A new sphere-tree generation method to speed up the collision detection pipeline

Omar Rodríguez González
Facultad de Ingeniería
Universidad Autónoma
de San Luis Potosí
San Luis Potosí, México
omarg@uaslp.mx

Marta Franquesa Niubó
Departament de Llenguatges
i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, España
marta@lsi.upc.edu

10th May 2005

Abstract

In this paper, a novel sphere-tree generation method used for collision detection is proposed. Using existing consumer-level graphics cards and its programmable graphics processing units (GPU) a sphere-tree is constructed in real-time inside an animation. This guarantees that no construction or loading of a precomputated hierarchy is required.

With our method, core memory is managed in an efficient manner, allocating and releasing memory space as necessary. By this, out-of-core techniques can perform better in real-time situations. The animation tests maintain an above the average performance and the collision detection is fast and efficient.

Keywords: sphere-tree construction, collision detection, viewing volume, graphics hardware

1 Introduction

Collision detection is a key problem in areas like computer graphics, virtual reality, games, animation, CAD, robotics and manufacturing [LG98, JTT01]. Considered as a bottleneck within real-time environments, several authors have studied the detection of a collision and multiple solutions have been proposed.

The problem to identify interfering objects in complex systems (huge environments with a large number of objects) has been approached applying different hierarchical space subdivisions methods over the scene. The problem to determine which object parts are candidate to collide with other object parts has been addressed modelling bounding volume hierarchies over

the objects, usually called Bounding Volume trees, or BVtrees. Bounding volume trees allow discarding collision faster than using the geometry of the original models. One of the most commonly bounding volume hierarchy model used is the sphere-tree.

The hybrid collision detection, introduced by Kitamura *et al.* [KTAK94], refers to any collision detection method that first performs one or more iterations of approximate test to study whether objects interfere in the workspace and then, performs more accurate tests to identify the object parts causing the interference. Hubbard [Hub95] reports two phases: the *broad phase*, where approximate interferences are detected, and the *narrow phase* where exact collision detection is performed. O'Sullivan and Dingliana [O'S99, OD99] extended the classification pointing out that the *narrow phase* consists of several levels of intersection testing between two objects at increasing level of accuracy (*narrow phase: progressive refinement levels*) and, in the last level of accuracy, the tests may be exact (*narrow phase: exact level*). Franquesa and Brunet [FNB03, FNB04] extended the *broad phase* in two more subphases. In the first one, tests are performed to find subsets of objects from the entire workspace where collisions can occur, rejecting at the same time, all the space regions where interference is not possible (*broad phase: progressive delimitation levels*). In the second subphase, tests determine the candidate objects that can cause a collision (*broad phase: accurate broad level*). Figure 1 summarizes the complete hybrid collision detection pipeline including all its phases.

In recent times, the availability of high performance 3D graphics cards are common in personal computers. The power and fastness of the built-in GPUs and its own dedicated memory is being applied to a wider variety of applications, even those that the creators not originally intended to manage.

In this paper, a new method called *on-collide sphere-tree*, OCST for short, is introduced. This new approach works by detecting collisions among models with arbitrary geometry using the video card's GPU. Candidate parts of colliding objects are detected as the OCST is constructed in real-time.

Main contribution: The structures involved in the hybrid collision detection phases, space subdivision hierarchies and bounding volume (BV) hierarchies, have been approached as a precomputation to the simulation environment. Before entering the simulation, the structures must be loaded in core memory. In simulation run-time, updates to the BV hierarchies take place according to the position and orientation of the moving objects.

We present an algorithm that doesn't use precomputed BV hierarchical structures, instead uses an octree-based sphere-tree created in real-time. The detection of surface overlapping over the sphere-tree nodes is performed using occlusion queries. These queries exist in modern graphics hardware. The algorithm is addressed for rigid objects moving in large environments.

- Broad Phase:
  - Delimitation levels
    - Input: N-bodies
    - Use of hierarchical structures
    - Output: n-bodies (subset of N-bodies)
  - Accurate broad level
    - Input: n-bodies
    - Use of simple bounding representations
    - Output: 2-candidate bodies list
- Narrow Phase:
  - Refinement levels
    - Input: 2-bodies
    - Use of hierarchical structures
    - Output: Candidate parts of 2-bodies
  - Exact level
    - Input: 2-bodies or candidate parts of 2-bodies
    - Use the geometry of objects
    - Output: Colliding features

Figure 1: Hybrid collision detection phases

With this algorithm, the *narrow phase* of the hybrid collision detection problem is accelerated. When many objects interact, core memory is managed more efficiently. The access to secondary storage is improved when out-of-core techniques are used.

Organization: The rest of the paper is organized as follows. In section 2 we discuss the state of the art in related areas. The representation model to be used is detailed in section 3. Description of the sphere-tree construction algorithm is in section 4. OCST construction and collision detection in real-time are discussed in section 5. Experimental results are given in section 6 and conclusions are presented in section 7.

2 Previous Related Work

A bounding volume hierarchy approximates a representation of an object as a hierarchical structure, known as bounding volume tree (BVtree). One of the most used BVtrees in the literature is the sphere-tree [Hub93]. A sphere-tree represents an object by sets of spheres in a hierarchical way. Three methods are commonly used for the construction of a sphere-tree. The first one, consists of fitting spheres to a polyhedron and shrinking them until they just fit [RB79]. The second one is based on an octree [Sam90]. Thus, the octree-based sphere-trees [PG95, Hub96, OD99] performs a re-

cursive subdivision in 3D, creating spheres on child nodes that overlap the surface of the object. And the third and last, the medial-axis surface method [Qui94, Hub95, Hub96, BO03], uses Voronoi diagrams to calculate the object *skeleton* placing maximal sized spheres on it so the spheres fill the object.

The graphics-hardware-assisted collision detection algorithms started with Shinya and Forgue [MM91], and Rossignac *et al.* [RMS92]. After them, a more efficient algorithm was proposed by Myszkowski *et al.* [MOK95] using the stencil buffer. Baciu and Wonk [BW98] were the first to use common available graphics cards to compute image-based collision detection. Vassilev *et al.* [VSC01] use a technique for collision detection in deformable objects like clothes. Kim *et al.* [KOLM03] use graphics hardware to calculate Minkowski sums to find the minimum translational vector needed to separate two interfering objects. All those algorithms involve no precomputation, but perform image-space computations that require the reading back of the depth or stencil buffer, which can be expensive on standard graphics hardware.

Govindaraju *et al.* [GRLM03] use occlusion queries to compute a potentially colliding set (PCS) in the *broad phase*, followed by exact collision in the *narrow phase*. Fan *et al.* [FWG04] use occlusion queries to fast detect collision between a convex object and an arbitrarily shaped object. The advantage of using GPU based occlusion queries is that no read back of the depth or stencil buffer is necessary to obtain results. This kind of tests are faster than image-space computations.

3 Representation Model

As mentioned before, the BVtree selected in this work is the sphere-tree. To bound each candidate object to collision, octree-based representation for sphere-trees construction is used. As is well known, an octree is a hierarchical structure obtained subdividing recursively in 3D to form eight child nodes. Each one can be represented with three colors. Black color for child nodes completely inside the subdividing object. White for child nodes completely outside. Grey for child nodes in which the frontier of the object overlaps. Grey nodes will be subdivided until an user-defined depth for the octree is reached. When the octree depth is reached, the grey nodes become leaf nodes. An octree-based sphere-tree is an octree where each node is bounded by one sphere instead of a cube. Figure 2 shows an octree-based sphere-tree representation.

A sphere-tree based on octrees, with a certain depth level, gives enough proximity to the object's surface so conservative collision detection can be performed.

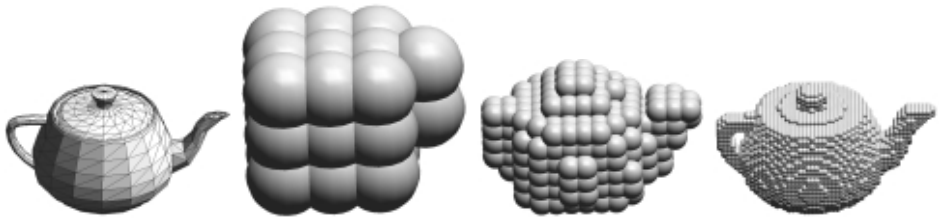

Figure 2: Original object and octree-based sphere-tree levels 2, 4 and 6

The cost to create sphere-trees can be high in terms of computing resources. Space subdivisions require floating-point operations, which are generally slow on CPU. The octree construction requires having the geometry object loaded in core memory aside the sphere-tree structure. Trying to create a sphere-tree on simulation run-time cannot be achieved using only the CPU. Therefore the construction of a sphere-tree has been treated as a precomputation step to the simulation. Having and maintaining all the sphere-tree structures in core memory when many objects are present, can be expensive during the life cycle of a simulation.

From the BVtrees methods, the simplicity of octree-based sphere-trees makes it good enough to implement them using graphics hardware (see section 4). The construction of sphere-trees in real-time is performed using occlusion queries. Thus, no precomputation is necessary, core memory is free of hierarchical structures at the beginning of the simulation because the sphere-trees are created only on-collision when required. To preserve memory, only branches of the sphere-tree for the parts of the objects that potentially can collide are computed. Newly created branches are maintained in core memory for future use during the simulation (see section 5).

4 Sphere-tree Construction

In this section, the algorithm to construct the octree-based sphere-tree model using the GPU is presented.

Different hardware designers have made various occlusion test implementations with differences in performance and functionality. In this way, the first occlusion query that we can find[1], returns a boolean answer if any incoming fragment passes the depth test. The second one[2], returns the number of fragments that pass the depth test but requires that the first query be supported by the graphic card. The third and most standard, GL_ARB_occlusion_query[3], is similar to the last named query, but does not

[1] http://oss.sgi.com/projects/ogl-sample/registry/HP/occlusion_test.txt
[2] http://oss.sgi.com/projects/ogl-sample/registry/NV/occlusion_query.txt
[3] http://oss.sgi.com/projects/ogl-sample/registry/ARB/occlusion_query.txt

require the first query to be available.

The GL_ARB_occlusion_query is used in our method to avoid stalls in the graphics pipeline. This query can manage multiple queries before asking for the result of any one, increasing the overall performance.

Let A be an arbitrarily shaped object. A sphere-tree root node for A is constructed creating an axis-aligned bounding box (AABB) for A. A bounding sphere for A is created bounding the AABB from A, with its center as the center of the AABB and its radius as half the distance of the AABB extreme vertices. Taking the AABB from the root node of A, we construct a new level for the sphere-tree subdividing it in 3D. For each new child node, a resulting octree subdividing AABB box is assigned and an overlap test is performed to verify if it can be a grey node.

To accelerate the overlap test for the detection of grey nodes, occlusion computations are performed. Based in the observation that, if the surface of A can be viewed in at least some part from inside the AABB of an octree node, then A is overlapping the octree node.

The overlap test performs one, two or up to three occlusion queries for each of the main axis. Three requirements are needed for each occlusion query:

- a viewing volume
- a camera position
- the occlusion test elements

The viewing volume is created using an orthographic frustum view limited by the AABB box of the octree node tested. The camera position is placed outside the viewing volume, centered at a box face, looking toward the box in parallel to a main axis, and with a distance equal to the length of the box in the looking direction. Figure 3 illustrates one of the three viewing volumes and its camera position. The first occlusion test element (the occluder), is the AABB box of the octree node. The second occlusion test element (the possibly occluded objects), is the surface of A.

An occlusion query reports if one or more occluders allow the possibility that occluded objects can be seen from inside a viewing volume. That is, if the surface of A can be seen from inside the AABB box (viewing volume) of the tested octree node, in at least one of the three main axis, then the surface of A is overlapping that octree node.

Algorithm 1 illustrates the overlap test. If the number of samples that passed the occlusion query is greater than zero in at least one of the three queries, the surface of A overlaps the tested octree node and it's a grey node. If it's

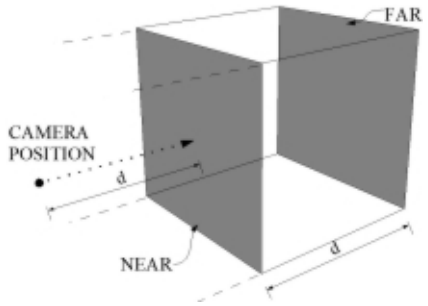

Figure 3: Viewing volume construction

a grey node, a sphere is created bounding the AABB box of the node and is inserted on the sphere-tree.

```
set occlusion query buffer;
render AABB box of the tested octree node;
clear depth buffer;
disable color and depth buffer;
disable cull face;
while not overlap found do
    select one of the three frustum views;
    set occlusion query;
    render surface of A;
    end occlusion query;
    get occlusion query results;
    if samples passed the test > 0 then
        overlap with surface of A is found;
    else
        if all queries finished then
            no overlap found;
        else
            select next frustum view;
        end if
    end if
end while
enable cull face;
enable depth and color buffer;
```

Algorithm 1: GPU based overlap test

5 OCST and Real-time Collision Detection

To achieve collision detection in real-time, OCST branches are constructed for objects only when it is needed. Thus, is necessary to load the geometry of

the objects into graphics card's memory, and construct a root OSCT for each of them at the beginning of the simulation. The OCST root is initialized with an AABB and a bounding sphere with the center as the center of the AABB, and its radius as half the distance of the AABB extreme vertices.

Let A and B be arbitrarily shaped objects in movement. The two objects collide each other only if the distance between their root sphere centers is equal or less than the sum of their respective radius. When a collision occurs, one level is constructed for the OCST (see algorithm 1) for objects A and B. If child nodes of object A collide with child nodes of object B, an additional level is constructed only for the colliding child nodes. This process continues up to an user-defined depth for the OCST. When the depth value is reached, and two leaf nodes collide, a collision between object A and B is reported. Using a bigger depth value, the approximation to the object surface is tighter, and the collision detection is better.

Algorithm 2 illustrates this process. Notice that all hierarchies sphere centers must be updated with the objects movement. When a new level for the OCST is created the number of updates increases. With a big user-defined depth value the maintaining cost to update all the animation OCSTs is higher.

```
while animation do
    for each object in the scene do
        found PCS for all OCST;
    end for
    if any collision occurs then
        for each pair-colliding do
            if depth level is reached for objects A and B then
                collision detected between A and B;
            else
                if don't exist a lower level for A then
                    create new branch for colliding nodes;
                end if
                if don't exist a deeper level for B then
                    create new branch for colliding nodes;
                end if
            end if
        end for
    end if
    update all OCST time-stamps;
    update all OCST spheres;
    update animation;
end while
```

Algorithm 2: Real-time collision detection

Figure 4: Example of input models: left to right, a *bunny* with 5110 triangles, a *dragon* with 5104 triangles, a *lamp* with 600 triangles and, a *cow* with 5144 triangles

To found the potentially colliding set, PCS, the sphere interference test described below is used. A list with pair-colliding spheres is computed and used to identify interfering object parts. In large environments [FN04, RF05], the PCS can be obtained using algorithms designed for the *broad phase* of the hybrid collision detection problem.

To increase the algorithm performance, the branches of the OCST created by older collisions are kept in core memory. These can be re-used on forthcoming collision tests.

To avoid the problem of a high computing resource cost caused for hierarchies updates, a time-stamp is assigned to the deeper OCST nodes. If a complete OCST level doesn't participate in a collision during a certain amount of time, it is deleted from core memory and the parent initialized with its own time-stamp. This will cause that an object gets back to its initial state (only the OCST root node), if it's not involved in any more collisions during a certain amount of time.

6 Experimental Results

In this section some relevant results of applying our method are exposed. To compare the actual results with existing others, the input data tested in other existing algorithms has been selected.

The algorithms have been implemented on a Dell Inspiron notebook with ATI Mobility Radeon 9600 graphics card with 128 MB VRAM and a Pentium M processor at 1.80 GHz. The algorithms were tested with commonly used complex models[4]. Figure 4 shows some of the models used.

[4]http://isg.cs.tcd.ie/spheretree/

6.1 Sphere-tree Construction Timings

Table 1 shows the time to construct one level of an OCST. The results are obtained with the objects already loaded in graphics card's memory as triangle-soup. No optimizations such as triangle-strips or triangles-fans have been made. The table shows the number of triangles for each model, the time used to construct the level (in seconds) and the number of occlusion tests performed.

Model	Triangles	Time	Occlusion
Dragon	1496	0.0099	13
Bunny	1500	0.0099	9
Cow	1500	0.0099	9
Lamp	600	0.0199	13
Dragon	5104	0.0199	13
Bunny	5110	0.0099	9
Cow	5144	0.0099	9

Table 1: OCST construction time

For each occlusion test, the complete model has to be render. Note that the object's geometry doesn't affect the time to construct one level for the OCST. The algorithm performance is affected only for the number of occlusion tests and the time each one lasts. That is, the worst case only occurs when all occlusion tests have to be considered. In these case, with eight possible child nodes and three tests per child node, for a total of 24 occlusion tests, the maximum time takes 0.03 seconds.

Construction of an OCST level using only the CPU can takes from 0.03 seconds for the simplest model, 0.1 to 0.5 seconds for the intermediate models, and 1 second and up for the largest models. Without the use of the GPU for the construction, the object's geometry indeed does affect the algorithm performance.

6.2 Real-time Collision Detection Performance

The algorithms were tested with a scenario where one object (with 5000 triangles approx.) follows a fixed trajectory in a 3D space. Collision occurs with other three objects (two of them with 5000 triangles approx. and one with 600 triangles approx.). Table 2 shows the performance with a 14 seconds animation and an user-defined depth level for the OCST as 5. The results are measured in frames-per-second (FPS). The number of occlusion queries performed in each time step is also showed.

Note that the FPS slowed down, only, when new levels for the OCSTs are generated. For example, at the second 5.00, when the occlusion queries are

Time	FPS	Occlusion queries
1.00	179.82	0
2.00	324.35	0
3.00	324.68	0
4.00	309.38	142
5.00	216.78	1056
6.00	252.75	742
7.00	256.49	610
8.00	255.74	690
9.00	230.54	1056
10.00	180.82	1596
11.00	215.57	1192
12.00	320.68	20
13.00	265.73	708
14.00	262.48	700

Table 2: Animation performance

1056 the FPS are 216.78. And, in the worst case, at the second 10.00, when the occlusion queries are 1596, the FPS slows down to 180.82. Although the speed of the FPS gets lower, the rate keeps on being good enough. Therefore, the animation can be maintained over 60 FPS and allows a smooth transition between frames.

The worst case occurs when objects are moving very fast and a collision occurs. This situation can cause that several levels of the OCST tree have to be constructed at once for each colliding object. In this case, the performance could slows down. Even so, stalls in the animation can occur, only, if the user-defined depth value is too high. Even thought, these stalls are due to the high number of occlusion tests that have to be performed to construct all the OCST branches, the running time is not affected when real-time simulations are computed.

Figures 5 and 7 shows the sequence of a collision between two objects. Figures 6 and 8 represents the same sequence showing the OCSTs created in real-time up to level 5.

7 Conclusions

In this paper, a new method that has been conceived to speed up the collision detection pipeline has been introduced. An algorithm, on-collide sphere-tree (OCTS), for fast construction of sphere-trees using the GPU from graphic cards has been presented. Its application in real-time environments has

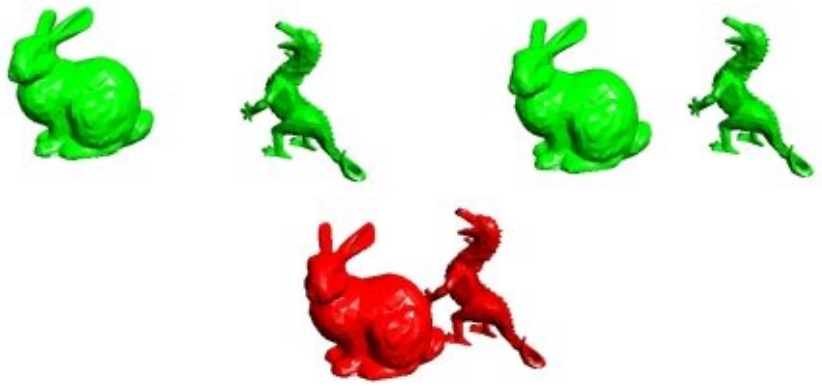

Figure 5: Original models animation

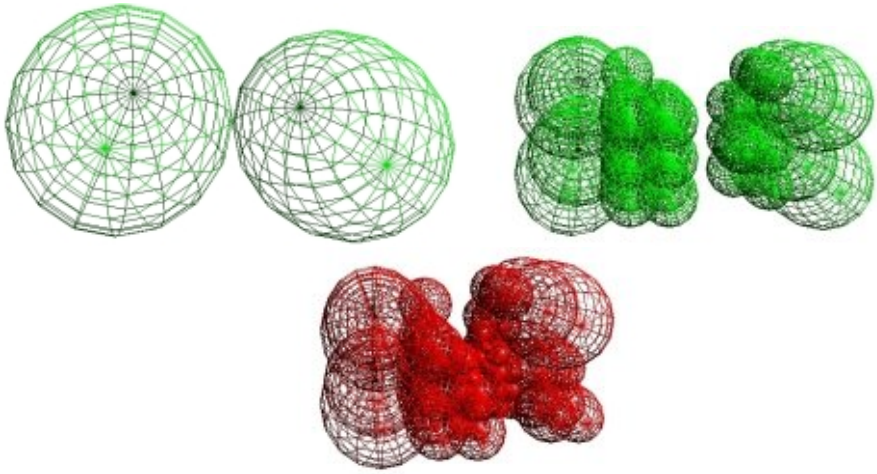

Figure 6: On-collision sphere-trees

been implemented using OCST (see section 4 for details). This method is fast enough to manage collision detection in real-time, as it can be seen in the section 6 where the experimental results are exposed. The speed and efficiency obtained with our method allow us to manage many concurrent objects in a scene.

The method's limitations are related to hardware constrictions. The overall performance is affected by several parameters. The amount and speed of the video memory built-in the graphic cards, the bus transfer speed, and the clock frequency of the GPU. The use of out-of-core methods in real-time could be degraded at reading time from secondary storage, and at sending time of the object's geometry to the graphic card memory.

Using of OCST reduces the amount of the model representation to be generated while decreasing considerably the collision detection time without loss

Figure 7: Original models animation

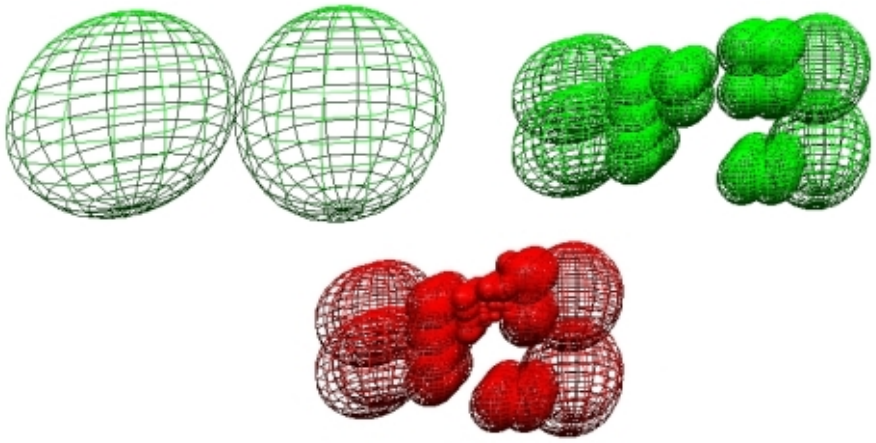

Figure 8: On-collision sphere-trees

of accuracy.

In the future, we would like to analyse different sphere-tree algorithms to achieve better object tightness. Testing optimized models, via triangle-strips or triangle-fans, could give us more efficiency. On the other hand, exact collision detection and collision response that are not considered in this paper, could be studied.

We detailed an algorithm related to the *narrow phase* of the collision detection pipeline problem. However, work related to the *broad phase* can be found in [FN04, RF05]. We are working on bringing together both methods, so a fully functional fast collision detection system for large environments could give us better results on our application environments.

Acknowledgements

This research has been partially supported by the Ministerio de Ciencia y Tecnología under the project MAT2002-0497-C03-02 and the Facultad de Ingeniería of the Universidad Autónoma de San Luis Potosí under the PROMEP program.

References

[BO03] G. Bradshaw and C. O'Sullivan. Adaptative Medial–Axis Approximation for Sphere–Tree Construction. *ACM Transactions on Graphics*, 22(4), 2003.

[BW98] G. Baciu and S.G. Wonk. Recode: An image–based collision detection algorithm. In *Proc. of Pacific Graphics*, pages 497–512, 1998.

[FN04] M. Franquesa-Niubò. *Collision Detection in Large Environments using Multiresolution KdTrees*. PhD thesis, Universitat Politècnica de Catalunya, March 2004.

[FNB03] M. Franquesa-Niubo and P. Brunet. Collision detection using *MKtrees*. In *Proc. CEIG 2003*, pages 217–232, July 2003.

[FNB04] M. Franquesa-Niubo and P. Brunet. Collision Prediction using MKtrees. In R. Scopigno and V. Skala, editors, *WSCG 2004, The 12–th International Conf. in Central Europe on Comp. Graphics, Visualization and Comp. Vision 2004*, volume 1, pages 63–70, February 2004. Plzen. ISSN 1213–6972.

[FWG04] Z. Fan, H. Wan, and S. Gao. Simple and rapid collision detection using multiple viewing volumes. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 95–99. ACM Press, 2004.

[GRLM03] 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 *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32. Eurographics Association, 2003.

[Hub93] P. M. Hubbard. Interactive collision detection. In *Proc. IEEE Symp. on Research Frontiers in Virtual Reality*, volume 1, pages 24–31, October 1993.

[Hub95] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, September 1995.

[Hub96] Philip M. Hubbard. Aproximating polyhedra with spheres for time–critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[JTT01] P. Jimenez, F. Thomas, and C. Torras. (3d) collision detection: A survey. *Computers and Graphics*, 25(2):269–285, August 2001.

[KOLM03] Young J. Kim, Miguel A. Otaduy, Ming C. Lin, and Dinesh Manocha. Fast penetration depth estimation using rasterization hardware and hierarchical refinement. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 386–387. ACM Press, 2003.

[KTAK94] Y. Kitamura, H. Takemura, N. Ahuja, and F. Kishino. Efficient collision detection among objects in arbitrary motion using multiple shape representation. In *Proceedings 12th IARP Inter. Conference on Pattern Recognition*, pages 390–396, October 1994.

[LG98] M.C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

[MM91] Shinya M. and Forgue M. Interference detection through rasterization. *Journal of Visualization and Computer Animations*, 2:131–134, 1991.

[MOK95] K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between computer solids using rasterizing graphics hardware. *The Visual Computer*, 11, 1995.

[OD99] C. O'Sullivan and J. Dingliana. Real–time collision detection and response using sphere–trees. In 15th Spring Conference on Computer Graphics, April 1999. ISBN: 80–223–1357–2.

[O'S99] C. O'Sullivan. *Perceptually–Adaptive Collision Detection for Real–time Computer Animation.* PhD thesis, University of Dublin, Trinity College Department of Computer Science, June 1999.

[PG95] I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 1995.

[Qui94] S. Quinlan. Efficient distance computation between non–convex objects. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994. San Diego, CA.

[RB79] J.O. Rourke and N. Badler. Decomposition of three–dimensional objects into spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI–1(3):295–305, July 1979.

[RF05] O. Rodríguez and M. Franquesa. Hierarchical structuring of scenes with MKTrees. Technical report, Software Dept. LSI. U.P.C., 2005. Ref: LSI–05–4–R. http://www.lsi.upc.edu/dept/techreps/techreps.html.

[RMS92] Jarek Rossignac, Abe Megahed, and Bengt-Olaf Schneider. Interactive inspection of solids: cross-sections and interferences. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 353–360. ACM Press, 1992.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990. ISBN 0-201-50255-0.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. In *Computer Graphics Forum*, volume 20(3), pages 260–267, 2001.