Eurographics Symposium on Virtual Environments (2006)
Roger Hubbold and Ming Lin (Editors)

Fast Continuous Collision Detection among Deformable Models using Graphics Processors

Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin, Dinesh Manocha

University of North Carolina at Chapel Hill
{naga,ilknurk,lin,dm}@cs.unc.edu

Abstract

We present an interactive algorithm to perform continuous collision detection between general deformable models using graphics processors (GPUs). We model the motion of each object in the environment as a continuous path and check for collisions along the paths. Our algorithm precomputes the chromatic decomposition for each object and uses visibility queries on GPUs to quickly compute potentially colliding sets of primitives. We introduce a primitive classification technique to perform efficient continuous self-collision. We have implemented our algorithm on a 3.0 *GHz Pentium IV PC with a NVIDIA* 7800 *GPU, and we highlight its performance on complex simulations composed of several thousands of triangles. In practice, our algorithm is able to detect all contacts, including self-collisions, at image-space precision in tens of milli-seconds.*

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computing Methodologies]: Hardware Architecture; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism; I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling;

1. Introduction

The problem of collision detection (CD) arises in geometric modeling, simulation and interaction for diverse areas, including virtual reality (VR), cloth simulation, haptic rendering, animation, rapid prototyping, CAD/CAM, robotics, and entertainment. In this paper, we mainly focus on collision detection *over a given time period* between deformable objects that includes inter-object collisions between disjoint objects and intra-object collisions (or self-collisions) within each deformable model. Most of the earlier work in CD has been restricted to collision detection at *discrete* time instances and these algorithms may not check for possible overlaps between successive time steps. As a result, it is possible to miss a collision and can result in visual artifacts, inconsistent state, or incorrect simulation that can significantly affect the sense of immersion and lead to break in presence (BIP) in a virtual environment (VE). Such problems can be especially challenging in environments composed of thin or fast moving objects, like cloth on virtual avatars.

In order to overcome the limitations of discrete CD algorithms, many techniques have been proposed that model the motion between successive time instances as a continuous path and check for collisions along these paths. These are classified as *continuous collision detection* (CCD) algorithms [Can86, RKC00, KR03, RKLM04a, RKLM04b]. However, current CCD algorithms are only able to handle rigid objects, articulated models, or simple deformable meshes (consisting of only a few hundreds of polygons) at interactive rates.

Main Results: We present a novel algorithm to perform CCD between general deformable models at interactive rates using commodity graphics processing units (GPUs). Our approach is general and makes no assumption about object motion. Each object is represented as a triangulated mesh and we assume that the mesh connectivity does not change during the simulation. We model the continuous motion using a piecewise linear motion between successive discrete instances and check for collisions between the resulting swept volumes.

Our algorithm precomputes an *improved* chromatic decomposition [GKJ*05] for each object and decomposes the problem into checking for overlap between adjacent and

non-adjacent primitives. We extend the Quick-CULLIDE collision culling algorithm [GLM05], which performs collisions at discrete time instances, to check for collisions between the swept volumes of non-adjacent primitives. In order to achieve high-culling efficiency, we classify the adjacent primitives into vertex-adjacent and edge-adjacent primitives. We pair the edge-adjacent primitives to reduce the number of penetrating contacts, as well as lowering the number of pairwise tests. Based on this decomposition, our algorithm directly checks for inter- and intra-object collisions and does not need to perform any special processing. The overall accuracy of our algorithm is governed by image-space resolution used to perform visibility queries (or 2.5D overlap tests) on the GPUs.

We have implemented the algorithm on a 3.0 GHz PC with a NVIDIA 7800 GPU and applied to complex cloth simulation benchmarks consisting of thousands of triangles. We are able to check for all collisions between the discrete time instances at $10-15$ frames per second at 1000×1000 image-space resolution, enabling real-time *continuous* collision detection for high-resolution deformable meshes with challenging contact scenarios in virtual environments. As compared to earlier work [GLM05, GKJ*05], our approach offers the following advantages:

- *High culling efficiency* using tight bounding swept volume representations for the mesh primitives.
- *Interactive performance* using an improved mesh decomposition scheme and a novel primitive classification technique, resulting in fewer false positives.

Organization: The rest of the paper is organized as follows. In Section 2, we briefly review the prior work on CCD and GPU-based collision detection algorithms. We give an overview of our approach in Section 3 and describe our primitive classification scheme. We present our algorithm in Section 4 and describe its implementation and performance on complex benchmarks in Section 5. We analyze its performance in Section 6 and highlight some of its limitations.

2. Related Work

The problem of collision detection has been extensively studied in the literature. A good overview of different algorithms is available in some recent surveys [Eri04, TMH*05, LM03]. In this section, we give a brief survey of earlier work related to continuous collision detection, GPU-based methods and collision detection between deformable models.

2.1. Continuous Collision Detection

Most of the prior work on collision detection has been restricted to checking for collisions at discrete time instances. Recently, many algorithms have been proposed for continuous collision detection (CCD). These approaches model the trajectory of the object between successive discrete time instances as a continuous path and check the result path for collisions. Different techniques have been used to model the trajectory including linear interpolation between the vertex positions, screw motion or arbitrary in-between motion, etc. At a broad level, the CCD algorithms can be classified into four approaches: algebraic equation solving approach [Can86, RKC00], swept volume (SV) techniques [AMBJ02], adaptive bisection [RKC02, SSL02], and kinetic data structures (KDS) [ABG*00]. These approaches have been used to perform CCD at interactive rates for rigid objects [RKC02, KR03] and articulated models or avatars in virtual environments [RKLM04b, RKLM04a]. However, no good algorithms are known for real-time CCD between general deformable models.

2.2. Collision Detection between Deformable Models

Many of the commonly used collision detection algorithms utilize spatial partitioning or bounding volumes hierarchies. Different bounding volumes including axis-aligned bounding boxes (AABBs) [BFA02, DKT98], OBBs [PBS02] and k-DOPs [MKE03, VT00] have been used to accelerate collision detection between deformable models. However, the cost of updating a hierarchy can be high. As a result, most algorithms for deformable models use spheres or AABBs as bounding volumes as it is relatively inexpensive to update these hierarchies. These include top-down and bottom-up techniques [LAM01, vdB97], models deformed by morphing [LAM03], and a sub-linear algorithm for deformable models expressed as linear superposition of precomputed displacements [JP04]. However, these hierarchies may not be able to perform significant culling and can result in a high number of false positives. As a result, they may not be able to check for collisions at interactive rates among complex deformable models.

Many specialized algorithms have been proposed for collision detection between cloth-like models. These include exact and approximate approaches. The exact algorithms use curvature and convexity properties to check for continuous self-collisions between highly tessellated surfaces [MKE03, Pro97, VT94]. The continuous self-collision test can also be applied in a hierarchical manner on large models, though it can be expensive for interactive applications [VT00]. Given the complexity of performing continuous collision detection, esp. self-collision, some interactive algorithms either do not check for self-collisions [CMT02, FGL03] or perform approximate collision detection using multiple layers [CMT02, KC02] or voxelized grids [MDDB00]. However, it may be difficult to give any bounds on the accuracy of the resulting simulation.

2.3. GPU-based Collision Detection Algorithms

Graphics processors (GPUs) have been used to perform interference and proximity computations between rigid and deformable models [HTG03, KP03, GLM05, RMS92]. These approaches are directly applicable to deformable models

as they do not involve pre-processing. The underlying algorithms exploit the computation power of rasterization hardware and are able to achieve interactive performance on complex simulations [HZLM01, VSC01, BW02, HTG03, GRLM03, GLM05]. Moreover, these algorithms are used to check for inter-object collisions between disjoint objects, as well as intra-object collisions (or self-collisions) within each object [GLM05, GKJ*05, HTG04].

Most GPU-based collision detection algorithms have been limited to check for collisions at discrete time instances. Recently, a few GPU-based methods have been used for CCD as well. Redon et al. [RKLM04a, RKLM04b] used the CULLIDE algorithm to perform continuous collision detection for articulated models and avatars. This algorithm models the motion of each link using a line swept-sphere volume and may not extend to general deformable models. Govindaraju et al. [GKJ*05] used a precomputed chromatic decomposition for CCD. However, this algorithm is only able to check for collisions at $1-2$ frames a second on complex cloth models and may not be fast enough for VR applications. Our new algorithm also uses a precomputed chromatic decomposition, but its classification techniques help to improve the overall performance, making it better suited for interactive VR applications (see Section 6.3).

3. Overview

In this section, we give an overview of our continuous collision detection (CCD) algorithm. We first formulate the problem of CCD among general deformable models. We present a novel classification algorithm to perform fast continuous collision culling between general deformable models.

3.1. Problem Formulation

Given a triangulated mesh M with a fixed mesh connectivity, we assume that the underlying simulator or tracking device specifies the position of the mesh triangles at discrete time instances. The continuous motion between two successive time instances is modeled using a piecewise linear motion of the vertices of the mesh. The continuous motion generates a swept volume for each primitive p (e.g. a triangle), and we represent the bounding swept volume of p using the symbol P. The problem of CCD computes the first time of contact among the swept volumes of the primitives between the two discrete time instances and is performed using elementary tests such as vertex-face and edge-edge tests [Pro97, BFA02].

3.2. Continuous Collision Culling

Our goal is to quickly compute a compact potential colliding set (PCS) of triangles in close-proximity using a broad-phase algorithm. The proximity among primitives is computed using their bounding swept volumes of primitives between the discrete time instances. We then perform exact overlap tests among the PCS using a narrow-phase algorithm. We use tight bounding swept volumes to achieve high culling efficiency. A tight bounding swept volume of a triangle is the union of bounding swept volumes of its edges. Each edge is tightly bounded using a tetrahedron defined by the edge vertices in the current time instance and the next time instance.

Given a connected mesh M with primitives $p_i, i = 1, \ldots, n$, we analyze the cases where penetrations occur between the bounding swept volumes of primitives. Our analysis is based on the mesh connectivity and we use the following classification for collision culling.

- **Vertex-adjacent primitives** share one common vertex. The bounding swept volumes of these primitives touch each other. For these primitives, we only need to perform one edge-edge elementary test for penetrations. As the number of such tests among all vertex-adjacent primitives is small, broad-phase culling may not be useful for vertex-adjacent primitives.
- **Edge-adjacent primitives** share a common edge. The bounding swept volumes of these primitives penetrate each other at the edge. Therefore, broad-phase culling among edge-adjacent pairs will not offer any culling.
- **Non-adjacent primitives** are neither edge-adjacent nor vertex-adjacent. Penetrations can occur among non-adjacent primitives in close-proximity.

As penetrations occur among every pair of edge-adjacent triangles, Quick-CULLIDE [GLM05] *cannot* cull triangles and will not work well for continuous collision detection. Furthermore, based on our classification, broad-phase culling is only useful for non-adjacent and edge-adjacent primitives. Therefore, we decompose the mesh into disjoint sets of non-adjacent primitives and perform collision culling among every pair of sets. Govindaraju et al. [GKJ*05] proposed chromatic decomposition to generate such disjoint sets of non-adjacent primitives. However, the resulting number of sets in chromatic decomposition is quite large to achieve interactive performance (See Figs. 1 and 2).

Given a mesh M with decomposition $S_1, \ldots, S_k$ such that $M = S_1 \cup \ldots \cup S_k$, we ensure the following key properties in the decomposition. These properties are used to reduce the number of penetration tests among non-adjacent and edge-adjacent primitives.

1. **Non-adjacency in a set:** No two triangles in the same set are adjacent to each other. This property is used to reduce penetration tests among non-adjacent primitives.
2. **Unique edge-adjacency relations among any two sets:** Every triangle in one set is edge-adjacent to at most one triangle in any other set. This property is used to reduce penetration tests among edge-adjacent primitives.

Since the penetrating contacts are tested only for non-adjacent or edge-adjacent primitives, the above two properties are sufficient to perform CCD culling tests. We compute a dual graph of the mesh and use graph coloring algorithms to decompose the mesh into disjoint sets.

Property 2 is a key property in our algorithm as we can uniquely pair edge-adjacent primitives for every two disjoint sets. The pairing eliminates overlap tests between the edge-adjacent primitives in the broad phase.

Theorem 1: Given two non-adjacent *triangles* $p_i, p_j \in S_1$ and a primitive $p_k \in S_2$, p_k is edge-adjacent to at most one of p_i and p_j.

Proof: Let us assume p_k is edge-adjacent to both p_i and p_j. This implies that p_i is adjacent to p_j. This leads to a contradiction that p_i and p_j are non-adjacent.

As a result of Theorem 1, given a pair of sets S_1 and S_2, we can uniquely compute edge-adjacent pairs of primitives $(p_1, p_2), p_1 \in S_1, p_2 \in S_2$.

For every pair of sets $S_i, S_j, j \neq i$, we treat each edge-adjacent pair of triangles between the two sets as one object, and the remaining unassigned triangles as separate objects for collision checking. The collision culling problem between any two sets reduces to N-body collision detection problem among these objects.

Our algorithm only performs inter-set collision culling. In contrast to the chromatic decomposition algorithm [GKJ*05], our algorithm does not require separate intra-set collision culling and is more efficient.

3.3. Algorithm

Our collision detection proceeds in two phases: a broad phase that computes potentially colliding primitives (PCS) that are not-adjacent, and a narrow-phase that is used to perform exact elementary tests among non-adjacent and adjacent primitives.

The broad phase of our algorithm proceeds in two stages:

1. We update an AABB hierarchy and use the hierarchy to compute the potentially colliding non-adjacent primitives.
2. We perform continuous collision culling among the objects corresponding to every pair of sets $S_1, S_2, S_1 \neq S_2$. We use the bounding swept volume representations of primitives and compute a set of potentially colliding primitives using the Quick-CULLIDE algorithm.

In the narrow phase, we first perform exact elementary tests among the potentially overlapping non-adjacent triangles. We then perform elementary tests among adjacent primitives. We further reduce the number of elementary tests if the non-adjacent edges belonging to a pair of non-adjacent primitives are not overlapping [GKJ*05].

4. Interactive Continuous Collision Detection

In this section, we present our multi-stages collision detection algorithm. We first describe the preprocessing phase followed by the runtime algorithm.

4.1. Pre-Process

Our algorithm uses an AABB hierarchy to quickly cull non-adjacent primitives that are not in close-proximity. We initially construct the AABB hierarchy using a standard top-down approach based on the mesh connectivity. Each leaf node contains a pointer to the mesh triangle and a list of adjacent primitives. The list of adjacent primitives are used to ignore overlaps between the adjacent primitives. Furthermore, each node in the hierarchy stores a pointer to an AABB. The AABB hierarchy is constructed as a pre-process and is updated at run-time for a deformable object.

During the pre-process, we decompose the mesh by applying a graph coloring algorithm to the dual graph. We compute the dual graph $G = (V, E)$ such that

- each vertex $v_i \in G$ corresponds to a primitive $p_i \in M$ and
- each edge $(v_i, v_j) \in G$ corresponds to a pair of vertex- or edge-adjacent primitives $p_i, p_j \in M$.

A graph coloring of G performs a mesh decomposition of M into independent sets and satisfies the two mesh decomposition properties (in Section 3.2). The complexity of the overall collision culling algorithm is a function of the number of independent sets generated by the graph coloring algorithm and the number of primitives in the mesh. The decomposition is performed using the DSATUR approximation algorithm [Bré79] for graph coloring.

After the mesh decomposition, our algorithm computes a data structure for each pair of sets. The data structure stores all the primitive pairs that belong to the two sets and are edge-adjacent. If a primitive is not edge-adjacent with any of the primitives in the set, we conceptually generate a primitive pair where one primitive is set to NULL. The pseudo-code for generating the pair data structure for sets S_1, S_2, where $S_1 \neq S_2$ is given in Algorithm 4.1.

4.2. Bounding Representations

We tightly enclose the linearly interpolated path of each triangle using a union of three edge-bounding-volumes, and the two discrete positions of the triangle. Our algorithm computes the tetrahedron obtained using the initial positions and the final positions of the edge as the edge bounding volume. Our bounding volume representation for the triangle is closed and tight. The bounding volume representation for a pair of triangles is the union of the bounding volumes of the two individual triangles. At run-time, our algorithm uses these *closed* bounding volume representations to perform efficient collision culling.

The bounding representation requires no additional overhead per frame as the representations are defined using the vertex indices of the primitive at the two discrete time instances. These indices are precomputed and do not change during the simulation. Therefore, we do not need to update the bounding representations at run-time.

```
GENSETTRIANGLEPAIRS()
1. For each triangle t1 in the mesh
2.       For each adjacent triangle t2 of t
3.             If EdgeAdjacent(t1,t2) and NotFind(t1,t2)
4.                   SetPairs(t1.color, t2.color).AddPair(t1,t2)
5.       End For
6.  End For

7. For each color c1 = 1,...,numcols
8.       For each color c2 = c1 + 1,...,numcols
9.             For each triangle t in the mesh
10.                  If t.color = c1 or c2 and NotInSetPair(t,c1,c2)
11.                        SetPairs(c1,c2).AddPair(t1,NULL)
12.            End For
13.       End For
14.  End For
```

ALGORITHM 4.1: *Primitive Pair Generation Algorithm: The data structure SetPairs stores the primitive pairs for each pair of sets. Each set is assigned a unique integer color that varies from* 1 *to numcols (line 7). Lines 1-6 compute the edge-adjacent pairs for each pair of sets. Line 3 checks if the primitives are edge-adjacent, and whether the primitive pair already exists in the data structure. Lines 7-14 compute the unassigned primitive pairs in each set and add them to the data structure (line 11).*

4.3. Run-time Algorithm

Our run-time algorithm consists of four stages:

- **Stage I:** We use the AABB hierarchy to compute triangles that potentially collide with other non-adjacent triangles. During each simulation time step, we update the AABB hierarchy using the mesh vertices of the two discrete instances. We traverse the hierarchy and test whether the AABBs of the triangles overlap with any of the non-adjacent triangles.
- **Stage II:** We perform 2.5-D visibility tests among each pair of sets. We use the bounding volume representations of the primitive pairs and combine it with Quick-CULLIDE culling algorithm to perform continuous culling. We use the results of stage I to reduce the number of visibility computations. A primitive pair (t_1,t_2) that corresponds to the sets (S_1,S_2) is potentially colliding, if and only if either t_1 or t_2 is potentially colliding with any other primitive in the two sets. We use the function RENDERPAIR shown in Algorithm 4.2 to render a primitive pair in Quick-CULLIDE. Observe that only triangles that are potentially colliding after stage I are used for visibility computations.
- **Stage III:** We perform edge-edge and vertex-face elementary tests among the potentially colliding non-adjacent primitives. We only perform these elementary tests, if the bounding swept volumes of primitives that are adjacent along the corresponding edges or vertices of the overlapping primitives are not touching or have tangential contacts.

```
RENDERPAIR()
1. Given primitive pair (t1, t2)
2. If t1 is in PCS, render t1
3. If t2 is not NULL, and t2 is in PCS, render t2
```

ALGORITHM 4.2: *Primitive Pair Rendering Algorithm: Only primitives which are in PCS after stage I are rendered (lines 2-3).*

- **Stage IV:** We perform elementary tests among adjacent primitives. Similar to stage III, we only perform elementary tests, if the bounding swept volumes of primitives that are adjacent along the corresponding edges or vertices are not touching.

5. Implementation and Results

We have implemented our algorithm on a Pentium IV PC with 2GB memory and NVIDIA GeForce 7800 GTX GPU. We use the OpenGL API under Windows XP and GL_NV_occlusion_query for performing the full visibility queries asynchronously. We improve the rendering performance by storing the mesh vertices using the GL_vertex_buffer_object extension on the GPU. We are able to achieve a throughput of 20 million triangles per second. We use an image resolution of 1000×1000 for collision computations, and used three axis-aligned views to perform collision culling.

We have applied our algorithm to detect collisions on general meshes used for cloth simulation. We have tested our algorithm on two complex benchmarks:

- **Benchmark I:** In this simulation, the cloth is modeled using 6K triangles. Our precomputation algorithm decomposes this mesh into using 11 colors as shown in Fig. 1(b). We have illustrated three sequences from this simulation in Figs. 1(e), (f), and (g). Using the stage I of our algorithm, the PCS reduces to $2000-2500$ triangles. The average update time for the AABBs is about 4ms and the average collision culling time spent in stage I is about 20ms. The stage II of our algorithm further reduces the size of the PCS to $100-200$ triangles. The average collision culling time in stage is about 70ms, and the average collision detection time is about 100ms per frame. The PCS computed after stages I and II are highlighted in red in Fig. 1 (c) and (d), respectively.
- **Benchmark II:** In this environment, the cloth wraps around a sphere. The cloth mesh is represented using $7K$ triangles, and our precomputation algorithm partitions the mesh into only 10 independent sets. Figs. 2 (e), (f), and (g) illustrates three snapshots from the simulation. The average PCS size after Stages I and II are around 3000 triangles and 200 triangles respectively. We highlight the PCS in red after the Stage I and II in Fig. 2 (c) and (d) respectively. The average collision culling time in the Stages I and II of our algorithm are 20 ms and 60 ms, respectively.

(a) Benchmark I

(b) Our Mesh Decomposition: 11 sets

(c) Chromatic Decomposition: 23 sets

(d) PCS computed (in red) after Stage I (2.5K primitives)

(e) PCS computed (in red) after Stage II (40 primitives)

(f) First snapshot of Benchmark I

(g) Second snapshot of Benchmark I

Figure 1: Benchmark I: The cloth is modeled using 7K triangles and our mesh decomposition results in 11 colors (as shown in Fig. 1 (b)). The chromatic decomposition algorithm [GKJ*05] results in 23 colors and the comparison is highlighted in Fig. (c). Using AABBs, the PCS is reduced to $2000-2500$ triangles and is highlighted in red in Fig. 1 (d). Using the Stage II of our algorithm, we further reduce the size of the PCS to 40 primitives and is shown in Fig. 1 (e). The sequence of images Figs. 1 (f), (g) show the various instances of a skirt due to the motion of the avatar. As the simulation progresses many complex folds and wrinkles arise. Our algorithm is able to detect all the collisions within 100 ms on a Pentium IV PC with a GeForce 7800 GPU.

6. Analysis

In this section, we analyze the performance of our algorithm. This includes the culling efficiency and different factors that govern the overall performance.

6.1. Performance

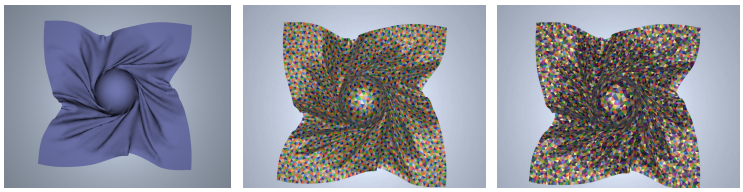
The performance of the Stage I of our algorithm is a linear function of the number of primitives in the mesh. The performance of the Stage II of our algorithm is linear to the number of the mesh primitives and the number of sets generated using the decomposition algorithm. We use the DSATUR algorithm [Bré79] to perform the mesh decomposition. In our benchmarks, we are able to partition complex cloth meshes into at most 10 independent sets. Furthermore, it may be possible to obtain optimal coloring for regular and subdivision meshes using our algorithm. Fig. 3 highlights the performance of our algorithm on benchmark I and II.

6.2. Culling Efficiency

The culling efficiency of our algorithm varies based on the relative position of the mesh primitives, and the bounding volume representations used to bound the continuous path of the primitives. The culling efficiency obtained by our algorithm is also a function of the view directions used to perform 2.5D overlap tests and the number of views. We have compared the culling efficiency obtained using stage I and stage II of our algorithm. Fig. 3(c) and 3(d) highlight the culling efficiency after each stage of our algorithm on the two benchmarks. Since the AABBs used to bound the primitive are more conservative than the tight bounding representations used in Stage II, our algorithm is able to reduce the PCS size after Stage II by $1-2$ orders of magnitude in comparison to Stage I.

6.3. Comparison

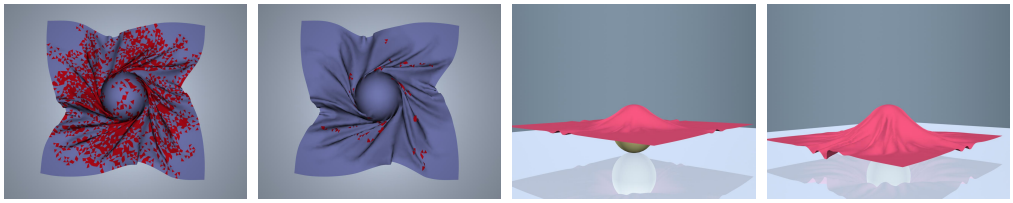
Our algorithm performs continuous collision detection between general deformable models at image-space precision. Our algorithm checks for collision detection between the time steps whereas Quick-CULLIDE [GLM05] checks for collisions only at discrete time instances. Due to the use of bounding swept volumes, the accuracy of our algorithm is always higher than Quick-CULLIDE. As compared to the chromatic decomposition algorithm [GKJ*05], our algorithm is relatively faster by almost 6 times. This is mainly due to the improved chromatic decomposition algorithm and



(a) Benchmark II

(b) Our Mesh Decomposition: 11 sets

(c) Chromatic Decomposition: 24 sets



(d) PCS (in red) computed after Stage I (1600 triangles)

(e) PCS (in red) computed after Stage II (30 triangles)

(f) First snapshot

(g) Second snapshot

Figure 2: Benchmark II: In this simulation, a cloth modeled using 6K triangles falls and drapes around a sphere. Our mesh decomposition algorithm partitions the mesh into 11 independent sets as shown in Fig. 2 (b). We show a comparison with the chromatic decomposition algorithm in Fig. 2 (c). Using the AABBs in the Stage I of our algorithm, the size of PCS is reduced to 1600 triangles. The Stage II of our algorithm further reduces the size of the PCS to 30 triangles. The PCS obtained after Stages I and II are highlighted in red in Fig. 2 (d) and (e) respectively. We illustrate the various instances of the simulation in Fig. 2 (f), (g) respectively. The average collision detection time is 80 ms.

edge-adjacent pair computation (as highlighted in Figs. 1 and 2). However, our algorithm performs collision queries at image-space precision, whereas the chromatic decomposition performs them at object-space precision. Furthermore, our algorithm does not check for touching contacts. Most of the object-space algorithms based on bounding volume hierarchies have been applied to relatively simpler benchmarks and are slower than our algorithm.

6.4. Limitations

Our approach has some limitations. First of all, the collision queries are performed at image-space resolution. Secondly, our algorithm does not detect all types of contacts, e.g. touching contacts. Furthermore, the performance of our algorithm varies as a function of the time step. For a large time step, the culling efficiency of the algorithm goes down and it performs a higher number of elementary tests.

7. Conclusions and Future Work

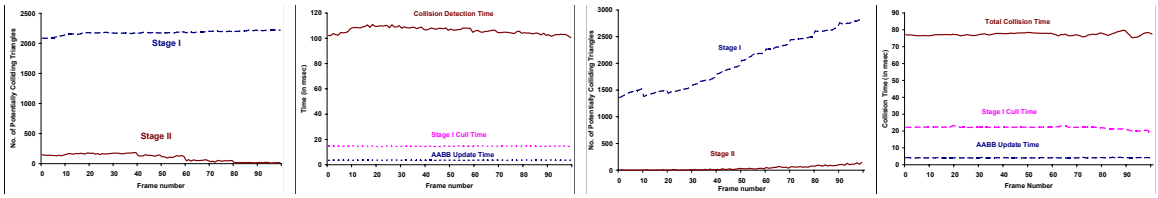
We have presented an interactive continuous collision detection algorithm for general deformable meshes in virtual environments. We decompose the problem into adjacent and non-adjacent collision detection. The algorithm takes advantages of primitive connectivity information to perform efficient collision culling between bounding swept volumes to ensure no collision is missed between time steps. The collision culling is performed using 2.5D queries on the GPUs. It is able to check for collisions, including self-collisions, in complex simulations consisting of many thousands of triangles. There are many avenues for future work. We would like to use this approach for other application including avatar motion and surgical simulation. Furthermore, we would like to perform other proximity queries including separation distance and penetration depth computation.

Acknowledgments

This research is supported in part by ARO Contract DAAD 19-02-1-0390, and W911NF-04-1-0088, NSF Awards 0400134, 0118743, DARPA and RDECOM Contract N61339-04-C-0043 and Intel Corporation. We thank the UNC GAMMA group for many useful discussions and support. We are also grateful to the reviewers for their feedback.

References

[ABG*00] AGARWAL P. K., BASCH J., GUIBAS L. J., HERSHBERGER J., ZHANG L.: Deformable free space tiling for kinetic collision detection. To appear. 2



(a) PCS size in Benchmark I

(b) Collision time in Benchmark I

(c) PCS size in Benchmark II

(d) Collision time in Benchmark II

Figure 3: Culling Efficiency and Performance on Benchmark I: We highlight the amount of culling efficiency obtained in each of the stages in Fig. 3(a) and 3(c). The graphs indicate over an order of magnitude improvement in culling efficiency using Stage II over Stage I. Fig. 3(b) and Fig. 3(d) highlight the time taken to update the AABB hierarchy, the collision time in Stage I and the total collision time. The average collision time is around 100 msec for Benchmark I and 90 msec for Benchmark II.

[AMBJ02] ABDEL-MALEKL K., BLACKMORE D., JOY K.: Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling* (2002). 2

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treament for collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH* (2002), 594–603. 2, 3

[Bré79] BRÉLAZ D.: New methods to color the vertices of a graph. *Communications of the ACM 22*, 4 (1979), 251–256. 4, 6

[BW02] BACIU G., WONG S.: Image-based techniques in a hybrid collision detector. *IEEE Trans. on Visualization and Computer Graphics* (2002). 3

[Can86] CANNY J. F.: Collision detection for moving polyhedra. *IEEE Trans. PAMI 8* (1986), 200–209. 1, 2

[CMT02] CORDIER F., MAGNENAT-THALMANN N.: Real-time animation of dressed virtual humans. *Computer Graphics Forum 21*, 3 (2002), 327–335. 2

[DKT98] DEROSE T., KASS M., TRUONG T.: Subdivision surfaces in character animation. *Proc. of ACM SIGGRAPH* (1998), 85–94. 2

[Eri04] ERICSON C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2004. 2

[FGL03] FUHRMANN A., GROSS C., LUCKAS V.: Interactive animation of cloth including self collision detection. *Journal of WSCG 11*, 1 (2003), 203–208. 2

[GKJ*05] GOVINDARAJU N., KNOTT D., JAIN N., KABAL I., TAMSTORF R., GAYLE R., LIN M., MANOCHA D.: Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)* (2005). 1, 2, 3, 4, 6

[GLM05] GOVINDARAJU N., LIN M., MANOCHA D.: Quick-CULLIDE: Efficient inter- and intra- object collision culling using GPUs. *Proc. of IEEE VR* (2005), 59–66. 2, 3, 6

[GRLM03] GOVINDARAJU N., REDON S., LIN M., MANOCHA D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), 25–32. 3

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M.: Real-time volumetric intersections of deforming objects. *Proc. of Vision, Modeling and Visualization* (2003), 461–468. 2, 3

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3 (2004), 145–152. 3

[HZLM01] HOFF K., ZAFERAKIS A., LIN M., MANOCHA D.: Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics* (2001), 145–148. 3

[JP04] JAMES D. L., PAI D. K.: BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH* (2004), 393–398. 2

[KC02] KANG Y., CHO H.: Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. *Computer Animation* (2002), 604–611. 2

[KP03] KNOTT D., PAI D. K.: CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface* (2003), 73–80. 2

[KR03] KIM B., ROSSIGNAC J.: Collision prediction for polyhedra under screw motion. *Symposium on Solid Modeling and Applications* (2003). 1, 2

[LAM01] LARSSON T., AKENINE-MÖLLER T.: Collision detection for continuously deforming bodies. In *Eurographics* (2001), pp. 325–333. 2

[LAM03] LARSSON T., AKENINE-MÖLLER T.: Efficient collision detection for models deformed by morphing. *Visual Computer 19* (2003), 164–174. 2

[LM03] LIN M., MANOCHA D.: Collision and proximity queries. In *Handbook of Discrete and Computational Geometry* (2003). 2

[MDDB00] MEYER M., DEBUNNE G., DESBRUN M., BARR A.: Interactive animation of cloth like objects in virtual reality. *Journal of Visualization and Computer Animation 12*, 1 (2000), 1–12. 2

[MKE03] MEZGER J., KIMMERLE S., ETZMUβ O.: Hierarchical techniques in cloth detection for cloth animation. *Journal of WSCG 11*, 1 (2003), 322–329. 2

[PBS02] PELECHANO N., BULL L., SLATER M.: *Fast collision detection between cloth and a deformable human body*. Tech. Rep. Technical Report, Department of Computer Science, University College London, 2002. 2

[Pro97] PROVOT X.: Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface* (1997), 177–189. 2, 3

[RKC00] REDON S., KHEDDAR A., COQUILLART S.: An algebraic solution to the problem of collision detection for rigid polyhedral objects. *Proc. of IEEE Conference on Robotics and Automation* (2000). 1, 2

[RKC02] REDON S., KHEDDAR A., COQUILLART S.: Fast continuous collision detection between rigid bodies. *Proc. of Eurographics (Computer Graphics Forum)* (2002). 2

[RKLM04a] REDON S., KIM Y. J., LIN M. C., MANOCHA D.: Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications* (2004). 1, 2, 3

[RKLM04b] REDON S., KIM Y. J., LIN M. C., MANOCHA D.: Interactive and continuous collision detection for avatars in virtual environments. In *Proceedings of IEEE VR Conference* (2004). 1, 2, 3

[RMS92] ROSSIGNAC J., MEGAHED A., SCHNEIDER B.: Interactive inspection of solids: cross-sections and interferences. In *Proceedings of ACM SIGGRAPH* (1992), pp. 353–60. 2

[SSL02] SCHWARZER F., SAHA M., LATOMBE J.-C.: Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)* (Dec. 2002). 2

[TMH*05] TESCHNER M., MANOCHA D., HEIDELBERGER B., GOVINDARAJU N., ZACHMMAN G., MEZGER J., FUHRMANN A.: Collision handling in dynamic simulation environments. *Tutorial, Eurographics* (2005). 2

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4 (1997), 1–14. 2

[VSC01] VASSILEV T., SPANLANG B., CHRYSANTHOU Y.: Fast cloth animation on walking avatars. *Computer Graphics Forum (Proc. of Eurographics'01) 20*, 3 (2001), 260–267. 3

[VT94] VOLINO P., THALMANN N. M.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.) 13*, 3 (1994), 155–166. 2

[VT00] VOLINO P., THALMANN N. M.: Accurate collision response on polygon meshes. *Computer Animation* (2000), 154. 2