A Collision Detection and Response Scheme for Simplified Physically Based Animation

Yalmar Ponce Atencio[1], Claudio Esperança[1], Paulo Roma Cavalcanti[1], Antonio Oliveira[1]
[1]PESC - Programa de Engenharia de Sistemas e Computação
COPPE / Universidade Federal do Rio de Janeiro
{yalmar, esperanc}@lcg.ufrj.br

Abstract

In this paper we describe a system for physical animation of rigid and deformable objects. These are represented as groups of particles linked by linear constraints, while a Verlet integrator is used for motion computation. Unlike traditional approaches, we accomplish physical simulation without explicitly computing orientation matrices, torques or inertia tensors. The main contribution of our work is related to the way collisions are handled by the system, which employs different approaches for deformable and rigid bodies. In particular, we show how collision detection using the GJK algorithm [9] and bounding sphere hierarchies can be combined with the projection based collision response technique described by Jakobsen [14].

1 Introduction

Physically based simulations usually require a great deal of computational resources in order to perform geometrically accurate collision detection. Physical attributes such as velocity, acceleration, forces, torques and momenta must be computed at each frame of the animation. Moreover, the response to the collision usually requires the computation of several expensive mathematical parameters, notably Jacobian matrices.

In 1999, Thomas Jakobsen [14] proposed a simplified scheme that addresses many of these difficulties. In a nutshell, his approach combines the following techniques: (1) physical objects (rigid bodies, ropes, cloth, etc.) are represented by means of particle systems with linear constraints; (2) particles dynamics are simulated through Verlet integration; (3) approximate rather than exact square roots are used in distance computations; (4) constraints are resolved using relaxation; (5) collision response is achieved by projecting penetrating vertices onto penetrated surfaces.

This approach is recommended for games, since it dismisses some expensive features which are necessary for a correct physical simulation. The resulting animation is very convincing, although not physically correct in a strict sense. Several important details, however, were left out of Jakobsen's work. In particular, the connection between the collision detection engine output and the collision response algorithms is only hinted at. More importantly, no strategy is suggested for dealing with even a modest increase in the number of linear constraints, which may compromise the overall system performance.

In this paper we try to address these issues by describing a prototype system in which several tentative solutions are introduced. More specifically, we describe (1) a scheme for separating colliding objects by moving vertices towards what we call a *projection point*, which is computed as a by-product of the collision detection engine, and (2) a linear constraint ordering scheme which enables the relaxation loop to converge more rapidly.

2 Previous work

Physically based modeling and animation have been extensively researched in the last years. Different approaches have been proposed (e.g. [3, 20, 12]) and great effort has been put in the construction of accurate and reliable algorithms.

Regardless of which physical formulation is employed, a key component of any simulation is the collision detection engine. This component is responsible for quickly answering geometric queries involving all objects in the simulation. These include not only *whether* two given objects intersect, but also *when* and *where* the collision occurred. The interested reader will find a comprehensive coverage of the subject in two recent surveys: [18, 16].

A key concept used in many implementations of collision detection engines is that of a bounding volume hierarchy. Axis-aligned bounding boxes (AABBs) are quite popular [6, 22], as are oriented bounding boxes (OBBs) [10].

Other polyhedra have also been proposed as bounding primitives – discrete orientation polytopes (DOPs) [17], for instance. Sphere hierarchies are also adopted by many researchers [23, 15] since spheres enjoy the nice property of being rotation invariant.

Once a pair of bounding volumes for potentially intersecting objects are found, one must perform an exact collision test between the objects themselves. If the objects are convex, one may resort to the the notion of *separating planes*. Given a pair of convex polytopes A and B, an intersection test will basically consist of finding a plane for which A and B lie in opposite sides. If none can be found, then A and B intersect. If the objects being tested have a large number of features, this search may be accelerated by employing an incremental algorithm, i.e., an algorithm that exploits frame coherence such as the Gilbert-Johnson-Keerthi (GJK) [9] method. Our implementation is based on a variant of the GJK algorithm, as described by Van den Bergen [25]

Once the collision detector reports a collision, the bodies involved must have their positions, orientations and other parameters altered in order to re-establish a non-colliding state. This procedure, known as *collision response*, is usually based on physical laws suitably adapted for a discrete computational environment. Hecker [12], Baraff [3, 4] and others have authored a wide range of contributions in this matter. Moore and Wilhelms [21] presented one of the earliest treatments of problems in dynamics simulation. Hahn [11] also pioneered dynamics simulation, modeling sliding and rolling contacts using impact equations. Baraff [3, 4] studied multiple rigid bodies in contact, and showed that computing contact forces in the presence of friction is NP-hard.

Jakobsen [14] presented a simplified approach to physically based animation by combining several techniques. As mentioned earlier, the two most important aspects of this approach are the use of a Verlet [26] integrator for the particle dynamics, and a projection-based scheme for collision response resolution. Unfortunately, the achieved rigid body simulation is limited, as it does not include rotational information and depends only on the linear constraints that compose the objects. It should also be mentioned that with this approach, a large number of constraints is necessary to simulate a simple rigid body.

3 Background

3.1 Verlet integrator

Particle dynamics is typically simulated using the so-called Euler integration scheme. Each particle is represented by three attributes: position, velocity, and acceleration/force. For each time step Δt, movement equations $x' = x + v \cdot \Delta t$ and $v' = v + a \cdot \Delta t$ are used to compute the particle's new position and velocity, whereas the acceleration a is computed using Newton's second law.

Jakobsen proposes the use of a different integration scheme created by Verlet [26] for molecular dynamics. This method is based on Taylor's theorem which is applied on the movement equations, yielding two expansions – forward and backward in time. By adding them up we obtain

$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + x''(t)\Delta t^2 + O(\Delta t^4), \quad (1)$$

where $O(\Delta t^4)$ represents truncated higher order terms. This is called the Verlet Integrator. In this scheme three attributes per particle are stored: current position, previous position and the force/acceleration. In particular, the particle's velocity needs not be represented. Another important feature of Verlet's approach is the reversibility in time which permits to approximately compute velocity when necessary.

3.2 Linear constraints

Typically, the simulation of soft objects such as cloth employs a model consisting of several interconnected springs between particles. This yields a system of differential equations which is solved by numerical methods. This method is generally unsuitable for real-time simulations. Jakobsen proposes a nice alternative by employing springs with infinite stiffness and ideal damping, that is, springs that instantly attain their rest length. The system then becomes solvable in a stable way and can be solved much faster.

The same idea can be employed in the simulation of rigid bodies. These are also modeled by particles linked by stiff springs which must exist in sufficient number so as to guarantee the body's rigidity.

An ideally stiff spring is computationally modeled by means of a linear constraint. In other words, if a linear constraint exists between two particles, their positions p_1 and p_2 are required to be maintained d units apart. Expressed mathematically, the constraint is given by $|p_1 - p_2| = d$.

4 Object representation

The system handles rigid bodies, some deformable objects as pieces of cloth and ropes. We distinguish two representations for each object: (1) a representation of its geometry and (2) a representation of its physical properties. The first is used for displaying the object and performing collision detection against other objects, whereas the second is responsible for estimating how the object should react to physical stimuli such as collision or the attraction of gravity.

All objects other than ropes are geometrically represented by means of triangulated surfaces. Thus, for instance, a cube is represented by 8 vertices, 12 triangles and

18 edges, ropes are represented by a polygonal line and pieces of cloth are represented by regularly triangulated surfaces.

As for the representations of their physical properties, all objects are modeled as particle systems. In particular, all particles are coincident with the vertices of their geometric representations and have identical mass. Additionally, each object type is modeled with a set of linear and/or angular constraints. For instance, a cube contains one linear constraint coinciding with each edge plus four linear constraints corresponding to the cube's main diagonals. A cloth object contains only linear constraints coinciding with its edges. It should be noted that, in general, for a body represented as a polygonal model, its rigidity may be enforced by creating a restricted 3D triangulation and taking all the edges of the resulting tetrahedra as linear constraints.

In our implementation, the geometry of each object is represented in two separate data structures. The first is a Half-Edge data structure [19] which supports operations such as visiting the object's faces, edges and vertices or determining incidence relationships between these elements. The second is a Sphere Tree [1] which is used for collision detection.

The physical properties of the object are mostly represented by constraint lists, which are described in Section 6.2. For convenience, however, the mass of each particle is stored as an attribute of the corresponding vertex inside the half-edge data structure rather than in a separate data structure.

5 Collision Detection

In a nutshell, the problem of collision detection consists of establishing for all objects in a scene all pairs of objects of the form (A, B), $A \neq B$ such that $A \cap B \neq \emptyset$. In our system this is accomplished by enumerating all pairs of objects and testing them for intersection. Strictly speaking, this is not very efficient since it entails testing $O(n^2)$ pairs, where n is the number of objects in the scene. Nevertheless, we are mostly interested in simulating the physical interaction among a small number of relatively complex objects. Thus, the performance penalty is not too high in practice, provided that the intersection test between two objects can be performed efficiently.

Our system employs two schemes for detecting the collision between two given objects. The first is an adaptation of the GJK algorithm [9, 25] and is employed when solving the collision between two rigid – i.e., non-deformable – objects. The second scheme is an adapted algorithm based on Sphere Trees [18, 15, 1, 13] and is used for solving collisions whenever at least one of the objects is deformable. It should also be mentioned that some deformable objects may collide with themselves and such collisions are also handled using the Sphere Tree approach.

5.1 The GJK algorithm

The Gilbert-Johnson-Keerthi algorithm [9] provides a very efficient method for detecting collision between convex objects. It relies on a few key concepts which are briefly outlined below:

Minkowski addition: Given two sets A and B, their Minkowski sum is defined as

$$A + B = \{x + y : x \in A, y \in B\}. \tag{2}$$

This definition does not seem correct since addition of points is meaningless. In this loose notation x and y should rather be understood as the vectors $\vec{x} = \mathbf{x} - \mathbf{0}$, where $\mathbf{0}$ is the origin of the world coordinate system.

Configuration Space Obstacle (CSO): For a pair (A, B) of convex objects, their CSO is given by $A - B$, i.e., the Minkowski sum of A and $-B$. This set is specially useful in collision detection because it can be proved that A and B intersect if and only if their CSO contains the origin:

$$A \cap B \neq \emptyset \equiv 0 \in A - B. \tag{3}$$

Moreover, their distance is given by

$$d(A, B) = \min\{\|x\|\colon x \in A - B\}. \tag{4}$$

Similarly, the penetration depth of pairs objects can be expressed in terms of their CSO as

$$p(A, B) = \inf\{\|x\|\colon x \in A - B\}. \tag{5}$$

For a pair of intersection objects, the penetration depth is realized by a point on the boundary of $A - B$ that is closest to the origin.

Support Mapping: The support mapping $S_A(v)$ is a function that, given a vector v and a convex set A, returns the most "extreme" point of A in the direction of v. Formally speaking,

$$S_A(v) \in A \mid v \cdot S_A(v) = \max\{v \cdot x : x \in A\}. \tag{6}$$

Separating Plane / Axis: Given two objects A and B, a plane $H(v, \delta)$ separates A and B if for every point $a \in A$, $v \cdot a + \delta \geq 0$ and for every point $b \in B$, $v \cdot b + \delta \leq 0$. Vector v is known as a *weakly separating axis* of A and B since there is at least one separating plane which is normal to it or, equivalently,

$$v \cdot S_A(-v) \geq v \cdot S_B(v). \tag{7}$$

The general idea of the GJK algorithm is to examine the CSO of two given objects A and B looking for a simplex which contains the origin. If this search ends with a negative

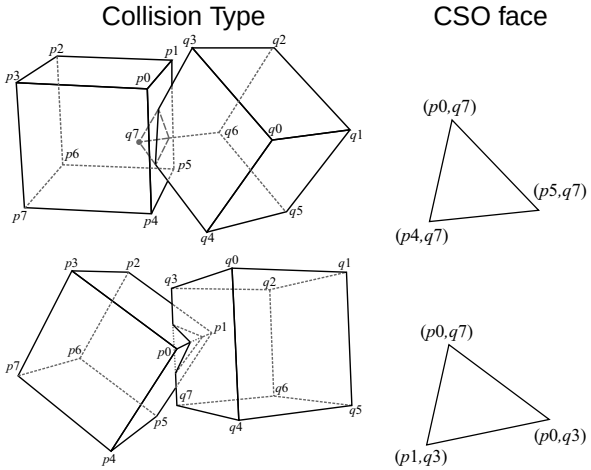


Figure 1. The two types of collision and corresponding CSO face: face-vertex (top) and edge-edge (bottom).

answer, i.e., the origin lies outside the CSO, then the objects do not intersect. In this case, the point of the CSO which is closest to the origin represents a separating axis of A and B, and this in turn can be used as a starting point for collision testing in subsequent frames.

On the other hand, if the search is successful, then the objects do intersect and in order to perform collision response several other details about the collision must be computed. For instance, typical schemes try to determine the *penetration depth* which, in turn, requires finding the point on the boundary of the CSO which is closest to the origin. Bergen [24] suggests an expanding polytope algorithm for this purpose. Our system, however, computes a related information: the *hit face*, i.e., the face on the CSO hull which is closest to the origin. By analyzing the vertices of this face, it is possible to determine which object features took part in the collision. Here, we distinguish two main cases: edge-edge and vertex-face collisions[1]. In order to understand how the features are identified, notice that each vertex of the CSO corresponds to a pair of vertices (a_i, b_j), $a_i \in A$, $b_j \in B$. For instance, a vertex of A colliding with a face of B would be characterized by having all three vertices of the hit face corresponding to the same vertex of A but to three different vertices of B (see Figure 1).

5.2 Sphere Trees

Collision detection of deformable bodies is usually supported by some hierarchical bounding volume scheme. The literature describes two types of bounding sphere hierarchies [1]: *wrapped hierarchies* wherein spheres enclose the geometry tightly, and *layered hierarchies* in which the sphere of a parent node merely encloses the spheres of its children (see Figure 2).

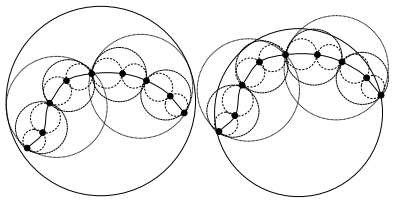


Figure 2. A layered sphere tree (left) and a wrapped sphere tree (right).

In general, a wrapped hierarchy is preferable to a layered hierarchy since the former is tighter than the latter. Another important aspect concerns the update strategies that must be employed after the object moves or deforms. In particular, if the bounded object is non-deformable, both hierarchies support an update procedure which can performed top-down in a "lazy" manner, i.e., the children of a node need only be updated if the node has been found to intersect another object or node. In the worst case, the update requires $O(n)$ time if all leaves take part in the collision, but more frequently, when only one or a small number of leaves is involved, the update will take $O(\log n)$ time. Wrapped hierarchies, on the other hand, may be updated in $O(n)$ expected time for some object classes bodies such as necklaces [2] [1], but in order to support generally deformable objects it is necessary to spend $O(n \log n)$ time in the worst case. For these objects, a layered sphere-tree which can be updated in $O(n)$ time is frequently more indicated.

Our system maintains sphere trees attached to all objects: wrapped hierarchies are used for rigid bodies and layered hierarchies for deformable bodies. We employ the traditional collision detection algorithm [1] which is based on a recursive descent on both hierarchies. At each step, an intersection test is performed on a pair of spheres (s_a, s_b), where s_a (or s_b) is a bounding sphere of the tree corresponding to object A (respectively B). If a leaf node is visited during the process, the enclosed object is then subject to an exact intersection test.

The trees are built in a manner similar to that reported by Quinlan [23]. Sphere trees which enclose deformable bodies are updated bottom-up in straightforward way, i.e., the leaf-nodes are updated by constructing new bounding spheres for the moved particles, whereas internal nodes are built as minimum bounding spheres for the children nodes' spheres.

[1] Although other configurations are possible – edge-face or vertex-edge, for instance – these can be mapped to the previous cases

[2] James and Pai [15] reported a sub-linear update algorithm which is restricted to objects subject to reduced deformations (as opposed to general deformations).

The top-down procedure for updating rigid body sphere trees takes advantage of the fact that any bounding sphere for a set of points may be constructed as the circumsphere of a subset of at most four points [27, 8], which are termed *support points*. During the construction of of the tree, links to the support points (i.e., particles) of each sphere are stored in the corresponding node. Thus, when the particles move, each bounding sphere may be re-computed in $O(1)$ time.

6 Collision response

The response to a collision can be divided in two distinct steps. The first step consists of separating the object features (vertex, edge or face) which were found to intersect. This is purely a geometric step since it entails only moving particles based on the collision geometry. The second step is an iterative relaxation process in which the remaining object features find their proper positions by enforcing the constraints.

6.1 Feature separation

If two non-deformable objects A and B collide, the feature separation requires three parameters which are computed by the collision detection engine. The first is p_A, the contact point on A, the second is p_B, the contact point on B and the third is what we call a *projection point*, i.e., a point q in space where the simulation assumes the two objects first touched. Jakobsen describes a fairly simple collision response approach [14] for a moving and a static object which is based on the concept of projection, although no details are given about the computation of the required parameters.

Let us analyze how the response is computed for A (B is handled in a similar way). Once p_A and q are known, p_A is moved so that it coincides with q. If the feature is a vertex, then this is trivial. If it is an edge or a face, then other vertices of the feature must be moved accordingly (see Figure 3 for an edge collision example). Since all particles

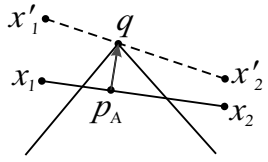


Figure 3. Collision response for an edge.

of an object have identical mass, the feature's new position depends only on the feature's geometry. For example, in Figure 3, if x_1 and x_2 are the endpoints of the edge and $p_A = (1-\alpha)x_1 + \alpha x_2$, then Jakobsen computes the edge's new position as

$$x_1' = x_1 + \frac{1-\alpha}{(1-\alpha)^2+\alpha^2}(q - p_A) \quad (8)$$

$$x_2' = x_2 + \frac{\alpha}{(1-\alpha)^2+\alpha^2}(q - p_A). \quad (9)$$

A face collision is handled in a similar way.

All it remains is to compute the parameters p_A, p_B and q. Two constructions are possible depending on the collision type – vertex-face or edge-edge (see Figure 4). If, say, a

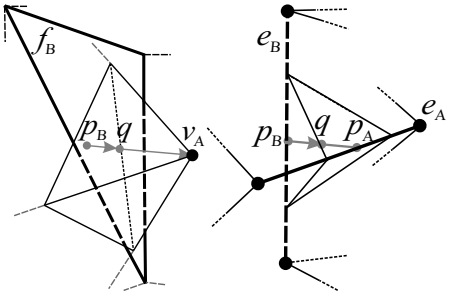


Figure 4. Vertex-face collision (left) and edge-edge collision (right).

collision occurred between vertex $v_A \in A$ and face $f_B \in B$, then $p_A = v_A$ and p_B is a point on f_B which is closest to p_A. Point q lies on the line segment $p_A p_B$ and its exact position depends on the body masses. If m_A and m_B are the masses of A and B then

$$q = \frac{m_B}{m_A + m_B} p_A + \frac{m_A}{m_A + m_B} p_B \quad (10)$$

For a collision occurring between edges e_A and e_B, the construction is similar (see Figure 4). p_A is the point of e_A which is closest to e_B and p_B is the closest point in e_B to e_A and q is computed by Eq. 10.

When a collision occurs between two deformable objects, then the feature separation is done in a somewhat cruder way. In particular, the sphere tree-based collision detection stops after detecting the intersection between two bounding spheres corresponding to leaf nodes in the tree, that is, it does not test the actual bounded features for intersection. This approach is justifiable for ropes and cloth pieces, provided that their geometries are finely sampled so that the leaf bounding spheres provide a good approximation for the actual object. Consider two features $f_A \in A$ and $f_B \in B$ whose bounding spheres s_A and s_B are found to intersect (see Figure 5). Let c_A (c_B) be the center and r_A (r_B) be the radius of s_A (s_B). Then all particles of f_A are displaced by $-\vec{d}/2$ and all particles of f_B are displaced by $\vec{d}/2$, where $\vec{d}$ is the vector given by

$$\vec{d} = \left(\frac{(r_A + r_B)}{||c_B - c_A||} - 1 \right) (c_B - c_A) \quad (11)$$

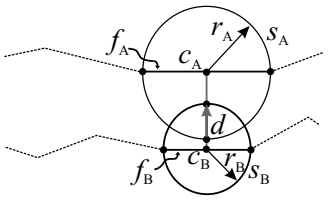


Figure 5. Separating two features enclosed in bounding spheres.

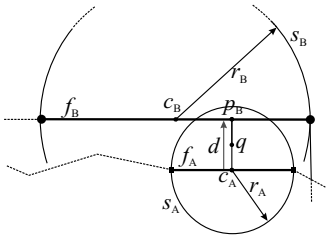


Figure 6. Separating features: non-deformable object vs. deformable object.

Finally, when the collision occurs between a deformable object A and a non-deformable object B, the feature separation procedure is a mix of the two cases described above. The system will consider the intersection between a bounding sphere s_A and an actual feature f_B a collision only if c_A, the center of s_A, is inside B (see Figure 6). Feature f_B is moved as in first case (non-deformable objects), whereas feature f_A is moved as in the second case (deformable objects). The projection point q is computed as the midpoint of the line segment $c_A p_B$, where the contact point p_B is the closest point of f_B to c_A. Vector $\vec{d}$ is given simply by $p_B - c_A$.

6.2 Iterative Relaxation

The relaxation process is used to reestablish the constraints of the system after some particles have moved either because forces were applied to them or as a result of the feature separation procedure. In order to enforce a linear constraint after the movement of one or both particles, one must push the particles away from each other or pull them closer together depending on whether their present distance from each other is greater or smaller than that prescribed by the constraint.

Jakobsen [14] suggests that all existing constraints are processed sequentially a few times (around ten) in no particular order, and this will lead to a realistic simulation. Unfortunately, however, if a constraint was violated so that particles deviate too strongly from their prescribed distance, the relaxation may require many iterations. This is the case, for instance, when a rigid body containing a large number of constraints hits a wall at high speed. It is reasonable to suppose that, in this case, the particles that were moved as a consequence of the projection process should have their constraints enforced before the others. Moreover, these constraints should probably be processed more times than the others.

This observation prompted us to use a heuristic scheme by which constraints of rigid objects are enforced in different orders depending on which particles took part in the feature separation process. For each particle v, a list of constraints C_v is constructed wherein all constraints of the rigid body appear in increasing order of distance – in the sense of path length – from v in the *constraint graph*. This is a graph whose vertices represent particles of a given body and whose edges represent constraints between two particles. Thus, for instance, for particle v in Figure 7, list C_v would contain $< c_1, c_5, c_2, c_6, c_7, c_4, c_3 >$. In fact, in our implementation, constraints which correspond to neighbors of v (e.g. c_1 and c_5) are repeated n times in the list, where n is the distance of the farthest constraint from v (3, in this example, corresponding to the distance of c_3). Similarly, those which correspond to neighbors of neighbors of v are repeated $n/2$ times (e.g. c_2, c_6, c_7 and c_4) and so forth, with distant constraints appearing at least once in the list.

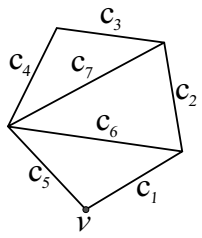


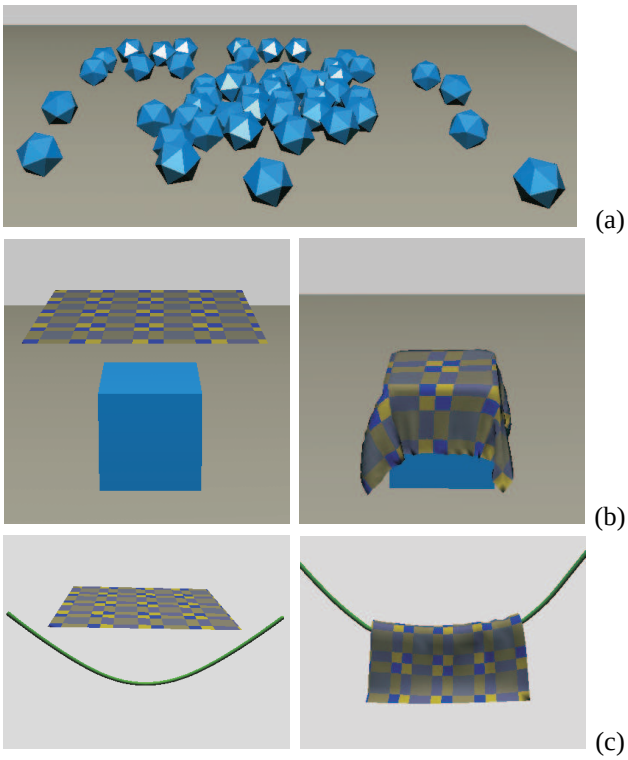
Figure 7. Vertex constraint graph.

The constraint lists for all particles of all object types are precomputed before the beginning of the simulation. During the simulation, if an object takes part in a collision, the particle which was subject to the largest movement is selected and its constraint list is used during the relaxation. If the object did not collide with any other, then its constraints are relaxed in any order, as per Jakobsen's approach.

7 Results

We have implemented the described system in the C++ language using OpenGL [7] for graphics rendering and the Computational Geometry Algorithms Library (CGAL) [5] which provided many geometric data structures. The system was used to run several experiments on a PC equipped with 512Mb main memory, a Pentium-IV 1.8 GHz processor and a ATI Radeon 9800 Pro graphics card.

The first experiment consists of a simple scene where several rigid icosahedra (36 constraints, each) are thrown in a cubical room (see Figure 8 (a)). The simulation runs in real time (around 20 fps) if up to 50 objects are placed in the scene. The chart in Figure 9 (a) shows the average frame rate as a function of the number of objects in the scene.

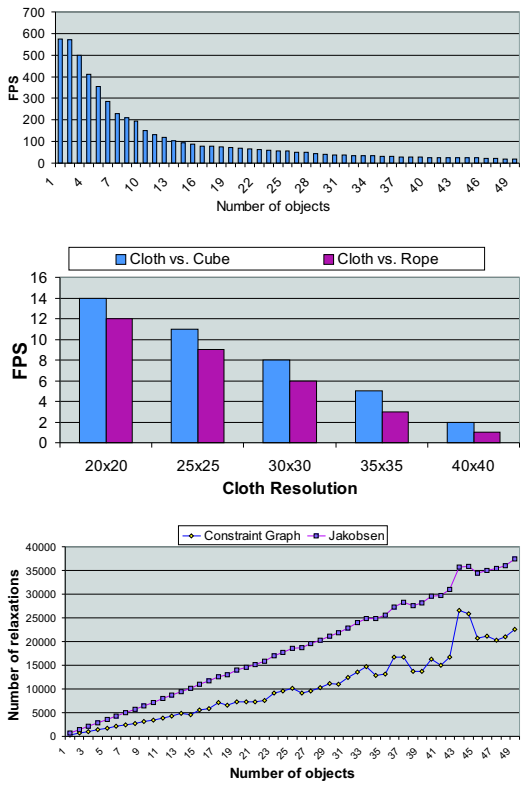


(a) (b) (c)

Figure 8. Experiments: Icosahedra on a cubic room (a), piece of cloth falling on a cube (b) and on a hung rope (c).

The same experimental setup was used to assess the effectiveness of the proposed relaxation heuristics based on constraint graphs by comparing it with Jakobsen's unordered relaxation. In this experiment, an object has each of its constraints relaxed only if it deviates more than 1% from their prescribed length. The chart in Figure 9 (c) shows the total number of required constraint relaxations needed for each relaxation scheme as a function of the number of objects in the simulation. The results suggests that the proposed scheme requires roughly half as many relaxations on the average than unordered relaxation.

Two other simulation experiments involving deformable objects were conducted. In the first, a piece of cloth is dropped on top of a cube and in the second, the cloth is dropped on a rope hanged by its two endpoints, as shown in Figure 8 (b,c). In these experiments the cloth resolution was varied from a coarse 20×20 mesh to a denser 40×40 mesh.



(a) (b) (c)

Figure 9. Simulation results: (a) rigid body frame rate, (b) deformable object frame rate, and (c) constraints relaxed per time step.

The hanging rope has 50 segments in all tests. The number of frames per second is plotted as a function of the cloth resolution in Figure 9 (b). These results show that finer cloth resolutions impose a severe impact on the collision detection engine. Even with a coarse mesh, the performance was only close to real time. This can be attributed to the cost of performing self-collision using a bounding volume hierarchy which is not very tight. Another point which deserves to be mentioned is that the simulation of "thin" objects must be conducted with very small time steps at the risk of allowing one object "go through" another.

8 Conclusions and Future work

We have described a system which uses a simplified physical animation framework originally proposed by Jakobsen [14] but which was extended with a true collision detection engine. In particular, we have described a scheme for resolving collisions between rigid bodies which does not require exact contact points to be computed but which nevertheless produce convincing collision responses. Similarly,

a sphere tree-based collision response scheme is described for coping with deformable objects.

It should be noted that the present paper is necessarily terse in the description of implementation details. For instance, we omitted details about the simulation of friction and some techniques borrowed from many sources when it comes to reducing the overhead of stacked objects (i.e., objects which continually collide but do not move).

We plan on extending the prototype system in many ways. As a first step, rotation physics will be incorporated using Baltman's [2] approach. Next, the sphere hierarchies will be put to much better use, probably by adopting some recently reported techniques [15]. Finally, the code needs some streamlining which should make the system a little less sluggish.

References

[1] P. Agarwal, L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Comput. Geom. Theory Appl.*, volume 28, pages 137–163, Amsterdam, The Netherlands, The Netherlands, 2004. Elsevier Science Publishers B. V.

[2] R. Baltman. Verlet integration and constraints in a six degree of freedom rigid body physics simulation. *Game Developers Conference*, 2004.

[3] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 23–34, New York, NY, USA, 1994. ACM Press.

[4] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–146, New York, NY, USA, 1996. ACM Press.

[5] CGAL. The computational geometry algorithms library. http://www.cgal.org.

[6] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. 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*, pages 189–ff., New York, NY, USA, 1995. ACM Press.

[7] Corporate OpenGL Architecture ReviewBoard. *OpenGL reference manual: the official reference document for OpenGL, release 1*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.

[8] B. Gartner. Fast and robust smallest enclosing balls. In *Proceedings of the 7th Annual European Symposium on Algorithms*, pages 325–338, 199.

[9] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three dimensional spaces. *IEEE Journal of Robotics and Automation*, April 1988.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, NY, USA, 1996. ACM Press.

[11] J. K. Hahn. Realistic animation of rigid bodies. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 299–308, New York, NY, USA, 1988. ACM Press.

[12] C. Hecker. Rigid body dynamics, 1998. http://www.d6.com/users/checker/dynamics.htm.

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

[14] T. Jakobsen. Advanced character physics. In *Proceedings of GDCONF'2001 Game Developer's Conference 2001*, 2001.

[15] D. L. James and D. K. Pai. Bd-tree: output-sensitive collision detection for reduced deformable models. In *ACM Trans. Graph.*, volume 23, pages 393–398, New York, NY, USA, 2004. ACM Press.

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

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

[18] M. C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[19] M. Mantyla. *Introduction to Solid Modeling*. W. H. Freeman & Co., New York, NY, USA, 1988.

[20] B. V. Mirtich. *Impulse-based dynamic simulation of rigid body systems*. PhD thesis, University of California at Berkeley, 1996.

[21] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1988. ACM Press.

[22] M. Ponamgi, D. Manocha, and M. C. Lin. Incremental algorithms for collision detection between solid models. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pages 293–304, New York, NY, USA, 1995. ACM Press.

[23] S. Quinlan. Efficient distance computation between non-convex objects. *IEEE Intern. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

[24] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[25] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[26] A. Verlet. Computer experiments on classical fluids: I. thermodymamic properties of leonard-jones molecules. *Phys. Review*, 159:98–103, 1967.

[27] E. Welzl. Smallest enclosing disks(ball and ellipsoids). In *New Results and New Trends in Computer Science, Volume 555 of Lecture Notes Comput. Sci.*, pages 359–370. Springer-Verlag, 1991.