Real time accurate collision detection for virtual characters

Andoni Mujika, David Oyarzun, Aitor Arrieta, María del Puy Carretero

VICOMTech - Visual Interaction and Communication Technologies Center
Mikeletegi Pasealekua, 57 - Parque Tecnológico
E-20009 Donostia - San Sebastián, Spain

{amujika, doyarzun, aarrieta, mcarretero}@vicomtech.org

ABSTRACT

This paper presents an accurate real time collision detection algorithm for interactively animated virtual characters using sphere-trees as Bounding Volume Hierarchies. We build upon a fast mathematical method for on-demand sphere refitting during the animation and improve it for being applicable to any object, without dependency on its geometrical level of detail or its dynamic/static behavior. It uses sphere-plane intersection test as the exact test in the collision detection algorithm instead of the usual triangle-triangle one. Corner-trees, a special hierarchy that ensures the utilization of the plane-sphere intersection test is right, are also presented. In the worst case, the optimization decreases in 25% the time needed to process a frame in extreme conditions. The algorithm has been successfully tested on a real time and collaborative 3D virtual world.

Keywords

Real-time Collision Detection, Bounding Volume Hierarchies, Virtual Character Animation.

1. INTRODUCTION

Collision detection (CD) is a key issue in almost all fields of computer graphics. Real time virtual objects and virtual characters' animation are not exceptions. In most of cases they need to have realistic behaviors that imply CD, i.e. not to penetrate other objects. Therefore, many algorithms have been proposed in recent years.

Accurate algorithms are usually very expensive computationally speaking. Then, applications that make use of collision detection algorithms have to balance between preciseness of the detection and velocity of the algorithm.

For instance, a very fast performance of the collision detection algorithm is needed in Massively Multiplayer Online Games (MMOGs) and a very precise detection is crucial in serious games and virtual prototyping. On the other hand, continuous collision detection (CCD) was presented to solve the main problem that discrete algorithms presented, the tunneling effect, i.e. the miss of some collisions. However, the velocity of the algorithm obtained was not appropriate for real time purposes.

Although the problem has been widely studied for rigid bodies, there is a lot to do regarding CD for real time deformable objects such as clothes, interactive virtual humans, etc. The problem increases in case of collaborative virtual worlds, with lot of avatars interacting among themselves and with objects at the same time. It is usual to see very fast but imprecise CD algorithms.

Therefore, in this article, we focus on real time humanlike animation in collaborative virtual worlds and propose an algorithm to obtain a fast and precise CD for interactive virtual characters of high level of detail. In order to be used in both, virtual worlds and precise simulations, the algorithm is based on these features:

- A fast update of the spheres in the Bounding Volume Hierarchy.
- Utilization of the sphere-plane intersection test instead of the slower triangle-triangle test.
- Implementation of the corner-trees, a novel hierarchy for the correct and fast performance of the algorithm.

The paper is structured as follows. In section 2 we summarize the related work. In section 3 we describe our virtual character animation platform and its collision detection algorithm. In section 4 we analyze the major problem we found for a fast performance of the algorithm and in section 5 we describe two

methods to solve it. In section 6 we present and compare the obtained results and finally, in section 7, we analyze future extensions to improve the performance of our CD algorithm.

2. RELATED WORK

When detecting collision detection between a virtual character and its environment, first, the character's movement has to be computed, i.e. the new position of the avatar's vertices has to be calculated. And then, the second stage will be the CD itself, taking into account the new positions of the vertices. Sections below deal with related work in each of these stages.

Virtual Character Animation

Virtual character animation has been widely studied in computer graphics. In this research field, one of the main goals is the realistic simulation of human movements. Especially in 3D animation, many efforts have been done in recent years. Although there are some methods such as Blend Shape Deformation [Moh03a] and Free-Form Deformation [Sed86], skeletal animation systems are the most used. The primitives that form the virtual character are transformed depending on the movements of a skeleton. We can classify these methods by the way they skeleton affects the primitives. Linear Blend Skinning [Moh03b] manipulates the triangle-mesh associating each vertex to a group of joints of the skeleton and giving a weight for each joint (the sum of the weights is one). Then, the transformation of the vertex is a linear combination of the joints' transformations. Spherical Blend Skinning [Kav05a] works similarly, but the relation between joints' transformations and vertices' transformations is not linear. It is based on Spherical Linear Interpolation.

Collision Detection

When detecting collisions between two objects, testing each primitive-couple is too costly. Detecting collisions between two objects with n and m primitives would cost $n * m * b$ operations, where b is the number of basic operations needed in an intersection test between primitives. Therefore, a method that detects which primitives are more likely to be colliding (broad phase) is used before executing the exact test between primitives (narrow phase) [Mol97, Tro05].

Usually, Bounding Volume Hierarchies (BVHs), i.e., sets of volumes that bound the object getting different levels of tightness, are used in the broad phase. During the collision detection, the volumes in the hierarchies of the objects are tested to be colliding. If they don't collide, all the primitives inside the volumes don't collide, but if they do collide, next levels of tightness are checked. Once the algorithm finds two colliding leaf-nodes, i.e. volumes that enclose only one primitive, the exact intersection test between primitives is called.

The number of operations needed to detect collisions between bounding volumes is much lower than between primitives. For instance, a collision test between spheres consists of 10 operations and the best collision test between triangles consists of 96 operations.

We can sort these methods by the type of volume they use:

- Spheres [Qui94, Hub96]
- Axis-Aligned Bounding Boxes (AABBs) [Van98]
- Oriented Bounding Boxes (OBBs) [Got96],
- k-Discrete Orientation Polytopes (k-DOPs) [Klo98]

Most of these methods were presented for collision detection between rigid objects. Nevertheless, CD for deformable objects also makes use of BVHs. Once again, different types of BVHs appear such as spheres [Bro01] and AABBs [Lar01, Zac06].

Regarding collision detection for avatars, i.e. virtual characters, there have been different approaches in recent years. Kavan et al. use spheres to create the BVH. They refit the sphere-tree for bodies that are moved based on a skeleton. They proposed collision detection methods for Linear Blend Skinning [Kav05b] and Spherical Blend Skinning [Kav06].

All the results shown so far are discrete, i.e. they sample objects' motions. As opposed to these methods, continuous collision detection (CCD) methods compute the first time of contact during the collision detection. Six different approaches to CCD have been presented in the literature: algebraic equation-solving [Cho06], swept volumes [Abd02], adaptive bisection [Red02], kinetic data structures (KDS) [Aga01], the configuration space approach [Van04], and conservative advancement [Cou06] However, these methods performance is not as fast as is required.

There are also some continuous collision detection results for avatars. Zhang et al. [Zha07] use OBB-trees and create AABBs during the motion interpolation using Taylor Models, i.e. a generalization of interval arithmetic. Instead, Redon et al. [Red04] use swept volumes (SV) for CCD in scenes with a simple articulated avatar.

3. ALGORITHM OVERVIEW

The developed collision detection algorithm is a discrete collision detection method and works with spheres as bounding volumes. Spheres were chosen because of the fast performance of the sphere-sphere

intersection test and the low space needed to store the data.

Virtual Character Animation

Regarding the animation stage, in our system, the vertices are associated to a unique joint and the transformation of a vertex is obtained computing the product of the transformations of all joints upon the associated joint in the skeleton-tree and the weighted transformation of the associated joint.

$$C_v = \left(\prod_{i \in I} C_i \right) * w_a * C_a$$

where I is the group of joints upon the associated joint in the skeleton-tree, C_v the transformation of the vertex, C_is the transformations of the joints in I, C_a the transformation of the associated joint and w_a the weight associated to the vertex. This way of animation provides an adequate balance between performance and realism for its use in collaborative virtual worlds.

Collision detection

The collision detection algorithm begins with the sphere-tree construction. This construction of the sphere-tree is based on Quinlan's work [Qui94]. First a binary tree is constructed: in each step, the triangles of a sphere are divided in two groups and two spheres are constructed enclosing each group [Gae99]. In this case, to make the division, the triangles are ordered depending on their position in one of the axes, so as to get two spheres as far as possible one from the other. Moreover, the axis is chosen to be the one where the spheres are most spread. As in [Kav05b], the binary tree is turned into a n-ary tree eliminating the spheres the radius of which is similar to their parent's radius. This way, when testing for collision, tests between similar spheres are avoided.

Since each vertex is associated to a single joint in our platform, instead of creating a unique tree, a tree is constructed for the group of vertices associated to each joint, so as to prevent the algorithm having spheres affected by no-adjacent-joints. To merge all the trees, an enclosing sphere for all vertices is computed as the root of the main tree and a sphere for each extremity to form the second level are created.

Sphere update

The sphere update of our algorithm is inspired by the main contribution of Kavan and Zara [Kav05b]. In the preprocess, all the vertices of a sphere are visited to compute the minimum and maximum weights for each joint affecting this sphere.

Then, during the animation, when a joint is visited to update the vertices associated to it, the spheres containing vertices associated to this joint are also visited. For each visited sphere, two new spheres (one if maximum and minimum weights are the same) are created applying the same transformation as to the vertices to the center of the sphere but using the maximum and minimum weights. The radii are the same as the original sphere.

$$c_{min} = \left(\left(\prod_{i \in I} C_i \right) * min_s * C_a \right) * c_s$$

$$c_{max} = \left(\left(\prod_{i \in I} C_i \right) * max_s * C_a \right) * c_s$$

where c_s is the center of the sphere, c_{min} and c_{max} are the new centers and max_s and min_s are the precomputed maximum and minimum.

Finally, the enclosing sphere of the new spheres is created, ensuring that all the vertices are inside the new sphere.

$$c_u = \frac{\sum_{i=1}^{n} c_i}{n}$$

$$r_u = \max_{1<i<n} (\| c_u - c_i \| + r_i)$$

where c_u and r_u are the center and the radius of the final updated sphere and c_i and r_i are the centers and the radii of the spheres obtained with all the maximum and minimum weights.

Narrow phase

During the collision detection, when two spheres in the lowest level of the hierarchies are colliding, an exact collision test between the triangles enclosed by those spheres is called. We use the fast algorithm presented by Tropp et al. [Tro05]. When detecting intersection between edges of a triangle and the other triangle, all the redundant operations to calculate determinants are discarded.

4. OPTIMIZATION

Since we want our platform to cope with virtual characters containing more than 40000 vertices, the algorithm needs some optimization. It has to be able to handle the big amount of spheres generated with this number of vertices.

In order to reduce the number of triangles that take part in the collision detection algorithm, we implemented an optimization proposed by Curtis et al. [Cur08]. They realized that many collision tests

between primitives are made more than once and developed a method to avoid these duplications. In our case, we assume that each edge of the triangle-mesh has to be tested once. Then, taking into account a triangle surrounded by three triangles that have already been taken into account is not necessary (see Figure 1). Therefore, we assume this triangle doesn't exist for the collision detection. One may think that some collision may be skipped this way. In fact, the penetration of a smaller triangle in a "not existing" triangle without touching its edges wouldn't be detected, but we have seen that in practice, this extreme case doesn't occur with avatars of so high level of detail. After implementing the optimization, the number of triangles used for the collision detection decreased 40%.

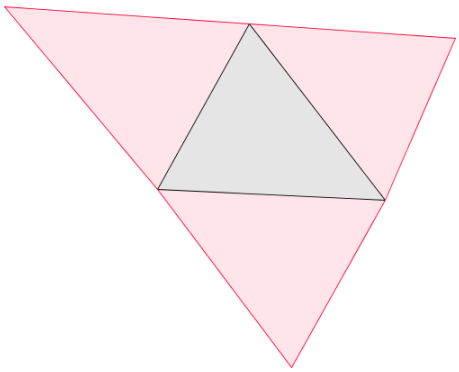

Figure 1 The triangle among the other 3 triangles is not taken into account in the CD algorithm.

Big triangles, a problem

When an avatar is walking in an environment, usually the triangles that compose the environment (walls, tables, windows, etc.) are much bigger than the ones that compose the avatar. This fact is a serious drawback when trying to get a fast performance of the collision detection system.

The leaf-node of the sphere-tree that corresponds to a big triangle is a big sphere. So, when the avatar is near a big triangle, it's possible that all the spheres in the BVH of the avatar are inside the big enclosing sphere of the triangle. This leads to a huge number of collision tests between spheres and a huge number of exact collision tests between triangles, since all the leaf-nodes of the avatar hierarchy are inside the leaf-node of the environment. We have checked that the algorithm can't cope with this number of operations, especially because of exact tests.

5. SOLVING BIG TRIANGLES' PROBLEM

A solution for the problem with big triangles could be just to divide big triangles in smaller triangles. Nevertheless, it is not always possible to manipulate the model received and dividing all big triangles until the leaf-nodes are small enough can increase the weight of the model drastically.

From now on, we denote the small triangle in the avatar's triangle mesh that takes part in an exact intersection test as t and the big triangle of the object in the environment as T. We denote their enclosing spheres, i.e. their leaf-nodes in the sphere-tree as S_t and S_T respectively.

Sphere Division

Although the division of the model's vertices may be impossible to carry out, a similar approach can be applied.

We want the avatar not to be inside S_T. So, we create a hierarchy inside the enclosing sphere of the big triangle to ensure that when exact test is called the primitives are really close. When the triangle is too big (we use a border value for the lengths of the edges), the triangle is divided in four new triangles joining the intermediate points of the edges and four new enclosing spheres are created to form the next level in the hierarchy (see Figure 2). The division finishes when the triangles are smaller than the threshold.

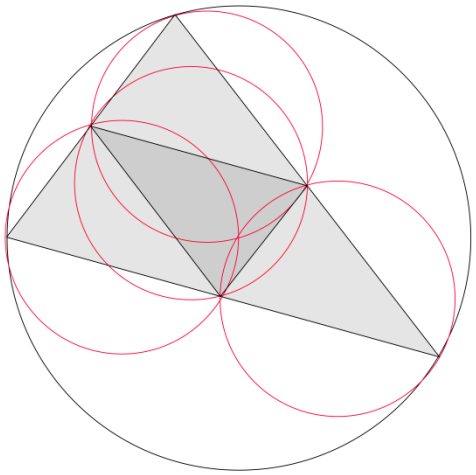

Figure 2 The enclosing sphere (black) of a triangle. The triangle divided in four triangles and their enclosing spheres (red).

During the animation, the collision detection algorithm runs as before, calling the spheres of the lower levels if the ones in upper levels collide, but in this case, the leaf-nodes doesn't enclose a triangle. They point to the big triangle T.

This way, when the avatar is not really close to T, only intersection tests between spheres are called. So the algorithm's performance is much faster. Moreover, when the avatar is close to the object that contains T, the exact collision test is only called for those triangles that are really close, avoiding the huge number of exact intersection test we had before.

The results obtained with this implementation were satisfactory, but we saw that a better performance could be obtained. Results will be shown in section 6.

Plane-Sphere intersection test

Virtual characters with high level of detail are composed by very small triangles comparing with the triangles that compose some objects of the environment. When the enclosing sphere S_t, of the small triangle, t, is colliding with a big triangle, T, t is colliding with T or it is very near. Therefore, testing t and T and testing S_t and T are nearly the same.

If the intersection test between a triangle and a sphere is not very costly, it is worth to use it instead of the exact test between two triangles. Nevertheless, we can see in [Eri05] that the sphere-triangle test is quite costly.

However, a simple and very efficient collision test between spheres and planes is presented in [Eri05] (see Algorithm 1) and it seems that T can be considered as a plane when testing with S_t. That way, the biggest bottle-neck in our algorithm would be solved due to the substitution of the exact test between triangles.

```
bool SpherePlaneTest(sphere s, triangle t){
    edge1 = t.v1 – t.v0;
    edge2 = t.v2 – t.v0;
    p = edge1 × edge2;
    n = s.center – t.v0;
    return ( |p · n| < s.radius * ||p|| );
}
```

Algorithm 1 Plane-sphere intersection test for sphere S and tirangle t

The problem of this substitution is that it is usual to find a leaf-node in the hierarchy of the avatar inside S_T which is not colliding with T, but colliding with the plane defined by T. This leads to a not existing collision detection.

So, before calling the plane-sphere collision test, we have to ensure that the sphere S_t is in front of the triangle T and it is not in the part of the enclosing sphere that the triangle doesn't occupy.

Working as in the latest subsection, we can create a quaternary tree inside the enclosing sphere S_T. Then, when the collision detection algorithm reaches leaf-nodes and calls the plane-sphere test, we can be sure that the sphere is in front of T and we can consider it as a plane.

Besides, a smaller tree than the quaternary-tree can be used without losing any property. For instance, when dividing the triangle in four smaller triangles, we can assume that if S_t is colliding with the enclosing sphere of the central triangle it is in front of T.

So, in the algorithm that recursively creates the hierarchy inside S_T, only triangles that have an edge that matches one of T's edges are divided into four new triangles again. We call the new hierarchy Corner-tree (see figure 3).

One may think that it is better to continue dividing the central triangles, because of the higher cost of the plane-sphere test. Nevertheless, it is less costly one plane-sphere test (30 operations) than four sphere-sphere tests (4x10 operations).

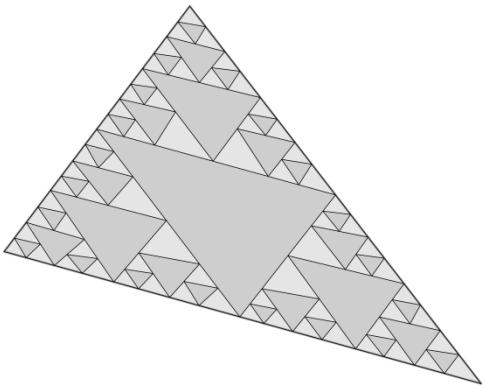

Figure 3 Division of a triangle to create the corner-tree.

Moreover, the number of the spheres in the hierarchy decreases drastically with this new algorithm. If n is the number of levels of the hierarchy, in the quaternary-tree the number of spheres in the nth level is 4^n. A huge number comparing to the new algorithm, which creates $12 * (2^{n-2} - 1)$ spheres in each level (see table 1).

Level	1	2	3	4	5	6	7
Corner-tree	1	4	12	36	84	180	372
Quaternary	1	4	16	64	256	1024	4096

Table 1 Number of spheres in the th level in the Corner-tree and in the quaternary-tree

6. RESULTS

The collision detection algorithm has been applied in our platform for the animation of virtual characters in collaborative virtual worlds successfully [Oya07] (see Figure 4).

We have checked our algorithm with an avatar composed by 44345 vertices in two different scenarios: a virtual museum with 9482 vertices and virtual living-room with 133139 vertices.

Both virtual worlds have triangles that are bigger than the avatar and we have checked the performance of the algorithm in extreme conditions, i.e. when the avatar is very close to these triangles without colliding. Moreover, the collision detection algorithm was run without any optimization and with the sphere division optimization to compare them with the latest version. All the tests were made with an Intel Core 2 Duo CPU at 2.20 GHz.

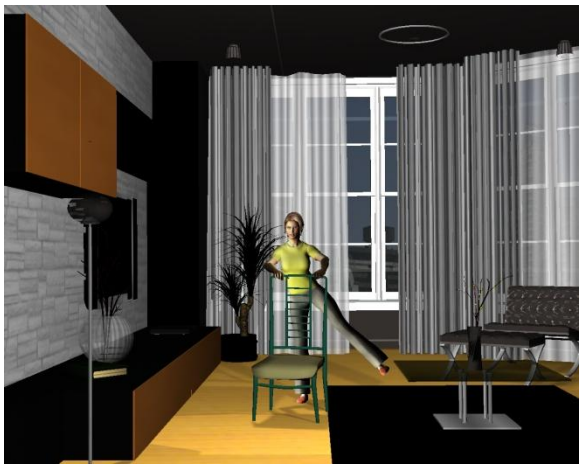

Figure 4 A virtual character in a virtual living-room.

First, we counted the basic operations (sum and multiplication) needed in each basic collision test: 10 operations in the sphere-sphere test, 96 in the triangle-triangle test [Tro05] and 30 in the plane-sphere test. Then, we ran the animation platform counting the number of these basic tests per frame so as to obtain the maximum number of operations made in a frame.

Table 2 and table 3 show the results obtained. The space needed to store sphere hierarchies, maximum times the intersection tests are called in one frame, the sum of basic operations in those maxima and the duration of the frame in the case of maximum operation.

In both cases, the space to store the information about the sphere hierarchies is much bigger when the algorithm has an optimization. Nevertheless, the space needed is not big enough to be a problem. As we stated before, we can see that the corner-tree is smaller than the quaternary-tree.

Virtual Museum	Original	Sphere Division	Plane-Sphere
Data	548 KB	65 MB	24 MB
Sph-Sph tests	≈200000	≈15000	≈5000
Tri-Tri tests	≈150000	≈150	
Pla-Sph tests			≈2000
Operations	≈17000000	≈165000	≈110000

Table 2 Results obtained for the animation in the virtual museum.

Living-room	Original	Sphere Division	Plane-Sphere
Data	3.21 MB	49.2 MB	37.5 MB
Sph-Sph tests	≈80000	≈80000	≈8000
Tri-Tri tests	≈70000	≈50000	
Pla-Sph tests			≈275
Operations	≈7000000	≈5000000	≈90000

Table 3. Results obtained for the animation in the virtual living-room.

As wished, sphere division optimization decreases the number of exact tests, especially in the virtual museum. This leads to a decrease in the duration of a frame.

Moreover, the plane-sphere optimization decreases the number of tests made in both the broad phase and the narrow phase. Combining this with the lower complexity of the plane-sphere collision test, we obtain a very fast performance.

In conclusion, we can see in the tables that increasing the stored data, i.e. creating bigger sphere hierarchies, we can decrease the time spent detecting collisions. In the virtual museum, the difference between the optimizations is not considerable, but in the living room, the time gained with the plane-sphere optimization is twice as the time gained with the sphere division optimization.

7. CONCLUSIONS AND FUTURE WORK

This article presents a fast and precise collision detection algorithm for real-time virtual character animation.

The utilization of the intersection test between a sphere and a plane instead of the triangle-triangle test resulted in a much faster performance of the algorithm.. We also presented the corner-tree, a novel sphere hierarchy that makes the algorithm detect collisions correctly.

We implemented the algorithm in a virtual world composed of several interactive avatars of high level of detail and objects of different levels of detail. The velocity obtained is fast and the collision detection is precise enough. We also implemented the collision detection for an online version of our platform.

Since discrete collision detection methods sometimes miss collisions (tunneling effect), continuous collision detection is becoming an important topic of research. Most of the new CCD methods are based on discrete methods, so it seems natural to try to convert our contribution into a CCD algorithm.

In recent years, the utilization of the GPUs has become very important when accelerating algorithms' performance. Since collision detection is one of the most important bottle-neck in animation, it is important to study how GPUs can accelerate the collision detection.

8. REFERENCES

[Abd02] Abdel-Malek, K., Blackmore, D. and Joy, K. Swept volumes: foundations, perspectives, and applications. International Journal of Shape Modeling. 2002.

[Aga01] Agarwal, P. K., Basch, J., Guibas, L. J., Hershberger, J. and Zhang, L. Deformable free space tiling for kinetic collision detection. In Workshop on Algorithmic Foundations of Robotics, 83–96. 2001.

[Bro01] Brown, J., Sorkin, S., Bruyns, C., Latombe, J. C., Montgomery, K. and Stephanides, M. Real-time simulation of deformable objects: Tools and application. Computer Animation 2001, 2001.

[Cho06] Choi, Y.-K., Wang, W., Liu, Y. and Kim, M.-S. Continuous collision detection for elliptic disks. IEEE Transactions on Robotics 22, 2. 2006

[Cou06] Coumans, E. Bullet Physics library. http://www.continuousphysics.com. 2006.

[Cur08] Curtis, S., Tamstorf, R. and Manocha, D. Fast collision detection for deformable models using representative-triangles. Proceedings of the 2008 symposium on Interactive 3D graphics and games, 61-69. 2008

[Eri05] Ericson, C. Real-time Collision Detection. The Morgan Kaufmann Series in Interactive 3-D Technology. 2005

[Gae99] Gaertner B.: Fast and robust smallest enclosing balls. In ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms , Springer-Verlag, pp. 325–338. 1999.

[Got96] Gottschalk, S., Lin, M. C. and Manocha, D. Obb-tree: A hierarchical structure for rapid interference detection. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 171 – 180, 1996.

[Hub96] Hubbard, P.M. Approximating polyhedra with spheres for time-critical collision detection. ACM Transactions on Graphics (TOG), Volume 15 , Issue 3:179 – 210, 1996.

[Kav05a] Kavan L. and Zara J. Spherical blend skinning: a real-time deformation of articulated models. Proceedings of the 2005 symposium on Interactive 3D graphics and games. 9 – 16. 2005.

[Kav05b] Kavan, L. and Zara J. Fast collision detection for skeletally deformable models. Computer Graphics Forum, 2005.

[Kav06] Kavan, L., O'Sullivan, C. and Zara, J. Efficient collision detection for spherical blend skinning. Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, Fast graphics:147 – 156, 2006.

[Klo98] Klosowsky, J. T., Held, M., Mitchell, J.S.B., Sowizral, H. and Zikan, K. Efficient collision detection using bounding volume hierarchies of k-dops. IEEE Transactions on Visualization and Computer Graphics, Volume 4 , Issue 1:21 – 36, 1998.

[Lar01] Larsson, T. and Akenine-Möller, T. Collision detection for continuously deforming bodies. Eurographics, pages 325–333, 2001.

[Moh03a] Mohr, A. and Gleicher, M. Building Efficient, Accurate Character Skins from Examples, ACM Trans. Graph., Vol. 22, No. 3, pp. 562-568. 2003.

[Moh03b] Mohr, A., Tokheim L. and Gleicher M. Direct manipulation of interactive character skins. Proceedings of the 2003 symposium on Interactive 3D graphics. 27 – 30. 2003.

[Mol97] Möller, T. A fast triangle-triangle intersection test. journal of graphics tools, 2(2):25–30, 1997.

[Oya07] Oyarzun, D., Lehr, M., Ortiz, A., Carretero, M. P., Ugarte, A., Vivanco, K. and García-Alonso, A. Using Virtual Characters as TV Presenters. Technologies for E-Learning and Digital Entertainment, 225-236. 2007.

[Qui94] Quinlan, S. Efficient distance computation between non-convex objects. International Conference on Robotics and Automation, 1994.

[Red02] Redon, S., Kheddar, A. and Coquillart, S. Fast continuous collision detection between rigid bodies. Proc. Of Eurographics (Computer Graphics Forum). 2002.

[Red04] Redon, S., Kim Y.J., Lin, M.C., Manocha, D. and Templeman, J. Interactive and Continuous

Collision Detection for Avatars in Virtual Environments. IEEE Virtual Reality Conference 2004 (VR 2004). 2004.

[Sed86] Sederberg, T. W. and Parry, S.R. Free-form deformation of solid geometric models, In SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM Press, pp. 151-160. 1986.

[Tro05] Tropp, O., Tal, A. and Shimshoni, I. A fast triangle to triangle intersection test for collision detection. Journal of Graphics Tools, Volume 2 , Issue 2:25 – 30, 2005.

[Van98] Van Den Bergen, G. Efficient collision detection of complex deformable models using aabb trees. Journal of Graphics Tools, Volume 2 , Issue 4:1 – 13, 1998.

[Van04] Van Den Bergen, G. Ray casting against general convex objects with application to continuous collision detection. Journal of Graphics Tools. 2004.

[Zac06] Zachmann, G. and Weller, R. Kinetic bounding volume hierarchies for deformable objects. Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications, Session F5:189 – 196, 2006.

[Zha07] Zhang, X., Redon, S., Minkyoung, F. and Kim, Y.J. Continuous collision detection for articulated models using Taylor models and temporal culling. International Conference on Computer Graphics and Interactive Techniques. 2007