Self-Collision Detection in Tubular Objects Approximated by Spheres

Enrique Ayala [1], Francisco A. Madera[2] and Francisco Moo-Mena[3]

**[1,2,3] Facultad de Matemáticas, Universidad Autónoma de Yucatán
Mérida, Yucatán 97110, México**

Abstract

We investigate the performance of an algorithm to detect self-collisions in a tubular object approximated by spheres. The approach utilizes the Bounding Volume Hierarchy (BVH) to arrange the spheres and it was implemented using sequential and parallel algorithms. The tubular object has a snake-like motion, and the algorithm calculates the closer pairs of spheres considering the hierarchy and the parallelism. Experiments were carried out to analyze the performance of the implementations with different object's motions.

Keywords: *Computer Graphics, Collision Detection, Bounding Volume Hierarchy.*

1. Introduction

The widespread use of animation has resulted in a strong demand for accurate and believable collision detection. The contact points between two objects can usually be approximated at a small number of locations on a subset of the object's primitives. Accurate self-collision detection is challenging to perform in real time because many adjacent or nearby primitives of a deforming mesh are always in close proximity.

Detecting self-collision for cables and similar objects is an important part of numerous models in many areas of computer simulation such as hair modeling [1], robotics [2], rope simulations [3], virtual intestines [4], and protein folding [5], just to name a few areas of application. The model utilized in this work is a tubular object which has been approximated with spheres using the method proposed in [6]. Thus, we work with a set of joining spheres which we call the chain of spheres. We implemented CPU and GPU versions of the self-collision detection algorithm. The CPU version compares all the pairs of spheres in a brute force manner. In the GPU versions, spheres are arranged in a hierarchy to speed up the collision detection process.

Recent advances in parallel processors such as multicore CPUs and many-core GPUs have made parallel computing ubiquitous, and such trends are expected to continue in the future. We implemented a parallel version with GPUs to explore the performance of the tree construction and tree traversal.

The remainer of the paper is organized as follows: In Section *2* previous work is presented, Section *3* gives an overview of the several versions of the algorithm, in Section *4* the hierarchy construction is described. The hierarchy traversal is explained in Section *5*, empirical results are presented in Section *6* and the conclusion is described in Section *7*.

2. Previous Work

Efficient collision detection algorithms are commonly accelerated by spatial data structures such as BVH or spatial partitioning. Such object representation is commonly built in pre-processing stage and performed very well for rigid and deformable objects. Bounding volumes are utilized in many applications because of their ability to represent the shape of objects and the reduced cost of testing against another BV (Bounding Volume). Spheres have been used in a wide range of applications since they are easy to represent, have a fast overlap test, and are rotationally invariant [7,8]. AABB (Axis Aligned Bounding Box) also has a fast overlap check, which is accomplished via a simple comparison of its coordinate values [9, 10]. The Oriented Bounding Box (OBB) can bound the object tighter than AABB because it is oriented to best align with the underlying geometry [11], however it does require a more expensive overlap test. Other volumes are discrete oriented polytopes, sphere-swept volumes, and convex hulls.

Research on tubular surfaces are widely found in the literature. Li et al. [12] proposed an approach to extract multi-branch tubular structures using minimal user input.

A novel 4D iterative key point searching method is proposed and utilized to detect multi-branch tubular structures with only one initial point. Tubular shapes are found in vessels and airways taken from Computerized Tomography and Magnetic Resonance Imaging volumes [13]. Luboz et al. introduce a simple model of vessel deformation based on few pre-deformed vessel shapes to take into account the action of the balloon and the stent on the vasculature [14].

In [15], a sweep-and-prune algorithm for detecting self-collisions of a deforming cable comprising linear segments is investigated. Rather than using spheres, in this reference cables are represented by a set of segments. For cable models whereby the current cable configuration is found by computing the energy minimizing configuration, adjustments to all cable segments in each simulation step is applied.

The implementation of algorithms in GPUs using CUDA has been investigated, in particular the enhancements of the memory usage. The Barnes-Hut n-body algorithm was implemented in [16] running in six kernels. The kernels are optimized to minimize memory accesses and thread divergence and are fully parallelized within and across blocks. Rosen presented an approach [17] to investigating the memory behavior of CUDA kernels focusing on identifying representative warps and performing detailed analysis of those warps. Zhang and Kim [18] proposed a method of computing adaptive distance fields on a GPU. Based on the notion of a p-partition, the algorithm distributes the workload of BVH traversal among multiple processing cores, while minimizing the memory overhead.

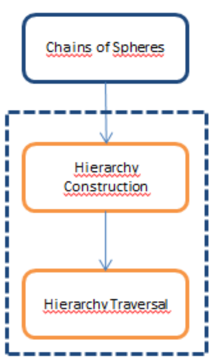

Fig. 1 Stages of the method proposed to detect self-collisions in tubular objects.

3. The implemented versions of the self-collision detection

The algorithm employed detects self-collisions by testing the overlap between the spheres that bound the object. A chain is formed by a set of spheres that cover the mesh of the object. As depicted in Figure 1, the entrance of the method is the chain of spheres and the two stages are the construction of a hierarchy and the hierarchy traversal. The goal is to cull away non-closer spheres.

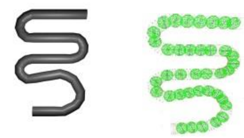

Fig. 2.Tubular objects are approximated with spheres [1].

We introduce an algorithm to build a hierarchy of spheres (BVH) and then traverse the tree *T* to detect self-collisions. One of the aims of this work is to compare the different versions of the algorithm implemented.

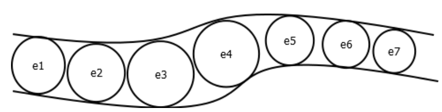

Fig. 3. A tubular object represented with seven spheres.

In the first algorithm (CPU1), sphere e_1 is compared against the other spheres to determinate overlaps. The algorithm requires $O(n^2)$ time, a costly operation. The algorithm GPU1 employs the GPU to improve the process CPU1, each sphere is compared with the others through one thread per sphere. The time per sphere is $O(n)$. However, as we have n spheres and n threads, the time remains as $O(n)$ since threads run in parallel.

The following algorithms require a hierarchy to detect collisions. This way, there are two stages: the hierarchy construction and the hierarchy traversal.

Algorithm aCPU2 considers the list of the spheres as the leaves of the tree and takes n pairs to form their parents which represent an upper level. The number of spheres has been reduced in $n/2$. To form the next upper level, we take spheres in pairs again. As the number of levels of the hierarchy depends on the number of leaves, we repeat the process $\log n$ times. Therefore, the time required is $O(n \log n)$.

To construct the hierarchy using GPUs, we take advantage on the threads allowed. For n leaf spheres, we take n

threads to perform the same process as algorithm aCPU2. This is the algorithm bGPU2. When the number of leaf spheres is greater than number of threads allowed per multiprocessor, then, we utilize algorithm aGPU2, which divides the leaf spheres in subtrees.

In the hierarchy traversal stage, three algorithms were developed. The sequential algorithm (CPU2) compares e_1 vs T, the hierarchy. This takes $O(log\ n)$ time for each sphere. As we have n spheres, then the process takes $O(n\ logn)$ time, that is, the levels of T. The parallel algorithm (GPU2) employs the GPU to improve the CPU2 algorithm, each sphere is compared using the hierarchy, through one thread per sphere. The time per sphere is $O(log\ n)$. As we have n spheres and n threads running in parallel, the time remains as $O(log\ n)$. Algorithm GPUv2 considers 2 threads per node, one for the left child and the other for the right child. This reduces the time in the half.

4. Hierarchy Construction

Let $e_1, e_2, \ldots, e_n$, be a set of n spheres joined in a sequential order as depicted in Figure *3*. The second (CPU2) and fourth (GPU2) algorithms need a hierarchy for detecting collisions. This stage is implemented in CPU by algorithm aCPU2 or in GPU by algorithms aGPU2 and bGPU2.

4.1 The sequential version: Recursive Algorithm (aCPU2)

Spheres are taken in pairs, parents are generated in a bottom-up manner. This way, we have n leaf spheres in level h, $\frac{n}{2}$ spheres in level h-1, and so on until we achieve the root of the hierarchy in the first level. This means that the tree has $h=\lceil log_2\, n+1 \rceil$ levels. From Figure *4*, we can see that e_8 is parent of e_1, e_2; and e_{11} is parent of e_8, e_9. The children of sphere i are represented as $e_i(e_j, e_k)$, where e_j is the left child and e_k is the right child. Binary trees were chosen since the tubular objects are approximated by joining aligned spheres, where a sphere has only two neighbor spheres. Other kind of trees could be employed.

Tree nodes are stored in data structures as shown in Figure *5*. Both arrays, Tree and Spheres, have $2n$-1 elements. The first n locations of the array are occupied by the leaves of the hierarchy, the next n-1 locations are occupied by the inner nodes and are linked with the array locations of their children as illustrated in Figure *6*.

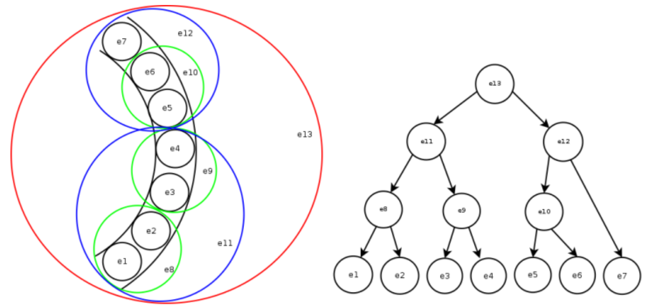

Fig. 4.Hierarchy construction using a bottom-up approach in a binary

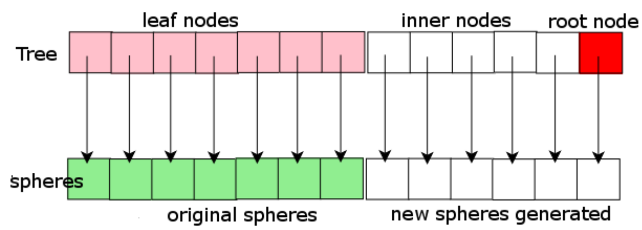

Fig. 5.The data structure to store the leaf spheres.

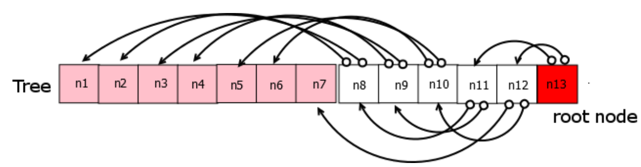

Fig. 6. Links of the resultant tree with 7 spheres.

Spheres are taken in pairs in level h-1 to generate their parents: $e_8(e_1, e_2)$, $e_9(e_3, e_4)$, …. The number of inner nodes is n-1. This way, the routine is recursively called with input E, the set of spheres. E is increased as the generation of new nodes, and the routine is called again in a recursive manner taken the children from e_n, testing the size of the spheres to envelop the corresponding spheres and setting the radius and center of the new sphere e_j. The maximum number of spheres created is n-1. Complexity time is $O(n\ log\ n)$.

A parent of two spheres is computed as illustrated in Figure *7*. The distance between two spheres $d(A,B)$, using the 3-vector Euclidean norm d is shown in equation (*1*).

$$\sqrt{(c2.x - c1.x)^2 + (c2.y - c1.y)^2 + (c2.z - c1.z)^2} \quad (1)$$

If d + *e-min.radius*≤ *e-max.radius* then the new *radius* is *r3* = *e-max.radius*, otherwise *r3* = *(d + r1 + r2)/2*. Where *e-min* is the sphere with the smaller *radius*, and *e-max* is the sphere with the greater *radius*.

The coordinates for the new center are also obtained:
if *(d + e-min.radius < e-max.radius)* then

c3 = e-max.center

else $c3 = r1\gamma + r2\gamma + \gamma$.
Where $\gamma = c1c2 \,/\, \| c1c2 \|$, is the unitized vector from *c1* to *c2*. So, the new sphere *C* encloses its children *A* and *B*.

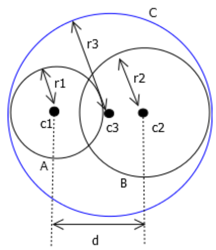

Fig. 7.Sphere C is constructed from spheres A and B.

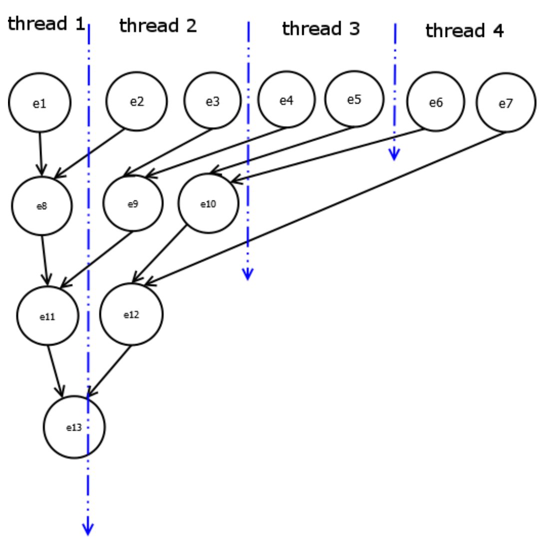

Fig. 8.The parallel hierarchy construction.

4.2 The basic parallel version bGPU2

Assume *n* leaf spheres in level *h*, we require a thread for a couple of spheres, that is $\frac{n}{2}$ threads. In the next level *h-1*, $\frac{n}{4}$ threads are required since there are $\frac{n}{2}$ spheres (Figure *8*). The process continues until the root tree is achieved. Complexity time is *O(log n)*, the height of the tree. Some graphics cards support at most *768* threads, that is *1,536* spheres, thus in the case we have more spheres, we use aGPU2, a new parallel algorithm explained in the next section.

4.3 The advanced parallel version (aGPU2)

There are two stages. In the first stage a kernel is launched that divides the *n* spheres in groups in such a way that they can be processed independently. As a result, subtrees are created: $SA=\{sa_1, sa_2, \ldots sa_r\}$ with $r \leq 768$. The roots of these groups form a new group of spheres $R=\{r_1,r_2,\ldots,r_r\}$, which can be processed using again the parallel algorithm aGPU2. This first kernel takes *O(log s)*, where *s* is the number of leaves of the tree. The second kernel accesses *R*, then it uses the basic parallel algorithm bGPU2 to generate the upper part of the tree, generating the set of spheres $S=\{s_1,s_2,\ldots,s_{r-1}\}$, so that it requires *O(log r)* time, being *r* the number of subtrees; in other words it is the cardinality of *SA*, |*SA*|.

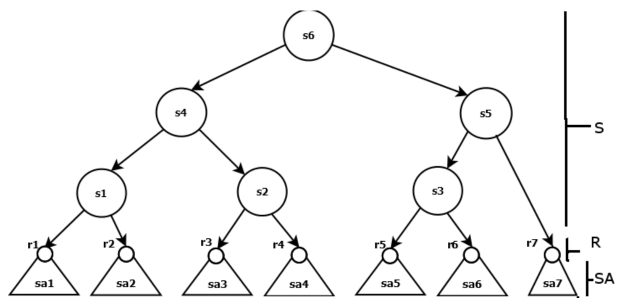

Fig. 9.Advanced parallel hierarchy construction.

Therefore, the hierarchy is formed by the union of the sets of spheres $SA \cup R \cup S$ as shown in Figure *9*. A different distribution of nodes is necessary in the array, as illustrated in Figure *10*. The algorithm aGPU2 needs a set of consecutive locations of nodes to be processed. *R*, the roots of subtrees, is written at the end of the *SA*s locations, the subtrees. Therefore, when the second kernel is launched, the spheres in *R* will be in the right locations to build the upper part of the tree.

Subtree sa_i can have at most *192* threads, where each one operates a pair of spheres. This results in *384* leaf spheres per sa_i. The number of subtrees sa_i depends on the number of threads allowed, *768*, which can operate *1,536* spheres, the roots of sa_i. Therefore, the maximum number of leaf spheres allowed is *1536* x *384* = *589,824*.

IJCSI International Journal of Computer Science Issues, Vol. 11, Issue 5, No 1, September 2014
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org

Fig. 10.New distribution of spheres in the array for the advanced parallel hierarchy construction

A summary of the time required by the three algorithms to construct the hierarchy is shown in Table *1*.

Table 1. Time required to construct the hierarchy.

Algorithm	**Time**
aCPU2	*O(n log n)*
bGPU2	*O(log n)*
aGPU2	*O(log s+r)*

5. Hierarchy Traversal

Bounding volumes are created in the hierarchy construction stage while self-collisions are determined in the hierarchy traversal stage.

Given two spheres $A(c_A, r_A)$ and $B(c_B, r_B)$, an overlap occurs between them if

$$d(A,B)^2 \leq (rA+rB)^2 \qquad (2)$$

This inequality verifies the squared distance between two spheres, using the 3-vector Euclidean norm and the squared of sum of radius of spheres.

To detect collisions, the animation is required. Spheres are updated automatically when object deforms due to the spheres depends on the polygons locations. New call to hierarchy construction algorithm is required when spheres modify its attributes.

5.1 The sequential version: Recursive Algorithm (CPU2)

Neighbor spheres are not considered as a collision. The tree traverse is performed in the node's children: left and right. A sphere is tested against the two nodes of level *1*, and cull away the sphere that is not colliding. For instance, *e1* is compared with *e11* and *e12*. The sphere colliding provides its two children to check for collisions. If *e1* collides with *e11*, then *e1* must be compared with *e8* and *e9*. This process continues until a collision with a leaf node occurs or no more children exist. It could be possible that more than one couple of spheres collide. As it can be seen, this is a top-down approach.

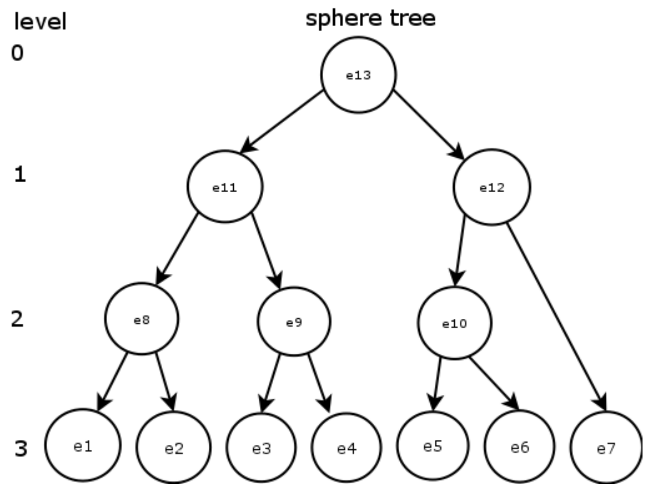

Fig. 11.Tree traversal to self-collision detection.

The process consists of testing e_i vs T. This process is required for each sphere, so that it takes *O(n log n)* time. At the end, an array of collisions is returned, where each location counts the number of collisions with other spheres in the chain that represent the tubular object.

5.2 The parallel version (GPU2)

This implementation utilises the GPU to improve the sequential version, the same manner each sphere is compared using the hierarchy, but one thread per sphere, as depicted in Figure *11*. Recursive calls are not supported in GPUs, so we use a stack to keep the array keys processed to get explicit recursion. Tree traversal is performed for each sphere e_i vs T running in parallel simultaneously. Then it takes *O(log n)* time.

In order to improve the parallel version, we launch two threads per sphere, instead of one (GPUv2). Tree traversal is performed as the same manner, but thread *1* processes the left subtree and thread 2 the right subtree for each sphere e_i vs T running in parallel simultaneously. This way, the time required is ½ *O(log n)*.

A summary of the time required by the three algorithms to traverse the hierarchy is shown in Table *2*.

Table 2. Time required to traverse the hierarchy.

Algorithm	Time
CPU2	*O(n log n)*
GPU2	*O(log n)*
GPU2v2	½ *O(log n)*

6. Empirical results

The algorithms were run in a PC desktop Intel Xeon CPU E5620 with *12.0* GB RAM DDR, operating system Windows *7* of *64* bits, NVIDIA GeForce 590 GTX Graphics Card. We used Microsoft Visual C++ and CUDA SDK *5.0*. The experiments compare algorithms by testing their implementations.

Animation was required to make the experiments. The animation consists in determine a number of points and then generate the *path* where the snake-like object (spheres) go through. The *path* is formed by the coordinates of keyframes generated by interpolation. The interpolation method utilizes splines via a cubic tracer. In Figure *12*, path *1* is defined as a sinusoidal signal shape, while in Figure *13*, path *2* is defined as circles.

Two different paths were used to test the algorithms. CPU1 algorithm, brute force, needs *1,500 ms* or more to process *10,000* spheres so we decided not to execute more cases for this version. The other algorithms took at most *1* ms with *10,000* spheres or less. Thus, Table *3* and Table *4* contain runtimes for *10*, *20*, *30*, *40*, *50*, *100* and *150* thousands of spheres, for path1 and path2, respectively. Both paths generate a maximum of *45* pairs of collisions with *56* and *114* keyframes of animation.

Results for hierarchy construction are shown in Figure *14* and Figure *15*, while results for hierarchy traversal are shown in Figure *16* and Figure *17*. The algorithm aGPU2 is faster than algorithm aCPU2, using *20,000* spheres or more (Figures *14*, *15*).

For hierarchy traversal, despite the use of the GPU, the GPU1 algorithm is the slowest. The CPU2 version, that uses the hierarchy, has a discrete performance, with good runtimes till *50,000* spheres, but it is exceeded by algorithms GPU2 and GPUv2, when the number of spheres increases (Figures *16*, *17*). The latter algorithms, have a good performance in most of the cases.

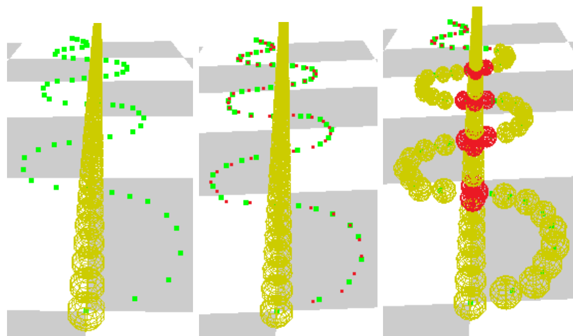

Fig. 12. Path 1: starting points, interpolation and collision detection.

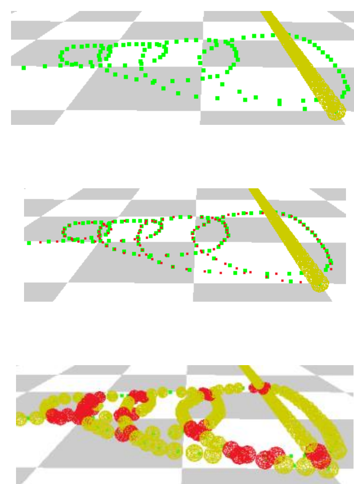

Fig. 13. Path 2: starting points, interpolation and collision detection.

4. Conclusion

Two versions of hierarchy construction algorithm and five versions of the algorithm to detect collisions were implemented. We investigated the performance of the implemented versions. The object used was a tube approximated with spheres and the animation employed has a snake-like motion.

Results shown that the parallel versions are suitable for more than *20,000* spheres, where the power of parallelism is exploited. GPU architecture has very high available parallelism that our algorithms take advantage to get a better performance. However, it would be possible to improve the throughput of the algorithms through the use of shared memory, constant memory, or minimizing the divergence. Finally, we would like to explore other applications of our algorithms, such as collision detection in fluids or particles, and other kind of trees: octree, quadtree.

Acknowledgments

We would like to thank to the Universidad Autónoma de Yucatán and the CONACYT México for their financial support.

References

[1] Sobottka, G., Varnik, E.,Weber, A.: *Collision detection in densely packed fiber assemblies with application to hair modeling*. In: The 2005 International Conference on Imaging Science, Systems, and Technology: Computer Graphics, pp. 244–250 (2005)

[2] Craig, J.J.: *Introduction to Robotics: Mechanics and Control*, 2nd edn. Addison-Wesley, Reading (1989).

[3] Kubiak, B., Pietroni, N., Ganovelli, F., Fratarcangeli, M.: *A robust method for real-time thread simulation*. In: Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology, pp. 85–88 (2007).

[4] Schmidl, H., Walker, N., Lin, M.C.: *CAB: fast update of OBB trees for collision detection between articulated bodies*. J. Graph. GPU Game Tools 9, 1–9 (2004).

[5] Lotan, I., Schwarzer, F., Halperin, D., Latombe, J.-C.: *Efficient maintenance and self-collision testing for kinematic chains*. In: Proceedings of the Eighteenth Annual Symposium on Computational Geometry, pp. 43–52 (2002).

[6] F. Madera, C. Herrera, S. Laycock. *Ray-Triangle Collision Detection to Approximate Objects with Spheres. The IASTED International Conference on Computer Graphics and Imaging, 2013*.

[7] Bradshaw and C. O'Sullivan. *Adaptive medial axis human approximation for sphere-tree construction.*ACM Transactions on Graphics, 23(1):1–26, 2004.

[8] D. James and D. Pai. *Bd-tree: Output-sensitive collision detection for reduced deformable models.*ACM Transactions on Graphics, 23(3), 2004.

[9] G. van den Bergen. *Efficient Collision Detection of complex Deformable Models using AABB Trees.* Journal of Graphics tools, vol. 2, No. 4, pp. 1-14, 1997.

[10] Thomas Larsson, Thomas Akenine-Moller. *Collision Detection for Continuously Deforming Bodies*, Eurographics, Manchester, UK, pp. 325-333, 2001.

[11] S. Gottschalk, M.C. Lin, D. Manocha. *A Hierarchical Structure for Rapid Interference Detection*, Proceedings on SIGGRAPH, New York, pp. 171-180, 1996.

[12] Li H., Yezzia J., Cohen L. *3D Multi-branch tubular surface and centerline extraction with 4D iterative key points*,In MICCAI (1) (2009), Yang G.-Z., Hawkes D. J.,Rueckert D., Noble J. A., 0002 C. J. T., (Eds.), vol. 5762of Lecture Notes in Computer Science, Springer, pp. 1042–1050.

[13] R. Wiemker, T. Klinder, M. Bergtholdt, K. Meetz, I. Carlsen, T. Bulow. *A Radial Structure Tensor and its use for Shape-Enconding Medical Visualization of Tubular and Nodular Structures.* IEEE Transactions on Visualization and Computer Graphics.Vol. 19, No. 3, pp 353 – 366, 2013.

[14] V. Luboz, J. Kim-Tun, S. Sen, R. Kneebone, R. Dickinson, R. Kitney, F. Bello.*Real-time stent and ballon simulation for stenosis treatment.*The Visual Computer Journal, vol. 30, No. 3, pp 341 – 349, 2014.

[15] Evan Shellshear. *1D sweep-and-pune self-collision detection for deforming cables.*The Visual Computer Journal, vol. 30, No. 5, pp 553 – 564, 2014.

[16] M. Burtscher, K. Pingali. *An efficient CUDA implementation of the tree-based barnes-hut n-body algorithm.* In GPU Computing Gems Emerald edition, pp 75 – 92, Morgan Kaufmann, 2011.

[17] Paul Rosen. *A Visual Approach to Investigating Shared and Global Memory Behavior of CUDA Kernels.* EuroGraphics Conference on Visualization.pp 161 – 170, 2013.

[18] X. Zhang, Y. J. Kim.*Scalable Collision Detection using p-Partition Fronts on Many-Core Processors.* IEEE Trans. On Visualization and Computer Graphics, vol. 20 No. 3, pp 447 – 456, 2014.

Enrique Ayala is a lecturer in Computer Sciences at Universidad Autónoma de Yucatán, in Mérida, México. He received a Master Degree in Distributed Systems and Networks from the Instituto Tecnológico y de Estudios Superiores de Monterrey, México, in 2002. He received a BS in Computer Systems Engineering from the Instituto Tecnológico de Morelia, México, in 1993. His research interests include Computer Networks, Parallel and Distributed Computing and GPU Programming.

Francisco A. Madera received his B. Sc. Degree from the Universidad Autónoma de Yucatán, México; his PhD from the University of East Anglia, UK. Dr. Madera teaches subjects related to computer graphics and videogames development; and his research is focused on collision detection and GPU programming.

Francisco Moo-Mena is a Professor in Computer Sciences at Universidad Autónoma de Yucatán, in Mérida, Mexico. From the Institute National Polytéchnique de Toulouse, in France, he received a Master Degree in Computer Science and a PhD, in 2003 and 2007, respectively. He also received another Master Degree in Distributed Systems from the Instituto Tecnológico y de Estudios Superiores de Monterrey, México, in 1997. His research interests include Parallel and Distributed Computing, CUDA, Self-healing systems, and Web services Architectures.

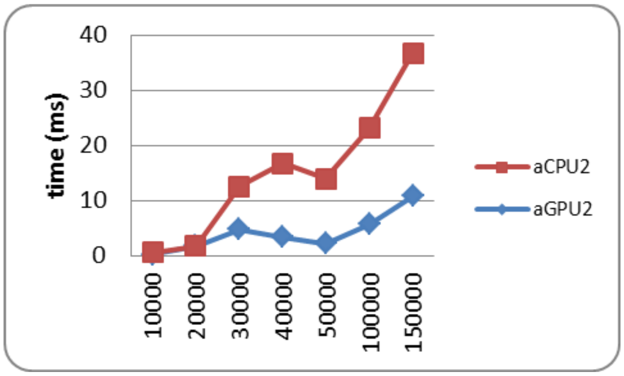

Fig. 14. Results for hierarchy construction with path 1.

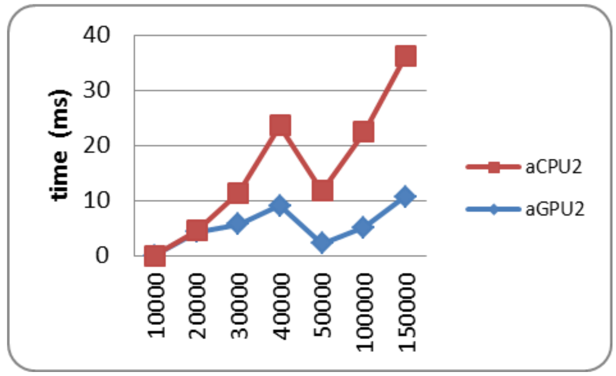

Fig. 15. Results for hierarchy construction with path 2.

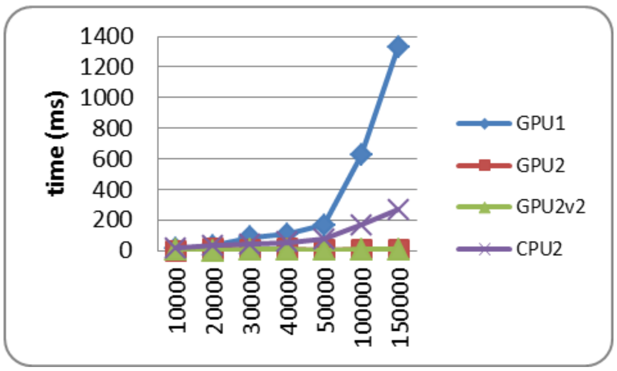

Fig. 16. Results for hierarchy traversal with path 1.

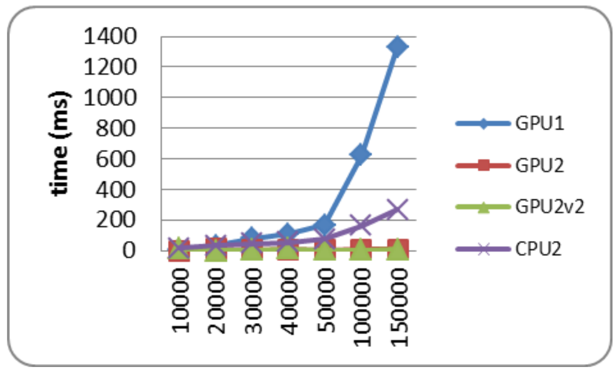

Fig. 17. Results for hierarchy traversal with path 2.

Table 3. Runtimes in *ms* for the hierarchy construction and hierarchy traversal with path1.

HIERARCHY CONSTRUCTION							
n	10,000	20,000	30,000	40,000	50,000	100,000	150,000
aGPU2	0,28	1,71	4,75	3,32	2,21	5,75	10,83
aCPU2	0,28	0,00	7,76	13,37	11,69	17,33	25,87
HIERARCHY TRAVERSAL							
GPU1	9,78	31,00	85,46	109,16	164,58	624,87	1331,55
GPU2	0,28	13,71	3,05	11,17	4,71	7,80	6,14
GPU2v2	3,66	0,28	6,98	5,80	3,05	5,32	6,07
CPU2	15,50	31,26	39,51	51,00	73,83	166,08	265,19

Table 4. Runtimes in *ms* for the hierarchy construction and hierarchy traversal with path2.

HIERARCHY CONSTRUCTION							
n	10,000	20,000	30,000	40,000	50,000	100,000	150,000
aGPU2	0,00	4,35	5,70	9,12	2,21	5,13	10,72
aCPU2	0,00	0,13	5,65	14,51	9,53	17,27	25,46
HIERARCHY TRAVERSAL							
GPU1	1,37	30,85	80,03	107,40	164,27	622,50	1333,05
GPU2	0,13	11,50	3,79	4,95	1,79	7,07	5,57
GPU2v2	11,20	0,00	3,45	12,43	1,46	4,22	7,90
CPU2	15,58	31,30	43,36	52,68	73,98	163,52	266,29