New trends in collision detection performance

Quentin Avril, Valérie Gouranton, Bruno Arnaldi

New trends in collision detection performance

Quentin Avril, Valérie Gouranton and Bruno Arnaldi
INSA de Rennes – France
INRIA Centre de Rennes – Bretagne Atlantique - France
IRISA, UMR CNRS 6074 – Rennes - France
Université Européenne de Bretagne, France
Email: {quentin.avril, valerie.gouranton, bruno.arnaldi}@irisa.fr

Abstract
Collision detection on industrial digital mock-up is one of the challenging problems for real-time interaction in virtual reality applications. The fast increase of graphics hardware performance, multiplication of cores number and recent improvements on their programmability, bring new directions to the optimisation of collision detection algorithms. Since few years methods appear handling General-Purpose Processing on Graphics Processing Unit (GPGPU) and more recently using multi-cores. We present in this survey an analysis on new trends in collision detection performance and we study the use from GPU to grid for virtual reality applications. We not only deal with algorithmic improvement but we propose a first approach on the new link-up between two fields of computer graphics, namely virtual reality (collision detection) and computers performance.

Keywords: collision detection, performance, parallel computing, scalable, graphics hardware, GPGPU, clusters, grids

1. Introduction

Industrial applications of virtual reality (VR) become more and more sized and the performance level for a real-time interaction of users is no longer satisfying. From more than thirty years, collision detection became the principal bottleneck of real-time VR applications. Collision detection (CD) 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. In all of these applications, real-time performance, efficiency and robustness are more and more required.
During several years, people work on different collision detection approaches but recently, few papers appear dealing with a new type of problem: speeding up the detection using hardware power (CPU and GPU) [27, 44].
In the wide range of collision detection algorithms we can notice that recent articles [11, 22] have a low complexity and provide perfect collision detection. But used with million and million of objects in a huge environment, they are inefficient for a real-time interaction. We may hope a constant evolution of processors power to resolve the real-time problem but trend is no more on that but rather in the processors multiplication. Hardware graphics is also subjected to an impressive power evolution. So it appears that having a different look on the real-time collision detection problem centred on hardware performance can not be ignored.
In the following section 2 gives a brief survey on collision detection algorithms. In section 3 we describe the hardware and architecture evolution, followed by software and middleware evolution in section 4. We present in section 5, a survey on the new link-up between collision detection and architecture performance. At the end we expose our personal point of view on this new link-up.

2. Collision detection

We now expose a short survey on the collision detection, for more details we refer to excellent surveys on the topic [24, 30, 32, 46]. Collision detection has generated a wide range of problems families: convex or non-convex objects, 2-Body or N-Body simulations, rigid or deformable objects, continual or discrete methods. Algorithms are also linked to geometric used models (polygonal, Constructive Solid Geometry (CSG), implicit or parametric functions). All of these problems reveal the high complexity and difficulty of this field of study.
Given n moving objects in a virtual environment, testing all objects pairs tend to perform $O(n^2)$

pairwise checks. When *n* is large it becomes a computational bottleneck. Collision detection is, since Hubbard [23], represented and built as a pipeline. This one is composed by three parts namely, broad-phase, narrow-phase and exact-phase (core-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. As input, the pipeline takes all the geometric data of the simulation and feeds in output the collision response module. Nowadays pipeline is shared in two phases: broad and narrow phases (exact phase being included in narrow-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 apparently not in collision. In the other hand, narrow-phase is in charge of determining exact collision detection.

2.1 Exact collision detection

The narrow phase constitutes the pairwise tests within subgroups of potentially colliding objects. We present important parts of exact collision detection starting by basic tests between polyhedrons, following with bounding volumes and their hierarchies.

2.1.1 Basic tests

In case of polygonal representations, tests are made on objects primitives. With CSG representations, tests are made on elementary geometrical objects (cubes, cones…). Implicit function or parametric surface representations involve tests with membership function. We present different algorithms used to detect collision with polygonal representation.

Detection between convex polyhedrons

It is easy to imagine that two objects separated by a plan don't intersect. Beginning with this assumption, different methods have been proposed to compute this separating plan. Some of them use a vectorial product exploiting temporal coherence or compute a bounded distance between objects to build a separating plan [16].
Other methods proposed to determine inter-objects distance. The most famous one is the GJK algorithm [15] that uses Minkowski difference on polyhedrons. A lot of GJK algorithm improvements have been proposed:
- 'Enhancing GJK' [8] uses *hill climbing* method to optimize computing time.
- 'ISA-GJK' [6] uses *data caching* to increase performances and uses a faster method to build separating plan using temporal coherence.
- MS [26] couples GJK algorithm with Lin-Canny algorithm.
Lin-Canny approach [33] or *Voronoï Marching* was the first algorithm working with objects primitives. Space around objects is divided in Voronoï regions that allow detecting closest features pairs between polyhedrons. The V-Clip algorithm [34] operates on a pair of polyhedron, defining closest points between pairs in terms of closest features of the polyhedron. Another way to compute distance between objects is to perform pre-computations on polyhedrons in order to reduce intersection detection complexity.
It is also possible to compute the interpenetration between two objects. Several approaches use GJK algorithm to compute penetration depth [8]. Gregory et al. [20] propose to extend Voronoï marching method coupling with temporal coherence. Given direction, depth penetration can be compute with Dobkin's hierarchy [12]. The use of normales space to find translation direction has been proposed [28]. More recently, a method detects interpenetration with ray casting [22].

Detection between non-convex polyhedrons

Algorithms described previously are suited for convex polyhedrons, use it with non-convex objects and they would do mistakes on the non-collision detection or collision non-detection.
To avoid expensive computing time, most of approaches use a bounding volume hierarchy that allows creating series converging to minimal distance [25]. Quinlan [38] uses spheres to prune parts of his models, Sato et al. [41] propose to couple Quinlan's spheres-trees with GJK algorithm.
Other methods propose to find a separating plan with points of one object in the negative side and points of the other in the positive side [49].
It is also possible to test objects primitives in order to determine if a vertice of an object is inside another one or if a segment intersects an object face [7]. Comparisons on objects triangles [35] or rectangles are possible.
To compute penetration depth between non-convex objects, Fisher et al. [14] show that using Minkowski sum can have a $O(n^6)$ complexity. Dobkin et al. [12] propose a method to compute this penetration between a non-convex object and a convex one, but with two non-convex objects, complexity becomes too high. A trivial approach consists in separating object into convex parts and computing penetration between overlapping convex parts [28]. Distance fields can also be used for the penetration depth computing [14].

2.1.2 Bounding Volume

Most of strategies use bounding volume hierarchies to perform collision tests. There are a lot of

bounding volume such as sphere [23], Axis-Aligned-Bounding-Box AABBs [5], Oriented-Bounding-Box OBBs [17], discrete oriented polytopes (*k*-DOP) [48] or convex hulls. As explained in [13], many other types of volume have been suggested as bounding volume namely, sphere-swept volumes, cones, cylinders, spherical shells, ellipsoids and zonotopes.
Using bounding volume (BV) to perform tests before testing the object itself, highly improves performance. Although tests have been simplified, to test collision between two objects, the same pairwise check is still performed. Bounding volume hierarchies (BVH) allow arranging BV into a tree hierarchy in order to reduce the number of tests to perform. An excellent description on these BVH and a comparison between their performance can be found in [13, 30]. Construction of these trees is performed according to three primary categories: *top-down*, *bottom-up* and *insertion*.
Deformable objects are very challenging for BVH because hierarchy structures (trees) have to be updated when an object deforms itself. As trees can not be updated at each time step, better solutions are compulsory [5, 46].

2.2 Accelerative steps

Broad-phase algorithms are classified into three main families [30], namely brute force method with bounding volumes, spatial hashing and topological and cinematic methods.

2.2.1 Brute force

Brute force approach is based on comparing bounding volumes of the overall objects pairs to determine if they are in collision or not. This test is very exhaustive because of its $O(n^2)$ pairwise checks.

2.2.2 Spatial partitioning

Spatial hashing method is based on a simple rule: two objects situated in distant space sides have no chance to collide during next time step. To divide space into unit cells several methods have been proposed: regular grid [36], octree [3], quad-tree, Binary Space Partitioning (BSP), *k-d tree* structure [4] or voxels. Subdivisions made on space can be independent from the environment [4] and can also be given by it. This technique is only accurate when the environment is static (2-body context). It is also possible to use tetrahedron meshing or use a constraints set projected on a high-dimensional space (six dimensions) [9]. Teschner et al. [45] employ a hash function to compress a potentially infinite regular spatial grid.

2.2.3 Topology and cinematic

Topology 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” [10] approach and consists in projecting objects coordinates on axis. If the projection reveals an overlapping of objects coordinates, they are probably in collision and they are then given to narrow-phase. This method is, in general, used with bounding volume like AABBs [10] or OBBs [40] (Figure 1).
On the contrary, the cinematic approach takes care of the objects movement, if objects are moving away, they can not collide. Vanecek [47] used cinematic of the objects and back-face culling technique to speed up collision detection.

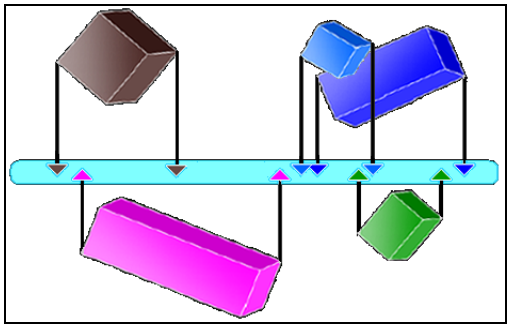

Figure 1: *“Sweep and Prune” with oriented-bounding boxes.*

3. Hardware and Architecture evolution

We now expose the evolution of computer hardware and architecture that can be used to improve collision detection. First we start with a presentation of graphics hardware (GPU) that has been highly improved those last years. We then present evolution of CPU from simple-core to multi-core and many-core. We introduce several features of the architecture evolution in order to take care of it during collision detection optimisation. At last, we briefly present the cluster and grid architectures, their use and different problems that can be encountered in performing real-time collision detection.

3.1 GPU evolution

Recent years have seen the evolution of graphics hardware from fixed function units toward an increasingly programmable graphics pipeline. Contrary to CPU, Graphics Processing Unit (GPU) has a very important power evolution since few

years (cf. Figure 2). This impressive evolution can be explained by the way that in one hand, CPU is a generalist processor (cf. Figure 4) that deals with ordinary data expressing a high level of dependencies, several of its components are in charge of the data stream control while its memory latency period is hidden by data caching. In the other hand GPU processors are well-suited to highly parallelisable computations (cf. Figure 5a). It deals with independent data so it does not need a sophisticated data stream control and its memory latency period is hidden by computations. For instance, we compare principal graphics cards of ATI[1] and Nvidia[2] in 2008 (cf. Figure 2).

ATI Radeon HD 4870 (RV770)	**Nvidia GeForce GTX 280 (G200)**
800 ALUs = 160 x 5 512Mo Ram(GDDR5) Bw =115,2Gb/s 1,2 TFlops	240 ALUs + 60 SFU 1024Mo Ram(GDDR3) 141,7 Gb / s 933,12 GFlops

Figure 2: *Comparison of recent graphics cards of Nvidia and ATI.*

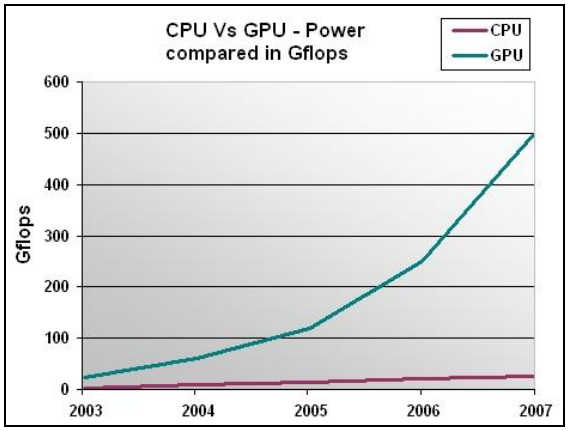

Figure 3: *Comparison of the CPU and GPU evolution from 2003 to 2007*

GPU can be imaged (in the GeForce 8800 version) as a big computing unit with 16 cells of 8 ALUs (Arithmetic Logical Unit) handling each one 4 threads. These 16 cells are able to manipulate all current instructions on 512 threads, with a flow of 256 operations per cycle. GeForce 8800 has been described as a GPU equipped with 128 processors allocated to 8 high frequency partitions (1350 MHz).
Bandwidth on GPU is also higher than on CPU [37] but a fundamental problem of performing computations on GPU is the bandwidth between CPU and GPU (only 4 GB/s). A recent solution* proposes to hide data-transferring time by using concurrent memory copy (between CPU and GPU) and execution on GPU.

* Workshop on GPU supercomputing – 16-01-2009 - Taiwan (H.Y. Schive, T. Chieuh & Y. C. Tsai)

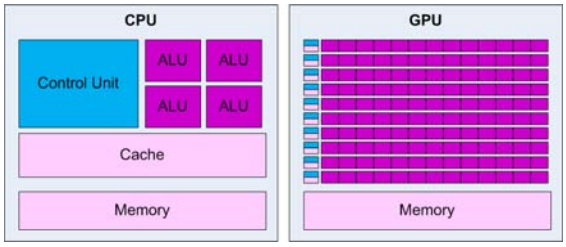

Figure 4: *Comparison of the architecture of CPU and GPU.*

3.2 CPU evolution

Compared to actual outlook, it seems clear that Gordon Moore was a lucky man. Since 1965, he predicts a duplication of the number of transistors on a microprocessor each two years. During more than forty years, this guesswork seems exact but we know now that physical limits (power and heat) prevent this duplication. Nowadays trend tends to be duplication of cores (cf. Figure 5b) in computers and parallel architectures. The first personal computer with a core-duo arrived in 2005 with AMD[1] followed by Intel[3]. In 2006 Sun[4] presented its new octo-core called Niagara2. Intel presents last year a 32 in-order x86 cores [42] and Sun recently announce 80 cores computer. Another emerging CPU concept is many-core: the computer dynamically adapts the number of active cores with respect to the user needs. Many-core is useful because when people do not need the entire power of cores, computer turns off some of them.
Until now, 3D objects and virtual environments grew up in parallel to processor power, so researchers were continuously looking for improvements on the collision detection algorithms in order to increase their precision, robustness and efficiency [11, 22]. But now, processors power stays roughly constant while virtual environments are more and more sized, so new scientific contributions are not only in the algorithms improvement but also in the algorithms architecture modification. As we can not hope a continual evolution of processors we have now to study what it is possible to do with multi-core in collision detection algorithms.
Nowadays it is impossible to present CPU without dealing with central memory handling. Indeed, on a multi or many cores, there is a very complex cache handling between cores. This handling is continously improved to increase computer

[1] http://www.amd.com/us-en/
[2] http://www.nvidia.com/
[3] http://www.intel.com/
[4] http://www.sun.com

performance. Cache and memory handling is another point that cannot be ignored in the optimisation of the collision detection performance.

3.3 Clusters and Supercomputers

A cluster, imaged as a computer grape, is a powerful engine with high performance that can be useful for real-time collision detection.
It is composed by localised machines connected through a local network (for instance Ethernet or Giga-Ethernet). In comparison to distributed units in GPU and multi or many cores in CPU, a cluster, with a higher scale, can be seen as a many or multi computers machine (cf. Figure 5c). A survey [39] has been done on different approaches that have been developed to use PC clusters for virtual reality applications. This survey also presents middle-ware allowing using maximum of cluster performances (section 4).
Differences between a cluster and a supercomputer become very thin because they use same CPUs and GPUs connected with a high performance network working on the same Operating System. Contrary to personal computers, clusters and supercomputers receive great attention on their communication architecture. The problem changes when you have to manage 10, 50, 500 or 2000 nodes. Several network topologies have been proposed, for instance: cubes, hypercube or torus.

3.4 Grids

A grid computing is an infrastructure composed by a mass of non-homogenous informatics resources (PC-clusters, computers, servers, mobiles…). Grid sites are geographically separated and it is usually used for huge scientific computations (cf. Figure 5d). A grid is a bit different than clusters and computers because it is not in the parallel computing world but rather in the distributed world. A middleware is integrated into grids in order to abstract all resources and so to exploit computation power (processor, memory…). Using a grid can reveal some more important problems than on clusters because of the resources volatility, the access rights and the latency time. Communication between grid resources is done with optical fibre and every one knows that its bandwidth physical limit is light speed. But as optical fibres go through signals amplifiers, bandwidth is reduced. This speed reduction has to be taking account into distributed computation for real-time interaction. Algorithms that would work on grid have to be more predictive than detective due to the latency. These algorithms have also to provide a dynamic resources management because of their volatility.

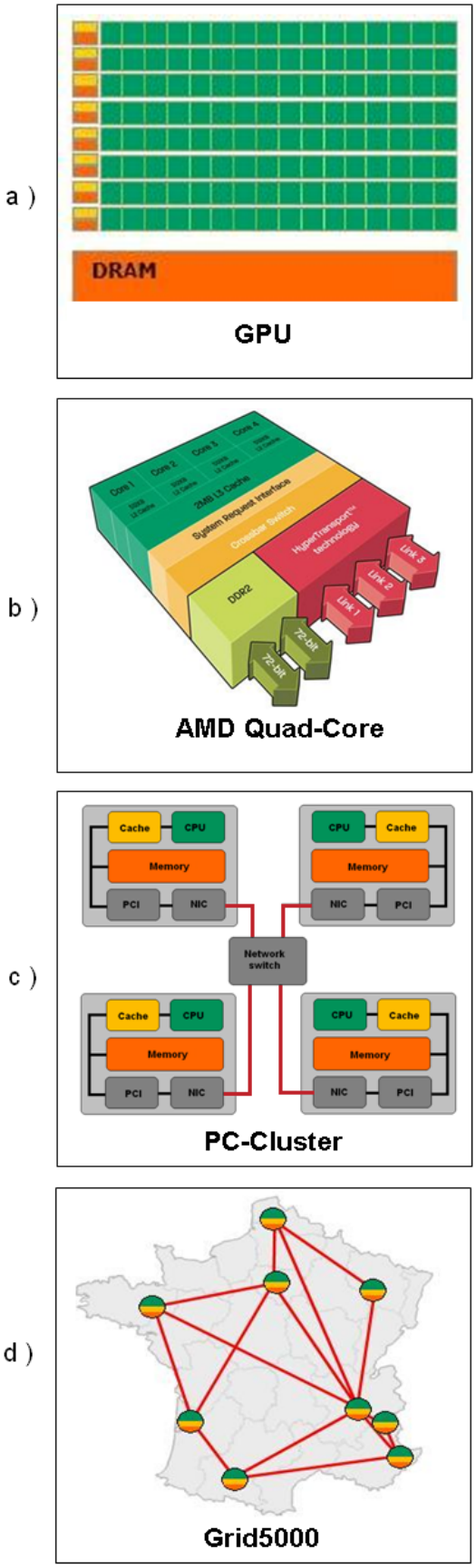

Figure 5: *Comparison of simplified architectures of GPU, quad-core, PC-Cluster and grid.*

3.5 Conclusion

For any architectural levels from GPU to grid, we can notice a fractal-like hierarchy (cf. Figure 5). In this figure, even if architectures schemes have been obviously simplified, they are still representative. In all of them, there are several computational units and a lot of memories and caches units through communication architectures.

4. Software and Middlewares evolution

Computer architecture is able to provide high computational performance. To fully exploit this power we need well-suited software tools and middleware. Given that our future goal is not to propose a new architecture disposition or a new cache handling, we now focus on available tools allowing the use of the overall computer power.
We follow the same previous plan to present middleware from GPU to grid.

4.1 GPGPU

General-purpose Processing on Graphics Processing Unit is the technique allowing graphics hardware (GPU) to perform computations traditionally reserved to CPU. A survey has been published [37] on GPGPU focusing on a simple presentation of GPGPU applications.
Using graphics cards in order to increase mathematical computations is not recent. During the nineties, some researchers use rasterizer and Z-Buffer of the graphics cards to accelerate path, for instance, path finding or Voronoï printing. But revolution appears in 2003 with evolved shaders allowing matrix computations on graphics cards. From this year, a *SIGGRAPH*[5] section is dedicated to this new computation technique. To handle GPU in 2003, OpenGL or Direct3D were essential. Brook was the first C language extension that allowed using GPU as a co-processor for parallel computations.
Recently (2007) Nvidia developed a language and a software called CUDA[6] (Compute Unified Device Architecture) exploiting GPU's power, using principles of parallel programming with threads. This API can be seen as a C language extension and its assembly language is PTX. As depicted in Figure 6, CUDA offers two API:

- An high level one : CUDA Runtime
- An low level one : CUDA Driver

The third software layer is a librairies set: CUBLAS for linear algebra computations and CUFFT for signal treatment computations.

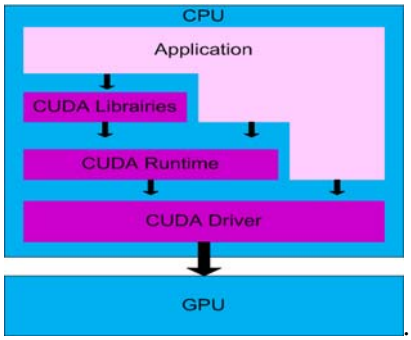

Figure 6: *CUDA Architecture.*

ATI/AMD develops its own language for graphics cards, called Brook+. Runtime uses CAL for the GPU backend. Even if AMD technology is as efficient as Nvidia's (or even more), Brook+ is less used than CUDA, due to a lack of documentation on it and to a higher difficulty to code solution. In Figure 7, we present several high level languages used in GPGU.

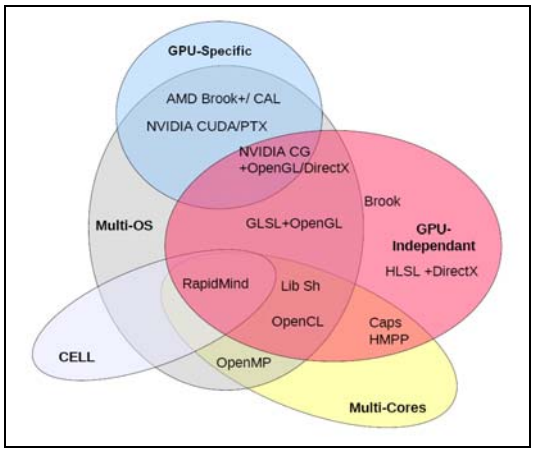

Figure 7: *Presentation of high level language for GPGPU*

4.2 OpenCL

OpenCL[7] (Open Computing Language), performed by Khronos[7,] is the first open standard for general-purpose parallel programming (GPGPU) of heterogeneous systems. It is analogous to OpenGL and OpenAL that are open industry standards for 3D graphics. It is a framework for writing programs executed on heterogeneous platforms (CPUps, GPUs and other processors). It includes a language based on C99 that is a modern dialect of C.
Khronos argues that molecular and fluid dynamics simulations will match very well to GPUs with OpenCL. The OpenCL 1.0 specifications and

[5] http://www.siggraph.org/
[6] http://www.nvidia.com/object/cuda_home.html
[7] http://www.khronos.org/opencl/

header files are available but not the implementation yet.

4.3 Middleware

In order to have an efficient use of computer or cluster power, a well-suited middleware is needed. The goal of a middleware is to give an abstract view of all hardware components to connect applications code to run-time infrastructure. In our study, we focus on grid and cluster middlewares providing a thin granularity of the architecture to users but having their own specificities.

Cluster Middleware

The main difficulty of a cluster middleware is first, to provide an efficient assembling and distribution of the overall components, and also to keep the application coherence.
Kerrighed[8] is an operating system working on cluster and providing a unique view of the overall nodes. It can be compared to a multi-processors machine with shared memory. FlowVR [1] is a cluster middleware dedicated to VR applications and it can also be implemented on a grid environment. These clusters middlewares provide both an abstract view and a thin granularity of resources.

Grid Middleware

To present grid middleware, we take a project example called *XTreemOS*[9]. The main objective of the *XtreemOS* project is the design, implementation, evaluation and distribution of an open source Grid operating system that can work on a wide range of platforms, from clusters to mobiles. The system is installed on each participating machine offering a single system view and giving the illusion to use a traditional computer. As we noticed previously, latency time is an incompressible physical limit, so even if grid middlewares provide good resources abstraction, collision detection algorithms have to be adapted to take into account this latency. Algorithms have also to provide an efficient dynamic resources management.

5. Collision detection and new architectures

Recent years have seen an increasing interest of performing collision detection algorithms taking into account computer architecture. We present three main families of architecture-based algorithms: GPU, Multi-threads and Multi-cores.

5.1 GPU-based algorithm

Image-based algorithms have been proposed to exploit the growing computational power of graphics hardware. GPU is very efficient in rasterisation of polygons; GPU-based collision detection algorithms rasterise the objects and perform either 2D or 2.5-D overlap tests in screen space [19]. Given several objects meshes, it returns pairs of objects primitives that are then computed on the CPU. A good advantage of using graphics hardware is the un-use of precomputed volumetric data structures and its use with rigid or deformable objects. GPU can also be useful to compute distance fields using a uniform spatial grid [43].
Furthermore, visibility computations can be performed using occlusion queries and used to compute both intra- and inter-object collisions among multiple objects [18].
A technique using image-space have been proposed and compared to a CPU-based implementation; results show that GPU accelerates collision detection in complex environments but CPU-based methods provides more flexibility and better performance in small environments [21]. Broad-phase is also made with GPU using image-space visibility queries [19].
Cinder [29] is an algorithm exploiting GPU to implement a ray-casting method to detect collision. When a ray strikes an edge, a count of the difference in the number of back-facing and front-facing polygons lying between the edge point and the ray's origin at the viewport is made.
GPU-based algorithms for self-collision and cloth animation have also been introduced by Govindaraju et al. [18]. An efficient backward voxel-based AABB hierarchy method was proposed to handle deformable surfaces that are highly compressed using graphics hardware [2].

5.2 Multi-threads based algorithm

Since few years, researchers are working on the implementation of multithreaded algorithms in collision detection and more precisely in dynamics molecular simulation. Lewis et al. [31] propose a new multithreaded algorithm to simulate planetary rings. An evaluation of the performance of a parallelized back-end of the pipeline has been made by Zachmann [49] and shows that if the environment density is large compared to the number of processors, then good speed-ups can be noticed. This evaluation did not parallelize the other phases of the pipeline.

5.3 Multi-cores based algorithm

Very recently, few papers appear dealing with new parallel collision detection algorithms using multi-cores. Tang & al. [44] propose to use a hierarchical

[8] http://www.kerrighed.org/
[9] http://www.xtreemos.org/

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 & al [27] 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.

6. Revisited CD pipeline

Trough this survey, it appears that the architecture of collision detection algorithms needs to be improved to face real-time interaction. In this way, we are thinking about reviews of the collision detection pipeline. We work on a global parallelisation of the pipeline in order to avoid starvation of each phase. They would not stop and run continuously during simulation. Broad-phase is still pruning un-colliding objects pairs and feeds an object pairs buffer that is, in the same time, used by narrow-phase to provide exact tests. This pipeline can be imagined as a double-buffer with access control. This parallelised pipeline would adapt itself on different hardware architectures. For instance this architecture would provide a very efficient, robust and fast broad-phase that would work on several GPUs. Then architecture would provide a narrow-phase working on multi-cores.
We have designed a novel view of a tri-dimensional collision detection pipeline (cf. Figure 8). The sequential pipeline has been revisited as a parallel pipeline working with broad and exact phase using buffers. Contrary to sequential pipeline, we propose to add a third dimension. This new dimension is the architecture showing that one phase can be done on GPUs and another on CPUs or one part of phase. This 3D pipeline might be dimensioned for a cluster or grid architecture. We also imagine running a collision detection system on grid, taking into account latency time. On the overall distributed machines predictive algorithms would compute possibly colliding sets of objects and local machines would compute exact collision.

7. Conclusion

We have presented in this survey new trends in collision detection performance quest. With the wide range of available architecture from GPU to grid, we have shown that developing new models taking into account computer power is essential to expect real-time interaction in large-scale environment. The link-up between virtual reality and computer performance has to be more and more studied and reinforced. Future algorithms providing a real-time interaction have to be performed through the use of GPGPU, multithreads, multi or many-cores processors with memory and cache handling and cluster or grid architecture.

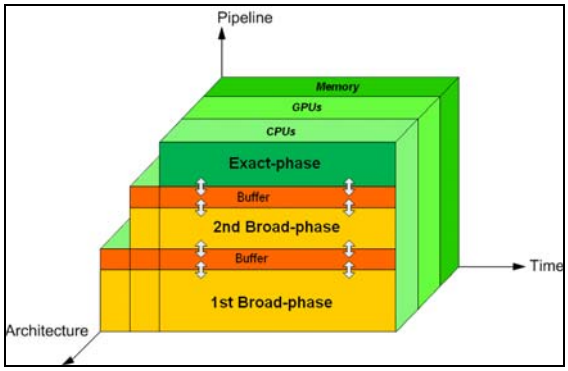

Figure 8: *Example of a new tri-dimensional collision detection pipeline.*

References

[1] Jeremie Allard, Valerie Gouranton, Loick Lecointre, Sebastien Limet, Bruno Raffin, and Sophie Robert. FlowVR: A middleware for large scale virtual reality applications, August-September 2004.

[2] 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.

[3] 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.

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

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

[6] Gino Van Den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *J. Graph. Tools*, 4(2):7–25, 1999.
[7] J. W. Boyse. Interference detection among solids and surfaces. *Communications of the ACM*, 22(1):3–9, January 1979.

[8] Stephen Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra, January 27 1997.

[9] J. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:200–209, 1986.

[10] 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.

[11] Daniel S. Coming and Oliver G. Staadt. Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Trans. Vis. Comput. Graph*, 14(1):1–12, 2008.

[12] Dobkin, Hershberger, Kirkpatrick, and Suri. Computing the intersection-depth of polyhedra. *ALGRTHMICA: Algorithmica*, 9, 1993.

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

[14] Susan Fisher and Ming C. Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields, *UNC*. July 17 2001.

[15] 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.

[16] Elmer G. Gilbert and Chong Jin Ong. New distances for the separation and penetration of objects. In *ICRA*, pages 579–586, 1994.

[17] 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.

[18] Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. Fast and reliable collision detection using graphics processors. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2005.

[19] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In M. Doggett, W. Heidrich, W. Mark, and A. Schilling, editors, *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 025–032, San Diego, California, 2003. Eurographics Association.

[20] Arthur D. Gregory, Ajith Mascarenhas, Stephen A. Ehmann, Ming C. Lin, and Dinesh Manocha. Six degree-of-freedom haptic display of polygonal models. In *IEEE Visualization*, pages 139–146, 2000.

[21] Bruno Heidelberger, Matthias Teschner, and Markus H. Gross. Detection of collisions and self-collisions using image-space techniques. In *WSCG*, pages 145–152, 2004.

[22] Everton Hermann, Francois Faure, and Bruno Raffin. Ray-traced collision detection for deformable bodies. In *GRAPP*, pages 293–299, 2008.

[23] 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.

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

[25] David E. Johnson and Elaine Cohen. A framework for efficient minimum distance computations. In *ICRA*, pages 3678–3684, 1998.

[26] D. d'Aulignac K. Sundaraj and E. Mazer. A new algorithm for computing minimum distance, October 13 2000.

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

[28] Young J. Kim, Ming C. Lin, and Dinesh Manocha. Deep: Dual-space expansion for estimating penetration depth between convex polytopes. In *ICRA*, pages 921–926. IEEE, 2002.

[29] Dave Knott and Dinesh K. Pai. Cinder: Collision and interference detection in real-time using graphics hardware. In *Graphics Interface*, pages 73–80, 2003.

[30] 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.

[31] 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.

[32] 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.

[33] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. Technical report, *UNC*. March 19 1991.

[34] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph*, 17(3):177–208, 1998.

[35] Moller. A fast triangle-triangle intersection test. *JGTOOLS: Journal of Graphics Tools*, 2, 1997.

[36] Overmars. Point location in fat subdivisions. *IPL: Information Processing Letters*, 44, 1992.

[37] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. 2007. Warning: the year was guessed out of the URL.

[38] Sean Quinlan. Efficient distance computation between non-convex objects. In *ICRA*, pages 3324–3329, 1994.

[39] Bruno Raffin and Luciano Soares. Pc clusters for virtual reality. *IEEE-VR*, pages 215–222, 25-29 March 2006.

[40] Stéphane Redon, Abderrahmane Kheddary, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–288, September 2002.

[41] Y. Sato, M. Hirata, T. Maruyama, and Y. Arita. Efficient collision detection using fast distance-calculation algorithms for convex and non-convex objects. In *Proc. IEEE Intern. Conf. on Robotics and Automation (Minneapolis, Minnesota 1996)*, pages 771–778. IEEE, 1996.

[42] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM SIGGRAPH'08 Transactions on Graphics*, 27(3), August 2008.

[43] Avneesh Sud, Miguel A. Otaduy, and Dinesh Manocha. Difi: Fast 3D distance field computation using graphics hardware. *Comput. Graph. Forum*, 23(3):557–566, 2004.

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

[45] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *Proceedings of the Conference on Vision, Modeling and Visualization 2003 (VMV-03)*, pages 47–54, Berlin, November 19–21 2003. Aka GmbH.

[46] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. pages 119–140, September 2004.

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

[48] Gabriel Zachmann. Rapid collision detection by dynamically aligned DOP-trees. pages 90–97, March 1998.

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