A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures

Quentin Avril, Valérie Gouranton, Bruno Arnaldi

Proceedings of Virtual Reality International Conference (VRIC 2010), 7-9 April 2010, Laval, France.
RICHIR Simon, SHIRAI Akihiko Editors. International conference organized by Laval Virtual.

A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures

Quentin AVRIL, Valérie GOURANTON and Bruno ARNALDI
Université Européenne de Bretagne, France
INSA, INRIA, IRISA, UMR 6074, F-35043 RENNES
Email: quentin.avril, valerie.gouranton, bruno.arnaldi @irisa.fr

***Abstract*—Recent years have seen the impressive evolution of graphics hardware and processors architecture from single core to multi and many-core architectures. Confronted to this evolution, new trends in collision detection optimisation consist in proposing a solution that maps on the runtime architecture. We present, in this paper, two contributions in the field of collision detection in large-scale environments. We present a first way to parallelise, on a multi-core architecture, the initial step of the collision detection pipeline: the broad-phase. Then, we describe a new formalism of the collision detection pipeline that takes into account runtime architecture. The well-known broad-phase algorithm used is the *"Sweep and Prune"* and it has been adapted to a multi-threading use. To handle one or more thread per core, critical writing sections and threads idling must be minimised. Our model is able to work on a n-core architecture reducing computation time to detect collision between 3D objects in a large-scale environment.**

***Index Terms*—Collision Detection, Broad Phase, Multicores Architecture**

I. Introduction

Digital mock-up and industrial applications of virtual reality (VR) become more and more sized. The performance level for a real-time interaction of users into a large-scale environment is no longer satisfying. Collision detection is one of the important bottleneck of real-time VR applications from more than thirty years. Recently, researchers's interest focuses on the evolution of hardware architecture and its friendly use in order to increase computation power of collision detection algorithms. Recent papers appear dealing with a new type of problem: speeding up the detection using specificities of new hardware architectures (Multi-cores and GPU) [1], [2], [3]. In the wide range of collision detection algorithms, we can notice that recent articles have a low computational complexity and provide an efficient collision detection. But used with million of objects in a large-scale environment, they are inefficient for a real-time interaction. Today, taking the most efficient and fast collision detection algorithm and running it on a parallel architecture will definitely not insure the best performance, algorithms have to be adapted. Few years ago it was possible to lean on Gordon Moore's law that predicts multiplication of transistors number each two years. Now, we know that this multiplication is prevented by physical limits (power and heat). Trend is now in the multi-cores architecture and many-cores. Hardware architects strategy is to increase number of cores and to reduce their frequency to have a low energy consumption and heat emission. Consequently, without worrying about it, a virtual reality application that only uses one core to work will probably be slower in few years. We also know that 3D objects and virtual environments will still grew up in size and complexity, so non-parallelised applications will be less and less efficient. Hardware graphics is also subjected to an impressive power evolution. So it appears that having a different approach on the real-time collision detection problem centered on hardware performance can not be ignored. This is the reason of our revisited collision detection pipeline and, as broad-phase is in charge of breaking down the pairwise checks complexity, we have decided to focus on it for a parallelisation. In the following, section 2 deals with related works on collision detection, the architecture evolution is presented in section 3 and section 4 presents the new link-up between these two fields of study. In section 5 we present our revisited *Sweep and Prune* algorithm following by the novel view of the collision detection pipeline. In section 6 we present results of the parallelisation, then we conclude and open on future works.

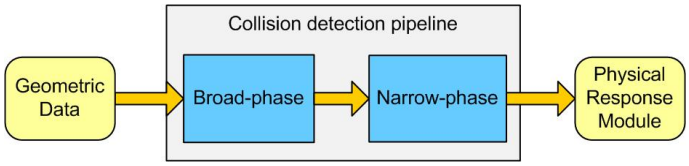

Fig. 1. Collision detection pipeline.

II. Collision Detection

Last decade have seen an impressive evolution of virtual reality applications and more precisely of collision detection algorithms in term of computational bottleneck. Collision detection is a wide field dealing with, apparently, an easy problem: determine if two (or several) objects collide. It is used in several domains namely physically-based simulation, computer animation, robotics, mechanical simulations (medical, biology, cars industry), haptics applications and

video games. All these applications have different constraints (real-time performance, efficiency and robustness) so it has generated a wide range of problems: convex or non-convex objects, 2-Body or N-Body simulations, rigid or deformable objects, continual or discrete methods. Algorithms are also dependent of the geometric model formalism (polygonal, Constructive Solid Geometry (CSG), implicit or parametric functions). All of these problems reveal the diversity of this field of study. For more details we refer to excellent surveys on the topic [4], [5], [6], [7].

Given n moving objects in a virtual environment, testing all objects pairs tend to perform n^2 pairwise checks. When n is large it becomes a computational bottleneck. Collision detection is represented and built as a pipeline (cf Figure 1) [8]. This one is composed by two main parts: broad-phase and narrow-phase. The goal of this pipeline is to apply successive filters in order to break down the $O(n^2)$ complexity. These filters provide an increasing efficiency and robustness during the pipeline traversal. In the following, we present two main parts of the pipeline, broad-phase in section 2.1 and narrow-phase in section 2.2.

A. Broad-phase

The first part of the pipeline, called the broad-phase, is in charge of a quick and efficient removal of the objects pairs that are not in collision. Broad-phase algorithms are classified into four main families [7]:

Brute force approach is based on the comparison of the overall bounding volumes of objects to determine if they are in collision or not. This test is very exhaustive because of its n^2 pairwise checks. A lot of bounding volume have been proposed such as sphere, Axis-Aligned-Bounding-Box (AABB) [9], Oriented-Bounding-Box (OBB) [10] and many others.

Spatial partitioning method is based on the principle that if two objects are situated in distant space sides, they have no chance to collide during next time step. Several methods have been proposed to divide space into unit cells: regular grid, octree [11], quad-tree, Binary Space Partitioning (BSP), k-d tree structure [12] or voxels.

Topological methods are based on the positions of objects in relation to others. A couple of objects that are too far one to the other is deleted. One of the most famous methods is called the *Sweep and Prune* [13] approach that consists in projecting objects coordinates on axis. An overlapping of objects coordinates on the three axis (x,y,z) means that their bounding volumes collide.

Kinematic approach takes care of the objects movement, if objects are moving away, they can not collide. Vanecek [14] used kinematic of objects and back-face culling technique to speed up collision detection.

B. Narrow-phase

Colliding objects pairs are then given to the narrow-phase that perform an exact collision detection. We can classify narrow-phase's algorithms into four main families [7]:

Feature-based algorithms work on objects primitives: faces, edges and vertices. This family appears in 1991 with the Lin-Canny approach [15] or *Voronoï Marching* that proposed to divide space around objects in Voronoï regions that allows detecting closest features pairs between polyhedrons.

Simplex-based algorithms of whom the most famous one is the GJK algorithm [16] that uses Minkowski difference on polyhedrons. Two convex objects collide if and only if their Minkowski difference contains the origin.

Image space-based algorithms work using image-space occlusions queries that are suitable to be used on graphics hardware (GPU). They rasterise objects to perform either 2D or 2.5D overlap test in screen space [17].

Bounding volume-based algorithms are used in most of strategies to perform collision tests and it highly improves performances. Bounding volume hierarchies (BVH) allow arranging bounding volume into a tree hierarchy (binary tree, quad tree...) in order to reduce the number of tests to perform. A description on these BVH and a comparison between their performance can be found in [7]. Deformable objects are very challenging for BVH because hierarchy structures have to be updated when an object deforms itself [9], [6].

III. Parallel Collision Detection

The parallel solution of collision detection algorithms is a well explored but still active field in high performance computing. Zachmann [18] made an evaluation of the performance of a parallelized back-end of the pipeline and showed that if the environment density is large compared to the number of processors, then good speed-ups can be noticed. Several algorithms have been proposed working on multi-processors machines [19], [20], [21], [22]. Depth-first traversal of bounding volumes tree traversal (BVTT) [23], [24] and parallel cloth simulation [25], [26] are good instances of this kind of work. Few papers also presented multi-threading use on single processor (Lewis et al. [27] propose a multithreaded algorithm to simulate planetary rings).

Very recently, few papers appeared dealing with new parallel collision detection algorithms using multi-cores. Thomaszewsk et al. [28] propose a new task splitting approach for implicit time integration and collision handling on multi-cores architecture. Tang et al. [2] propose to use a hierarchical representation to accelerate collision detection queries and an incremental algorithm exploiting temporal coherence, the overall is distributed among multiple cores. They obtained a 4X-6X speed-up on a 8-cores based on several deformable models. Kim et al [1] propose to use a feature-based bounding volume hierarchy (BVH) to improve the performance of continuous collision detection. They

also propose novel task decomposition methods for their BVH-based collision detection and dynamic task assignment methods. They obtained a 7X-8X speed-up using a 8-cores compared to a single-core. Hermann et al. [3] propose a parallelization of interactive physical simulations. They obtain a 14X-16X speed-up on a 16-cores compared to a single-core.

IV. Broad phase adaptation

We present here two contributions on the collision detection optimisation, the first one concerns the parallelisation of the *"Sweep and Prune"* algorithm and the second one describes a new way to represent the collision pipeline .

A. Parallel Algorithm

Related works let appear that the architecture of collision detection algorithms needs to be improved to face real-time interaction. In this way, we focus on an essential step of the collision detection pipeline: the broad-phase. More precisely, our algorithm is an implementation of the *"Sweep and Prune"*[13] on a multi-cores architecture.

Sweep and prune is also known as *sort and sweep* [29] being called that way at David Baraff Ph. D thesis in 1992 [30]. Later works like the 1995 paper about I-COLLIDE by Cohen et al. [13] refer to this algorithm. It is one of the most used methods in the broad-phase algorithms because it provides an efficient and quick pairs removal and it does not depend on the objects complexity. The sequential algorithm of *"Sweep and Prune"* takes in input the overall objects of the environment and feeds in output a collided objects pairs list. The algorithm is divided in two principal parts. The first one is in charge of the bounding volume update of each active virtual objects. Bounding volumes used are AABBs that are aligned on the environment axis (cf. Figure 2). The second part is in charge of the detection of overlapping between objects. To do that a projection of higher and upper bounds on the three axis of coordinates of each AABBs is made. Then, we obtain three lists with overlapping pairs on each axis (x, y and z). We can notice two related but different concepts on the way the *Sweep and Prune* operates internally: by starting from scratch each time or by updating internal structures. To differentiate them a name was given to each method, the first type is called brute-force and the second type persistent. We work with the brute force one and after projection on axis there are $\frac{(n^2-n)}{2}$ objects pairs to test. Pair that is still alive before this test mean that its objects are considered as in potential collision. This pair is then given to the narrow-phase.

Multi-core architecture allows us to separate collision detection computations on available cores. But computations can not be separated on the way without a special data structure. To fully exploit multi-cores architecture, critical sections, threads idling and cores synchronisation have to be taken account and minimised or avoided. To parallelise the algorithm we have decided to use OpenMP because

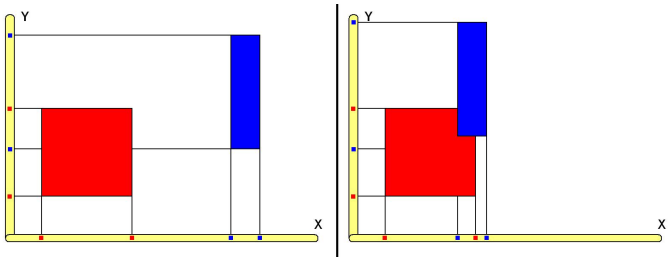

Fig. 2. "Sweep and Prune" algorithm on x and y axis with a non-overlapping condition (left) and an overlapping one (right).

of the directives that allow to keep the same code (with few algorithmic modifications on the data structure) and to focus on the directives. Even if IntelTBB provides better performances, it is more complex to program with and it generates specific code, enable to work without IntelTBB libraries.

A simplified scheme of our model is in Figure 3. We can notice parallelisation of the two principal parts of the algorithm with a synchronisation between both. Number of threads that are created depends on the number of available cores. As a thread is only in charge of geometric computations and does not wait for anything, create more than one thread per core will increase computation time. In the first step of the algorithm, each thread works on $\frac{n}{c}$ objects where n is the number of objects in the environment and c the number of cores. It is possible to divide objects per threads because AABB update computation does not depend of the object complexity, time spent per object by a thread is almost homogeneous. Compared to the sequential algorithm where new computed bounding volume is written on the way in a data structure, we can not use the same scheme without avoiding critical writing section between threads. That is why we introduce a new smallest data storage used by each thread to put new computed bounding volume. This new structure is an array dynamically allocated in relation to the number of cores and objects. Synchronisation between this two steps is compulsory to merge all the new bounding volumes in the same data structure. We only merge threads array pointers to reduce synchronisation time.

In the second part of the algorithm, each thread works on $\frac{(n^2-n)}{2}/c$ pairs of objects where c is still the number of cores. Like in the first part, each computation made by a thread is an overlapping test between objects coordinates so it does not depend on the object complexity. To avoid critical section between threads we use a similar technique where each thread is fitted with its own data storage to put objects pairs with overlapped coordinates. All pairs of objects in collision are merged at the end of the overall computation to create the pair list of objects in collision. Then, this new pairs list is given to the narrow-phase that performs an exact collision detection test.

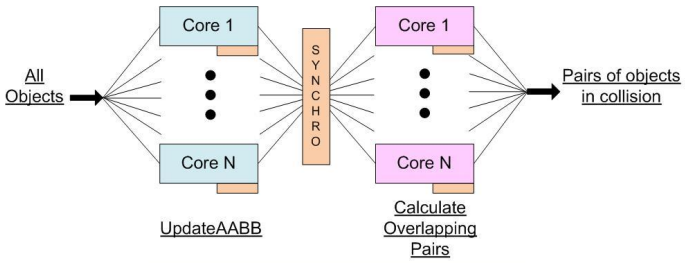

Fig. 3. Our parallel broad-phase algorithm. Parallelisation of the update AABB part and the calculate overlapping pair one with a synchronisation point between them.

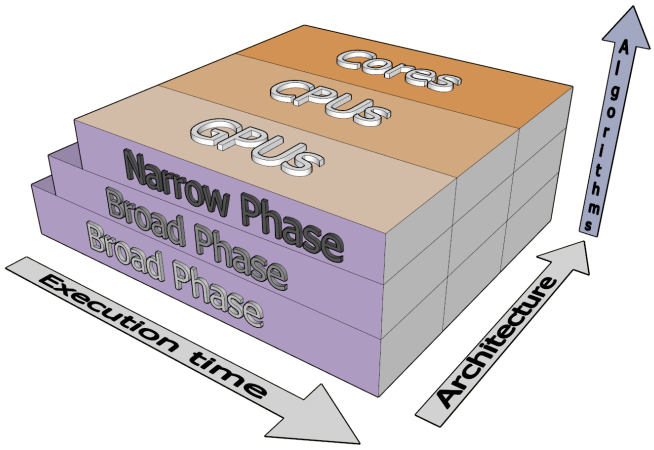

Fig. 4. The new 3D formalism of the collision detection pipeline.

B. Revisited Pipeline

We use new 3D formalism of the collision detection pipeline [31] to represent our new parallel algorithm. We can see it on the Figure 4. This formalism allows to represent in same time the type of algorithms used and hardware mapping techniques performed. We have developed a parallel broad-phase algorithm separated in two main parts using multi-threading techniques on a multi-cores architecture. People who want to develop real-time algorithms for large-scale virtual environment have to take into account the runtime architecture. This formalism recalls us that architecture must be an essential point for the development of one block of this 3D pipeline.

In the near future, we want to focus on each block and propose a generic mapping model for each of them on several architecture components.

V. Results

In this section we present main results of computation time speed-up. Those tests were performed through several benchmark models (cf Figure. 7). We only performed test on n-body simulation of rigid bodies using AABB as bounding volume. To obtain homogeneous results, we have only worked on a 8-cores computer using 1, 2, 4 or 8 cores. We work on Windows XP Professional x64 Edition Version 2003 with Intel Xeon (2*Quad) CPU X5482 of 3.20 GHz and with 64 GB of RAM.

	Cubes	Balls	Skittles
1 core	8,89ms	4,45ms	1,6ms
2 cores	4,96ms	2,48ms	0,9ms
4 cores	2,76ms	1,4ms	0,5ms
8 cores	1,52ms	0,74ms	0,27ms

Fig. 5. Time spent for updating AABB for each benchmark model from 1 core to 8 cores.

	Cubes	Balls	Skittles
1 core	53,339ms	26,7ms	10,71ms
2 cores	31,65ms	15,748ms	6,35ms
4 cores	18,76ms	9,51ms	3,742ms
8 cores	11,43ms	5,82ms	2,314ms

Fig. 6. Time spent to calculate overlapping pairs for each benchmark model from 1 core to 8 cores.

We present here time results for all used benchmark models (Cubes, Balls and Skittles). Numerical results for the first part of the algorithm is presented in tab 5. The reduction of the overall running time is shown on the graphic in Figure 8. We can see a percentage of time reduction for the first part of the algorithm concerning the AABB update. For one scenario four blocks show time spent from 1 to 8 cores and we can notice that time decreases when the number of cores goes up. The overall running time is reduced by 56.04% by using 2 cores, 31.49% for 4 cores and 17,03% for 8-cores. Numerical results for the second part of the algorithm is presented in tab 6. This second part of the algorithm is shown in the graphic Figure 9 and we notice the same gain of time than the first part. The overall running time is reduced by 59.2% by using 2 cores, 35.34% for 4 cores and 21.56% for 8-cores.

The general speed-up of our parallel algorithm is shown in Figure 10, on this graphics our work is represented by the pink line bounded by the blue one which is the optimal speed-up for a parallel execution whose we wanted to get closer. We have also performed measures on the computation time spent by t threads shared on c cores and the assumption made at the beginning on using more than one thread per core seems to be exact. Time spent by 3 threads on 2 cores is slower than 2 threads but better than 1. So using more than one thread per core is not justified and appears to be less efficient.

VI. Conclusion & Future Works

We have presented a contribution on the collision detection optimisation centered on hardware performance. The revisited collision pipeline, presented with a new third dimension allows to merge hardware components with collision detection pipeline. Starting with this new pipeline, we focus on the first

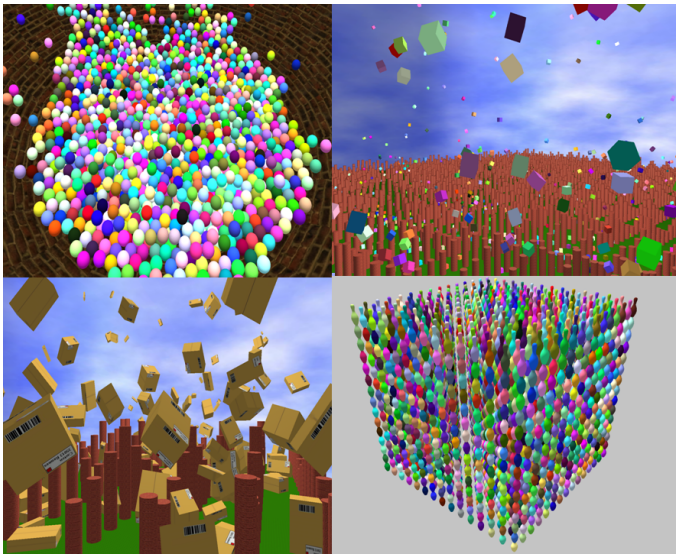

Fig. 7. **Benchmarks:** We used several benchmark models to measure collision detection time: 10K balls of 2K polygons each falling in simple environment of 600 polygons (= 1.1M polygons), 20K cubes of 12 polygons each fallen on complex environment of 300K (= 420K polygons) and 3.5K concave shapes (skittles of 20K each) falling on a plan. We only performed test on n-body simulation of rigid bodies using AABB as bounding volume.

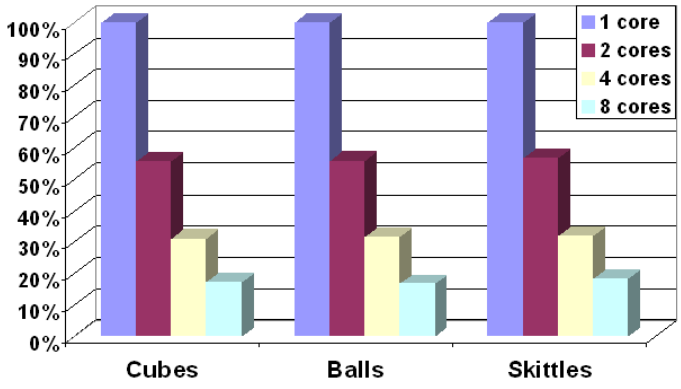

Fig. 8. The AABB update execution time in relation to the number of cores. The overall computation time is reduced by 17.03% by using 8 cores on this benchmark.

step (Broad-phase) and propose a new way of parallelisation of the well-known *Sweep and Prune* algorithm. This model takes into account run-time architecture and more precisely the number of available cores. Multi-cores architecture allows us to distribute geometric computations with use of multi-threading. Critical writing section and threads idling have been minimised by introducing new data structures for each thread. Programming with directives, like OpenMP, appears to be a good compromise for code portability.

Results show that it is not easy to pretend obtaining optimal speed-up on a multi-cores architectures. We are also focusing on a coupling of our multi-cores model with graphics hardware in order to speed-up again collision detection algorithms

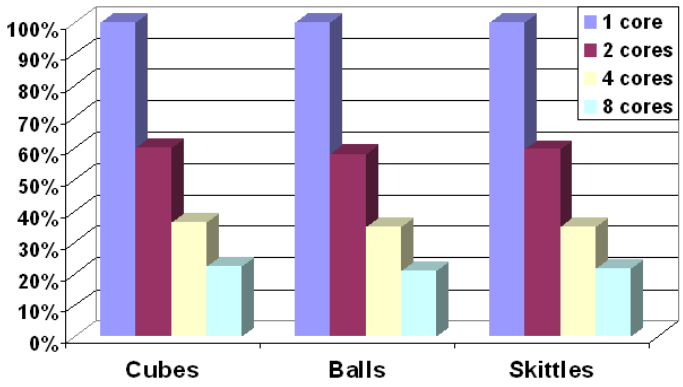

Fig. 9. The execution time of the overlapping pairs checks in relation to the number of cores. The overall computation time is reduced by 21.56% by using 8 cores on this benchmark.

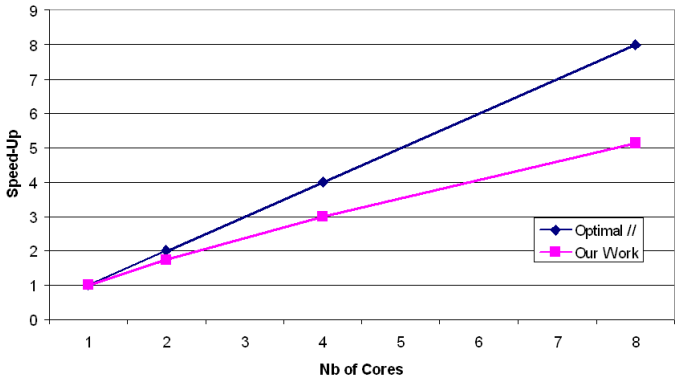

Fig. 10. The overall gain of the execution. A speed-up of 5,1 is obtained on a 8-cores computer.

and it seems to be an efficient way of optimisation. Indeed, to work on a GPU introduces several points that are interesting to focus on, like working on several memory hierarchical levels (registers, caches...). Others blocks of our revisited pipeline have also to be optimised through the use of multi or many cores architecture and GPGPU. Having a reflexion on technical characteristics and time constraints of each pipeline blocks in order to speed-up algorithms is one of our principal goal. We also imagine some parts of the pipeline working on a cluster or a grid environment taking into account particularities of each architecture. Future algorithms providing a real-time interaction have to be implemented through the use of GPGPU, multithreads, multi or many-cores architecture with memory and cache handling and maybe on cluster or grid architecture.

VII. Acknowledgments

This research is supported by the Bretagne Region under project GriRV N4295. Authors want also to thanks M. Florian Nouviale for his great help in engineering and debugging.

REFERENCES

[1] DukSu Kim, Jea-Pil Heo, and Sung eui Yoon. Pccd: Parallel continuous collision detection. Technical report, Dept. of CS, KAIST, 2008.

[2] Min Tang, Dinesh Manocha, and Ruofeng Tong. Multi-core collision detection between deformable models. In *Computers & Graphics*, 2008.

[3] Everton Hermann, Bruno Raffin, and François Faure. Interactive physical simulation on multicore architectures. In *Eurographics Workshop on Parallel and Graphics and Visualization, EGPGV'09, March, 2009*, Munich, Allemagne, 2009.

[4] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In Robert Cripps, editor, *Proceedings of the 8th IMA Conference on the Mathematics of Surfaces (IMA-98)*, volume VIII of *Mathematics of Surfaces*, pages 37–56, Winchester, UK, September 1998. Information Geometers.

[5] Pablo Jimnez, Federico Thomas, and Carme Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[6] Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, Marie-Paule Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Straßer, and Pascal Volino. Collision detection for deformable objects. *Comput. Graph. Forum*, 24(1):61–81, 2005.

[7] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and Richard Rowe. Collision detection: A survey. *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 4046–4051, Oct. 2007.

[8] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, September 1995. ISSN 1077-2626.

[9] Gino Van Den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[10] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the ACM Conference on Computer Graphics*, pages 171–180, New York, August 4–9 1996. ACM.

[11] Srikanth Bandi and Daniel Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. *Comput. Graph. Forum*, 14(3):259–270, 1995.

[12] Jon Louis Bentley and Jerome H. Friedman. Data structures for range searching. *ACMCS*, 11(4):397–409, 1979.

[13] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *SI3D*, pages 189–196, 218, 1995.

[14] G. Vaněček, Jr. Back-face culling applied to collision detection of polyhedra. *The Journal of Visualization and Computer Animation*, 5(1), January–March 1994.

[15] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. Technical report, University of Berkeley, California, March 19 1991.

[16] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4:193–203, 1988.

[17] George Baciu and Wingo Sai-Keung Wong. Image-based collision detection for deformable cloth models. *IEEE Trans. Vis. Comput. Graph*, 10(6):649–663, 2004.

[18] Gabriel Zachmann. Optimizing the collision detection pipeline. In *Proc. of the First International Game Technology Conference (GTEC)*, January 2001.

[19] Yoshifumi Kitamura and Andrew Smith. Parallel algorithms for real-time colliding face detection. *Robot and Human Communication*, pages 211–218, November 07 1995.

[20] Gao Shuming Wan Huagen, Fan Zhaowei and Peng Qunsheng. A parallel collision detection algorithm based on hybrid bounding volume hierarchy. *CAD/Graphics2001*, August 2001.

[21] R. Lario, C. Garcia, M. Prieto, and F. Tirado. Rapid parallelization of a multilevel cloth simulator using openmp. In *Proceedings of the 3rd European workshop on OpenMP.*, pages 21–9, 2001.

[22] Ulf Assarsson and Per Stenstrom. A case study of load distribution in parallel view frustum culling and collision detection. In Rizos Sakellariou, John Keane, John R. Gurd, and Len Freeman, editors, *Euro-Par 2001: Parallel Processing, 7th International Euro-Par Conference (7th Euro-Par'01)*, volume 2150 of *Lecture Notes in Computer Science (LNCS)*, pages 663–673, Manchester, UK, August 2001. Springer-Verlag (New York).

[23] Andrew Smith, Yoshifumi Kitamura, Haruo Takemura, and Fumio Kishino. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. *Proc. IEEE Virtual Reality Annual International Symposium*, pages 136–145, March 1995.

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

[25] Bernhard Thomaszewski and Wolfgang Blochinger. Physically based simulation of cloth on distributed memory architectures. *Parallel Computing*, 33(6):377–390, 2007.

[26] Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Trans. Vis. Comput. Graph*, 15(2):339–350, 2009.

[27] Mark Lewis and Berna L. Massingill. Multithreaded collision detection in java. In Hamid R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications & Conference on Real-Time Computing Systems and Applications (PDPTA'06)*, volume 1, pages 583–592, Las Vegas, Nevada, USA, June 2006. CSREA Press.

[28] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Blochinger. Parallel techniques for physically based simulation on multi-core processor architectures. *Computers & Graphics*, 32(1):25–40, 2008.

[29] Christer Ericson. *Real-time Collision Detection*. Morgan Kaufmann, 2005.

[30] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Cornell University, 1992.

[31] Quentin Avril, Valérie Gouranton, and Bruno Arnaldi. New trends in collision detection performance. In Simon Richir & Akihiko Shirai, editor, *Laval Virtual VRIC'09 Proceedings*, pages 53–62, BP 0119 53001 Laval Cedex FRANCE, April 2009.