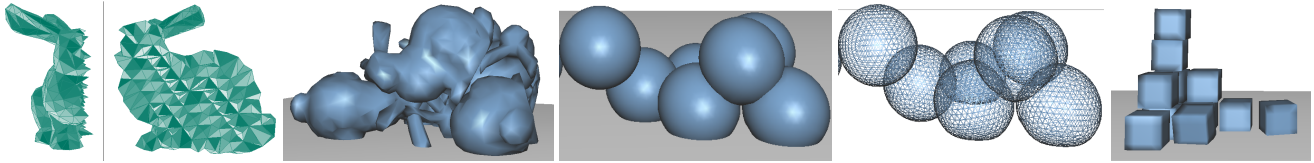


Simulation of Deformable Bodies Based on Tetrahedral Meshes and Shape Matching

Guina Sotomayor Alzamora*

Yalmar Ponce Atencio

Claudio Esperança

 Universidade Federal do Rio de Janeiro
 Laboratório de Computação Gráfica - LCG


Abstract

We present a simplified approach for animation of geometrically complex deformable objects represented by tetrahedral meshes. Our prototype system detects and responds to collisions of objects subject to elastic deformations of variable stiffness. The proposed approach combines several techniques, namely, collision detection using spatial hashing, collision response through a contact surface with consistent penetration depth obtained by propagation, estimated displacement vectors for the deformation region, and binary search to separate objects. The dynamics is based on shape matching and a modal analysis scheme, using an Euler explicit-implicit integrator. Preliminary results show that collisions between objects containing several hundreds tetrahedra can be animated in real-time.

Keywords:: Collision detection, physically based animation, deformable bodies

Author's Contact:

{yalmar, esperanca}@lcg.ufrj.br
 *guinas@gmail.com

1 Introduction

Physically based animation is a computational process, where animated objects behave in a physically plausible manner. This is in contrast to “physics animation”, which is employed to design simulations where physically correct behavior is sought. This work adopts the former rationale, since it employs simplified approaches, emphasizing performance over accuracy. Nevertheless, as with other physically based models, motion is still produced based on the same physical principles; namely, dynamics is based on Newton's laws.

Methods for animation of both rigid bodies and deformable bodies are commonly based on particle systems. In addition to the requirements for the animation of rigid bodies, deformable objects also require some physical deformation model such as mesh-based methods like mass-spring [Baraff and Witkin 1998] and finite elements, or meshless methods [Müller et al. 2004; Müller et al. 2005; Pauly et al. 2004]. Deformable objects also require a more involved representation, since their shapes can change in time, due to interaction with other objects or themselves.

This work focuses on the use of a simple methodology for animating geometrically complex deformable objects represented by tetrahedral meshes. The goal is obtain a stable dynamic simulation with physically plausible behavior. The presented method combines several techniques employed for the animation of deformable bodies, namely:

- A broad phase collision detection based on bounding spheres for filtering the set of potentially colliding objects.
- Narrow phase collision detection employing the *Spatial Hashing* approach [Teschner et al. 2003].

- Collision response is achieved by computing the penetration depth [Heidelberger et al. 2004] of all collided vertices. For asymmetric collisions we use the free obstacle method described in [Jakobsen 2001]. Finally, the separating process is based on a binary search [Heidelberger et al. 2004], which computes the contact surface.
- Dynamics is based on shape matching [Müller et al. 2005] which does not need a mesh representation of the model – the tetrahedral mesh is used only in the collision detection and response stages.
- The system integration uses semi-implicit Euler method [Müller et al. 2005].

Preliminary results are shown interacting objects with several hundred of tetrahedra in real time. On the other hand, the material properties are not yet modeled in our prototype.

The remainder of text is organized as follows: Section 2 reviews related works in collision detection and deformable models. Section 3 describes some required prerequisites. Section 4 describes the collision detection process in its two phases, broad and narrow. Section 5 describes the collision response process. Section 6 describes the shape matching technique employed for the dynamics. Section 7 describes the integration method. In Section 8 some results using our prototype are shown. Finally, concluding remarks and suggestions for future work are discussed in Section 9.

2 Related Work

Many methods and models have been proposed in computer graphics to simulate deformable objects including finite difference approaches [Terzopoulos et al. 1987], mass-spring systems [Baraff and Witkin 1998] Finite Element Methods (FEM) [Müller et al. 2002; Müller et al. 2004], modal analysis [Shen et al. 2002] and mesh-free particle systems [Müller et al. 2005]. Most of these approaches focus on the accurate simulation of elasto-mechanical properties which demand a costly processing, notwithstanding several acceleration strategies such integration schemes for large time-steps [Baraff and Witkin 1998] and shape matching in place of accurate modal analysis [Müller et al. 2005].

Collision detection encompasses a whole research area by itself. Approaches are usually divided into broad and narrow phase methods:

- The Broad phase aims at reducing the quadratic number of potentially colliding pairs of objects to a more manageable number. Several schemes using spatial data structures such as Octrees, Sphere Trees or BSP-trees have been proposed [Samet 2006]. The popular *sweep and prune* technique [Cohen et al. 1995] maintains a sorted list for each of the principal axes, where the elements in the list are intervals obtained by projecting the geometry of the object onto the axis.
- The goal of the Narrow phase is to detect pairs of objects which indeed collide. Methods are usually classified into four groups: Bounding volume (Axis Aligned Bounding

Box, Oriented Bounding Box, Bounding Sphere, discrete oriented polytope, among other) hierarchies [Cohen et al. 1995; Gottschalk et al. 1996; van den Bergen 1997; Bradshaw and O'Sullivan 2004; James and Pai 2004], spatial subdivision (grids) [Ganovelli et al. 2000], spatial subdivision hierarchies (Octrees, k-d Trees, BSP-Tree, among other) [Samet 2006; Luque et al. 2005] and image-space techniques [Shinya and Forgue 1991; Knott and Pai 2003; Heidelberger et al. 2004].

Among these, the preferred narrow phase methods for detecting the collision of rigid bodies tend to be those based on bounding volume hierarchies. Unfortunately, they are not very suitable for handling deformable objects, since the cost for updating the hierarchies cannot be amortized with the pre-processing of fixed shapes. On the other hand, spatial subdivision schemes lend themselves more easily to this task. In particular, we chose to employ the *spatial hashing* method from Teschner [Teschner et al. 2003], which supports efficient updating.

Collision response consists of updating properties such as shape, position and velocity of objects involved in collisions. A common approach is to compute penalty forces for penetrating object vertices as a function of their penetration depth, i.e., the minimum translation for separating them [van den Bergen 1999]. In this work we use Heidelberger's consistent penetration method [Heidelberger et al. 2004], which computes the penetration depth for all collided vertices. Later, that information is used for separating the objects.

3 Preprocessing

Firstly, a spatial hash data structure must be defined. In essence, the idea consists of using a grid to subdivide the space where collisions are expected to occur into cells of equal size. Similarly, each moving body is subdivided into tetrahedra. The method then detects collisions by keeping track of voxels (cells) intersected by tetrahedra associated with different objects. Rather than using a 3D array of cells, however, a more memory efficient scheme consists of mapping grid cells to a much smaller array using a hash function. Thus, the data structure is dependent on the following parameters:

Hash table size. Its optimal size is related with the number of primitives in the scene, and must be a high prime number in order to minimize address collisions in the hash table.

Grid cell width. Clearly, this should be of the same size order as the objects' tetrahedra. Thus, a reasonable choice is to employ the tetrahedra's average edge length.

Hash function. Maps a grid cell to an arbitrary hash table address. An efficient function should be able to achieve a good distribution. In our prototype, following function is used:

$$h = \text{hash}(i, j, k) = (i c_i \oplus j c_j \oplus k c_k) \bmod n, \quad (1)$$

where $\oplus$ stands for bitwise exclusive-or operation, i, j, k are grid coordinates, c_i, c_j and c_k are high prime numbers and n is the hash table size.

Additionally, the minimum bounding sphere for each object is pre-computed. These are used in the broad phase collision detection (see Section 4.1). In order to accommodate the stretching of deformable objects, an error margin is added to the sphere radius which is dependent on the object's stiffness constant α (see Section 7). In particular, for a deformable object with stiffness α whose rest shape has a bounding sphere of radius r , we consider a sphere of radius $\frac{2r}{1+\alpha}$.

4 Collision detection

In this work, we are mainly interested in simulating a relatively small number of objects (a few tens), although each object may be arbitrarily complex. This has guided the choice of collision detection schemes detailed below.

4.1 Broad phase

The goal of this phase is to select regions on the grid where collisions may have taken place. Therefore, a simple verification between each pair of objects is done, considering only their bounding spheres. For each pair of objects A and B , collision potentially occurs if the distance between their centers is less than the sum of their radii, i.e.,

$$|c_1 - c_2| < r_1 + r_2. \quad (2)$$

For each pair of objects, the vertices involved in the region of potential collision ($v \subset A \cap B$) are collected in order to be later checked for collision during the narrow phase. For this purpose, it suffices to verify if a vertex $v \in A$ is inside the bounding sphere of B and vice-versa (see Figure 1).

An important difference between our method and Teschner et al.'s approach is that rather than updating the grid with the new positions of all tetrahedra, only those associated with objects potentially involved in collisions are considered. Thus, a tetrahedron t_A associated with object A , whose bounding sphere collides with that of B , will be updated in the grid only if it also intersects B 's bounding sphere.

It should also be noted that *timestamps*, as discussed in [Teschner et al. 2003] are also used in our prototype. The idea is to avoid cleaning up the grid after each frame by labeling each grid cell with a timestamp associated with the moment it was last updated. Thus, a collision with primitives inside a given grid cell is considered only if it was updated in the current iteration.

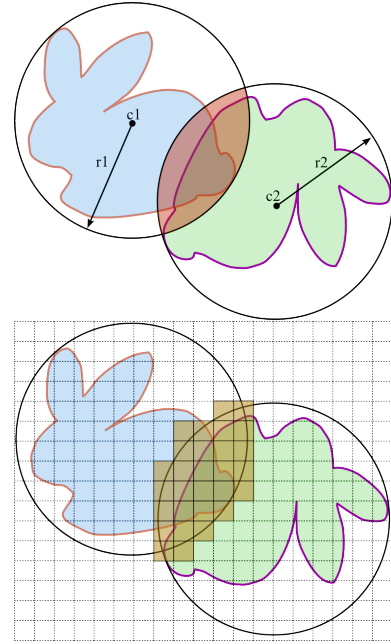


Figure 1: Interference check during broad phase collects potentially colliding vertices inside intersecting bounding spheres (top); narrow phase will be restricted to grid cells intersecting a region of potential collision (bottom).

4.2 Narrow phase

Once potential collision regions are found, participating primitives are submitted to an exact collision test. The test consists in using the hash function to visit the cell C associated with each potentially colliding vertex v . An exact intersection test is then conducted between v and each tetrahedron t_C referenced in C , provided that v is not one of the vertices of t_C .

5 Collision response

In a nutshell, the collision response process starts by computing the penetration depth of all collided vertices. Next, deformation

regions for all objects are computed. Finally, the objects are separated by approximately half of the penetration depth distances of all colliding vertices using binary search.

5.1 Penetration depth

The penetration depth between intersecting objects is the minimum translation necessary for separating them. Generally speaking, collision response for a pair of colliding rigid bodies can be computed based solely on this information. For deformable objects, however, the penetration depth of all colliding vertices must be computed. The use of tetrahedral meshes is very helpful in this task, since its incidence information can be used to track the deformation region and to compute the penalty forces, thus leading to a realistic collision response [Heidelberger et al. 2004].

The idea is to process the colliding vertices ranked by their penetration depth. Firstly, colliding vertices closest to the surface are evaluated. Later, this information is propagated to deeper colliding vertices using the tetrahedral meshes' incidence relations. As a result, penetration depth d and direction $\vec{r}$ are estimated for all relevant vertices.

Additionally, the process needs to find a contact triangle for each *border vertex* (described below) v . This triangle is a face on the penetrated object surface that intersects a ray with origin in v and direction $\vec{r}_v$.

Classification of colliding vertices: if a colliding vertex has one or more non-colliding incident vertices, it is called a *border vertex*, otherwise, it is called an *internal vertex* (see Figure 2).

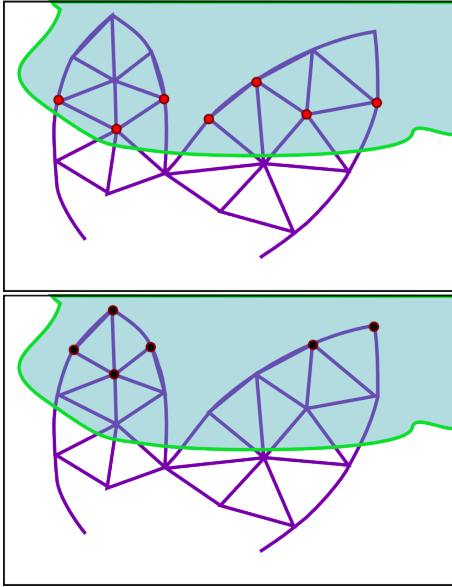


Figure 2: border (top) and internal (bottom) vertices.

Penetration depth of the border vertices: the penetration depth of border vertices depends on the computation of *contact points* and their normals. The underlying idea is to identify edges that contain one non-colliding point and one colliding point. These edges are called *intersecting edges*. Moreover, the exact intersection point of each edge with the surface of the penetrated object must be computed. This can be done efficiently by visiting all grid cells which intersect each edge and then testing all surface triangles in them against the edge.

The triangle-edge intersection is done in two passes: first, the triangle plane and the edge are tested for intersection. If successful, the barycentric coordinates (t_1, t_2, t_3) of the intersection point p is computed. Then, p is inside the triangle if $t_i > 0$ for $i = 1, 2, 3$. Additionally, the surface normal $\vec{n}$ is linearly interpolated from the three triangle vertex normals $(\vec{n}_1, \vec{n}_2, \vec{n}_3)$, i.e., $\vec{n} = \sum t_i \vec{n}_i$.

If more than one intersection point for a given edge is found, the point closest to the colliding vertex is chosen. The chosen points are called *contact points*. Finally, this data is used to compute the border vertices' penetration depth (see the Figure 3).

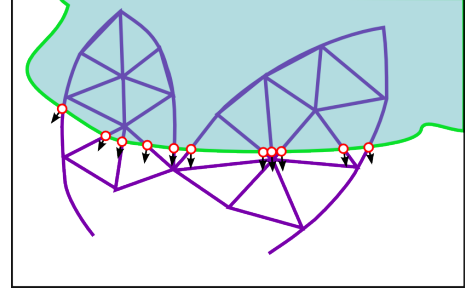


Figure 3: contact points with their normals on the intersected faces.

For each border vertex v , its penetration depth $d(v)$ and its penetrating direction vector $\vec{r}(v)$ are computed as follows: for each intersecting edge e_i incident in v , a weight w_i between the contact point p_i and v is given by:

$$w(p_i, v) = \frac{1}{\|p_i - v\|^2}. \quad (3)$$

Next, $d(v)$ and $\vec{r}(v)$ are computed using the following equations:

$$d(v) = \frac{\sum_{i=1}^k w(p_i, v)(p_i - v) \cdot \vec{n}_i}{\sum_{i=1}^k w(p_i, v)}, \quad (4)$$

$$\vec{r}(v) = \frac{\sum_{i=1}^k w(p_i, v) \vec{n}_i}{\sum_{i=1}^k w(p_i, v)}, \quad (5)$$

where k is the number of intersecting edges incident in v and p_i and $\vec{n}_i$ represents the i -th contact point and normal to be evaluated, respectively.

Penetration depth for internal vertices: After all border vertices are processed, their penetration information is employed for computing the penetration depth of associated internal vertices u . This is done by propagation, i.e., once the first layer of internal vertices are processed, edge-neighbor internal vertices are processed in turn. Thus, the penetration depth computation is done by levels, until the penetration depth information for all colliding vertices is determined.

Given an internal vertex u , its penetration depth $d(u)$ and penetration direction $\vec{r}(u)$ are computed using weights. First, for a non-processed internal vertex u incident in an already processed vertex v_j , a weight is computed as

$$\mu(u, v_j) = \frac{1}{\|v_j - u\|^2}.$$

Next, $d(u)$ and $\vec{r}(u)$ are computed by a weighted average of the contributions of all incident processed vertices using the following equations:

$$d(u) = \frac{\sum_{j=1}^k \mu(u, v_j)((v_j - u) \cdot \vec{r}(v_j) + d(v_j))}{\sum_{j=1}^k \mu(u, v_j)}, \quad (6)$$

$$\vec{r}(u) = \frac{\sum_{j=1}^k \mu(u, v_j) \vec{r}(v_j)}{\sum_{j=1}^k \mu(u, v_j)}, \quad (7)$$

where k is the number of processed vertices incident in u . Figure 4 illustrates the penetration directions of colliding vertices.

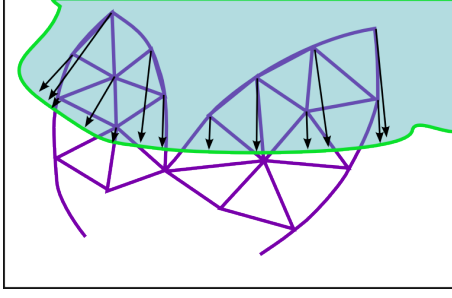


Figure 4: penetration directions for colliding vertices.

5.2 Deformation region

After the computation of the penetration depth for all colliding vertices, a separation process must be executed in order to reach a non-overlapping state. To this end, a region of deformation defined by colliding vertices is obtained. In addition to these vertices, the deformation region also includes vertices on colliding faces (contact faces), which not necessarily are colliding vertices (x_i , x_j and x_k in Figure 5). Collisions with this configuration are called *asymmetric collisions*.

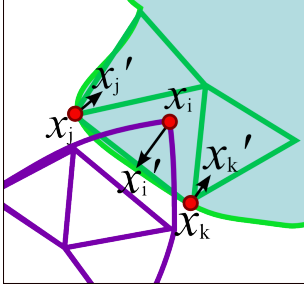


Figure 5: treating asymmetric collisions.

The handling of asymmetric collisions in our approach deviates from the technique proposed in [Spillmann and Teschner 2005]. Rather than using a scheme based on barycentric weights, we employ the simpler projection technique described in [Jakobsen 2001]. The idea consists in projecting the colliding vertices out of the obstacle, moving them until they are intersection-free. For example, in Figure 5, the displacement vector of x_i is its penetration depth, and the displacement vectors for the non-colliding vertices are

$$\vec{s}_j = \frac{\alpha_1}{\alpha_1^2 + \alpha_2^2} (x'_i - x_i),$$

$$\vec{s}_k = \frac{\alpha_2}{\alpha_1^2 + \alpha_2^2} (x'_i - x_i),$$

where x_i is the colliding vertex, x'_i is its projection on contact edge x_j-x_k , $\vec{s}_j$ and $\vec{s}_k$ are the displacement vectors for x_j and x_k respectively, and α_1 and α_2 represent the proportional displacement factors for the vertices on the contact edge, subject to $\alpha_1 + \alpha_2 = 1$. Notice that a contact edge in 2D corresponds to a contact triangle in 3D, and constants α_i are proportional areas. Namely, if the contact triangle has vertices x_k , x_j and x_l and x'_i is the projection of the colliding point onto that triangle, then, $A = \text{Area}(x_j, x_k, x_l)$, $\alpha_1 = \text{Area}(x_k, x_l, x'_i)/A$, $\alpha_2 = \text{Area}(x_j, x_l, x'_i)/A$, and $\alpha_3 = \text{Area}(x_j, x_k, x'_i)/A$.

Once all displacement vectors for colliding vertices have been computed, a contact surface is defined. Conceptually, the contact surface should be equidistant from the colliding vertices, and thus an initial estimate of its position is obtained by translating collided vertices by half of their associated displacement vectors. This estimate is then refined by means of a binary search scheme [Spillmann and Teschner 2005]: if x^1 is the initial estimate for a surface vertex, and s is its associated displacement vector, then the i^{th} estimate for that vertex is given by

$$x^i \leftarrow x^{i-1} \pm \frac{1}{2^{(i+1)}} s, \quad (8)$$

where the direction of the movement is determined by the sign of the pressure difference. In practice, three or four iterations are sufficient to produce a contact surface with adequate accuracy.

6 Dynamics based on shape matching

In order to estimate the dynamic behavior of collided objects, a shape matching technique [Müller et al. 2005] is used. A large part of its appeal is that, being a meshless method, only a set of particles (the object vertices) are considered. The idea is to match vertices in their initial states $\mathbf{x}_i^0$ with their positions in the current state $\mathbf{x}_i$, i.e., the positions of the vertices as a result of deformation. The matching consists of determining an affine¹ transformation matrix using a least-squares approach. This transformation is a model for a plausible deformation and provides a simple way for the object to recover its original shape (see the Figure 6). Once the transformation is computed, goal positions $\mathbf{g}_i$ are estimated for each vertex. These are then used to estimate the new position and velocity of each particle for the next time step.

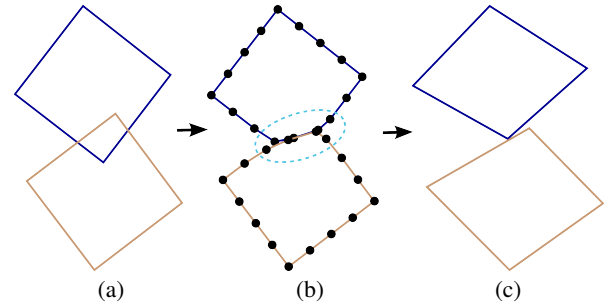


Figure 6: (a) two colliding objects, (b) after a contact surface is determined, and (c) deformed with shape matching.

The algorithm has two main components: (1) find the optimal rigid transformation that approximates a new position and orientation for the object (matching problem) and (2) move the vertices for the goal positions applying a linear deformation model.

Considering the particles weights, the matching problem consists of finding translation vectors $\mathbf{t}$ and $\mathbf{t}_0$ and rotation matrix $\mathbf{R}$ which minimize:

$$\sum_i w_i (\mathbf{R}(\mathbf{x}_i^0 - \mathbf{t}_0) + \mathbf{t} - \mathbf{x}_i)^2.$$

For simplicity, $w_i = m_i$ is used, where m_i is the mass of particle i . The optimal translation vectors between the *centers of mass* of the initial position and the current position are given by

$$\mathbf{t}_0 = \mathbf{x}_{cm}^0 = \frac{\sum_i m_i \mathbf{x}_i^0}{\sum_i m_i}, \quad \text{e} \quad \mathbf{t} = \mathbf{x}_{cm} = \frac{\sum_i m_i \mathbf{x}_i}{\sum_i m_i}.$$

The optimal rotation $\mathbf{R}$ may be found by first determining a general linear transformation matrix $\mathbf{A}$. Notice that $\mathbf{A}$ not necessarily is orthonormal. Considering the relative particle positions defined by

$$\mathbf{q}_i = \mathbf{x}_i^0 - \mathbf{x}_{cm}^0 \quad \text{and} \quad \mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{cm},$$

the correspondence problem may be solved in the least-square sense. Thus, the optimal linear transformation matrix $\mathbf{A}$ is found minimizing the term:

$$\sum_i m_i (\mathbf{A} \mathbf{q}_i - \mathbf{p}_i)^2.$$

By setting to zero all the first derivatives of this expression with respect to the coefficients of $\mathbf{A}$, we obtain

$$\mathbf{A} = \left(\sum m_i \mathbf{p}_i \mathbf{q}_i^T \right) \left(\sum m_i \mathbf{q}_i \mathbf{q}_i^T \right)^{-1} = \mathbf{A}_{pq} \mathbf{A}_{qq}, \quad (9)$$

¹[Müller et al. 2005] also considers quadratic transformations.

where $\mathbf{A}_{pq}$ is a correlation matrix and $\mathbf{A}_{qq}$ is a symmetric matrix that can contain only scale information but not orientation. Therefore, for finding $\mathbf{R}$, it is necessary to find the rotation component of $\mathbf{A}_{pq}$. Following [Müller et al. 2005], we use a polar decomposition method [Shoemake and Duff 1992] to obtain $\mathbf{R} = \mathbf{A}_{pq}\mathbf{S}^{-1}$, where symmetric matrix $\mathbf{S} = \sqrt{\mathbf{A}_{pq}^T\mathbf{A}_{pq}}$. In order to obtain $\mathbf{S}^{-1}$, matrix $\mathbf{A}_{pq}^T\mathbf{A}_{pq}$ is diagonalized using from 5 to 10 Jacobi rotations [Sumali 1992].

Finally, all particles are moved to their goal positions computed as

$$\mathbf{g}_i = \mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm},$$

and the process is repeated in every time step.

7 Integration

As in [Müller et al. 2005], for the integration we use a modified Euler's method, which includes an explicit part (velocity updating) and an implicit part (position updating). The integration scheme is given by

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \alpha \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{\Delta t} + \frac{\Delta t}{m_i} \mathbf{f}_{ext}(t), \quad (10)$$

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t), \quad (11)$$

where $\mathbf{v}_i(t + \Delta t)$ is the velocity after the time step, $\mathbf{v}_i(t)$ is the current velocity, $\alpha \in [0, 1]$ is a parameter which controls the rigidity of the object, $\mathbf{g}_i(t)$ is the goal position for vertex $\mathbf{x}_i(t)$, Δt is the time step, m_i is the particle mass and $\mathbf{f}_{ext}(t)$ are the external forces (gravity, wind, among others).

8 Results

A prototype implementation of the proposed system was developed using the C++ language and the OpenGL and Glut libraries. For performance reasons, *Vertex Buffer Objects* (VBOs) were used to store object geometry.

Several experiments were conducted with this prototype in order to evaluate the correctness and speed of the technique. These involved simulations of collections of objects with different shapes. All experiments were conducted in a PC running Linux (Fedora 8) operational system and equipped with a Intel Core 2 Duo processor running at 2.4Ghz, 1GB memory and an NVidia GeForce 8600 GS graphics board.

The experiments aimed to gauge the system performance as a function of both the number and geometric complexity of the objects involved in the simulation. Table 1 shows the object resolutions (number of vertices, number of faces and number of tetrahedra). In

Table 1: objects with different resolutions.

Object	surface vertices	total vertices	faces	tetrahedra
bunny	436	510	868	1750
duck	424	519	846	1819
sphere	386	729	768	2560

all experiments, the rigidity coefficient α was set to 0.8, i.e., quasi-rigid objects. Additionally, it was collected information about: number of colliding primitives (vertices in potential collision, faces, tetrahedra and vertices in real collision) processed at each time step, and the percentage spent in each sub-process (broad phase collision detection, narrow phase collision detection, penetration depth computation and shape matching) in milliseconds.

A first experiment consists of the simulation of a scene with different kinds of objects (see Figure 7). The scene contains 3 ducks, 2 bunnies and 3 spheres (see Table 1). This experiment demonstrates that the prototype is able to handle objects with arbitrary geometry, provided that a 3D triangulation for them is known. From the experiment, some information was extracted regarding the various

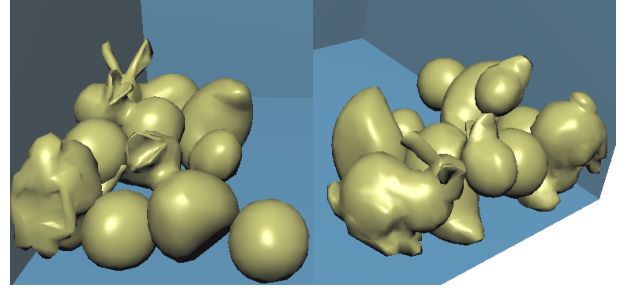


Figure 7: eight objects in contact: 3 ducks, 2 bunnies e 3 spheres. The scene contains 2952 vertices and 9917 tetrahedra that are animated at approx. 32 fps.

stages of our algorithm. Figure 8 shows as a function of simulation time the time in milliseconds spent in the four main sub-processes: broad phase collision detection, narrow phase collision detection, penetration depth computation and shape matching. Notice that the collision detection, in both stages, spends much more time than the other sub-processes. The penetration depth computation is almost constant and the time spent on shape matching is negligible.

A better notion about the behavior of the collision detection algorithms may be apprehended from Figure 9, which shows, for the same experiment, the quantity of primitives in collision (vertices in potential collision, faces, tetrahedra and vertices in real collision) for each time-step. Notice that the broad phase filters out a significant number of non colliding primitives and that, on average, only about 10% of the potential colliding vertices are effectively colliding vertices. Additionally, other three experiments were conducted, involving a varying number of spheres. Figure 10, shows simulations of scenes with 8, 18 and 27 spheres. The number of primitives contained in each scene as well as the simulation frame rates are tabulated in Table 2. A chart with the frame rate for each time-step is shown in Figure 11.

Table 2: primitives for experiments with spheres.

Number of spheres	vertices	tetrahedra	average fps
8	5832	20480	62
18	19683	46080	41
27	19683	69120	22

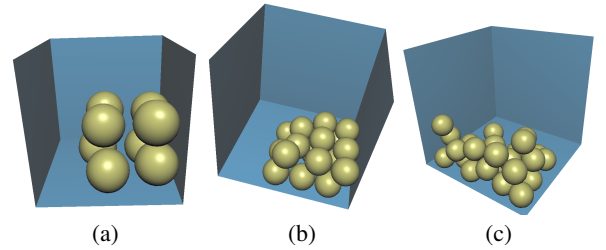


Figure 10: experiments with 8 (a), 18 (b) e 27 (c) spheres.

9 Conclusions and future work

We have described a method for physically based animation of deformable objects which combines several techniques such as collision detection using spatial hashing [Teschner et al. 2003] and bounding spheres, collision response through the surface contact computation that utilizes penetration depth computation by propagation [Heidelberger et al. 2004], the computation of displacement vectors [Heidelberger et al. 2004] and handling of asymmetric collisions [Jakobsen 2001], binary search in order to separate the objects [Spillmann and Teschner 2005], and a deformation and dynamics model based on shape matching and an explicit-implicit Euler integrator [Müller et al. 2005].

The original structure of the collision detection was extended in order to use a broad phase mechanism, which detects regions on potential collision avoiding to update the hash table unnecessarily. The narrow phase is restricted to cell grids where potential collision was detected. In addition, asymmetric collisions are handled by projection [Jakobsen 2001].

The prototype system enables the simulation of scenes with plausible physics behavior and containing objects with complex geometry. Collision response is implemented in a robust manner and handles stacked objects.

As future work, we plan to extend this prototype with quadratic and plastic deformations. Both the broad and narrow phase collision detection can be improved using more efficient techniques. For instance, the bounding sphere scheme and vertex/tetrahedron tests can easily be implemented in GPU. Other aspects of the system may be harder to port to GPU, but we intend to investigate recent advances in this area such as the CUDA [NVIDIA™2007] technology.

Acknowledgements

We would like to acknowledge the support of Brazilian agencies CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) and CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

References

- BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 43–54.
- BRADSHAW, G., AND O'SULLIVAN, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Graph.* 23, 1, 1–26.
- COHEN, J. D., LIN, M. C., MANOCHA, D., AND PONAMGI, M. 1995. I-collide: an interactive and exact collision detection system for large-scale environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 189–ff.
- GANOVELLI, F., DINGLIANA, J., AND O'SULLIVAN, C. 2000. Buckettree: Improving collision detection between deformable objects. In *Spring Conference in Computer Graphics (SCCG)*.
- GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. Obb-tree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 171–180.
- HEIDELBERGER, B., TESCHNER, M., KEISER, R., AND MÜLLER, M. 2004. Consistent penetration depth estimation for deformable collision response. In *Proceedings of Vision, Modeling, Visualization VMV'04*, 157–164.
- JAKOBSEN, T. 2001. Advanced character physics. In *Proceedings, Game Developer's Conference 2001*, GDC Press, SJ, USA.
- JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)* 23, 3 (Aug.).
- KNOTT, D., AND PAI, D., 2003. Cinder: Collision and interference detection in real-time using graphics hardware.
- LUQUE, R. G., JO A. L. D. C., AND FREITAS, C. M. D. S. 2005. Broad-phase collision detection using semi-adjusting bsp-trees. In *ISD '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 179–186.
- MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, 49–54.
- MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 141–151.
- MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. In *Proceedings of SIGGRAPH'05*, 471–478.
- NVIDIA™, 2007. CUDA Environment – Compute Unified Device Architecture. http://www.nvidia.com/object/cuda_home.html.
- PAULY, M., PAI, D., AND GUIBAS, L. 2004. Quasi-rigid objects in contact. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 109–119.
- SAMET, H. 2006. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- SHEN, C., HAUSER, K. K., GATCHALIAN, C. M., AND O'BRIEN, J. F. 2002. Modal analysis for real-time viscoelastic deformation. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*, ACM, New York, NY, USA, 217–217.
- SHINYA, M., AND FORGUE, M.-C. 1991. Interference detection through rasterization. In *The Journal of Visualization and Computer Animation*, vol. 2, 132–134.
- SHOEMAKE, K., AND DUFF, T. 1992. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface '92*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 258–264.
- SPILLMANN, J., AND TESCHNER, M. 2005. Contact surface computation for coarsely sampled deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'05*, 16–18.
- SUMALI, H. 1992. *A New Adaptive Array of Vibration Sensors*. PhD thesis, Mechanical Engineering Virginia Polytechnic Institute and State University, Virginia, USA.
- TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. 1987. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 205–214.
- TESCHNER, M., HEIDELBERGER, B., MÜLLER, M., AND POMERANETS, D. 2003. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03* *Proceedings of SPM 2005*, 47–54.
- VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools* 2, 4, 1–13.
- VAN DEN BERGEN, G. 1999. A fast and robust GJK implementation for collision detection of convex objects. *Journal Graph. Tools* 4, 2, 7–25.

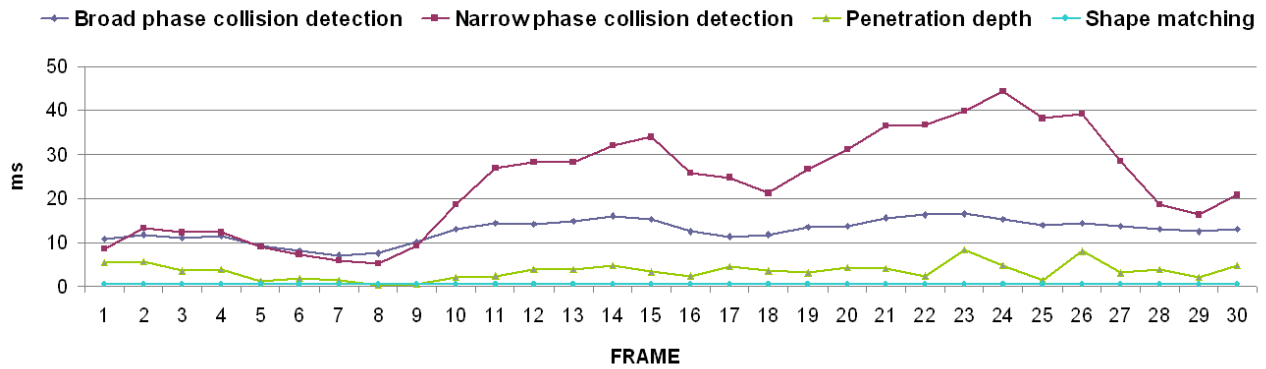


Figure 8: spent time in milliseconds for each sub-process at each time-step.

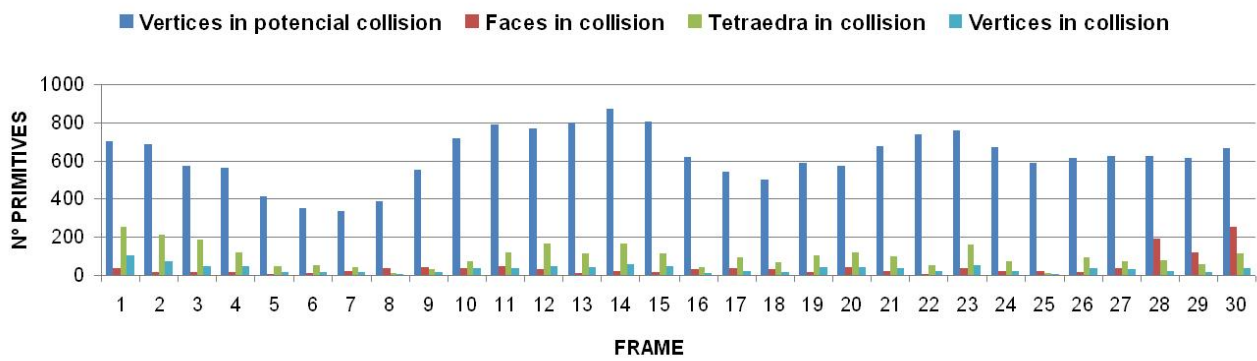


Figure 9: quantity of primitives in collision for each time-step.

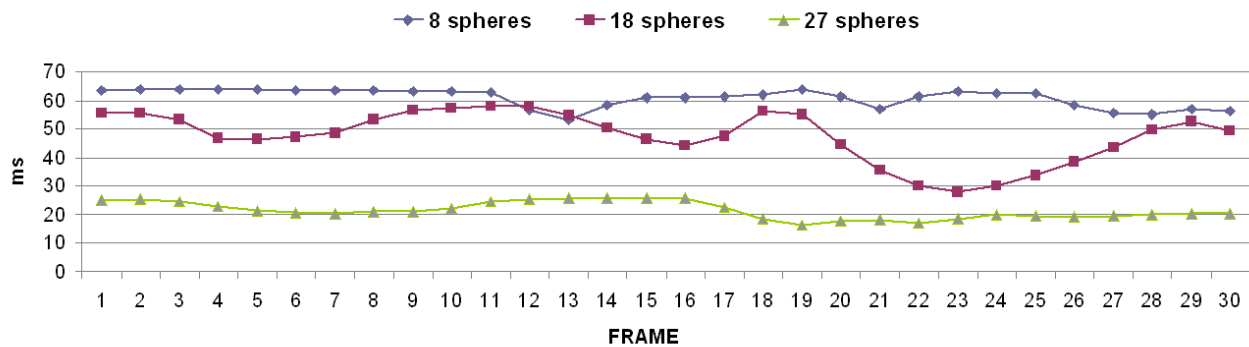


Figure 11: frame rate for each time-step for scenes with 8, 18 and 27 spheres.