig. 3. Note that any two polygons belonging to the same independent set have a gap of at least two polygons between them.

4.2 Graph Coloring

We transform mesh decomposition into a graph coloring problem and compute independent sets. A set of vertices in a graph is called an *independent set* if no two vertices of the set are connected in the graph by an edge. Graph coloring assigns each vertex a color such that no two adjacent vertices of the graph have the same color. All vertices with the same color are grouped into a *color class* [Jensen and Toft 1995]. Graph coloring results in a decomposition of the vertices into different color classes. The minimum number of colors in any proper coloring is called the *chromatic number* of a graph.

Our goal is to partition the primitives of a mesh into independent sets. Given a mesh, we construct the extended-dual graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as follows:

- Each primitive $p \in \mathbf{M}$ corresponds to a vertex $V(p) \in \mathbf{G}$.
- An edge $(V(p_l), V(p_m)) \in \mathbf{G}$ if and only if one of the following edge constraints are satisfied:
  - **Edge Constraint 1:** p_l and p_m are vertex-adjacent i.e., $Adj(p_l, p_m)$ is true, or
  - **Edge Constraint 2:** There exists a primitive $p \in M$ such that both $Adj(p_l, p)$ and $Adj(p_m, p)$ are true.

Note that if we use only Edge Constraint 1, we obtain the standard dual graph of the mesh. We extend the standard dual graph by placing additional edges among the vertices based on Edge Constraint 2. These two edge constraints guarantee that the mesh decomposition computed using graph coloring on $\mathbf{G}$ satisfies the constraints on the primitives described in Section 4.1. The first edge constraint ensures that adjacent primitives belong to different sets. The non-adjacency is ensured as an edge exists between every pair of adjacent primitives in the graph. Graph coloring assigns different colors to the corresponding vertices and they are in different independent sets. The second edge constraint ensures that no two primitives $p^i_l, p^i_m \in \mathbf{S}^i$ are adjacent to a common primitive $p^j \in \mathbf{S}^j$. Otherwise, both $Adj(p^i_l, p^j)$ and $Adj(p^i_m, p^j)$ would be true and there would be an edge $(V(p^i_l), V(p^i_m))$ in $\mathbf{G}$ because of Edge Constraint 2.

Given a graph $\mathbf{G}$, its decomposition into color classes is not unique. In particular, we can either minimize the number of independent sets, say k, or compute a decomposition such that the number of vertices in each independent set is bounded by some constant. In this case, our goal is to compute a decomposition that minimizes the number of independent sets. This is mainly because our self-collision detection algorithm (presented in Section 5) has an expected complexity of $\mathcal{O}(kn)$, where n is the number of primitives in $\mathbf{M}$. As a result, our problem reduces to the minimal graph coloring problem [Jensen and Toft 1995]. The minimal graph coloring problem is a well-studied problem and has been shown to be NP-hard for general graphs. In many cases, the mesh has a rectangular connectivity and can be mapped to a quad-tree. Eppstein et al. [2002] have presented a simple linear-time algorithm for coloring the squares of balanced and unbalanced quad-trees. We extend this algorithm to take into account our edge constraints and color meshes with rectangular connectivity, i.e. every vertex has four neighbors and every polygon is rectangular, in linear time. An example is shown in Fig. 3. In order to handle general meshes, we use an extension of the DSATUR algorithm [Brélaz 1979] to compute a graph coloring. The overall algorithm is fast and has a worst case complexity of $\mathcal{O}(\|\mathbf{E}\| \log \|\mathbf{V}\|)$. This algorithm is near-optimal for a large variety of graphs. In practice, this algorithm works well and the number of color classes is close to the chromatic number of the extended-dual graph. Theoretically, a simple upper bound on the chromatic number of any graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is $(0.5 + \sqrt{2||\mathbf{E}|| + 0.25})$. Based on our graph construction algorithm, we expect $||\mathbf{E}|| = \mathcal{O}(||\mathbf{V}||)$ and hence, a loose upper bound for a mesh $\mathbf{M}$ is $\mathcal{O}(\sqrt{||\mathbf{M}||})$. Theoretical bounds on k also exist as

a function of the maximum face size and the maximum valence in the mesh [Sanders and Zhao 1998].

The extended-dual graph coloring computes k disjoint sets of primitives $\mathbf{S}^i, i = 1, \ldots, k$, where the vertices corresponding to the primitives in a set are assigned the same color in the extended-dual graph. Given a decomposition of a mesh into independent sets, our algorithm reorders the primitives within each pair of independent sets. The order of the primitives is used by our self-collision detection algorithm which requires that every pair of primitives, (p_l^i, p_m^j), in a pair of independent sets $(\mathbf{S}^i, \mathbf{S}^j), 1 \leq i, j \leq k$ satisfy the following constraints:

$$\begin{aligned} \text{Adj}(p_l^i, p_l^j) &= \text{true}, \qquad (1) \\ \text{Adj}(p_l^i, p_m^i) &= Adj(p_l^j, p_m^j) = Adj(p_l^i, p_m^j) = \text{false}, l \neq m. \end{aligned}$$

It is possible that a primitive $p_l^i \in S^i$ has no adjacent primitive $p_l^j \in S^j$ since the graph coloring algorithm only ensures that at most one primitive in a set is adjacent to another primitive in any other independent set. In such cases, we treat p_l^j as $NULL$. The pseudo-code for our simple reordering algorithm for a pair of independent sets is given in Algorithm 4.1.

```
REORDER(S^i, S^j)
1  O = NULL /* output */
2  for each primitive P in S^i
3        for each primitive Q in S^j
4              if Q is adjacent to P,
5                    append the pair (P, Q) to O and mark Q
6        if no primitive Q in S^j is adjacent to P,
7              append the pair (P, NULL) to O
8  for each unmarked primitive Q in S^j,
9        append the pair (NULL,Q) to O
9  return O
```

ALGORITHM 4.1: *Reordering algorithm: The reordering algorithm computes pairs of adjacent primitives between two independent sets, and outputs a list of such pairs* $\mathbf{O}$. *Our edge constraints ensure that a primitive in a set* $\mathbf{S}^i$ *can have at most one adjacent primitive in another set* $\mathbf{S}^j, j \neq i$. *If* $j = i$, *then the adjacent primitive is itself. If a primitive* $P \in \mathbf{S}^i$ *has no adjacent primitive* $Q \in \mathbf{S}^j$, *then we append the pair* $(P, NULL)$ *(Lines 6 and 7). Similarly, we output* $(NULL, Q)$ *for unmarked primitives* $Q \in \mathbf{S}^j$ *(Lines 8 and 9).*

5 Collision Detection

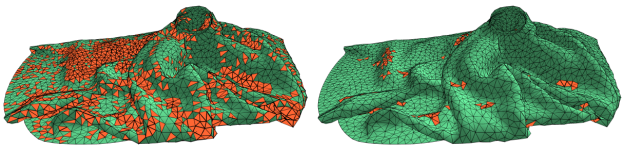
In this section, we present our self-collision detection algorithm. We use properties of chromatic mesh decomposition and perform set-based culling to compute potentially colliding, non-adjacent primitives. Finally, we use elementary tests for exact collision detection. We initially compute a prism P_l for each primitive p_l and also compute a bounding box around each prism. Our collision detection algorithm proceeds in four stages as described in Section 3.

5.1 Set-based Culling Using AABB Hierarchy

We dynamically update the AABB hierarchy for the mesh in a bottom-up manner. We update leaf nodes taking into account their vertex positions. The size of the boxes are increased based on the distance threshold. Next, we update the AABBs of the parent node based on the children's AABBs. We use the AABB hierarchy to cull primitives that are not overlapping with any other non-adjacent primitive. We compute a bounding box for each primitive P_m^i, check for overlaps using the hierarchy, and ignore the overlaps with adjacent primitives. The overlap tests are performed efficiently by testing the hierarchy against itself. We show the potentially colliding set of triangles in a cloth model computed using the AABB hierarchy in Fig. 5(a).

5.2 Set-based Culling Using 2.5D Overlap Tests

In this section, we present our linear-time algorithm that performs reliable 2.5D overlap tests using GPUs. For each independent set, our algorithm computes a PCS associated with that independent set. We also present techniques to reduce the number of 2.5D overlap tests by performing 1D tests on the CPU and use temporal coherence.



(a) *Collision Culling with AABB hierarchy*

(b) *Collision Culling with AABB hierarchy + 2.5D overlap tests*

Figure 5: **Improved culling performance with 2.5D overlap tests:** *The cloth model consists of* $12,296$ *triangles. The left image shows the* $6,442$ *potentially colliding triangles (in orange) computed using an AABB hierarchy (i.e., the output of Stage I). This results in more than* $65K$ *pairwise elementary tests (VF + EE). The right image shows the* 524 *potentially colliding triangles (in orange) computed after the 2.5D overlap tests (i.e., the output of Stage II), which results in* $2,737$ *pairwise elementary tests.*

Given a decomposition of a mesh into independent sets, as expressed in Eq. (1), we check the non-adjacent primitives within each set for collisions. Given an independent $\mathbf{S}^i$ and a primitive P_m^i, we define two subsets $\mathbf{S}_{<m}^i = \{P_1^i, \ldots, P_{m-1}^i\}$ and $\mathbf{S}_{>m}^i = \{P_{m+1}^i, \ldots, P_{\|\mathbf{S}^i\|}^i\}$. Given this formulation, we further decompose the NACD problem into two subproblems:

- **Intra-Set Collision Detection** $(\mathbf{S}^i, \mathbf{S}^i)$**:** Test each triangle $P_m^i \in \mathbf{S}^i$ for overlap with its non-adjacent primitives in $\mathbf{S}^i$ i.e., $\mathbf{S}_{<m}^i$, and $\mathbf{S}_{>m}^i$.

- **Inter-Set Collision Detection** $(\mathbf{S}^i, \mathbf{S}^j,\ i \neq j)$: Test each primitive $P_m^i \in \mathbf{S^i}$ for overlap with its non-adjacent primitives in $\mathbf{S}^j$ i.e. $\mathbf{S}_{<m}^j$, and $\mathbf{S}_{>m}^j$. Similarly, test each primitive $P_m^j \in \mathbf{S}^j$ for overlap with its non-adjacent triangles in $\mathbf{S}^i$ i.e. $\mathbf{S}_{<m}^i$, and $\mathbf{S}_{>m}^i$.

As a result, the collision detection problem reduces to performing $\mathcal{O}(k)$ intra-set collision detections and $\mathcal{O}(k^2)$ inter-set collision detections.

5.2.1 2.5D overlap tests

We extend the set-based culling algorithm, CULLIDE [Govindaraju et al. 2003], to perform self-collision detection. CULLIDE checks whether one object, say o_i, overlaps with a group of objects, say $\mathbf{O}$, by performing visibility computations between them. If o_i is fully visible with respect to all the objects in $\mathbf{O}$ along any viewpoint, then o_i does not overlap with any object in $\mathbf{O}$. We refer to this test as the *2.5D overlap test* between o_i and $\mathbf{O}$. The 2.5D overlap test implicitly checks whether there exists a separating surface of unit depth complexity from a given view direction. We implement this test using occlusion queries on a GPU. An occlusion query returns the number of pixels that pass the depth test. In order to perform the 2.5D tests using occlusion queries, we first render all the objects in $\mathbf{O}$ and compute its visible surface in the depth buffer. We then render the object o_i using an image-space occlusion query and test whether the depth of all rasterized fragments of o_i are in front of the visible surface in the depth buffer. We overcome image-precision errors in occlusion queries by adding an offset to the bounding swept volume of each primitive [Govindaraju et al. 2004]. The size of the offset is a function of the pixel resolution used to perform the overlap tests. In practice, the 2.5D overlap test is less conservative as compared to bounding volume based tests and culls a larger fraction of non-overlapping primitives in close proximity.

CULLIDE performs a linear-time front-to-back and back-to-front traversal of the list of objects and computes a potentially colliding set (PCS) of objects. However, CULLIDE is unable to check for self-collisions or perform inter-set culling between independent

sets. If we treat each polygon in a connected mesh as a separate object, CULLIDE would not be able to cull any primitive and the PCS would consist of all the primitives in the set. In the same manner, if we apply CULLIDE to a union of two sets, say $\mathbf{S}^i$ and $\mathbf{S}^j$, it would perform no culling because every primitive in $\mathbf{S}^i$ has an adjacent primitive in $\mathbf{S}^j$.

5.2.2 Self-Collision Culling

We present a modified linear-time algorithm that uses 2.5D overlap tests to perform self-collision culling. We take advantage of chromatic decomposition and properties of independent sets to cull the non-overlapping primitives within each PCS. Our mesh decomposition algorithm ensures that every primitive $P^i_m \in \mathbf{S}^i$ has one adjacent element $P^j_m \in \mathbf{S}^j$ and P^j_m partitions $\mathbf{S}^j - \{P^j_m\}$ into two subsets: $\mathbf{S}^j_{<m}$ and $\mathbf{S}^j_{>m}$. We make two linear passes over these sets.

- **First pass:** Traverse the primitives in $\mathbf{S}^i$ from the last to the first. During the traversal, we test if the current primitive $P^i_m \in \mathbf{S}^i$ is fully visible against the previously rendered primitives in $\mathbf{S}^j$ using a 2.5D overlap test (i.e., with primitives in $\mathbf{S}^j_{>m}$). We then render the potentially intersecting primitive $P^j_m \in \mathbf{S}^j$ into the frame buffer.

- **Second pass:** Traverse the primitives in $\mathbf{S}^i$ from the first primitive to the last. During each scan, we only test the primitive $P^i_m \in \mathbf{S}^i$ which was fully visible in the first pass for potential overlap with the PCS in $\mathbf{S}^j$ (i.e. $\mathbf{S}^j_{<m}$). We then render $P^j_m \in \mathbf{S}^j$ into the frame buffer.

The two passes are sufficient to test each primitive in $\mathbf{S}^i$ against non-adjacent, potentially intersecting primitives in $\mathbf{S}^j$. We repeat the same algorithm to compute the PCS for the set $\mathbf{S}^j$ in $(\mathbf{S}^i, \mathbf{S}^j)$. We use multiple view directions along the three world-space axes. The pruning algorithm is used repeatedly till the PCS does not decrease between successive instances. The same two-pass algorithm is used for the inter-set and the intra-set culling. The 2.5D overlap tests can cull away a high number of non-overlapping primitives in close proximity (Fig. 5(b)).

5.2.3 Reducing 2.5D overlap tests

The self-collision algorithm described above performs $2kn$ 2.5D overlap tests, where k is the number of independent sets and n is the number of primitives. In the case of a complex model, n can be very high while k is typically in the range $10 - 20$. As a result, the overall performance of the algorithm is governed by the number of 2.5D overlap tests used for self-collision culling. In this section, we present three techniques that use spatial relationships and temporal coherence to reduce the number of 2.5D overlap tests.

View Coherence: We utilize coherence between different views to reduce the number of 2.5D overlap tests between each pair of sets $(\mathbf{S}^i, \mathbf{S}^j)$. For example, if a primitive P^j_m does not overlap with any of the primitives $\mathbf{S}^i_{>m}$ along some view, then P^j_m does not overlap with any primitive in $\mathbf{S}^i_{>m}$. Therefore, we do not need to render P^j_m while testing the overlap status of primitives in $\mathbf{S}^i_{>m}$ along any view. Specifically, we do not render P^j_m after testing the full-visibility of P^i_m in the first pass. We use a similar sufficient condition to reduce the rendering operations in the second pass based on the overlap between P^j_m and $\mathbf{S}^i_{<m}$. Intuitively, we are reducing the depth complexity of the PCS associated with $\mathbf{S}^j$ against the PCS associated with $\mathbf{S}^i$ by rendering only the potentially overlapping primitives in $\mathbf{S}^j$. As a result, view coherence accelerates the performance of the culling algorithm.

1D overlap tests: We reduce the number of 2.5D overlap tests by first performing 1D overlap tests on the CPU. Our 1D algorithm uses view coherence relations along the world-space axes. We compute the 1D overlap relations before performing the 2.5D overlap tests. In particular, we project the AABBs of the primitives along each axis and apply our culling algorithm described in Section 5.2.2. The 1D overlap test is performed using the same two-pass algorithm. In the first pass, we maintain and update the minimum and maximum values of the AABB of the primitives in $\mathbf{S}^j_{>m}$ along each axis and check for overlap against the AABB of P^i_m. The second pass is performed in the same manner. Combined with the conditions used to check for view coherence, 1D overlap tests can reduce the number of 2.5D overlap tests.

```
GPUCULL(S^i, S^j)
1  Mark each primitive P^i_k in S^i as potentially colliding /* Initialization */
2  n = ||S^i|| = ||S^j||
3  for each view
4        for k= n to 1 /* first pass */
5              if P^i_k ≠ NULL and is not fully visible in first pass
6                    Render P^i_k with an occlusion query
7                    If fully visible, mark P^i_k as first-pass-fully-visible
8              if P^j_k ≠ NULL and is not fully visible in second pass
9                    Render P^j_k
10       for k= 1 to n /* second pass */
11             if P^i_k ≠ NULL and is not fully visible in second pass
12                   Render P^i_k with an occlusion query
13                   If fully visible, mark P^i_k as second-pass-fully-visible
14             if P^j_k ≠ NULL and is not fully visible in fully pass
15                   Render P^j_k
```

ALGORITHM 5.1: *Self-collision culling algorithm: Our self-collision culling algorithm proceeds in two passes. Lines 4-9 perform the first pass and test each primitive $P^i_k \in \mathbf{S}^i$ for overlap with $\mathbf{S}^j_{>k}$. Similarly, lines $10 - 15$ perform the second pass and test each primitive P^i_k against $\mathbf{S}^j_{<k}$. Each visibility test is performed using an occlusion query (Lines 6 and 12) and the rendering operation during the occlusion query is performed with the depth test GL_GEQUAL. The remaining rendering operations (Lines 9 and 15) are performed using the depth test GL_LESS. Primitives which are fully visible in both first and second passes do not collide with any of the primitives in $\mathbf{S}^j$ and are pruned from the PCS of these pair of sets.*

Temporal Coherence: The 2.5D overlap tests are used to check whether a primitive is fully visible with respect to the rest of the primitives along a view direction. In most interactive applications, there is high coherence in the visibility relationships between the primitives. We rearrange the primitives based on the order in which they were fully visible in the previous frame or the last viewing direction. Due to the temporal coherence, we expect the primitives to be fully visible in nearly the same order. This optimization is useful for reducing the required number of views used to compute the PCS. As a result, we perform fewer 2.5D overlap tests.

Our overall GPU-accelerated culling algorithm is simple and efficient. A pseudo-code description to compute the potentially colliding primitives of a set $\mathbf{S}^i$ against a set $\mathbf{S}^j$ is given in Algorithm 5.1.

5.3 Exact Tests: Non-Adjacent Primitives

The set-based culling algorithm computes a PCS for each independent set. In the third stage, we check for exact intersections by performing elementary tests between non-adjacent pairs. We first merge the PCS of all the independent sets by computing their union. Next, we perform an N-body test between the elements to compute potentially intersecting pairs. We use our AABB hierarchy to quickly compute the potentially intersecting pairs. Finally, we perform elementary EE and VF tests between the pairs to check for exact intersections. We explicitly maintain a list of all overlapping, non-adjacent primitives and store this information in the data structures used to represent the mesh. Even though each edge or vertex can be shared by more than one polygon, we do not perform redundant EE or VF tests.

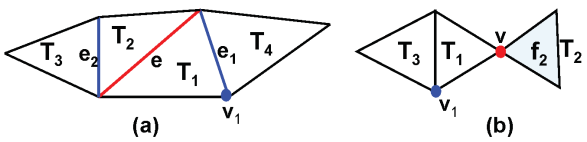


Figure 6: **Culling elementary tests between adjacent primitives:** *If non-adjacent triangles in these examples are not overlapping, we do not need to perform the elementary test between* (e_1, e_2) *and* (v_1, T_2) *as shown in (a) and* (v_1, f_2) *as shown in (b).*

5.4 Exact Tests: Adjacent Primitives

In the last stage of the algorithm, we need to check all adjacent primitives for exact intersection. We perform elementary EE and VF tests between adjacent polygons, but we do not test the shared edge or vertex for co-planarity conditions. For example, given two triangles sharing a vertex, we perform only 4 VF and 5 EE tests. Similarly if two triangles share an edge, we perform just 2 VF and 2 EE tests.

We use the results of non-adjacent, overlapping primitive pairs to cull many of the elementary tests between adjacent primitives (i.e. the false positives). We illustrate our algorithm with a figure. For example, in Fig. 6(a), T_1 and T_2 are an adjacent pair of triangles that share an edge. Let us assume that their neighboring triangles, T_3 and T_4, do not overlap. As a result, we do not need to perform elementary tests between (e_1, e_2) and (v_1, T_2). In Fig. 6(b), the triangles T_1 and T_2 share a vertex. If T_2 and T_3 do not overlap, then we do not need to perform an elementary test between the pair (v_1, f_2). In this manner, we can cull a high number of elementary tests between the adjacent primitives by using the results of Stage III.

6 Implementation and Performance

We have implemented our algorithm on a PC running Windows XP with a 3.4 GHz Pentium IV CPU, an NVIDIA GeForce 6800 GPU, and 2 GB of main memory. We use hyper-threading and the Pentium IV SSE2 optimizations of the Intel compiler 8.0 to improve the performance. Moreover, we use the asynchronous GL_NV_occlusion_query queries in OpenGL to implement fast and reliable 2.5D overlap tests on the GPU. We optimize our rendering throughput by using the GL_ARB_vertex_buffer_object extensions and avoid the stalls by batching multiple visibility queries. In practice, we are able to achieve 1.1M occlusion query throughput per second.

The chromatic mesh decomposition algorithm is performed once as a pre-process for a mesh. We use the DSATUR algorithm [Brélaz 1979] to compute the color classes. In our benchmarks, this algorithm works well and takes approximately two seconds on a mesh with 10K triangles. The number of independent sets is typically in the range $10 - 20$.

The 2.5D overlap tests are performed along the three world-space axes using orthographic projections. The reliable tests are performed using a viewport resolution of $1K \times 1K$. The prisms, P_m^i are computed based on the deviation and height of the final position of each vertex with respect to the plane corresponding to the initial positions. When the time steps used in the simulation are small, the prism tightly bounds the swept volume of the polygon. Moreover, the prism computation time is relatively low. In our implementation, we are able to compute the prisms on the CPU and transfer to the GPUs at the rate of 10K prisms in 10 msec. Moreover, view coherence, 1D overlap tests and temporal coherence reduce the number of 2.5D overlap tests by 20-60% in our benchmarks. We have implemented the VF and EE elementary tests using interval arithmetic. We compute the coefficients of the cubic equations and check for an existence of a root in the time interval. We also take into account distance thresholds. It takes about 3 usec on average to perform one elementary test.

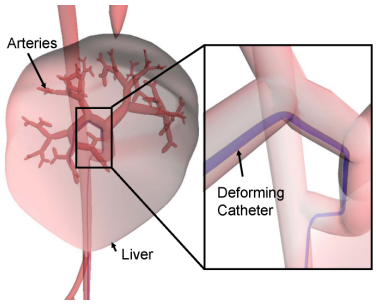


Figure 7: **Path planning for a deformable catheter:** *Our collision detection algorithm is used for path planning of a deforming catheter in liver chemoembolization. The catheter is modeled as a mass-spring system with* $10K$ *triangles. The geometric model of static arteries and the liver has more than* $83K$ *triangles. Our algorithm computes all the contacts between the catheter and the environment in* 60-90 *msec.*

6.1 Performance

We use our collision detection algorithm in cloth simulation as well as surgical planning of a catheter insertion for liver chemoembolization. We highlight the results on three cloth simulations shown in Fig. 1, Fig. 2, and Fig. 8. The number of polygons in the mesh used to model the cloth vary from 10K to 40K. The time to check for all collisions, including self-collisions, is in the range of $100 - 550$msec. The performance depends on the input complexity, output complexity and the number of independent sets generated using chromatic decomposition. Specifically, collision detection takes around $100 - 150$msec when the number of independent sets is around 10, and $300 - 550$msec when the number of independent sets is around 20. We observed that checking for self-collisions takes 3-5 times longer than cloth-object collision.

We also used our collision detection algorithm for surgical planning of a catheter in liver chemoembolization [Gayle et al. 2005]. The catheter is modeled as a mass-spring system undergoing real-time deformation. As the catheter deforms, we check for self-collisions as well as collisions with the arteries. The catheter is modeled with 10K polygons and the arteries are modeled with 83K triangles (Fig. 7). Our collision detection algorithm takes about 60-90msec to compute all contacts during each time step. A collision free path computed by the simulator is also shown in the video.

We give a breakup of the running time of our collision detection algorithm among different stages as a function of model complexity in Fig. 9(a). The stage I takes about $40 - 50\%$ of time to update the AABB hierarchy and perform culling. The stage II of the algorithm takes about $35 - 45\%$ of the total time. We spend less than 10% of the time in the elementary tests.

6.2 Comparison

We have compared the performance and culling efficiency of our algorithm against prior collision detection algorithms for deformable models. Current GPU-based algorithms either do not check for self-collisions or their accuracy is limited by image-precision. Therefore, we have limited our comparison to object-space algorithms based on bounding volume hierarchies. Most collision detection algorithms use AABBs or spheres as bounding volumes, since the cost to update the hierarchy is relatively low [Larsson and Akenine-Möller 2001; van den Bergen 1997]. These hierarchies are updated during each time step and used to compute potentially overlapping pairs of primitives. Finally, these algorithms perform elementary tests between the primitives at the leaf nodes. We do not perform elementary tests between the shared edges or vertices.

We compare the performance of our algorithm with AABB and sphere hierarchies. In our benchmarks, the culling obtained by sphere trees is rather poor as compared to the AABB trees. As the cloth deforms, many triangles become long and skinny and the bounding spheres can become rather large. Our algorithm also computes an AABB hierarchy, but only uses the hierarchy to cull away non-adjacent primitives that do not overlap. Fig. 9(b) shows the

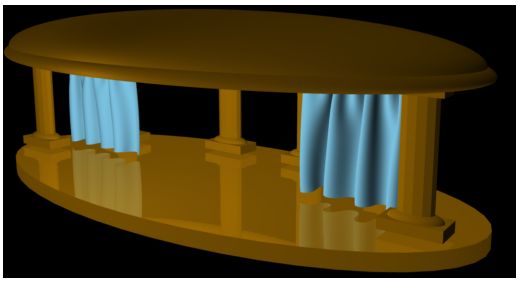


Figure 8: **Folding Curtains**: *This simulation highlights a stage scene with two folding curtains. Each curtain is modeled using* $32,500$ *triangles and we decompose each mesh into* 10 *independent sets. As the simulation progresses, the curtains are pulled closer and generate complex folds and wrinkles. Our algorithm computes all self-collisions within each curtain in about* 100 *msec.*

improvement in the overall running time obtained by our algorithm in a cloth simulation, where the cloth has about 32K triangles. Fig. 9(d) compares the culling efficiency of our algorithm in terms of elementary VF and EE tests.

Speedups: Our algorithm obtains speedup due to fewer number of elementary tests between the primitives. In our benchmarks, we have observed five times improvement in the running time even without 2.5D overlap tests, i.e. without Stage II. This performance improvement is due to our decomposition of the problem into ACD and NACD. Our algorithm results in relatively fewer false positives between non-adjacent primitives. Furthermore, we use the results of NACD to cull away a high fraction of elementary tests between the adjacent primitives. If we perform 2.5D overlap tests on the GPUs (i.e. Stage II), we achieve up to $15-20$ times overall speedup, because of high culling efficiency in close proximity scenarios. As a result, the use of GPU-based 2.5D overlap tests results in additional speedup of $3-4$ times.

7 Analysis and Limitations

We analyze the complexity of our algorithm by analyzing each stage of the algorithm separately. Updating the AABB hierarchy and performing culling tests during the first stage takes $\mathcal{O}(n)$ time, where n is the number of primitives. The complexity of the second stage also depends on the number of independent sets. Given a pair of independent sets $(\mathbf{S}^i, \mathbf{S}^j)$, the collision culling algorithm requires $\mathcal{O}(\|\mathbf{S}^i\| + \|\mathbf{S}^j\|)$ operations. Therefore, each set $\mathbf{S}^i$ requires an average of $\mathcal{O}(\frac{k\|\mathbf{S}^i\|+n}{2})$ operations to test for overlaps against other sets. As a result, the run-time complexity of the self-collision culling algorithm is $\mathcal{O}(kn)$. The complexity of the third stage is a function of the number of non-adjacent primitives that are in close proximity or overlapping. Finally, the fourth stage takes linear time.

In our current benchmarks, most of the query time is spent on the first two stages: 75- 90% of the query time. Since our algorithm culls a very high fraction of non-overlapping primitive pairs, we need to perform relatively fewer elementary tests. The 2.5D overlap tests are useful when a high number of non-adjacent primitives are in close proximity. If the PCS computed after the first stage is small, we need not use the 2.5D overlap tests, therefore eliminating the second stage of the algorithm all together.

The decomposition of the original problem into adjacent and non-adjacent primitives, along with a fast set-based culling algorithm enables us to achieve almost interactive performance on complex deformable models. Since we first check for overlaps only among non-adjacent primitives, we are able to perform significant culling and significantly reduce the number of false positives. Furthermore, we use the results from non-adjacent primitives to cull away a very high fraction of elementary tests between the adjacent primitives.

Limitations: Our algorithm has a few limitations. We restrict our inputs to be polygonal meshes with fixed connectivity. The culling efficiency of our algorithm can vary based on the relative configuration of the primitives in the mesh and their placement with respect to the viewing directions used for 2.5D overlap tests. Our set-based culling algorithm works well when the number of overlapping pairs is relatively small. If there are a high number of colliding or penetrating primitives, the resulting PCS can be high and we may not benefit much from set-based culling algorithm. The 2.5D based overlap tests are more effective when the primitives are in close proximity. Finally, our chromatic decomposition algorithm based on graph coloring may produce a high number of independent sets, even though the number of sets have varied in the range $10-20$ in our benchmark.

8 Conclusions

We present a novel algorithm for collision detection between deformable models. We decompose the problem into ACD and NACD and precompute a chromatic decomposition of the mesh. We use a linear-time culling algorithm that performs 1D and 2.5D overlap tests. We have applied our algorithm to multiple benchmarks in cloth modeling and medical simulation. Our initial results are promising and we have observed more than an order of magnitude improvement in running time over previous approaches.

There are many avenues for future work. We can improve the performance for complex deformable models by grouping the primitives into small clusters (e.g. 2-4 polygons) and then computing potentially colliding sets of clusters using the AABB-hierarchy and 2.5D overlap tests. Currently, we only compute pairs of overlapping primitives. It may be possible to combine our algorithm with curvature-based tests to further improve the culling performance. We would like to relax our assumption about fixed mesh connectivity and incrementally update the chromatic decomposition, whenever the connectivity changes. We would like to apply our algorithm to higher-order primitives including NURBS or subdivision surfaces. In this case, we may be able to directly compute the chromatic decomposition based on subdivision rules, as opposed to using graph coloring algorithms. Finally, we plan to use our collision detection algorithm in other applications including avatars in virtual environments.

Acknowledgments

Our work was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0118743, 0400134, 0429583 and 0404088, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043, Alias and Intel. We would like to thank NVIDIA corporation for hardware and driver support. We would like to acknowledge Avneesh Sud for video editing and Brandon Lloyd, Mike Henson, and other members of UNC GAMMA group for their feedback. We are also grateful to the reviewers for their comments.

References

BACIU, G., AND WONG, S. 2002. Hardware-assisted self-collision for deformable models. *ACM Symposium on VRST*, 129–136.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. *Proc. of ACM SIGGRAPH*, 43–54.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *Proc. of ACM SIGGRAPH*, 862–870.

BARAFF, D. 1992. *Dynamic simulation of non-penetrating rigid body simulation*. PhD thesis, Cornell University.

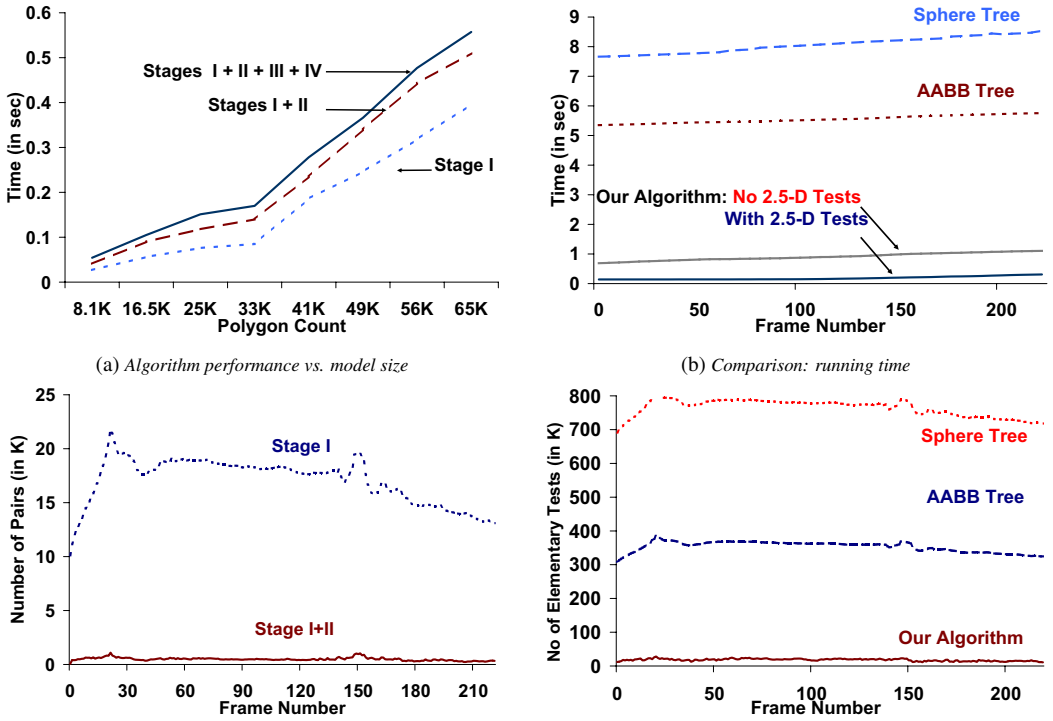
BREEN, D., HOUSE, D., AND WOZNY, M. 1994. Predicting the drape of woven cloth using interacting particles. *Proc. of ACM SIGGRAPH*, 365–372.

BRÉLAZ, D. 1979. New methods to color the vertices of a graph. *Communications of the ACM 22*, 4, 251–256.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treament for collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH*.

COHEN, J., LIN, M., MANOCHA, D., AND PONAMGI, M. 1995. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, 189–196.

CORDIER, F., AND MAGNENAT-THALMANN, N. 2002. Real-time animation of dressed virtual humans. *Computer Graphics Forum 21*, 3.



(a) *Algorithm performance vs. model size*

(b) *Comparison: running time*

(c) *Comparison: culling efficiency in broad phase*

(d) *Comparison: culling performance*

Figure 9: **Performance comparison and culling efficiency:** *We compare the performance of our algorithm with prior approaches based on AABB and sphere hierarchies. We show the breakup of the total time in different stages of the algorithm in (a). We highlight the benefit of GPU-based 2.5D overlap tests over AABB hierarchies in (c). We compare the performance of our algorithm with prior approaches based on AABB and sphere hierarchies in (b) and (d). We achieve more than an order of magnitude improvement in terms of culling efficiency and the overall running time. Even without the 2.5D overlap tests (i.e. no Stage II), the chromatic decomposition and separate handling of adjacent and non-adjacent primitives result in a speedup of almost* 5 *times over prior algorithms, as shown in (b).*

DeRose, T., Kass, M., and Truong, T. 1998. Subdivision surfaces in character animation. *Proc. of ACM SIGGRAPH*.

Eppstein, D., Bern, M., and Hutchings, B. 2002. Algorithms for coloring quadtrees. *Algorithmica 32*, 1, 87–94.

Fuhrmann, A., Gross, C., and Luckas, V. 2003. Interactive animation of cloth including self collision detection. *Journal of WSCG 11*, 1.

Gayle, R., Segars, P., Lin, M., and Manocha, D. 2005. Path planning for deformable robots in complex environments. Tech. rep., University of North Carolina-Chapel Hill. To appear in Proc. of Robotics: Science and Systems.

Govindaraju, N., Redon, S., Lin, M., and Manocha, D. 2003. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 25–32.

Govindaraju, N., Lin, M., and Manocha, D. 2004. Fast and reliable collision detection using graphics hardware. *Proc. of ACM VRST*.

Govindaraju, N., Lin, M., and Manocha, D. 2005. Quick-CULLIDE: Efficient inter- and intra- object collision culling using GPUs. *Proc. of IEEE VR*.

Heidelberger, B., Teschner, M., and Gross, M. 2003. Real-time volumetic intersections of deforming objects. *Proc. of Vision, Modeling and Visualization*.

Heidelberger, B., Teschner, M., and Gross, M. 2004. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3.

Huh, S., Metaxas, D., and Badler, N. 2001. Collision resolutions in cloth simulation. *Computer Animation, IEEE*, 604–611.

James, D. L., and Pai, D. K. 2004. BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH*.

Jensen, T., and Toft, B. 1995. *Graph Coloring Problems*. Wiley InterScience.

Kang, Y., and Cho, H. 2002. Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. *Computer Animation*, 604–611.

Kimmerle, S., Nesme, M., and Faure, F. 2004. Hierarchy accelerated stochastic collision detection. In *Proceedings VMV*.

Knott, D., and Pai, D. K. 2003. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, 73–80.

Larsson, T., and Akenine-Möller, T. 2001. Collision detection for continuously deforming bodies. In *Eurographics*.

Larsson, T., and Akenine-Möller, T. 2003. Efficient collision detection for models deformed by morphing. *Visual Computer 19*, 164–174.

Lin, M. C., and Manocha, D. 2004. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL, ch. 35, 787–807.

Meyer, M., Debunne, G., Desbrun, M., and Barr, A. 2000. Interactive animation of cloth like objects in virtual reality. *Journal of Visualization and Computer Animation 12*, 1, 1–12.

Mezger, J., Kimmerle, S., and Etzmuβ, O. 2003. Hierarchical techniques in cloth detection for cloth animation. *Journal of WSCG 11*, 1.

Provot, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, 177–189.

Rossignac, J., Megahed, A., and Schneider, B. 1992. Interactive inspection of solids: cross-sections and interferences. In *Proceedings of ACM SIGGRAPH*, 353–60.

Sanders, D., and Zhao, Y. 1998. On d-diagonal colorings of embedded graphs of low maximum face size. *Graphs and Combinatorics 14*, 81–94.

Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., and Volino, P. 2004. Collision detection for deformable objects. *Eurographics*.

van den Bergen, G. 1997. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4, 1–14.

Vassilev, T., Spanlang, B., and Chrysanthou, Y. 2001. Fast cloth animation on walking avatars. *Computer Graphics Forum (Proc. of Eurographics'01) 20*, 3, 260–267.

Volino, P., and Thalmann, N. M. 1994. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.) 13*, 3, 155–166.

Volino, P., and Thalmann, N. M. 2000. Accurate collision response on polygon meshes. *Computer Animation*.