# Parallel Collision Detection Oriented to Distributed Memory Architectures for High-Resolution Meshes

Marcos Novalbos* and Alberto Sánchez

ETSII, Universidad Rey Juan Carlos, Madrid, Spain

**ABSTRACT**

*Higher resolution meshes should be used in graphics applications to make them as realistic as they can. However, they imply a high computational. Several approaches have been built to solve collision detection, although most of them do not take into account this feature. This paper presents a scalable parallel algorithm for collision detection designed for working with high resolution meshes. The algorithm is based on distributed memory architectures taking advantage of their benefits and overcoming their drawbacks.*

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Parallel processing).

## 1. Introduction

The goal of most graphics applications is to create images that are as realistic as possible. Moreover, current animated graphics, such as games, animation movies, special effects, etc. are focused on this idea. There is a relation between realism and render time. Thus, in most cases where a fast answer is required, this means focusing on real-time handling incorporating character animation and physics simulation. This limits the realism that can be obtained. In other cases where real time is not a must, higher resolution meshes are used in order to make it as realistic as they can. Scene objects are represented as sets of triangles whose number depends on the amount of detail needed. Nevertheless, too much time is required to render these models and work with them.

Collision detection is a fundamental technique used by several graphic applications. High-resolution meshes imply a high computational cost to simulate and detect collisions. In this sense, many researchers have chosen using low-resolution meshes -e.g. [GKJ*05, SGG*06, VM06] simulate only a few amounts between 5 and 80 thousand triangles-. Most simulation techniques could fail if mesh resolution is increased because of two problems: robustness and tractability [SSIF09].

This article is aimed at improving the collision detection time for high resolution meshes. Collision detection techniques were traditionally run using a single processor. Nowadays some parallel high performance alternatives have been proposed mainly by using shared memory multicore architectures [TPB07, TMT09, KHY09]

Our proposal is to cover other architectures to exploit their benefits. We propose a load-balancing detection collision algorithm between different objects represented by high-resolution meshes using distributed memory systems.

## 2. Related work

Collision detection is a key element to provide efficient and realistic computer graphics. Nowadays there are several initiatives focused on providing parallelism to collision detection techniques to improve the performance of the graphic process. These algorithms depend on the parallel infrastructure where they are applied. Thus this dependency could cause non-scalability performance. These works can be easily divided by the high performance architecture where parallelism is applied.

Several research works [LL02, TPB07, TMT09, KHY09] use shared memory systems as architecture where the parallel algorithms are develop. For instance, [TMT09] presents a parallel algorithm for continuous collision detection between deformable models using bounding volume traversal tree. Furthermore they use a Bounding Volume Test Tree (BVTT) front in parallel exploiting the temporal coherence. But the BVTT front only works properly when it is used in scenes that keeps temporal coherency.

Others authors have used GPUs [SGG*06, LGS*09] and hybrid systems [KHH*09] to improve the collision detection performance. For instance, [KHH*09] parallelizes the collision tree navigation in shared-memory systems and use the GPU to make the triangle tests. This method avoids problems with scenes that do not keep temporal coherency. Its implementation shows better performance results compared to [TMT09]. Furthermore, its load balancing shows a good scalability when it works with Uniform Memory Access (UMA) architectures.

* Corresponding author

Finally, there are only a few works that have been used distributed memory systems [GW07, SSI*09] because this kind of architectures implies a bottleneck in the communications between processes. Our proposal selects this last alternative taking advantage of their benefits and overcoming their drawbacks. As [SSIF09], we take advantage of the scalability capacity of these systems to deal with high-resolution meshes. However, both approaches are different. Whereas in [SSIF09], the authors distribute mesh pieces between nodes, we propose that all nodes have a complete copy of the mesh only sharing the results regarding the triangles that have collided. For them, their approach means a high cost to update the border mesh pieces shared between nodes. In addition, their evaluation shows that their results scaled to 4 threads (this may be due to the use of a quad-core with shared memory). When the interconnection network is required, this solution does not properly scale.

## 3. Algorithm

We propose a parallel collision detection algorithm focused in scalability and efficiency for Non Uniform Memory Access (NUMA) architectures. As starting point, we have used the algorithm HPCCD proposed by Kim et al. [KHH*09]. Although this algorithm was developed for hybrid (CPUs and GPU) systems we have adapted it to distributed memory architectures, including load balancing.

The HPCCD algorithm takes advantage of the shared memory architecture to share data. Threads use global variables and queues to share their workload and distribute it in a load balanced way. In distributed memory architectures, it should be advisable to use cache structures to bring closer data avoiding remote access. If any process modifies data, there is then a consistency problem, and the cache must be updated. Cache updating and memory management has a high computational cost. It is required thus to design cache-friendly parallel algorithms minimizing the number of shared memory accesses and updates.

Our algorithm avoids shared memory updates keeping a copy of the whole meshes close to the process that requires them. Every process accesses the local copy to work with it, and it shares with the other processes its results. Every process is running in an isolated computational node, so it cannot access to other node data unless data is sent. For doing this, we have developed a protocol to synchronize every process. A single node acts as a master distributing the workload between the other nodes in a load balanced way.

We use two kinds of processes:

Working nodes: these processes perform the actual collision tests. They keep a complete copy of the meshes, so they can start a collision test at any point given two nodes of a Bounding Volume Hierarchy (BVH). When the simulation starts, they ask the query server for a "test" job, and then they analyze this job creating new sub-jobs. When they have no more tests, they ask again to the query server. Also, they can share their unfinished jobs with idle nodes. Once the collision test finishes, every working node shares the triangle ID pairs that were detected in collision. Each working node can calculate the new vertices position for every mesh in the scene, by doing the same physic simulation to avoid sharing parts of the scene between working nodes

Query server: This process contains the first test jobs to share between the working nodes. It also keeps an updated list that shows the workload of every working node. When the simulation starts, it receives requests from the working nodes. Then, it sends the previously queued collision test. Once the job queue is empty, it answers the requests by sending a message to the most loaded working node. The selected node will share its job with the idle node. As the query server does not know the actual workload of the nodes, it tries to guess the most loaded node. If this decision fails, it will search a new working node. If all nodes are idle, the collision test is finished and it will send an end flag to the working nodes.

## 4. Evaluation

This section analyzes in depth the different benefits of the proposed approach. This analysis aims at demonstrating the efficiency and scalability of this proposal for high-resolution meshes on distributed memory architectures in different scenarios.

In this case, the work environment is designed to test the benefits of the proposal on distributed memory vs. shared memory architectures. Thus, a NUMA environment has been selected. Specifically, a SGI PRISM machine with 16 Intel Itanium processors and 32GB of RAM memory, arranged in 8 biprocessor nodes with 4 GB of RAM memory each connected by a NUMALink network, has been used. This system can be used as both a 32 GB shared memory machine being programmed with any thread library or a cluster with 8 nodes with 4 GB of RAM and 2 cores each. In case of using it as a shared memory machine, data is spread along the 8 nodes knowing that any process running is able to access any other node's memory.

In order to compare the results, we have implemented both HPCCD parallel collision detection algorithm for shared memory and our algorithm for distributed memory on the previous environment. HPCCD algorithm has been implemented using pthread library, whereas our proposal uses MPI as the message library interface.

We focus our simulations in collision detection tests, without implementing a complete physic simulation, i.e. our simulations consist of detecting which pair of triangles is colliding, and then using a set of prerecorded simulations that emulates the physical response. Every frame is read from a file and then the object's BVH is updated. We only are aware of the triangles that are in collision in the given frame of animation, without performing a complete continuous collision detection. Our MPI implementation shares its collision test results sending a list with pairs of triangle id which represents the triangles that collide. In this way, every node is able to solve the collision in a real physic environment.

The first scenario consists of a low-resolution cloth composed of 91,470 triangles falling over a ball of 760 triangles. Figure 1 shows the collision detection times with both algorithms.

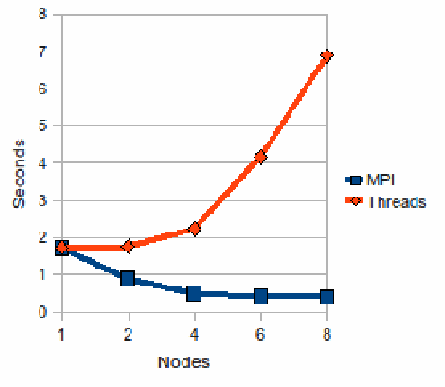


**Figure 1:** *Comparison between the collision detection times of HPCCD and the proposed collision detection algorithm for scenario1*

In this scenario, we observe that the HPCCD parallel collision detection algorithm does not obtain any improvement. This is due to the NUMA memory accesses and updates from every processor. Every thread accesses the shared mesh triangles, causing data communication between the memories of the processors. Therefore the parallel collision algorithm does not scale.

Our proposal obtains better simulation times. The collision test times are reduced when new working nodes are added, although the simulation time increases because of the update times (but still, it is smaller than in HPCCD). Also, a small time is required to share the results between the nodes. When new working nodes are added, it is necessary to send and receive results from more nodes, increasing the communication times.

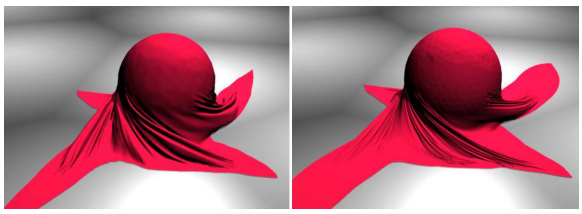

**Figure 2:** *A cloth draping over a ball. The right figure represents a low-resolution cloth composed of 91,470 triangles (scenario 1). In the left figure,the same cloth is represented by high resolution with 524,288 triangles (scenario 2). It can be seen that the high-resolution cloth represents more details*

The second scenario tries to measure the impact of using a high-resolution mesh. A similar simulation of the first scenario is used, but in this case the cloth is composed of 524,288 triangles and the sphere of 1,280 triangles (see Figure 2). Figure 3 shows the simulation times with both algorithms, stating the benefits of our proposal

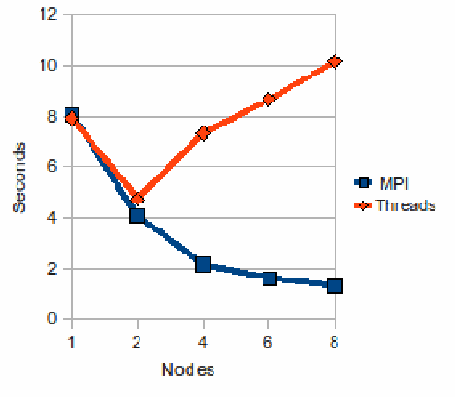


**Figure 3:** *Comparison between the collision detection times of HPCCD and the proposed collision detection algorithm for scenario 2(high-resolution meshes)*

We can see how the collision times are reduced using the distributed memory-based proposal while HPCCD collision detection algorithm does not obtain any improvement. This scenario is defined to work with more triangles, and our distributed-based algorithm takes advantage of the parallelism when there are more tests. HPCCD collision detection algorithm obtains higher performance when it works with two threads. This happens because the threads are allocated in the same computing node of the machine, which means that they can take advantage of sharing memory in a UMA environment.

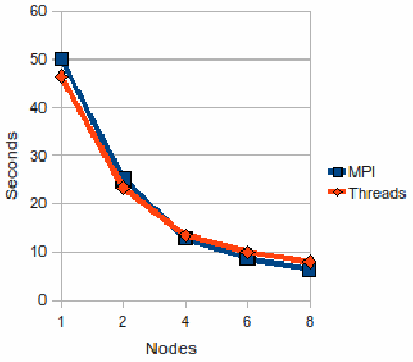


**Figure 4:** *Comparison between the collision detection times of HPCCD and the proposed collision detection algorithm for scenario 3*

The last scenario is focused on testing a case of study where most of the simulation time is spent in collision detection. In this case, we test one of the high-resolution cloth frames from the second scenario with the same frame overlapped during five frames of simulation. It means that every triangle in both meshes is colliding with another triangle. More than 50% of the simulation time will be spent in collision detection. Thus, it is a good test to see the time reduction thanks to the parallel approach. This scenario generates a high amount of triangle tests, so the size of data generated that needs to be shared is increased.

Figure 4 shows the results of the collision detection tests for this scenario. Both algorithms scale when we add more processors/nodes. We observe that the shared memory-based algorithm works well vs. previous scenarios. We can see that with 8 nodes our proposal obtains better results than the HPCCD algorithm.

As a summary, it can be observed that for all scenarios, our proposal obtains better results than the shared memory-based approach. HPCCD algorithm is limited by memory bottlenecks in the first two scenarios, only achieving a good performance under the characteristics of the third scenario. Our proposal on distributed memory obtains higher performance than threads implementation when more than 2 nodes are used.

## 5. Conclusions and Future Work

Collision detection refers to the problem of detecting the intersection of two or more objects, which can be represented by meshes. This means a computational problem that many researches have tackled from a parallel perspective. Nevertheless, the high resolution of the meshes, because of the current need of realism, brings a new scale to this problem.

In search of a scalable solution, we have presented a distributed memory parallel algorithm for collision detection designed for high resolution meshes. Efficiency and scalability has been proved by experimental results in Section 4 comparing the results of the distributed memory approach with the HPCCD memory shared parallel collision detection algorithm.

Regarding future work, we are planning to test the benefits and improvements that our proposal entails in hybrid architectures joining shared memory multicore UMA machines, GPUs and distributed memory. At the same time, we continue improving our algorithm to include self-collision detection. On the other hand, our implementation uses prerecorded simulations, which adds a bottleneck when several nodes want to update their state. It should be advisable to add a real physic simulation to obtain good performance on global time simulation. Finally, we are working on providing different mesh pieces to different groups of nodes.

## Acknowledgments

This work has been partially supported by the Madrid Regional Authority (Comunidad de Madrid) under the CCG10-URJC/TIC-5185 contract.

## References

[GKJ*05] N. GOVINDARAJU, D. KNOTT, N. JAIN, I. KABUL, R. TAMSTORF, R. GAYLE, M. LIN, AND D. MANOCHA, Interactive collision detection between deformable models using chromatic decomposition. ACM Trans. Graph. (SIGGRAPH Proc.), 24, 3, 991–999, 2005.

[SGG*06] A. SUD, N. GOVINDARAJU, R. GAYLE, I. KABUL, AND D. MANOCHA,. Fast proximity computation among deformable models using discrete Voronoi diagrams. ACM Trans. Graph. (SIGGRAPH Proc.), 25, 3, 1144–1153, 2006.

[VM06] P. VOLINO AND N. MAGNENAT-THALMANN, Resolving surface collisions through intersection contour minimization. ACM Trans. Graph. (SIGGRAPH Proc.), 25, 3, 1154–1159, 2006.

[SSIF09] ANDREW SELLE, JONATHAN SU, GEOFFREY IRVING, AND RONALD FEDKIW. Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction. IEEE Transactions on Visualization and Computer Graphics 15, 2 339-350. 2009

[KHH*09] DUKSU KIM, JAE-PIL HEO, JAEHYUK HUH, JOHN KIM, SUNG-EUI YOON. HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs. Computer Graphics Forum, 28, 7, 1791-1800, 2009

[TPB07] B. THOMASZEWSKI, S. PABST, AND W. BLOCHINGER, Exploiting parallelism in physically-based simulations on multi-core processor architectures. Proc. EuroGraphics Symposium on Parallel Graphics and Visualization, 69-76. 2007.

[LGS*09] C. LAUTERBACH, M. GARLAND, S. SENGUPTA, D. LUEBKE, D. MANOCHA. Fast BVH Construction on GPUs. Computer Graphics Forum, 28, 2, 375-384, 2009

[GW07] I. GRINBERG AND Y. WISEMAN, Scalable Parallel Collision Detection Simulation, Proc. Signal and Image Processing (SIP-2007), 380-385, 2007

[LL02] ORION SKY LAWLOR AND LAXMIKANT V. KAL. A voxel-based parallel collision detection algorithm. Proc. 16th International Conference on Supercomputing (ICS '02). ACM, 2002

[KHY09] DUKSU KIM, JAE-PIL HEO, SUNG-EUI YOON. PCCD: Parallel Continuous Collision Detection. SIGGRAPH Posters, 2009

[TMT09] MIN TANG, DINESH MANOCHA, AND RUOFENG TONG. Multi-core collision detection between deformable models. Proc. 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling (SPM '09). ACM, 2009

[SGG*06] AVNEESH SUD, NAGA GOVINDARAJU, RUSSELL GAYLE, ILKNUR KABUL, AND DINESH MANOCHA. Fast proximity computation among deformable models using discrete Voronoi diagrams. ACM Trans. Graph. 25, 3, 285-293 2006.