Rapid Cloth Collision Detection using Bounding Sphere Trees

Fangyuan Ding*
Advanced Computer Graphics
Spring 2013

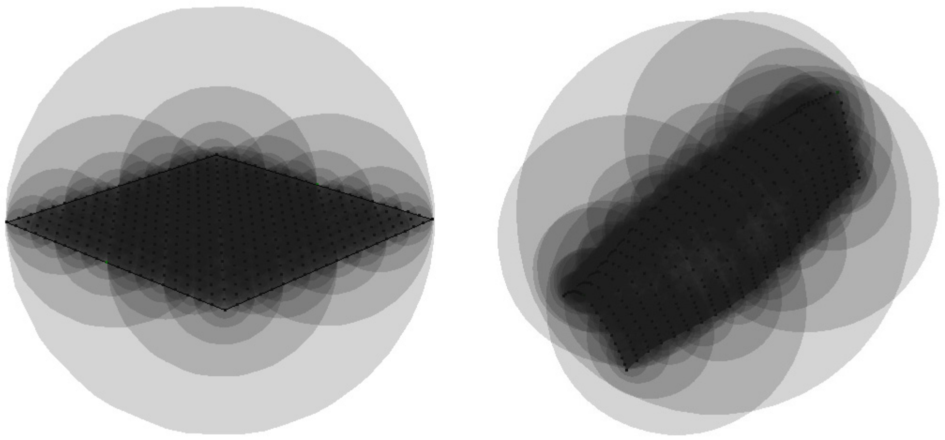

Figure 1: *A bounding sphere tree applied to a flat surface and draped cloth, respectively*

Abstract

The accurate detection of object-to-object collisions has numerous practical applications in the fields of computer simulation and graphics. However, naive collision detection and handling is a very costly process that increases exponentially with the size and complexity of the scene geometry, making it impractical for moderate to large-scale simulations.

This paper presents a method for efficiently handling collision detection in semi-rigid mass-spring particle systems such as cloth. In particular, this paper will outline the usage and implementation of a bounding sphere tree as a spatial accelerator, as well as other optimizations that can exploit this hierarchical structure.

Keywords: bounding sphere tree, cloth simulation, collision detection

Links: DL PDF

1 Introduction

The mechanics of collisions among rigid bodies is well-defined by the laws of classical mechanics. By interpolating the position of a rigid object along its velocity, we can derive its position at any point in time. In a similar manner, we can also derive the velocity and acceleration. However, this is not necessarily the case for semi-rigid and deformable surfaces, such as cloth, which are subject to internal forces as well as external ones.

Additionally, while the mechanics of such collisions may be well-defined, the process of detecting such a collision is less so. Given that modern meshes are usually composed of at many thousands of polygons, a per-polygon comparison for each object would be impractically slow. To aid in this endeavor, rigid bodies often employ bounding volumes, which are simple, geometric shapes that enclose the object. Because the intersections of bounding volumes are much easier to calculate, collisions between rigid objects can be computed in a reasonable time, at the expense of some accuracy.

An interesting physical-based approach for simulating cloth was presented by [Provot 1995], which treated the cloth as a grid of particles, each connected to its neighbors by springs. Under this model, each particle is treated a rigid body, which allows for the use of classical mechanics to approximate the cloths movement. A later paper by [Provot 1999] also details how to apply a simple collision detection algorithm to this system.

The goal of this paper is to expand upon the foundations presented in these papers, as well as those found in traditional rigid body collision algorithms, in order to build a fast and efficient model for collision detection within a mass-spring cloth system.

2 Collision Detection

The general case of collision detection involves the superposition of one object, be it a mesh, edge, or polygonal face, over another object at some time t. Let t_0 be a previous time at which these two objects were not in contact with each other. We can then define an interval $[t_0, t_0 + \Delta t]$, where Δt is the difference in time between t and t_0, over which the collision occurred. Because we know both

*e-mail:dingf@rpi.edu

the positions and velocities of each particle at time t_0, we can calculate their respective positions at time t.

The relatively simple nature of the mass-spring system means that there are only two types of collisions that need to be checked for: point-to-triangle collisions and edge-to-edge collisions. Note that these collisions must be checked reciprocally in order to be accurate; given two triangles, checks must be made both from the vertices of the first triangle to the second and from the vertices of the second triangle to the first.

2.1 Point-to-Triangle Collisions

Consider the case of a point and triangle in our simulation. In general, neither the point nor the triangle will remain stationary due to the various forces that are being applied to them. Therefore, let $P(t)$ denote the position of the point at time t, and $A(t)$, $B(t)$, $C(t)$ denote the positions of the vertices of the triangle ABC at time t, respectively. We can define $P(t)$, $A(t)$, $B(t)$, and $C(t)$ as follows:

$$\begin{aligned} P(t) &= P_0 + tV_P \\ A(t) &= A_0 + tV_A \\ B(t) &= B_0 + tV_B \\ C(t) &= C_0 + tV_C \end{aligned}$$

where P_0, A_0, B_0, and C_0 denote the positions of the corresponding points at time t_0, and V_P, V_A, V_B, and V_C denote the velocities of these points at time t_0, respectively.

When $P(t)$ intersects with the triangle ABC, we note that $P(t)$ is coplanar with ABC. Let us define the vectors $\overrightarrow{AP}(t)$, $\overrightarrow{AB}(t)$, and $\overrightarrow{AC}(t)$ as follows:

$$\begin{aligned} \overrightarrow{AP}(t) &= P(t) - A(t) = P_0 + tV_P - A_0 + tV_A \\ \overrightarrow{AB}(t) &= B(t) - A(t) = B_0 + tV_B - A_0 + tV_A \\ \overrightarrow{AC}(t) &= C(t) - A(t) = C_0 + tV_C - A_0 + tV_A \end{aligned}$$

When $P(t)$ is coplanar to the ABC, $\overrightarrow{AP}(t)$ will be perpendicular to the normal of the plane. Therefore, we can define this coplanarity in terms of the scalar triple product of the three vectors, $\overrightarrow{AP}(t)$, $\overrightarrow{AB}(t)$, and $\overrightarrow{AC}(t)$:

$$\overrightarrow{AP}(t) \cdot (\overrightarrow{AB}(t) \times \overrightarrow{AC}(t)) = 0 \tag{1}$$

From our definitions above, we can see that this equation is a cubic polynomial of t. Solving for (1) produces up to three values of t, which correspond to the different times at which the point and triangle are coplanar. However, we are only interested in the value t_c that corresponds to our original interval, $[t_0, t_0 + \Delta t]$. If no such t_c exists, then we can safely conclude that there is no collision. If multiple valid t_c values exist, we take the smallest value instead.

Recall that the solutions to this polynomial only define the times at which the point is coplanar with the triangle. In order to determine whether the point is actually intersecting with the triangle, we first extrapolate both the point and the triangle to their respective locations at time t_c. Then, the position of the point $P(t_c)$ is converted from the Cartesian coordinates (x_p, y_p, z_p) to the barycentric coordinates $(\alpha_p, \beta_p, \gamma_p)$ with respect to ABC. If α_p, β_p, and γ_p are all within the interval $[0, 1]$, then we confirm that the point is in the triangle. Otherwise, we reject t_c as a valid collision time.

2.2 Edge-to-Edge Collisions

Let us consider the general case of two edges moving simultaneously. We will define $AB(t)$ to be the edge defined by $(A(t), B(t))$ and $CD(t)$ to be the edge defined by $(C(t), D(t))$. When $AB(t)$ and $CD(t)$ intersect, we observe that the points $A(t)$, $B(t)$, $C(t)$, and $D(t)$ must be coplanar. Therefore, to derive the time of collision, we will use the following equation, which is similar in nature to (1):

$$\overrightarrow{AC}(t) \cdot (\overrightarrow{AB}(t) \times \overrightarrow{CD}(t)) = 0 \tag{2}$$

where $\overrightarrow{AC}(t)$ is the vector from $A(t)$ to $C(t)$, $\overrightarrow{AB}(t)$ is the vector from $A(t)$ to $B(t)$, and $\overrightarrow{CD}(t)$ is the vector from $C(t)$ to $D(t)$.

Once again, solving the cubic equation (2) for t_c is not sufficient in determining whether a collision actually occurred between $AB(t)$ and $CD(t)$. To determine whether the edges are intersecting, we find the line (P_{AB}, P_{CD}) such that the distance between $AB(t_c)$ and $CD(t_c)$ is minimized. If the distance between P_{AB} and P_{CD} is approximately zero, then the two lines that contain $AB(t_c)$ and $CD(t_c)$ collide at time t_c. However, this is already known from the previous step. Instead, we check whether or not P_{AB} and P_{CD} are contained within their respective edges; if the distance from the midpoint of $AB(t_c)$ and $CD(t_c)$ to P_{AB} and P_{CD}, respectively, is less than or equal to half of the length of the corresponding edge, then we find that there is a collision between $AB(t)$ and $CD(t)$ at time t_c.

3 Bounding Sphere Tree

Specifically, a bounding sphere S is a sphere with center c_s and radius r_s such that the object that it is bounding is completely contained within its bounds. That is, every point in the bounded object must be within r_s distance from c_s. This makes the detection of any collision within a bounding sphere trivially easy. Given a set of points however, finding the exact minimum bounding sphere that encompasses those points is computationally difficult. Instead, we use a fast approximation algorithm, along with known properties of our cloth, in order to approximate a very tight fitting minimum bounding sphere.

Recall that in a mass-spring system, cloth particles are always aligned in a rectangular $w \times h$ grid. Therefore, under normal deformation constraints, we may assume that the center of this grid, $(\frac{w}{2}, \frac{h}{2})$ is reasonably close to the center of the exact minimum bounding sphere. From this point, we construct an axis-aligned bounding box that contains every cloth particle. We may then trivially construct an initial bounding sphere with center c_s equal to the center of the bounding box and radius r_s equal to half the length of the box's diagonal.

Once we have our initial bounding sphere, we recursively subdivide the parent bounding sphere into four children bounding spheres, based on the positions in 2D texture space of its corresponding bounded grid. If a parent sphere bounds a grid with dimensions $w \times h$, then its children will bound a grid of dimensions $\frac{w}{2} \times \frac{h}{2}$ with centers at $(\frac{w}{4}, \frac{h}{4})$, $(\frac{3w}{4}, \frac{h}{4})$, $(\frac{w}{4}, \frac{3h}{4})$, and $(\frac{3w}{4}, \frac{3h}{4})$, respectively. If, due to excessive deformation, the radius of the child bounding sphere is not large enough to bound its allocated particles, then we extend its radius using the axis-aligned bounding sphere method described above. Note that, unlike many other space partitioning structures, this subdivision scheme does not completely partition the parent's volume among its children. It is however, guaranteed to fully bound the cloth at any depth of the tree.

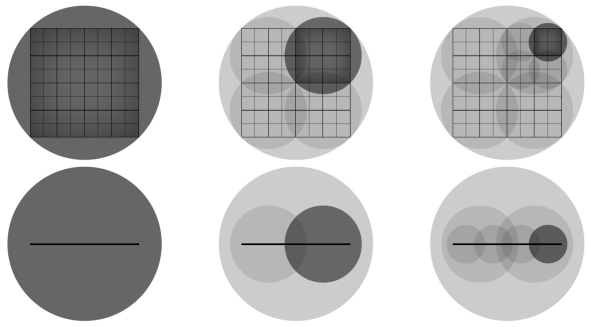

Figure 2: *Top and side view of the bounding sphere tree after two iterations. As depth increases, the volume quickly converges to that of the bounded surface.*

Once a certain depth is reached, the subdivision algorithm halts. This depth varies based on the size of the bounded cloth; bounding spheres will continue to subdivide until the number of particles that they bound, and consequently, their area, is less than a threshold value. These terminal bounding spheres are then marked as leaf nodes in the tree, which store pointers to their bounded cloth particles and surrounding neighbors, but do not have any children of their own.

When performing a collision test, we can then quickly eliminate large portions of non-intersecting particles by only traversing those nodes through which the bounding sphere intersects the object's volume. Intersecting leaf nodes are then collected and individually checked for collisions. Note that, in order to be accurate, we must consider the object over the interval $[t_0, t_0 + \Delta t]$. For a point $P(t)$, this becomes a line segment of the form $(P(t_0), P(t_0+\Delta t))$. For an edge $AB(t)$, this will be a quadrilateral of the form $(A(t_0), B(t_0), B(t_0 + \Delta t), A(t_0 + \Delta t))$ instead.

Because of the dynamic nature of deformable systems such as cloth, a previously generated bounding sphere tree is not guaranteed to remain accurate for future calculations. Therefore, at each iteration of the simulation, the bounding sphere tree must be generated again in order to be valid. This adds some undesirable, albeit necessary, overhead to the simulation's running times.

4 Normal Masks

For the purposes of this section, let us only consider the case of self-collisions. In such a scenario, we note that a collision can only occur over a given area if the curvature of that area is sufficiently large. That is to say, a relatively flat piece of cloth will not have any self-collisions whatsoever. More formally, we can describe this as follows: if there exists a vector V such that, for each normal vector N in the area, $N \cdot V > 0$, then no self-collision can occur within that area.

Our method builds upon the work of [Volino and Thalmann 1998], who used a hierarchical bounding box structure for spatial acceleration. First, we construct a reference table of 26 vectors, one from the center of a unit cube to each of its corners, edges, and face centroids. Then, for each leaf node, we calculate which of these 26 vectors have a positive dot product with each of the normal vectors within the area bounded by the bounding sphere. Because we are only interested in whether a vector has a positive dot product with the normals of a given area, we can record the results for each of the 26 vectors within a single 32-bit bitmask. Normal masks must also be generated for time $t_0 + \Delta t$, in addition to time t_0, in order to ensure correctness.

Once the normal masks for the leaf nodes have been constructed, the normal masks for previous depths may also be trivially constructed: for any non-leaf node, any vector V that has a positive dot product with every face in its bounded area will also be common to each of its children nodes. Therefore, its normal mask is simply the intersection of the normal masks of its children. Note that, because we only use the 26 previously described vectors for comparison, there may exist a vector V for a node, even if its normal mask is completely empty. In practice however, this does not occur very often.

When performing self-collision checking over two nodes that bound areas of the same cloth, we first test their normal masks; if the intersection of their normal masks at both time t_0 and time $t_0 + \Delta t$ is not zero, and the intersection of these intersections is also non-zero, then there exists a vector V that is common to both areas. In that case, we may safely assume that self-collisions will not occur between the two areas, and no further collision testing is performed.

5 Results and Discussion

Our algorithm produces correct bounding spheres trees that fully bound the surface of the cloth at any time t. Because the surface of the cloth is constantly changing however, new bounding sphere trees must be generated at each iteration of the simulation. As mentioned earlier, this adds some overhead to the simulation, usually on the order of 20-40% of the original running time. Nevertheless, we find that our algorithm is still practical for producing real-time cloth simulations.

As the bounded cloth becomes increasingly deformed however, the performance of the bounding sphere tree begins to decrease. This is largely due to two factors: first, the normal masks between nodes are far less likely to rule out self-collisions. And second, the number of bounding spheres intersecting a given object is likely to increase in a very tightly deformed area. In the worst case, if every particle were close enough such that all of the leaf bounding spheres intersected each other, the bounding sphere tree would provide almost no benefit at all. However, the same could be said for any other bounding volume hierarchy as well.

One more thing to note is that volume is not conserved between depths in a bounding sphere tree. In fact, the combined volume bounded by four children spheres is usually much smaller than the volume of their parent bounding sphere itself. Indeed, as depth increases, the combined bounding volume of all nodes at that depth converges to that of the surface itself, making it ideal for bounding flat, deformable surfaces such as cloth. As the cloth becomes deformed, the bounding volume of a parent node is also not guaranteed to fully bound the bounding volume of any of its children. However, any volume in a child sphere that lies outside of the volume of its parent is largely irrelevant, for it will neither contain cloth particles nor be reachable by collision testing.

Cloth	Grid Size	Frame Rate*	Frame Rate†
singularity.txt	15x15	45.33	37.17
drape.txt	20x20	27.91	19.62
bigsingularity.txt	100x100	1.26	0.95

*without bounding sphere tree calculations
†with bounding sphere tree calculations

Table 1: *Rendering times for our cloth simulation, both with and without bounding sphere tree calculations, for various cloth samples. Note that actual collision detection was not performed in any of these test cases.*

6 Limitations

Because our underlying cloth model is based off of the one presented by [Provot 1995], we use explicit Euler integration to determine the position, velocity, and acceleration of every particle for each iteration of the simulation. This is also required for our collision detection, which uses Euler integration to extrapolate the position of a point, triangle, or edge at a future time. However, the accuracy of Euler integration is also proportional to the step size used, and therefore becomes unstable when a sufficiently large timestep is used. This severely limits the range of cloth that this model is capable of simulating.

Additionally, recall that our algorithm relies on an axis-aligned bounding box to generate bounding spheres. While this is a fast approximation method that produces passable results for most situations, it also fails to produce the absolute minimum bounding sphere for a given set of points. An alternative method presented by [Welzl 1991] introduces a method for producing the absolute minimum bounding sphere in linear time. However, due to time constraints, we chose not to implement this algorithm in our solution.

Finally, and perhaps most important of all, our collision detection algorithm is not currently functional. While we are able to construct the bounding sphere tree and calculate the correct solutions to the cubic equations (1) and (2), actual collision detection remains disabled for the time being. A naive collision detection method was attempted; however, this method proved impractically slow, even with normal mask and boundary sphere culling. Additionally, while this algorithm above did produce the correct results eventually, it failed to perform any collision handling afterwards, producing no visible results in the underlying cloth.

References

PROVOT, X. 1995. Deformation constraints in a mass-spring model. *Graphics Interface*.

PROVOT, X. 1999. Collision and self-collision handling in cloth model dedicated to design garments. *Graphics Interface*.

VOLINO, P., AND THALMANN, N. M. 1998. Collision and self-collision detection: Efficient and robust solution for highly deformable surfaces.

WELZL, E. 1991. Smallest enclosing disks (balls and elipsoids). *New Results and New Trends in Computer Science*.