Performance Evaluation of a Hybrid Algorithm for Collision Detection in Crowded Interactive Environments

Rafael de Sousa Rocha[1] Maria Andréia Formico Rodrigues[2] Leandro da Silva Taddeo[2]
[1]*Informática, Universidade de Fortaleza (UNIFOR), Fortaleza, Brazil*
[2]*Mestrado em Informática Aplicada, Universidade de Fortaleza (UNIFOR), Fortaleza, Brazil*
rafaelrocha@edu.unifor.br, mafr@unifor.br, taddeo@unifor.br

Abstract

Crowded interactive environments composed of a large number of objects need a fast, accurate and scalable mechanism for collision detection. This work presents a detailed performance analysis of a hybrid collision detection algorithm for highly interactive and crowded environments. Extensive tests were conducted and the performance of the algorithm was evaluated in terms of output quality and running time, by applying a usability criteria. The results show that interactive frame rates for environments composed of 1000 colliding objects can be successfully achieved with a good level of user satisfaction using the Sweep & Prune algorithm together with sphere-trees generated by the Combined algorithm.

1. Introduction

In the last years, new computer technologies have been developed to achieve more realism and enhance the level of interactivity supported by virtual environments [1]. An important issue related to both realism and level of interactivity of virtual environments is the approach used for collision handling [2].

Basically, the collision handling can be divided in two steps: collision detection and response. The collision detection aims at reporting collisions and delivering specific information to the collision response, so that the latter can compute the responses accordingly.

Collision detection algorithms can be classified into the following categories: continuous and discrete [3]. Continuous algorithms utilize time-parameterized equations to compute the first time of contact between colliding objects, as well as their contact state. By contrast, discrete algorithms sample the object's trajectory and report interpenetrations. Despite the former being more accurate, the latter fits better in interactive environments with high scalability, such as those composed of a large number of objects.

To speed up the collision detection, a traditional approach is to separate the problem into two distinct phases: broad and narrow. The broad phase collision detection aims at efficiently cull out pairs of non-intersecting objects, discarding as many pairs of objects as possible. The collision detection process is refined in the narrow phase, when the remaining pairs of objects are analysed more accurately. The objects' geometry or an approximation of it may be used in the narrow phase. Pairs of objects are then validated, issued as colliding objects, and passed to the response module.

This work presents a performance evaluation of collision detection strategies in highly interactive and crowded environments, in which the user interacts with 3D moving objects via a joystick with force feedback. As the main motivation of our work, we address the question of whether there exists an effective trade-off between speed, accuracy, and scalability of some existing collision detection schemes that can be combined to achieve the best possible performance for the scenarios we implemented. In particular, for the broad phase collision detection, we analysed, measured, and compared four algorithms: Brute Force, Grid [4], Octree [5] and Sweep & Prune [6]. For the narrow phase, we implemented an algorithm based on sphere-trees [7], and compared two algorithms for building this hierarchical structure: Spherical Octree [2] and Combined [8]. The main contribution of our work is the demonstration that a hybrid algorithm that uses Sweep & Prune (for the broad phase) and sphere-trees generated by the Combined algorithm (for the narrow phase) is fast, accurate, and scalable. We show that this approach is recommendable for applications with a large number of colliding objects

(approximately 1000, no matter their geometric complexity) and that it also achieves good levels of effectiveness (frame rate) and user satisfaction. Specifically, we were able to achieve at least a rate of 20 frames per second (fps) for a number of colliding objects spanning from 100 to 700, and 10fps for objects spanning from 600 to 1000.

The remaining of the paper is organized as follows. In Section 2, we provide an overview of related work. Section 3 presents four traditional broad phase collision detection methods, as well as their main advantages and limitations. In Section 4, two algorithms for building sphere-trees for the narrow phase are evaluated. Section 5 presents the scenarios we designed and implemented. Various experiments and the obtained results are discussed in Section 6. The algorithms are then evaluated in terms of output quality of the running time, using a subjective analysis and usability criteria in Section 7. Finally, conclusions and future work are summarized in Section 8.

2. Related work

Before actually starting the narrow phase, bounding volumes (BVs) can be applied to quickly reject pairs of objects that do not collide. BVs are usually simple geometric objects, such as axis aligned bounding boxes (AABB) [6] and spheres [9]. Although they are very useful, testing all the BVs is $O(n^2)$, where n is the number of objects. Decreasing this complexity is the broad phase's role.

An important aspect of the broad phase is the concept of spatial coherence (objects that are located far away from each other do not need to be tested for collision). Spatial data structures can be used to decompose the virtual environment into cells (usually voxels) to take advantage of spatial coherence. Regular grids, octrees and BSP-trees [10] are some examples of spatial data structures used with this purpose. There are also approaches other than spatial partitioning data structures. The Sweep & Prune algorithm, for example, is based on temporal coherence. This approach can compute intersecting AABB pairs in nearly $O(n+m)$, where n is the total number of objects and m is the number of colliding objects. Hubbard uses space-time bounds to implement the broad phase [7]. This approach avoids the collision-tunneling problem (objects passing through each other between frames), which occurs in discrete algorithms.

In the narrow phase, the remaining pairs from the broad phase are further processed. For most interactive systems it is not possible to test the actual geometry of the objects. Instead, conservative approximations can be used. One such approximation is bounding volume hierarchies (BVHs). AABB [11], oriented bounding boxes (OBB) [12], spheres and k-DOPs [13] have been successfully used to build approximations of an object's geometry. In particular, Hubbard describes a collision detection algorithm for interactive applications based on sphere-trees [7] and shows three algorithms for automatically building this structure. Recently, Bradshaw has proposed a very efficient method for building sphere-trees that combines a set of sphere reduction algorithms [8].

Currently, image-based techniques which can be executed on graphics processing units (GPUs) have also received attention [14]. However, reading buffers from GPU memory is usually a very slow operation, making buffer readbacks unappealing [1].

Although many approaches have been proposed for collision detection, few papers have been published on crowded environments, taking the level of interactivity achieved during the simulations into account. By contrast, in our experiments, we implemented a highly interactive environment composed of a large number of colliding 3D objects, where the user can navigate and interact with them using a force feedback joystick.

3. Broad phase collision detection

In the following sections, we initially investigate a Brute Force algorithm for the broad phase, only for purpose of comparison with other well-known broad phase methods. This algorithm is used to test all the objects' bounding volumes for intersection and is $O(n^2)$. Then, to improve on this initial algorithm, in Section 3.1 we implemented two spatial partitioning data structures: grids and octrees. Finally, in Section 3.2, we implemented the Sweep & Prune algorithm, an approach that does not partition the space.

3.1. Grid and Octree algorithms

A simple way to enhance the initial approach is to partition the virtual environment space into cells and, for each cell, execute the Brute Force method. We have implemented an algorithm that uses regular 3D grids to partition the space into voxels [4]. Each voxel contains a list of objects within its region and has a Brute Force method to compute a set of pairs of objects whose BVs intersect. The main advantage of using grids is that these data structures can be used statically. It means that the grid granularity does not have to be always updated, and the objects can be updated in the grid very quickly. In fact, the granularity may never change and can be computed in

a pre-processing time. However, grids have still serious limitations: the difficulty of estimating the best grid granularity for different scenarios, the likelihood of objects to belong to more than one cell (requiring extra use of memory), and the fact that grids do not adapt themselves to objects' distribution in a scene.

To build the grid, we first create an AABB that surrounds all the space considered for collision. Then, we fill this AABB with smaller, non-overlapping AABBs, depending on the grid granularity, so that they cover all the space. Each of these AABBs corresponds to the voxels of the grid. Therefore, finding out the objects' location inside the grid is straightforward, by intersecting their bounding volume with the correspondent voxel. The center of the objects' bounding volumes may indicate an initial voxel to be tested. After that, a flooding algorithm is used to find out the remaining cells. Once all the objects are organized into the voxels, we loop through the voxels (not all of them, instead, when we update the objects in the grid, we keep track of those cells that had updates and pass them to the collision detection method) and test the objects in the voxels' list for intersection. For this, we invoke the Brute Force method for each voxel.

The grid may be extended in a hierarchical way to build octrees. With this in mind, we extended our grid implementation to support an octree structure, partitioning the cells in a new grid when required (with granularity 2x2x2). Differently from the grids, octrees have the property of self-adapting to the objects' distribution. However, the cost for this is very high because the structure needs to be updated constantly. In our implementation, the octree is updated at each frame, what causes a significant overhead. Another problem is that the octree requires large amount of memory due to the number of cells and because objects may be located in more than one cell.

The process of building the octree structure is as follows. Firstly, an AABB that surrounds the space considered for collision is created and becomes the root of the tree. After this, all the objects inside the root node are inserted in the node's object list, and the node is partitioned in a 2x2x2 grid. Each voxel of the grid becomes a new octree node, and this process is repeated using the objects of its parent's list, until a target depth (the distance from the root) is reached. After building the octree structure, we start processing the root node by traversing the tree down the leaves. Whenever a leaf node is reached, the Brute Force method is invoked to compute the object pairs among the objects, within the leaf. Otherwise, the algorithm recurs on the current node's children.

3.2. Sweep & Prune algorithm

An AABB can be represented by its three intervals (one for each axis) and, in particular, two AABBs intersect if and only if all their three intervals intersect [6]. The Sweep & Prune algorithm keeps all the AABBs' intervals in three separated sorted lists (one for each axis) and takes advantage of frame coherence. It means that the lists from the previous frame are nearly sorted in the current frame, because the objects do not move too far between frames. An insertion sort algorithm is then used to keep the lists sorted in nearly linear time [11]. Analysing the adjacent intervals in the lists can consequently derive all the colliding object pairs for the broad phase. In particular, the algorithm we implemented is described in [1] and does not need to tune any parameters, such as the grid granularity or the octree depth, for example. In some circumstances, however, we observed that there is a likelihood of generating some specific objects' distributions in the scene, such as line arrangements of objects. As a result, this behaviour may cause many intersection tests, where most of the intervals in one of the lists do intersect. In spite of this, the Sweep & Prune algorithm updates its structure (specifically the three lists) in a fast manner and requires less intersection tests than the methods discussed previously.

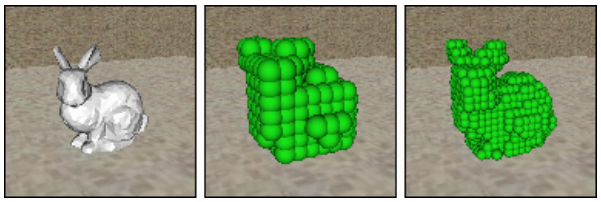
4. Narrow phase collision detection

In this Section, we implement a robust algorithm based on BVHs for the narrow phase [1]. We choose sphere-trees as BVHs because spheres are rotationally invariant, the intersection test between them is computationally very cheap, and there are many efficient algorithms for building sphere-trees. The simplest algorithm to build sphere-trees relies on octrees and is described in Section 4.1. Section 4.2 presents the Combined algorithm, a more sophisticated approach that integrates a set of sphere reduction algorithms to compute a very accurate approximation. We assume that the root of the sphere tree (level 0) corresponds to the object's bounding sphere.

4.1. Spherical Octree algorithm

A modified version of the octree structure developed for the broad phase can also be utilized in the narrow phase. We adapted our octree structure to partition polygons of an object instead of objects of a scene, that is, we need to test the polygons against AABBs, which represent the octree's cells. The octree can then be used as a hierarchical approximation of the

object, where the object's AABB is the root. Once the octree is built, a straightforward way for building a sphere-tree is to circumscribe each node of the octree, namely spherical octrees. In (b) and (c) of Figure 1, are shown the object bunny's spherical octree with 4 and 5 levels, respectively. Although this subdivision process is fast, octrees need many levels of subdivision to converge to the object's geometry. Consequently, spherical octrees may be deep, when compared to the approaches from Section 4.2, and does not generate an acceptable approximation. As a result, this approach degrades the performance of our narrow phase algorithm.



(a) (b) (c)

Figure 1. The object's geometry and the object's spherical octree with 4 and 5 levels are shown in (a), (b), and (c), respectively.

4.2. Combined algorithm

Motivated by the fact that spherical octrees do not fit objects tightly (because they do not take into account the object's geometry), Hubbard proposed an algorithm based on the medial axis (a skeleton-like shape structure based on the object's geometry), and uses this structure to build sphere-trees [7].

Many algorithms to build tight sphere-trees have been published, including variations of Hubbard's algorithm. For example, a simple way to enhance the Spherical Octree algorithm is to use hierarchical grids instead of octrees, allowing more freedom in the subdivision, and, additionally, optimizing the orientation of the grid and the size of the spheres. The algorithms Merge, Burst and Expand are all variations of Hubbard's algorithm. All these algorithms improve on Hubbard's algorithm by reducing the number of spheres, each in a different way, so that they generate tighter sphere-trees. Another relevant algorithm is the Spawn algorithm, which attempts to reduce the number of spheres in the sphere-tree, although it does not use the object's medial axis. Bradshaw implemented an approach that allows the use of different sphere reduction algorithms in conjunction [8]. In this way, for each set of spheres, the algorithm searches for the approach which results in the lowest error, converging to a very tight approximation of the object (Figure 2).

(a) (b) (c)

Figure 2. Sphere-tree generated using the Combined algorithm with 2, 3, and 4 levels are shown in (a), (b) and (c), respectively.

5. Scenarios of the application

The environment of the application is represented by a room with dimensions 100x25x100, which is populated with a large number of moving objects (from 100 to 4000), whose dimensions are approximately 1x1x1. The scenario is composed of two types of objects: boxes (composed of 12 triangles) and bunnies (composed of 1500 triangles). In Figure 3 is exhibited one of our scenarios with the avatar located in the center of the scene, circumscribed by a circle.

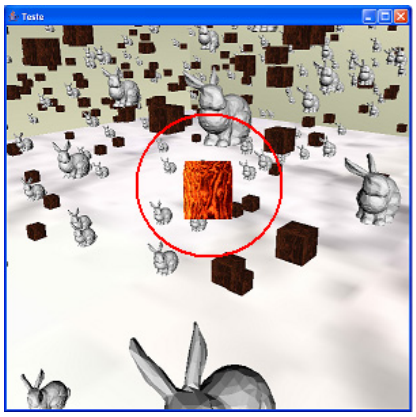


Figure 3. One of the scenarios implemented to run the experiments.

Each object in the environment has linear velocity and is not allowed to rotate. Thus, the AABBs do not need to be recalculated. When objects collide, one or more components of their velocities are negated as a response to the collision. The user of the application can control an avatar through a joystick with force feedback. A virtual joystick was also implemented so that the user can realize the latency of the movements applied to the real joystick. Additionally, the real joystick offers to the user a great immersion, particularly when there is a collision between the avatar and another object in the scene.

6. Experiments and results

The scenarios we designed to carry out the experiments were implemented with the graphics package Java3D [15]. The experiments were conducted by varying the number of objects and the algorithms for collision detection. These experiments were made on a 3.06 GHz PC with 768 MB of memory. Two different types of movements were implemented (in the plane XZ and in the space XYZ), but similar results were obtained. Therefore, only the results for the movements in the space XYZ are presented. The algorithms' performance was evaluated based on the average frame rate generated in each experiment. For calculating this value, we considered only the total time needed to process the collision detection. Actually, this value includes the time to update the structures involved in the detection, the time to process the broad phase algorithm, and the time to process the narrow phase.

Initially, we carried out extensive experiments exclusively using broad phase algorithms. The collision response was implemented applying a heuristic algorithm that considers the AABBs' intervals and determines which components should be negated. We evaluated four algorithms: Brute Force, Grid, Octree and Sweep & Prune. It is important to note that the algorithms that use grids and octrees need specific parameters: grid granularity and octree depth, respectively. For the scenarios implemented, the best parameters found in our experiments were a granularity of 10x4x10 and a depth of 3 for the grid and octree, respectively. We use these parameters to compare the broad phase algorithms.

The performance curve of the Brute Force, Octree, and Grid algorithms, with the number of objects varying from 100 to 500, is shown in Figure 4, where we can notice the superior performance achieved by partitioning data structures (grid and octree) when compared to the Brute Force algorithm. Furthermore, in this experiment, the Grid algorithm has superior performance over the Octree algorithm. For example, for 500 objects, the Grid algorithm has a performance of 12fps, while the Octree algorithm achieves a rate of approximately 34fps. This happens because the octree is dynamically built at each frame, while the grid is built in a pre-processing time and is not updated during the execution of the application.

Although the superior performance of the Grid algorithm over the Octree one, the former still presents serious performance limitations when compared to the Sweep & Prune algorithm (Figure 5). The latter reaches a high frame rate without drastically degrading its performance while the number of objects increases. For 1000 objects, for example, while the Grid algorithm has a performance of approximately 14fps, the Sweep & Prune algorithm achieves a rate of approximately 54fps. This demonstrates that this approach is not only fast, but also scalable and, therefore, the most suitable to detect collision in the broad phase of our interactive environment.

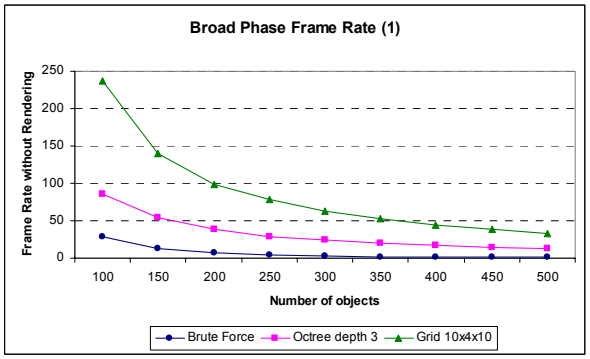


Figure 4. Performance curves of broad phase collision detection algorithms (Brute Force, Octree, and Grid).

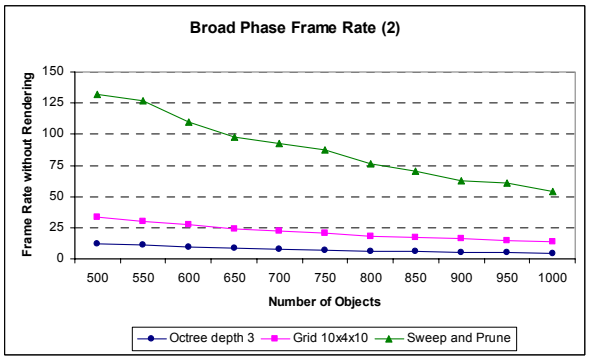


Figure 5. Performance curves of broad phase collision detection algorithms (Octree, Grid, and Sweep & Prune).

To analyze the differences between the Grid and the Sweep & Prune algorithms, we compared the time spent on updating the data structures and on processing the broad phase during 500 frames of the animation (Figure 6). In particular, there is a huge disparity between both algorithms in the time spent on processing the broad phase. The Sweep & Prune algorithm computes the colliding pairs in much less time. The data structure updating time is relatively greater in the Sweep & Prune algorithm than in the Grid one (the former varies from approximately 78% to 98%, whereas the latter varies from 53% to 62%). This fact motivated us to explore a hybrid approach

where the Sweep & Prune and the Grid algorithms are combined (each voxel of the grid has a Sweep & Prune method to compute the colliding pairs). However, the results found were inferior to the ones obtained using only the Sweep & Prune algorithm.

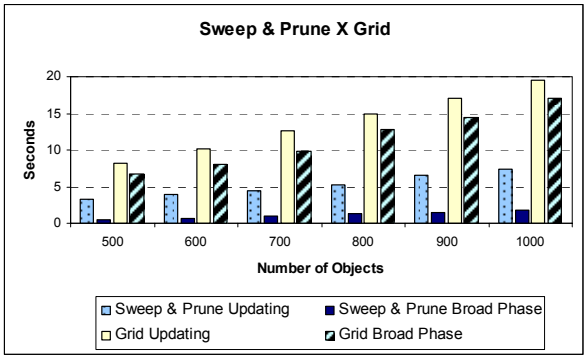


Figure 6. Time spent on updating the data structures and processing broad phase algorithms during 500 frames of the animation.

New experiments were conducted, now processing the narrow phase too. In this stage, we used approximations of the objects and carried out a refinement process in levels of details for the collision detection, by using sphere-trees. To automatically build the sphere-trees, two specific algorithms were used: Spherical Octrees and Combined. When there is a collision between two sphere-trees, the pairs of spheres (one from each sphere-tree) that do intersect are used to assist in the collision response (they help to choose which components of the velocity will be negated). The experiments in the narrow phase used Sweep & Prune, the algorithm that achieved the best performance in the previous tests for the broad phase.

The octree implemented in the broad phase was then extended to build sphere-trees. However, sphere-trees built in this manner are inaccurate, because they need many levels to converge to the object's geometry. For this reason, we built octrees with 4 and 5 levels to generate satisfactory approximations of the object bunny. On the other hand, due to the simplicity of the object box´s geometry, for generating its geometric approximation we built an octree with 2 levels only. As an alternative to this approach, we also used the Combined algorithm, presented at Section 4.2. This algorithm generates very accurate approximations of the object bunny. In addition, it provides a way to build sphere-trees with 2 and 3 levels, with approximations even better than the ones built with the Spherical Octree algorithm with 4 and 5 levels.

The results we have obtained with the implementation of these four approaches (Spherical Octree algorithm with 4 and 5 levels; Combined algorithm with 2 and 3 levels) are shown in Figure 7. The performance of the narrow phase algorithm supported by sphere-trees is directly affected by the structure's depth. Therefore, the Combined algorithm is more suitable for our scenarios because it builds more accurate sphere-trees, with low depth.

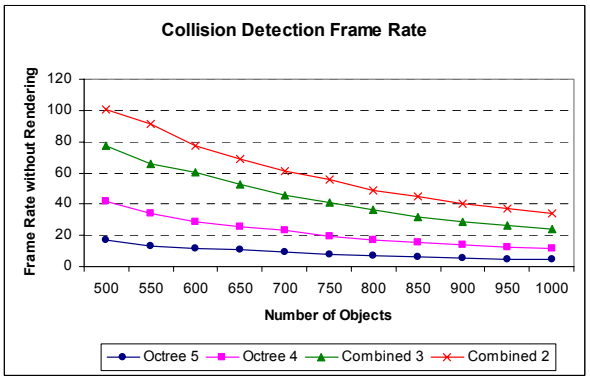


Figure 7. Performance curves of the hybrid algorithm with different approaches for building sphere-trees.

Figure 8 shows the time spent by the collision detection process (data structure updating, as well as broad and narrow phases) during 500 frames of the animation. For this experiment, we used the Sweep & Prune algorithm and sphere-trees with 2 levels, built by the Combined algorithm. Note that the time needed to update the lists of the Sweep & Prune algorithm clearly dominates the time spent during collision detection. Also, the time needed for the broad phase was greater than the one spent on the narrow phase.

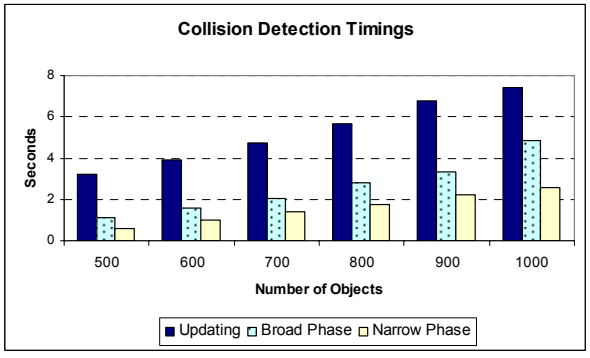


Figure 8. Time spent on updating the data structures and processing the collision detection, during 500 frames of the animation.

To evaluate then the scalability of our approach (besides speed and accuracy), we combined the Sweep & Prune algorithm (broad phase) with sphere-trees (narrow phase) in a scenario extremely crowded, with up to 4000 moving objects. Our experiments demonstrate that with a slightly more than 2000 objects, it is possible to obtain a performance of approximately 9fps (without rendering). For 4000 objects, however, the performance was unacceptable (approximately 2fps).

Finally, we evaluated the performance of the Sweep & Prune algorithm combined with sphere-trees, including the rendering process. Figure 9 shows the frame rate obtained considering the total time needed to render the scenes. Note that even when the environment is populated by 1000 moving objects, it is still possible to obtain a rate greater than 10fps, using sphere-trees with 2 and 3 levels. More specifically, sphere-trees with 2 levels resulted in a better frame rate (approximately 14fps) when compared to sphere-trees with 3 levels (approximately 12fps).

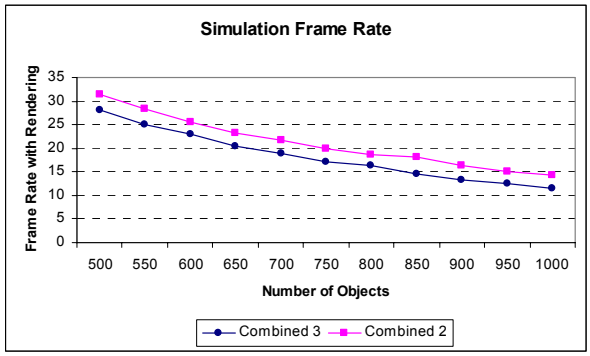


Figure 9. Performance curves of the hybrid algorithm, including rendering.

7. Usability Analysis

Usability is an important aspect to analyse the level of interactivity of virtual environments and is used in this work to evaluate the performance of the hybrid algorithm, based on the experiments from Section 6.

According to ISO 9241-11, the dimensions of usability are: effectiveness, efficiency, and satisfaction. Effectiveness measures usability from the point of view of the output of the interaction. Efficiency relates effectiveness of interaction to resources expended. Satisfaction refers to the comfort and acceptability of using the system. We validated the quality of the output of the interaction (perceptions) based on level of usability of the system implemented using the effectiveness (frame rate) and satisfaction (subjective user analysis) criteria [16]. Minimal frame rates that range from 6Hz to 20Hz are suggested as acceptable frame rates for interactive virtual environments [11], whereas values spanning from 60Hz to 75Hz are presented in the literature as theoretical targets still to be reached [17].

Table 1 shows the overall level of effectiveness and satisfaction obtained with the implemented hybrid algorithm. The results indicate a high level of user satisfaction for scenarios with up to 350 objects and a very good satisfaction for simulations with the number of objects ranging from 350 to 500 ($26 \leq$ frame rate ≤ 40). User satisfaction was also good for scenarios with the number of objects ranging from 500 to 1000 ($10 \leq$ frame rate ≤ 25).

Table 1 – The overall level of effectiveness and satisfaction obtained with the hybrid algorithm

Effectiveness / Frame Rate (fps)	Level of Satisfaction / Performance
< 10	low
≥ 10 and ≤ 25	good
≥ 26 and ≤ 40	very good
> 40	high

8. Conclusions and future work

We implemented and evaluated the performance of four algorithms for the broad phase (Brute Force, Grid, Octree, and Sweep & Prune). The Sweep & Prune algorithm achieves the best performance, and among the algorithms we tested is the most scalable for broad phase collision detection in interactive environments.

In the narrow phase, we used an algorithm supported by sphere-trees. Two algorithms for building these structures (Spherical Octree and Combined) were compared. We showed that the Combined algorithm builds much tighter sphere-trees, and therefore is more suitable for narrow phase collision detection in interactive environments. We also evaluated a hybrid algorithm that uses the Sweep & Prune algorithm and sphere-trees. This approach is fast (it carries out only simple intersecting tests), scalable (the Sweep & Prune algorithm discards non-intersecting objects very efficiently) and accurate (the Combined algorithm generates very precise sphere-trees). The hybrid algorithm was successfully used in crowded interactive environments. We were able to achieve very good frame rates (between 26 and 40) for scenarios with up to 500 objects and good frame rates (between 10 and 25) for scenarios with 1000 objects.

Finally, according to our analyses and experiments, the proposed hybrid algorithm achieves a much higher performance than other combined methods for the broad and narrow phases. For example, the Grid and

Spherical Octree algorithms obtain low values for accuracy, scalability and speed; the Grid and Combined algorithms obtain good levels of accuracy, but low scalability and speed; and the Sweep & Prune and Spherical Octree algorithms obtain intermediate values for speed and scalability, but with low accuracy. Besides, the investigation of pure methods shows that for the broad phase the Sweep & Prune algorithm is about 75% faster than the Grid one; for the narrow phase when using the Spherical Octree algorithm with 4 levels (instead of 5 levels) we speed up the algorithm by about 56%, and when using sphere-trees with 3 levels, built by the Combined algorithm (instead of Spherical Octrees with 4 levels), we gain approximately 45% in speed. Additionally, sphere-trees built by the Combined algorithm are much more accurate.

As future work, we aim at improving the Sweep & Prune algorithm motivated by our experiments using the hybrid algorithm. These experiments demonstrate that the time needed to update the interval lists of the Sweep & Prune algorithm dominates the total time spent during the collision detection. Additionally, we plan to optimize the Grid and Octree algorithms for the broad phase, as well as to implement other spatial partitioning data structures such as BSP-trees. As an improvement on the narrow phase, an interruptible algorithm may be also used to provide more control upon the time spent for collision detection in interactive environments.

Acknowledgements

Rafael de Sousa Rocha benefits of a PIBIC studentship and is grateful to the Brazilian supporting Agency CNPq.

References

[1] C. Ericson, *Real-Time Collision Detection*, Morgan and Kaufmann Publishers, 2005.

[2] C. O' Sullivan and J. Dingliana, "Real-Time Collision Detection and Response using Sphere-Trees", In Proceedings of the *15th Spring Conference on Computer Graphics*, 1999, pp. 83-92.

[3] S. Redon, "Continuous Collision Detection for Rigid and Articulated Bodies", *ACM SIGGRAPH Notes*, 2004.

[4] R.S. Rocha and M.A.F. Rodrigues, "Detecção de Colisão *Broad Phase* Utilizando *Grids* para Ambientes Interativos". *Revista Eletrônica de Iniciação Científica (REIC)*, SBC, June, Vol. 6(2), 2006, pp. 1-17.

[5] S. Bandi and D. Thalmann, "An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animating Rigid Bodies", *Computer Graphics Forum*, Vol. 14(3), 1995, pp. 259-270.

[6] J.D. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi, "I-Collide: An Interactive and Exact Collision Detection System for Large-Scale Environments", In Proceedings of the *ACM Interactive 3D Graphics Conference*, 1995, pp. 189-196.

[7] P.M. Hubbard, "Collision Detection for Interactive Graphics Applications", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1(3), 1995, pp. 218-230.

[8] G. Bradshaw and C. O'Sullivan, "Sphere-Tree Construction using Medial-Axis Approximation", In Proceedings of the *ACM SIGGRAPH Symposium on Computer Animation*, 2002, pp. 33-40.

[9] K. Storey, F. Lu, G. Morgan, "Determining Collisions Between Moving Spheres for Distributed Virtual Environments", In Proceedings of the 22nd *Computer Graphics International*, 2004, pp. 140-147.

[10] R.G. Luque, J.L.D. Comba, and C.M.D.S. Freitas, "Broad-Phase Collision Detection Using Semi-Adjusting BSP-Trees", In Proceedings of the *Symposium on Interactive 3D Graphics and Games*, ACM Press, 2005, pp. 179-186.

[11] G.V.D. Bergen, *Collision Detection in Interactive 3D Environments*, Morgan and Kaufmann Publishers, 2004.

[12] S. Gottschalk, M.C. Lin, and D. Manocha, "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection", *Computer Graphics Forum*, Vol. 30, 1996, pp. 171-180.

[13] J.T. Klosowski, M. Held, J.S.B. Mitchell, and H. Sowizral, "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4(1), 1996, pp. 21-36.

[14] N. Govindaraju, S. Redon, M.C. Lin, and D. Manocha, "CULLIDE: Interactive Collision Detection between Complex Models in Large Environments using Graphics Hardware". In Proceedings of the *ACM SIGGRAPH/Eurographics Conference on Graphics Hardware*, 2003, pp. 25-32.

[15] G. Rowe, *Computer Graphics with Java*, Palgrave, 2001.

[16] L.S. Taddeo, "Detecção de Colisão utilizando *Grids* e Octrees Esféricas para Ambientes Gráficos Interativos", MSc. Thesis, UNIFOR, Fortaleza, Brazil, 2005.

[17] B. Watson, N. Walker, W. Ribarsky, and V. Spaulding, "Effects on Variation in System Responsiveness on User Performance in Virtual Environments", *Human Factors (Special Section on Virtual Environments)*, Vol. 40(3), 1998, pp. 403-414.