Available online at www.sciencedirect.com
SCIENCE DIRECT
Computers & Graphics 30 (2006) 432–438
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Interruptible collision detection for deformable objects

C. Mendoza[a,*], C. O'Sullivan[b]

[a]Virtual Reality and Modeling Group, Deptal. II, Desp. 147, Rey Juan Carlos University, c/Tulipan s/n, 28933 Mostoles (Madrid), Spain
[b]Image Synthesis Group, Computer Science Department, Trinity College Dublin, Dublin 2, Ireland

Abstract

This paper presents an approach to performing time-critical collision detection for deformable objects. The deformable objects are represented by dense meshes and their deformations are steered by a coarser mesh (reduced model) based on explicit finite elements, to achieve an interruptible algorithm, we use a sphere tree constructed using an adaptive medial-axis approximation of the dense mesh. The bounding spheres are updated using the coarse mesh, thus balancing computational accuracy and speed.

Keywords: Collision detection; Interruptible algorithms; Deformable objects

1. Introduction

Physically based modeling of deformable objects and collision detection have been extensively researched. Different approaches have been proposed to reproduce physically correct behaviors of deformable objects in real-time. Mostly, they are based on simplifications of more complex representations or on the use of fast and stable integration techniques on their governing equations. More recently, different levels of resolution of the physical model, *reduced models*, have been used to speed up the simulation.

In collision detection, most effort has been focused on solving the collision detection problem for rigid bodies. Most of the proposed techniques tackle the problem in two phases. The first phase, a *broad phase*, culls out pairs of objects that cannot possibly be interacting. The second phase, *narrow phase*, carries out more detailed intersection calculations. To accelerate the collision queries, this narrow phase most often uses pre-computed data structures that are hierarchical representations of the objects. To ensure real-times rates throughout the simulation, it is desirable to use a time-critical system which can interrupt the collision detection process to fit a time budget. Unfortunately, these techniques cannot be applied directly in cases where the objects are deformable, since the data structures need to be updated after every deformation. The update process is normally slow and constitutes a major bottleneck for real-time computations. Real-time collision detection for deformable objects is therefore a growing research area.

1.1. Contributions and outline

We present an approach to performing time-critical collision detection for deformable objects. To our knowledge, all previous collision detection algorithms that used an interruptible, or *just in time*, system were focused on rigid bodies and none of them on deformable objects. Our approach is based on [1] since we also use a reduced model to update the hierarchy tree. However they do not consider a fully interruptible system and some artifacts arise due to the use of a simple bounding volume generator.

The main idea behind our approach is to trade accuracy for speed in order to guarantee that collision processing is always performed in less than or equal to a stipulated *critical time*. First we create a deformable model based on two different resolution representations: The first resolution is a dense triangular mesh that we use for graphical rendering and the second is a coarser mesh (*reduced model*) used for deformations. The coarser mesh uses inner tetrahedrons so explicit finite elements can be used. The

*Corresponding author. Tel.: +34 91 488 8300; fax: +34 91 488 7049.
E-mail addresses: cesar.mendoza@urjc.es (C. Mendoza), carol.osullivan@cs.tcd.ie (C. O'Sullivan).

0097-8493/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2006.02.018

dense mesh follows the deformations of the coarse mesh by using *rigid links*. Next, we produce multiple approximations of the object's real surface using a sphere hierarchy. Our collision detection algorithm checks for intersections between such hierarchies until it exceeds the given critical time. The sphere hierarchies are not updated using the vertices of the surface of the object, but rather the vertices of the reduced model, thus reducing the time to update the hierarchies.

The rest of the paper is organized as follows. Section 2 gives an overview of some related works in deformable objects and collision detection. Then, in Section 3, we describe how we simulate deformations by using two different resolution meshes. In Section 4 we present our approach to handling collision detection between deformable objects using an interruptible mechanism, followed by experiments and results in Section 5. Finally, some conclusions and future plans are discussed in Section 6.

2. Related work

In general, there are two types of deformable objects in computer graphics: *geometrically based* and *physically based*. Geometrically based deformable objects change their shape by moving some control points or by calculating implicit functions (e.g. Free Form Deformation models [2]). Physically based models use physical laws and material properties to model the object. Among the best known are the mass-spring dampers [3], boundary element [4] and finite element [5,6] methods. The latter are the most accurate, but interactivity can be lost if the objects have a large number of primitives. To handle this, we can use multiresolution models (i.e., a set of reduced models) to achieve real-time simulations. Debunne et al. [7] use space and time adaptive sampling to achieve dynamic real-time deformations. Kondo and Kanai [8] simulated dense meshes using an underlying reduced physical model. A similar approach has been proposed in [9].

A significant amount of research has been focused in rigid body collision detection. Most of the resulting techniques need to be modified for deformable objects (please refer to [10,11] for recent surveys). The general process is divided in a broad phase to cull out non interacting pairs of bodies and in a narrow phase that traverses hierarchical representations of the objects to find intersecting regions. In general, these representations are made of bounding volumes such, as oriented bounding boxes (OBB) [12], *k*-dops [13], including their special case, the 6-dop axis-aligned bounding boxes (AABBs) [14], and sphere trees [15,16]. For rigid body time-critical collision detection, spheres have also been used to generate approximate responses to contacts [17]. Time-critical collision detection, also known as *graceful degradation*, was first proposed by Hubbard [18]. The objects are represented by sphere trees and collisions are tested in round-robin order, progressively increasing the level of accuracy until the interruptible mechanism stops the process after a given time. Later, this approach was extended to improve the mechanism for collision scheduling, contact modelling and collision response [17]. Klein and Zachmann [19] proposed an average case approach (ADB-trees) to abort the traversal of the hierarchy in a time-critical framework. They consider the probability that a pair of bounding volumes contains intersection volumes. To date, these approaches have only been used for collision detection between rigid bodies.

In collision detection for deformable objects, the hierarchies must be updated at each timestep that the object deforms. This update process can be very slow and thus the simulation may not meet real-time demands. Van den Bergen compared AABBs and OBBs [14] for deformable objects and determined that AABBs are the best option. He also showed that, although the hierarchies can also be rebuilt, updating is almost ten times faster than rebuilding. Larsson and Akenine-Möller [20] compared different methods for the hierarchy updating process based on bottom-up and top-down strategies. They found that these methods depend on the number of deep nodes processed. Based on this, they proposed an hybrid method that uses both strategies. Mezger et al. [21] speeded up the process by updating the hierarchy alter a few time steps and then only those branches whose primitives that have moved farther than a given distance. Recently, an approach to update the hierarchies by means of a reduced model has been proposed by James and Pai [1]. Alternatively, hardware accelerated collision detection methods have shown promising results [22,23]. However, accuracy is still limited due to the non-floating point precision and the size of the frame buffer memory. Other techniques based on spatial hashing have shown good results [24].

3. Deformable model approach

In this section, we present our dynamic model for the simulation of object deformations. It is based on two meshes: a dense mesh composed of a large number of triangles and used for graphical rendering. The vertices of this mesh are repositioned at each time-step to simulate complex deformations. These new positions of the vertices are steered by a coarser mesh, *reduced model*, see Fig. 1. This reduced model can be any physically based elastic model. In our case, we have implemented an explicit finite-element model as proposed by O'Brien and Hodgins [5] and used a fourth-order Runge-Kutta integration scheme.

3.1. Linking the coarse and dense meshes

The procedure to link the dense and the coarse meshes is done in an off-line process. Suppose that the dense mesh has $\mathbf{P} = (p_1, p_2, \ldots, p_N)^{\mathrm{T}}$ vertices on its surface. Next, consider only the vertices on the surface of the coarse tetrahedral mesh: $\mathbf{Q} = (q_1, q_2, \ldots, q_M)$. These are the reduced coordinates of the model. Note that $M < N$. Since

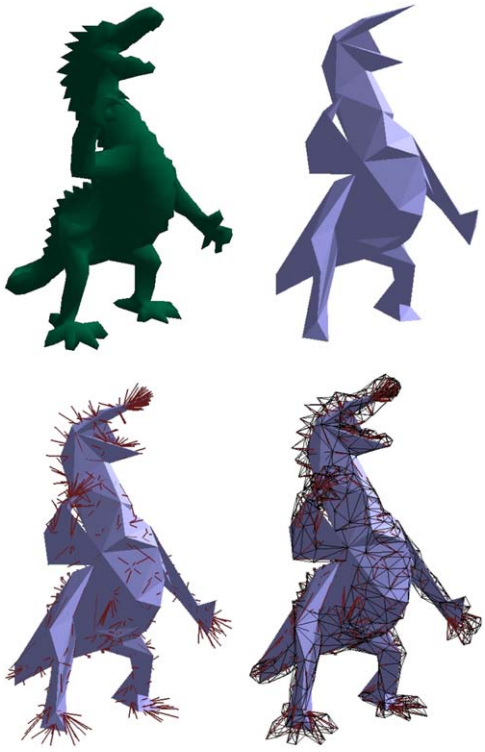

Fig. 1. Top left figure shows a dense mesh whose deformations are steered by a coarser mesh, top right. Rigid links are used to link both meshes, bottom.

the dense mesh vertices are steered by the reduced coordinates, we can establish that $\mathbf{P} = f(\mathbf{Q})$.

For each p_i we find its *three closest reduced coordinates* q_0, q_1, q_2. The idea is to find a *rigid link* between the vertex p_i and a point O weighted by the three closest reduced coordinates, see Fig. 2. The rigid link will never vary its length but its origin, O, and its orientation change. The orientation changes with respect to the reduced coordinates, q_i, $(i = 0, 1, 2)$. These changes are executed following precomputed weights in the undeformed reference axis.

To compute the origin of the rigid link, O, we use:

$$O = q_0\alpha_0 + q_1\alpha_1 + q_2\alpha_2, \tag{1}$$

where α_i represents the weight of each of the three reduced coordinates. These weights are obtained using the distances, in the undeformed configuration, between the reduced coordinates and its given vertex in the dense mesh, see Fig. 2. Let the distances be given by

$$d_i = \|q_i - p\|_2, \quad i = 0, 1, 2. \tag{2}$$

Therefore the weights can be computed as follows:

$$\alpha_0 = \frac{d_1 + d_2}{\sum_i d_i}; \quad \alpha_1 = \frac{d_0 + d_2}{\sum_i d_i}; \quad \alpha_2 = \frac{d_1 + d_0}{\sum_i d_i}. \tag{3}$$

These *weights are constant* and they are not recomputed during the simulation. When the vertices of the tetrahedral coarse mesh change their position (i.e. when a deformation occurs) only the new origin of the rigid link needs to be recomputed following Eq. (1).

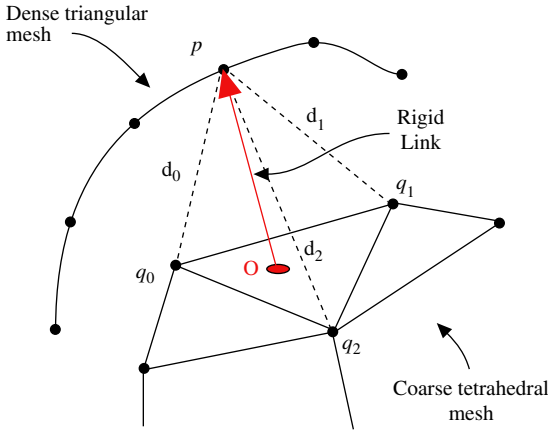

Fig. 2. The length of the rigid link remains constant but its orientation and origin changes to adapt to the new shape of the deformable object.

To keep the magnitude of the rigid link constant with respect to the three reduced coordinates, we continue as follows: let n be the normal of the plane spanned by the reduced coordinates, i.e.

$$n = (q_1 - q_0) \times (q_2 - q_0) \tag{4}$$

and set

$$a = (q_1 - q_0) \times n, \tag{5}$$

see Fig. 3.

Define a fixed and undeformed basis constituted by the vectors:

$$e^0 = n; \quad e^1 = (q_0 - q_1); \quad e^2 = a. \tag{6}$$

Let the magnitude of the projections of the rigid link onto these axes be $\|Pr_i\|_2$, $i = 0, 1, 2$ (e.g. $\|Pr_1\|_2$ is the projection of the rigid link onto the n axis). The sign, ξ_i, of these projections is given by:

$$\xi_i = \frac{e^i \cdot Pr_i}{\|e^i\|_2 \|Pr_i\|_2}; \quad i = 0, 1, 2. \tag{7}$$

Thus, we have that the rigid link is defined as:

$$R_l = \sum_{i=0}^{2} \xi_i Pr_i. \tag{8}$$

3.2. Steering the dense mesh

When the vertices of the coarse mesh are repositioned, meaning that there has been a deformation, we use the rigid links to update the vertices of the dense mesh. Using the new positions of the reduced coordinates, we recompute the origin of the rigid link O_{new} using Eq. (1), where the values of α_i have been computed off-line. To update the orientation of the rigid links, we use the new reduced coordinates to compute the new reference axis as we did for e^i. Define the normalized axis as e^i_{deformed}. Hence, the rigid

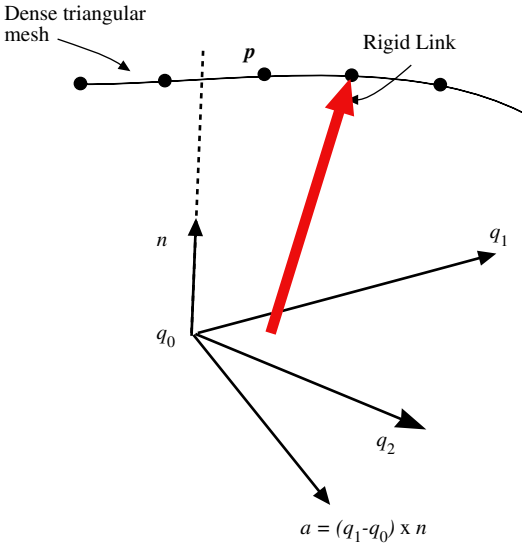

Fig. 3. The rigid link joins each vertex of the surface dense mesh to the tetrahedral coarse mesh.

link in the new basis is given by:

$$R_{l(\text{new})} = \sum_{i=0}^{2} \xi_i \|Pr_i\|_2 e^i_{\text{deformed}}, \quad i = 0, 1, 2. \tag{9}$$

Note that ξ_i and $\|Pr_i\|_2$ have been computed off-line. Therefore, the new position of the vertex, p, is computed using the new origin and the new orientation of the rigid link as follows:

$$p_{\text{new}} = O_{\text{new}} + R_{l(\text{new})}. \tag{10}$$

4. Collision handling

A time-critical collision detection mechanism "checks for collisions between successively tighter approximations of the objects' real surfaces" [15]. We use hierarchies of spheres, known as *sphere-trees*, as approximated representations of the objects. There are some advantages to using hierarchies of spheres: it is easy and fast to test for intersections between them; they are invariant to rotations and therefore they can be efficiently updated; and most importantly, they can easily be adapted to provide approximate contact modelling and collision response in an interruptible algorithm [17].

4.1. Sphere tree construction

There exist several methods to construct a sphere-tree [15,16]. The idea is to construct hierarchies of spheres in which each level of the hierarchy represents a tighter fit to the object. Guibas et al. [25] classified these hierarchies as being either *layered* or *wrapped*. In a layered hierarchy the spheres always enclose their child spheres, while in a wrapped hierarchy this is not necessarily true since regions of the child spheres can be outside of their enclosing parent sphere; Although less conservative, the wrapped hierarchy is always tighter than the layered hierarchy. We construct, in an off-line process, a wrapped hierarchy based on an *adaptive medial axis approximation* [16]. The medial axis approximation of the object represents its *skeleton* and it is used to place the sphere hierarchies on the surface of the object. The medial axis is updated, i.e. *adapted*, during the sphere-tree construction to ensure a higher degree of accuracy. Additionally, we use the data structure proposed in [17] to achieve fast hierarchy traversals and low memory storage.

4.2. Interruptible approach for collision handling

Traditional hierarchy traversal for collision, detection is performed using a *depth-first search* [11,14]. The general algorithm can be summarized as depicted in Fig. 4.

In this search method the hierarchy trees are examined in vertical directions, going down to the leaves for each search. See Fig. 5.

If the process is suddenly interrupted, many branches, from the root to the leaf, may remain untested, leading to missed collisions. For example, suppose that the hierarchies of Fig. 5 collides at the $(4, e)$ and $(12, m)$ leaf sphere pairs. If the process is interrupted at the fourth path search (i.e. $(1, f)$), the collision between the $(12, m)$ leaf sphere pair will have never been tested, and even worse, the sphere pair parents tests $((0, d), (3, d))$ will be missed as well. Therefore,

```
traverse ( O, a )
  if no overlap(0, a) then
      return;
  endif
  if a and 0 are leaves then
      return primitives tests for a and 0
  else
     for all children of a[i] and 0
        traverse( a[i], 0)
     endfor
  endif
```

Fig. 4. Pseudocode of traditional traversal algorithm.

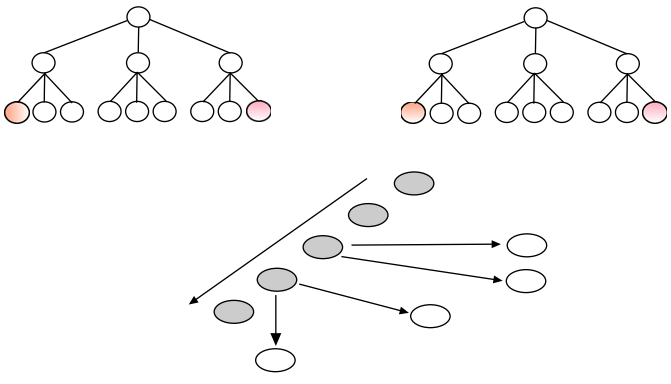

Fig. 5. Example of a traditional hierarchy traversal for collision detection.

depth-first search is certainly not an option for time-critical processing, unless the search is capped at a given depth.

Instead of the traditional traversal, we propose to use a *breadth-first search* to frame our algorithm. This search technique traverses the hierarchy horizontally, progressing successively through tighter approximations of the object. Therefore, if the allocated budget time expires, at least the algorithm would have tested collisions between approximations of the object without missing any branch.

In Fig. 6, if the process is interrupted at the fourth path search, then we would have tested all the branches although not reaching the leaf sphere pairs. Hence, we use approximations of the object surface to compute the collision responses. Fig. 7 summarizes our proposed algorithm:

There are several key aspects in our traversal algorithm:

- It contains an entity called *pair*, where we store spheres to be tested for overlapping. The order in which we store the spheres in the entity changes during the traversal. This allows us to traverse the hierarchy in a stepped and optimized way. See Fig. 6.
- It has two FIFO (first input first output) lists. PAIRL1ST stores the pairs of spheres to be tested for collision and COLLISIONLIST stores the colliding pairs of leaf spheres.
- The algorithm checks at each sphere collision test that the allocated time is not exhausted. If this is the case then the process is *interrupted*.

Ideally, if the process is not interrupted, we use the spheres in the pairs of COLLISIONLIST to compute the force responses. Note that we do not reach the primitives (e.g. facets) on the surface. We are trading accuracy for speed. However, if the process is interrupted then we use PAIRLIST and COLLISIONLIST to approximate the force responses. We assume, therefore, that untested sphere pairs in PAIRLIST are colliding.

4.3. Collision response

To compute the approximated contact force we used penalty techniques based on the proportional penetration distance between the colliding spheres as in [17]. Each

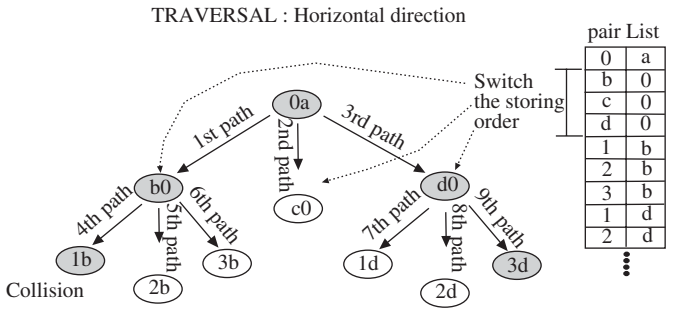

Fig. 6. Example of a horizontal hierarchy traversal.

```
traverse ( 0, a )
    Pair p;
    p.1 = 0; // root object 1
    p.2 = a; // root object 2
    List pairList;
    List collisionList;
    pairList.add( p );
    while ( pairList.isNotEmpty() .AND. notTimeCritical() )
        p = pairList.getFirstElement();
        pairList.RemoveFirstElement();
        if ( p.Overlaps())
            if ( p.2.isLeaf )
                if ( p.1.isLeaf )
                    collisionList.add( p );
                else
                        Pair pB;
                        pB.1 = p.2;
                        pB.2 = p.1;
                        pairlList.add( pB );
                endif
            else
                forall p.2.children;
                    Pair pB;
                    pB.1 = p.2.children[i];
                    pB.2 = p.1;
                    pairList.add( pB );
                endfor
            endif
        endif
    endwhile
```

Fig. 7. Pseudocode of interruptible traversal algorithm.

sphere contains information about the penetration with respect to the corresponding colliding sphere and the direction of the force repulsion vector. We compute the penetration for each pair of spheres in COLLISIONLIST (and eventually in PAIRLIST if the process is interrupted). The corresponding repulsion vector for the spheres is given by the vectors between the center of the spheres, i.e.

$$n = \frac{O_2 - O_1}{\|O_2 - O_1\|_2}. \tag{11}$$

The direction repulsion vector is given by $-n$ and n. Additionally, each sphere is related to a set of closest vertices in the coarse mesh. The spheres considered to be colliding apply their stored contact force to the closest coarse vertex, as in a classical boundary condition of finite element methods. Each coarse vertex averages the received forces to cause deformations. Finally, at the end of the simulation cycle, the finer mesh follow the deformations of the coarse mesh through the rigid links as described in Section 3.2.

4.4. Hierarchy update

Our hierarchy update is based on recent research by James and Pai [1], in which updates are performed using a set of coordinates of a reduced model instead of the vertices of the rendered and dense mesh. There are however some key factors that makes our update proposal different. They

assumed that each sphere contained at least a set of polygons with associated vertex points. This leads to large bounding spheres for objects with long thin triangles. The sphere-tree hierarchy generator that we use [16] avoids this situation since spheres can be placed on the surface of the triangles ensuring complete coverage and a higher degree of accuracy, as shown in Fig. 8.

To update the position and radius of the spheres we use the set of closest vertices in the coarse mesh, V_i, obtained in a pre-processing stage. The new position center is updated by adding the average displacements of the vertices in V_i to the original center. Results are better for a larger number of vertices in V_i, however increasing this number may lead to slower updates rates. The diameter of the spheres is updated using an heuristic as follows: we take the distance from the center of a given sphere to each of its associated coarse vertices at the undeformed configuration. At each deformation, these distances change; Let the maximum ratio between the new and the original distances be λ. Hence, the new radius is given by $r_{\text{new}} = \lambda r_{\text{original}}$.

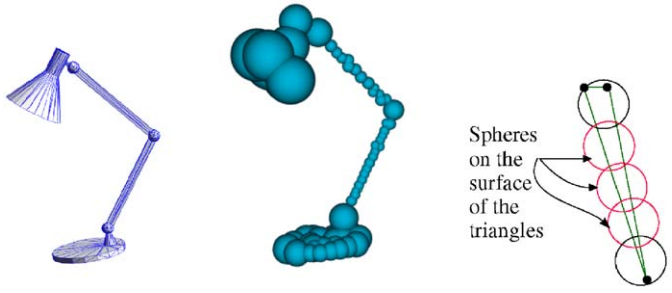

Fig. 8. Putting spheres on the surface of the triangles avoids large bounding spheres.

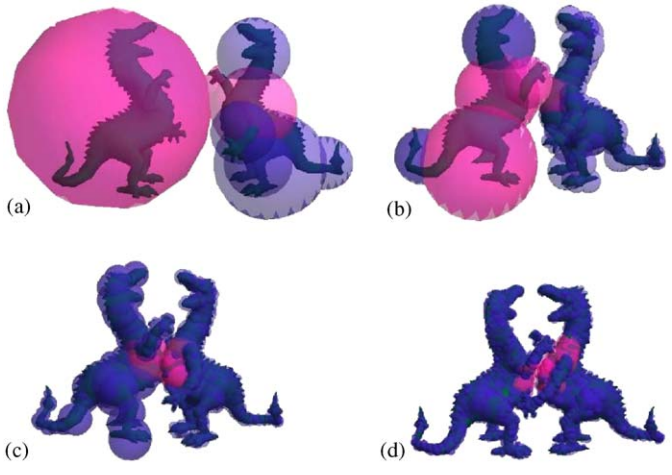

Fig. 9. Red spheres are used to compute contact responses at different critical times: (a) 0.0001 seg, (b) 0.001 seg, (c) 0.01 seg, and (d) 0.1 seg.

5. Results

In our tests we used a Pentium 4 CPU, 3.00 GHz with 1.00 GB RAM under a windows XP system. The graphics card is a NVIDIA GeForce 6800 GT. The dense surface mesh was simplified under a quadratic error metrics approach [26]. A tetrahedralization from the simplified surface mesh was obtained using *NetGen* [27].

To test our algorithm we have simulated the interaction of two virtual objects. They deform following the dynamic model described Section 3. We have run the simulation several times and each one has had a different time budget allocated to its collision detection module. Each simulation starts with identical initial conditions (e.g. same physical parameters, same object positions). The collision process of each simulation is interrupted at different instants. Fig. 9 shows four simulations whose collision detection mechanisms are interrupted. Simulations in Fig. 9(a,b,c) use spheres (colored in red) in the middle of the sphere-tree hierarchy to compute contact responses, while simulation in Fig. 9(d), having a larger budget of time for the collision process, uses leaf spheres.

Different types of interactions have been tested and the corresponding results are summarized in Table 1.

Notice that for all the interactions when the critical time is small the COLLISIONLIST is empty, i.e. the collision process is interrupted before it can store any leaf sphere in the list. When we increase the critical time allowed, the COLLISIONLIST starts storing elements which will provide a more accurate response. However, the number of elements in the PAIRLIST increases dramatically, meaning that the non-tested collision spheres pairs increased. This supposes that our approach is memory consuming. Moreover, the frames per second (FPS) increase as we allocate less time to the collision process. Hence, we can profit from this extra time and use it in other processes, e.g. the time-step of the integration scheme can be modified.

Table 1
Results of interactions

Critical time	Dragon *pairList*	Dragon *collisionList*	FPS (Dragons)	Bunny *pairList*	Bunny *collisionList*	PFS (Business)	Bunny-Dragon *pairList*	Bunny-Dragon *collisionList*	PFS (Bunny-Dragon)
0.0001	3	0	60	3	0	52	3	0	56
0.001	17	0	60	25	0	48	21	0	51
0.01	30	0	42	44	0	37	33	0	42
0.1	41	6	35	61	11	30	52	12	33
1	1345	30	25	1405	36	22	1387	28	23

6. Conclusion

In this paper we introduce an approach to performing time-critical collision detection for deformable objects. To our knowledge, previous fully interruptible collision detection methods were focused on rigid bodies and none of them to deformable bodies. Our approach is well suited for applications that do not require high precision but that need to keep a constant frame rate. Moreover, instead of interrupting the collision detection process using critical times, the same algorithm could be useful for accelerating collisions using level-of-detail as a metric. Thus, we can interrupt, for example, collisions occurring farther away and therefore high levels of visual accuracy during the collisions would not be needed (motivating the use of larger bounding spheres). We have also improved BD-trees by using an adaptive medial axis approximation and by using heuristics to associate vertices of the reduced model to spheres that do not enclose any vertex.

There are some limitations of this approach. Storing a set of primitives (in our case, a set of vertices of the tetrahedral mesh) represents a high level of memory consumption. Additionally, our interruptible mechanism is based on a breadth-first search that keeps a list of possible pairs of colliding spheres. For hierarchies whose levels have a large number of nodes, the storage in the lists may be very high. Future work includes the evaluation of the accuracy of our approach for large deformation and force computation using reduced models.

References

[1] James D, Pai D. BD-Tree: output-sensitive collision detection for reduced deformable models. ACM Trans Graphics (SIGGRAPH 2004) 2004;23(3).

[2] Faloutsos P, de Panne MV, Terzopoulos D. Dynamic free-form deformations for animation synthesis. IEEE Trans Vis Comput Graphics 1997;3(3):201–14.

[3] Kuhnapfel U, Akmak H, Maass H. Endoscopy surgery training using virtual reality and deformable tissue simulation. Comput Graphics 2000;24:671–82.

[4] James D, Pai D. Artdefo: accurate real-time deformable objects. In: Proceedings of ACM SIGGRAPH 1999; 1999. p. 65–72.

[5] O'Brien J, Hodgins J. Graphical modeling and animation of brittle fracture. In: Proceedings of the ACM SIGGRAPH 1999. New York: ACM Press/Addison-Wesley Publishing Co.; 1999. p. 137–46.

[6] Dorsey JMM, McMillan L, Jagnow L. Stable real-time deformations. In: ACM SIGGRAPH symposium on computer animation (SCA '02); 2002. p. 49–54.

[7] Debunne GM, Desbrun MPC, Barr A. Dynamic real-time deformations using space and time adaptive sampling. In: Proceedings of the ACM SIGGRAPH '01; 2001. p. 31–6.

[8] Kondo R, Kanai T. An interactive physically-based animation system for dense meshers. In: Proceedings of the short presentations and interactive demos, Eurographics '04; 2004. p. 93–6.

[9] Capell S, Green S, Curless B, Duchamp T, Popovic Z. A multiresolution framework for dynamic deformations. In: Proceedings ACM SIGGRAPH symposium on computer animation; 2002.

[10] Jimenez P, Thomas F, Torras C. 3d collision detection: a survey. Comput Graphics 2001;21(3):269–85.

[11] Teschner M, Kimmerle S, Heidelberger B, Zachmann G, Raghupathi L, et al. Collision detection for deformable objects. In: Schilick C, Purgathofer W, editors. Proceedings state of the art reports, Eurographics '04; 2004. p. 119–40.

[12] Gottschalk S, Lin MC, Manocha D. Obbtree: a hierarchical structure for rapid interference detection. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on computer graphics and interactive techniques. New York: ACM Press; 1996. p. 171–80.

[13] Klosowski JT, Held M, Mitchell JSB, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of *k*-dops. IEEE Trans Vis Comput Graphics 1998;4(1):21–36.

[14] van den Bergen D. Efficient collision detection of complex deformable models using aabb trees. J Graphic Tools 1997;2(4):1–13.

[15] Hubbard P. Approximating polyhedra with spheres for time-critical collision detection. IEEE Trans Vis Comput Graphics 1996;15(3): 79–120.

[16] Bradshaw G, O'Sullivan C. Adaptive medial-axis approximation for sphere-tree construction. ACM Trans Graphics 2004;23(1): 1–26.

[17] Dingliana J, O'Sullivan C. Graceful degradation of collision handling in physically based animation. Comput Graphics Forum 2000;19(3): 239–48.

[18] Hubbard P. Collision detection for interactive graphic applications. IEEE Trans Vis Comput Graphics 1995;1(3):218–30.

[19] Klein J, Zachmann G. Adb-trees: controlling the error of time-critical collision, detection. In: 8th international fall workshop vision, modelling, and visualization (VMV). Germany: University München; 2003. p. 19–21.

[20] Larsson T, Akenine-Möller T. Collision detection for continuously deforming bodies. In: Eurographics, short presentations; 2001. p. 325–33.

[21] Mezger J, Kimmerle S, Etzmuss O. Hierarchical techniques in collision detection for cloth animation. J WSCG 2003;11(2): 322–9.

[22] Manocha D, Lin M, Doggett M, Greene N, Hoff K, Kilgard M. Interactive geometric computations using graphics hardware. In: SIGGRAPH 2002 course notes. ACM SIGGRAPH; 2002.

[23] Govindaraju N, Lin M, Manocha D. Fast and reliable collision culling using graphics processors. In: IEEE transactions on visualization and computer graphics 2005; 2004 (Special issue on best papers of ACM VRST'04).

[24] Teschner M, Heidelberger B, Mueller M, Pomeranets D, Gross M. Optimized spatial hashing for collision detection of deformable objects. In: Proceedings of vision, modelling, visualization VMV'03; 2003. p. 47–54.

[25] Guibas L, Nguyen A, Russel D, Zhang L. Collision detection for deforming necklaces. In: 8th annual symposium on computational geometry. New York: ACM Press; 2002. p. 33–42.

[26] Garland M, Heckbert PS. Surface simplification using quadric error metrics. In: SIGGRAPH '97: Proceedings of the 24th annual conference on computer graphics and interactive techniques. Reading: ACM Press/Addison-Wesley Publishing Co.; 1997. p. 209–16.

[27] Schoberl J. Netgen—an advancing front 2d/3d mesh generator based on abstract rules. In: Computations in visualization and science; 1997. p. 41–52.