Multi-Core Collision Detection between Deformable Models

Min Tang
Zhejiang University
tang_m@zju.edu.cn

Dinesh Manocha
University of North Carolina at Chapel Hill
dm@cs.unc.edu

Ruofeng Tong
Zhejiang University
trf@zju.edu.cn

ABSTRACT
We present a new parallel algorithm for interactive and continuous collision detection between deformable models. Our algorithm performs incremental hierarchical computations between successive frames and parallelizes the computation among multiple cores on current CPUs. The main computations include front building and updating and performing the elementary tests between the triangle primitives. The overall algorithm can perform inter- and intra-object collisions at interactive rates on current commodity processors on models with many tens of thousands of triangles. In practice, the performance of our algorithm almost scales linearly with the number of cores.

Keywords
Continuous collision detection, deformable models, multi-core processor, parallel collision detection

1. INTRODUCTION
Continuous collision detection(CCD) is an essential component for physically-based high precision simulations, virtual prototyping, and robotics. One of the main challenges is to perform CCD computations at interactive rates on complex models that are frequently used in these applications, including cloth, soft-body, virtual characters.

There is extensive work on fast algorithms for CCD computations. Most of them are based on hierarchical representations and different methods have been proposed to accelerate the computations. At a broad level, the efficiency of CCD algorithms for deformable objects is mainly governed by the following:

- **Number of Elementary tests:** For two triangles which are potentially colliding, 15 elementary tests (EE and VF) are needed to be checked to perform CCD. Each elementary test reduced to solving a cubic equation. As compared to discrete collision checking, CCD is relatively more expensive.
- **False positives:** Most hierarchical CCD algorithms result in a high number of false positives between the triangle pairs.
- **Self-collisions:** Due to the random deformation of simulation, self-collisions need to be examined to prevent "leaks", which, in many complex simulations, about 70% – 90% of running time for CCD algorithms can be spent in handling self-collisions.

In this paper, we address the problem of designing parallel algorithms that can exploit the capabilities of current multi-core CPUs to design faster algorithms. One of the challenges is to distribute the computation among multiple cores. However, due to complexity of collision algorithms, it is hard to estimate the load during the hierarchial traversal and decompose the task evenly among different cores.

Main Results:

- An incremental CCD algorithm that exploits temporal coherence between successive frames during the simulation. A "front" is maintained to record the colliding pairs computed during the pervious time step.
- We present efficient parallel techniques for front building and front updating. We also distribute the elementary tests among different cores and thereby present an overall parallel CCD computation algorithm between deformable models. We have evaluated its performance and scalability on a 16-core workstation on different benchmarks.

2. RELATED WORK
Collision detection among objects is extensively studied in computer graphics, simulation, animation and virtual reality communities, so we focus our discussions on continuous collision detection, incremental collision detection, and parallel collision detection.

2.1 Continuous Collision Detection
Continuous collision detection has become a de factor tool for high precision simulation to compute the exact contact time during simulation time interval.

It has been applied for rigid bodies and articulated models extensively.For deformable objects, collision detection is more difficult for the occurrences of self-collisions [23].

Recently many researchers have proposed algorithms to improve the efficiency of CCD among deformable objects. By using these bounding volumes, the number of false positives is reduced. Tang et al. [22] used a table-based method to remove redundant elementary tests caused by feature sharing. Curtis et al. [3] proposed the concept of representative triangle to remove these redundancies efficiently. Tang et al. [20] used continuous normal cones and orphan set to remove redundant tests between non-adjacent triangle pairs and adjacent triangle pairs. By using continuous normal cones, triangle meshes with relatively flat area are skipped from self-collision detection. With orphan set, almost all the elementary tests involved by adjacent triangles are avoided [20].

2.2 Incremental Collision Detection

There is considerable prior work on exploiting temporal coherence to accelerate the process of collision detection. Lin and Canny [13] presented an incremental algorithm for Voronoi diagram based data structures. Ponamgi et al. [15, 16] extended this algorithm to general B-rep solid models and polygonal models. Klowoski et al. [10] proposed the concept of front tracking, to represent a temporal coherence data structure for BVH with k-DOPs. The front also be used for the sphere-based BVH [12] and for a convex hull based BVH [4]. Continuing to these works, Tropp et al. [26] designed an incremental algorithm for BVHs based on OBBs and AABBs. Although good speedup gains, all these works are limited to rigid objects under continuous motions.

2.3 Parallel Collision Detection

The detection of collisions is equal to traversing BVTT in depth-first manner. Many researchers designed parallel algorithm to accelerate depth-first traversal of BVTT. Rao and Kumar [17] pointed out the efficiency of parallel depth-first search is strongly influenced by the work distribution scheme and architectural features. Kumar and Grama [11] analyzed the scalability of several load balancing techniques for difference architectures. Reinefeld and Schnecke [18] compared load balancing strategies of two depth-first search methods and propose a scheme that uses fine-grained fixed-sized work packets. Kitamura et al. [8] used static and dynamic load balancing methods to acceleration colliding detection respectively, and finds out the communication between processers in the main cause for limited performance. Assarsson and Stenström [1] achieved three times speedup on eight processors for collision detection of some industrially related test scenes. It shows the superiority of applying some straightforward tricks to design a lock-free load distribution algorithm. Grinberg and Wiseman [6] proposed a method to extract parallelism by statically pre-processed task partition. Chen et al. [2] used tera-scale processors to accelerate off-line physical simulation applications. Thomaszewski et al. [24] and Selle et al. [19] accelerated collision detection for cloth simulation on distributed memory architectures. Thomaszewski et al. [25] used multi-threaded programming to accelerate the implicit time integration and collision handling for cloth-simulation. A stack is used to hold sub-tasks during the traversal of BVTT, and frame coherence is used to estimate load of sub-tasks. In parallel to our work, a self-collision detection based task decomposition has been used in [7] to map the computation to multi-core processors.

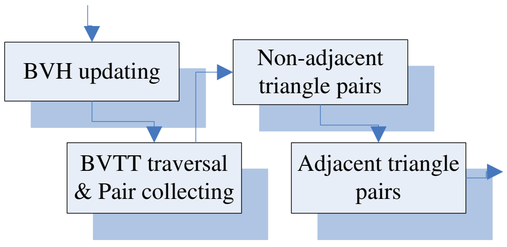

Figure 1: Running flowchart: The flowchart of [20] can be divided into 4 stages: BVH updating, BVTT traversal and pair collecting, Non-adjacent pairs processing, and adjacent pairs processing.

3. OVERVIEW

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

3.1 Notations

BVH: Bounding volume hierarchies (BVHs) are used to accelerate queries. We represent the scene containing deformable models by using a signle BVH, and perform intersection test from the root node of the BVH. our algorithm is independent to the shape of bounding volume. It can be axis aligned boxes (AABB) [27], k-DOPs [10], oriented bounding boxes(OBB) [5], sphere [14], etc.

BVTT: A bounding volume test tree (BVTT) represents the hierarchy of tests performed. Each node in the BVTT corresponds to a single test between a pair of BVs or a self-collision test of one BV. However, our formulation is different from prior works [12, 4, 26, 28], and we also integrate self-collisions in this formulation, so the overall BVTT is no longer a binary tree. All the adjacent triangle pairs are not recorded in the leaf nodes of the BVTT. Rather they are processed separately by using an orphan set.

Front: A front of the BVTT is a set of tree nodes where the traversal terminates during a given frame. It reflects collision occurrence at a given simulation time step.

3.2 Efficiency Bottlenecks

A key to designing a good parallel algorithm is to parallel the components where most of the running time in the serial algorithm is spent. In order to identify the bottlenecks in a serial algorithm, we analyze the time breakdown during various stages of the single-threaded implementation in [20].

As shown in Fig. 1, the overall CCD algorithm for deformable models can be divided into 4 stages:

- Updating BVH: recompute the bounding volumes for features (faces, edges, and vertices), and update the BVH structure to reflect model deformations.
- BVTT traversal & pair collecting: traverse the BVTT in depth-first manner, performing bounding volume

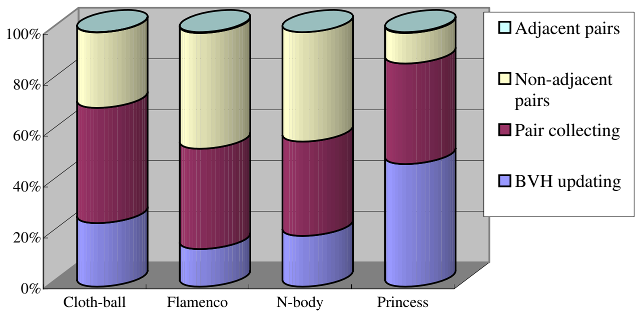

Figure 2: Running time of each component: This figure shows the running time ratio of each stage on several benchmarks.

tests, and collect potential colliding non-adjacent triangle pairs.

- Non-adjacent pair processing: All these potential colliding pairs collected are tested for overlap by executing the elementary tests on these pairs.
- Adjacent triangle pairs: By testing the precalculated "Orphan Set" [20], all elementary tests associated with adjacent triangle pairs are performed.

Based on experiments of several benchmarks, the ratios of running time of each stage are shown in Fig. 2.

As shown by Fig. 2, by using the "Orphan Set", the running time spent on adjacent pairs is negligible. Its ratios are below 0.4% in all our benchmarks. As a result, we mainly focus on developing good parallel techniques for the other three stages. In practices, developing parallel algorithms for BVH updating and non-adjacent pair processing are relatively easy. The main challenge is parallelization of BVTT traversal & pair collecting stage.

3.3 Our Approach

Instead of using the 4-stage algorithm described above, we first present an incremental CCD algorithm for deformable objects, which rather consists of three stages. Based on this incremental formulation, we propose a scheme to parallelize it effectively on multi-core architectures .

4. INCREMENTAL CCD FOR DEFORMABLE MODELS

We present an incremental algorithm that exploits temporal coherence between successive frames of a simulation. The main idea is to maintain a front which records collision information at the last simulation time step, and we update that information to check for collisions during the current frame.

4.1 Orphan Set based Front Reduction

In order to record the information about collisions at the last simulation time step, we store three kinds of node within front during at the traversal of BVTT: $\{N_i, M_j\}$, $\{N_i, L_j\}$, and $\{L_i, L_j\}$. Here, N_i and M_j stand for the internal nodes of the global BVH, L_i and L_j stand for leaf nodes of the global BVH. In $\{N_i, M_j\}$ and $\{N_i, L_j\}$, their nodes' bounding volumes do not overlap, so the tree traversal terminates at these nodes. $\{L_i, L_j\}$ represent the leaf node pairs.

All the potentially colliding triangle pairs that satisfy bounding volume tests can be classified into two categories based on their connectivity: adjacent triangle pairs and non-adjacent triangle pairs [20]. Moreover, all the non-adjacent triangle pairs are skipped during the BVTT traversal phase, and are processed separately by using "Orphan set" formulation.

We store the non-adjacent pairs in a front. All the adjacent pairs are removed from the front and processed separately. By deleting the node pairs containing adjacent triangle pairs, the size of the front can reduce by $60\% - 80\%$.

4.2 Front Building

The initial front is built by traversing the BVTT in top-down manner. The algorithm performs self-collision checking from the root node of the BVH . During the traversal of the BVTT, all leaf node pairs that contain the non-adjacent triangle pairs are recorded into the front. Given any internal node of the BVTT, if their bounding volumes do not overlap, they are also recorded into the front. This step is applied recursively till the traversal terminates.

4.3 Front Updating

As the objects deform during the simulation, the front needs to be updated accordingly. At the next simulation time step, all the colliding pairs are recollected by following rules:

- All $\{L_i, L_j\}$ pairs are tested again;
- All $\{N_i, M_j\}$ pairs are tested again. If the bounding volumes do not overlap, they can be skipped. Otherwise, the algorithm checks their children for overlap;
- $\{N_i, L_j\}$ is tested again. If the bounding volumes do not overlap, the traversal can be skipped. Otherwise, their children are checked for overlap.

In order to maintain a valid front, two operator "sprouting" and "pruning" are used in [12, 4, 26]. The front is sprouted to the level at which the bounding volumes do not overlap or the traversal reaches the leaf nodes. We do not used the pruning operator, since it is a relatively expensive operation. Only sprouting operator is used when scanning the front. If during two or three consecutive simulation time steps the front does not increase in length, this indicates that the quality of the front is decreasing, and we rebuilt a front from scratch.

So we are using a deferred rebuilding strategy to maintain the front. The front is updated using sprouting operator for several consecutive simulation time steps, then it will be rebuild.

4.4 Benefits

By using the incremental algorithm based on the front, the collisions are detected starting from the front, and these is no need to traversal the BVTT again. So the costs of full

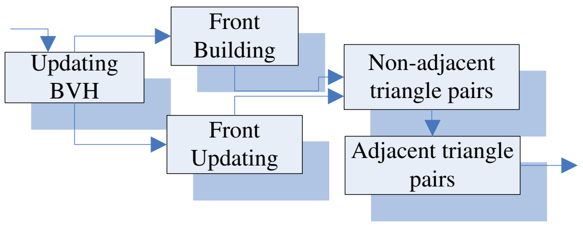

Figure 3: Running stages of the incremental CCD algorithm: BVH updating, Front building, Front updating, Non-adjacent pairs processing, and adjacent pairs processing.

BVTT traversal and its associated bounding volume tests are saved, and the overall performance improves. The numbers of bounding volume test are reduced by $53\% - 63\%$ compared to the 4-stage algorithm, and the overall running time are reduced by $68\% - 88\%$.

Now the overall flowchart is composed by five stages, as shown in Fig. 3. The stage named as *BVTT traversal & pair collecting* in Fig. 1 now is replaced by two stages: *Front updating* and *Front building*.

5. PARALLEL CONTINUOUS COLLISION DETECTION

5.1 Parallelism Analysis

For all the stages in Fig. 3, BVH updating and non-adjacent triangle pairs processing is relatively easy to parallelize on multiple cores. Adjacent triangle pairs processing only takes a small fraction of the total running time. The main bottlenecks in terms of a good parallel algorithm are the two stages: front building and front updating.

5.2 Parallel Front Building

The front computation is performed during the depth-first traversal of the BVTT. We use a list to buffer the collision tasks and self-collision tasks during the BVTT traversal. Later, these tasks are executed in parallel. As shown in Fig. 4, the front building is decomposed into a serial of inter-object collision and self-collision tasks.

For all these tasks, we use the number of bounding volume tests as a metric to estimate its cost. For the self-collision tasks, this number remains unchanged during simulation. In order to perform collisions between the BVH nodes, we used the method in [9] to evaluate the cost dynamically.

Base on the metric, we decide whether a task need to be decomposed further. And all such tasks are executed in parallel.

5.3 Parallel Front Updating

All the node pairs in the front are updated in parallel according to their categorization: $\{N_i, M_j\}$, $\{N_i, L_j\}$, and $\{L_i, L_j\}$. For each category, the update cost of node pairs are approximately equal, and thereby the algorithm achieves good scalable performance.

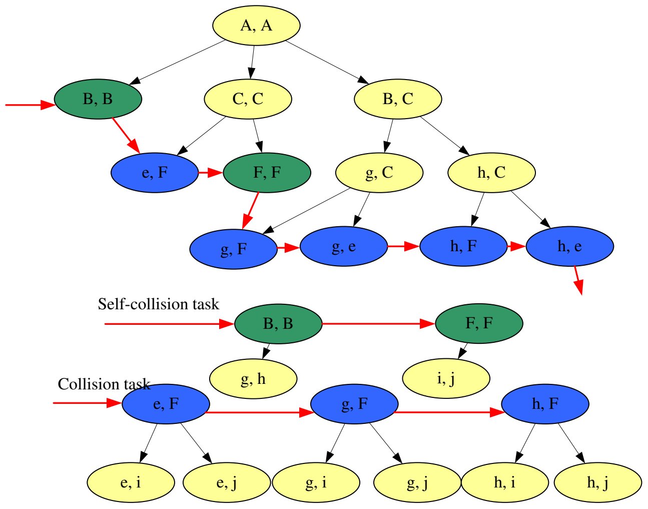

Figure 4: Task decomposition of front building: The collision tasks (blue nodes) and self-collision tasks (green nodes) are buffered in a list (red line). Then these tasks are executed in parallel.

6. IMPLEMENTATION AND PERFORMANCE

6.1 Experimental Platform

In this section, we describe our implementation and highlight the performance of our algorithm on many benchmarks. We have implemented our algorithm on a Intel Xeon machine with 16 cores (X7350 at 2.93 GHz) and 16 GB of RAM using Visual Studio 2005. OpenMP is used as API for multi-threaded programming. We use k-DOPs (specifically 18-DOPs) as bounding volumes.

Five benchmarks described in [21] are used: Cloth-ball(92K), N-body(34K), Princess(40K), BART(4K), and Flamenco(49K).

In Cloth-ball, Princess, and Flamenco benchmarks, there are a lot of self-collisions. In the N-body and BART benchmarks, only inter-object collisions exist.

All the benchmarks have multiple simulation steps. We perform CCD between each discrete steps and compute the first time-of-contact between those discrete positions.

6.2 Performance

We get good acceleration on the benchmarks. In our parallel implementation, we only build the front from scratch during the first time step. During the subsequent simulation time steps, we only update the front literately. So the overall performance is governed by the following three stages: BVH updating, front updating, and non-adjacent pairs processing. For all the benchmarks, the overall running time of our parallel algorithm is reduced by $76\% - 82\%$ by fully utilizing the computation ability of 16-cores. The sub-linear acceleration rate is a side prove of good scalability.

The speed-up of our parallel algorithm on the cloth-ball benchmark is shown in Table. 1. The running time ratios of all the stages decrease as the number of core increase. For all the benchmarks, we get $4X - 6X$ speedups by using 8 cores, and get $5X - 8X$ speedups by using 16 cores.

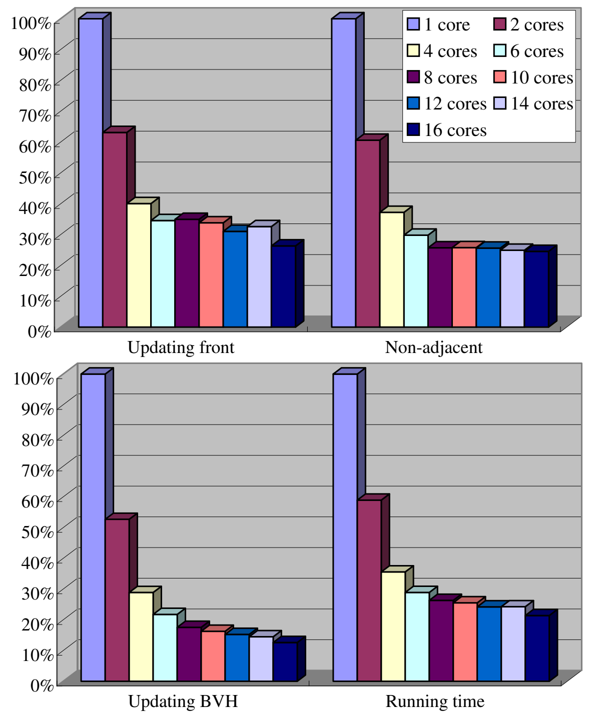

Figure 5: Parallel speed-up of the cloth-ball benchmark: When the core number is increasing, execution time of all stages are reducing. And the overall running time is reduced to 21.4% when all the 16 cores are used. It takes $73ms$ per frame to compute all the collisions.

7. COMPARISON AND LIMITATIONS

Comparison: As the kernel of an incremental algorithm, our front is similar to the notion of separate list described in [28].In terms of processing the self-collisions, Weller and Zachmann [28] skip all the adjacent triangle pairs while our algorithm processes them in a uniform frame work. This is critical for high-precision simulations. By processing the self-collisions efficiently, our algorithm offers improved performance over prior algorithms [24, 25, 19].

Comparing to prior parallel collision detection algorithms, such as [6, 2, 25], our algorithm avoids the difficulties of dynamic task partitioning on BVTT. The benefits of using a front-based approach aries due to two reasons. On one hand, the running time of bounding volume test and BVH traversal are reduced. Moreover, we obtain good parallel performance in terms of front building and front updating.

Limitations: As an incremental algorithm, the memory overhead can be high due to explicitly maintaining the front. As the number of colliding pairs increase, the length of the front are grow significantly. Moreover, we do not observe a linear speedup as a function of the number of cores in our benchmarks.

8. CONCLUSION AND FUTURE WORK

In this paper, we proposed an incremental CCD algorithm for deformable models by integrating the concept of front. Then based on it, parallelism is extracted for each stage of the incremental algorithm. And good parallel CCD algorithm is designed. We observed sub-linear acceleration rate on a serial of practical benchmarks.

Cores	Updating front	Non-adjacent pairs	Updating BVH	Running time
1 core	100%	100%	100%	100%
2 cores	64.42%	69.45%	57.99%	60.55%
4 cores	40.98%	52.85%	33.86%	37.34%
6 cores	35.25%	43.49%	24.09%	28.46%
8 cores	31.16%	38.43%	19.64%	24.06%
10 cores	31.75%	40.41%	17.33%	22.73%
12 cores	32.26%	40.34%	15.09%	21.27%
14 cores	31.12%	40.73%	13.63%	20.04%
16 cores	28.74%	40.46%	12.78%	18.94%

Table 1: Running time of each stage of the Princess benchmark: By utilizing all the 16 cores, sub-linear acceleration is achieved on the overall running time.

There are many avenues for future work. First, we only build the front once at the beginning of simulation. It will be interesting to rebuild the front after K frames during the process of simulation. In this way, the quality of front can be improved at cost of front buildings, which can be amortized over multiple frames. Second, we would like to devise methods to reduce the memory overhead.

As an extension of this work, we have used a front-based task decomposition(FBD) [21] to design an improved continuous collision detection algorithm between deformable objects, and achieve $7X$ and $13X$ speedups on 8 cores and 16 cores respectively.

Acknowledgments

We thank Sung-Eui Yoon, Sean Curtis, Rasmus Tamstorf, Naga Govindaraju, Avneesh Sud, Russ Gayle and Ming Lin for useful discussions and the benchmarks.

This research is supported in part by National Basic Research Program of China (No. 2006CB303106), MOE-INTEL special research fund on Information Technology (No. MOE-INTEL-09-05), ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043, Disney and Intel. Tang is supported in part by National Natural Science Foundation of China (No. 60803054), National Key Technology R&D Program, China (No. 2006BAF01A45-05), Natural Science Foundation of Zhejiang, China (No. Y107403), Doctoral subject special scientific research fund of Education Ministry of China (No. 20070335074).

9. REFERENCES

[1] U. Assarsson and P. Stenström. A case study of load distribution in parallel view frustum culling and collision detection. In *Lecture Notes in Computer Science*, page 663, 2001.

[2] Y.-K. Chen, J. Chhugani, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. Lin, A. D. Nguyen, E. Sifakis, and M. Smelyanskiy. High-performance physical simulations on next-generation architecture with many cores. *Intel Technology Journal*, 11(3), 2007.

[3] S. Curtis, R. Tamstorf, and D. Manocha. Fast collision detection for deformable models using representative-triangles. In *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D graphics and games*, pages 61–69, 2008.

[4] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *of Eurographicsąŕ2001*, pages 500–510, 2001.

[5] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, NY, USA, 1996. ACM.

[6] I. Grinberg and Y. Wiseman. Scalable parallel collision detection simulation. *Proc. of Signal and Image Processing*, 2007.

[7] D. Kim, J. Heo, and S. Yoon. PCCD: Parallel continuous collision detection. Technical report, http://sglab.kaist.ac.kr/PCCD/, Korea Advanced Institute of Science and Technology, South Korea, 2008.

[8] Y. Kitamura, A. Smith, H. Takemura, and F. Kishino. Parallel algorithms for real-time colliding face detection. *Robot and Human Communication, 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE International Workshop on*, pages 211–218, Jul 1995.

[9] J. Klein and G. Zachmann. The expected running time of hierarchical collision detection. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*, page 117, New York, NY, USA, 2005. ACM.

[10] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Visualization and Computer Graphics*, 4(1):21–37, 1998.

[11] V. Kumar and A. Y. Grama. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22:60–79, 1994.

[12] T.-Y. Li and J.-S. Chen. Incremental 3d collision detection with hierarchical data structures. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–144, New York, NY, USA, 1998. ACM.

[13] M. Lin and J. Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 2:1008–1014, 1991.

[14] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[15] M. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between solid models. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pages 293–304, New York, NY, USA, 1995. ACM.

[16] M. K. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64, 1997.

[17] V. N. Rao and V. Kumar. Parallel depth-first search, part i: Implementation. *International Journal of Parallel Programming*, 16:6–479, 1987.

[18] A. Reinefeld and V. Schnecke. Work-load balancing in highly parallel depth-first search. In *In Scalable High Performance Computing Conference*, pages 773–780, 1994.

[19] A. Selle, J. Su, G. Irving, and R. Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Transactions on Visualization and Computer Graphics*, 99(2), 2008.

[20] M. Tang, S. Curtis, S.-E. Yoon, and D. Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 25–36, New York, NY, USA, 2008. ACM.

[21] M. Tang, D. Manocha, and R. Tong. MCCD: Multi-core collision detection between deformable models using front-based decomposition. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2009.

[22] M. Tang, S. Yoon, and D. Manocha. Adjacency-based culling for continuous collision detection. *The Visual Computer, Proc. of CGI08 (Computer Graphics International 2008)*, 24(7-9):545–553, 2008.

[23] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects. In *Eurographics State-of-the-Art Report (EG-STAR)*, pages 119–139. Eurographics Association, Eurographics Association, 2004.

[24] B. Thomaszewski and W. Blochinger. Physically based simulation of cloth on distributed memory architectures. *Parallel Comput.*, 33(6):377–390, 2007.

[25] B. Thomaszewski, S. Pabst, and W. Blochinger. Special section: Parallel graphics and visualization: Parallel techniques for physically based simulation on multi-core processor architectures. *Comput. Graph.*, 32(1):25–40, 2008.

[26] O. Tropp, A. Tal, I. Shimshoni, and D. P. Dobkin. Temporal coherence in bounding volume hierarchies for collision detection. *International Journal of Shape Modeling*, 12(2):159–178, 2006.

[27] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[28] R. Weller and G. Zachmann. Kinetic separation lists for continuous collision detection of deformable objects. In *Third Workshop in Virtual Reality Interactions and Physical Simulation (Vriphys)*, Madrid, Spain, 6–7 November 2006.