Intersection Test for Collision Detection in Particle Systems

Evaggelia-Aggeliki Karabassi   Georgios Papaioannou   Theoharis Theoharis

Depatment of Informatics, University of Athens, Greece

Abstract. We present a method for detecting collisions between a system of spherical particles and an environment composed of triangles. The proposed algorithm takes into account the particles' volume and is based on an intersection test between a triangle and a cylinder with spherical caps (the trajectory of a particle). The algorithm also efficiently calculates a close estimate of the point of intersection, which is necessary for collision detection.

1. Introduction

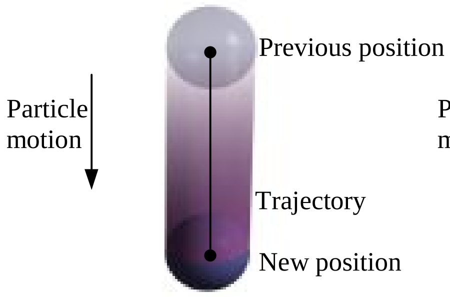
We describe a test for the intersection of a triangle with a cylinder-with-spherical-caps (see Figure 1). This work has been motivated by the need to compute the behavior of particle systems [Reeves 83] where the volume of the particles may not be ignored and the particles may not be considered as points. This is, for example, the case of materials like slime, mud, oil and snow. Furthermore, if the volume and mass properties are to be manipulated for simulation purposes, the collision detection problem may not be reduced to simple ray/surface intersections, as suggested in [Miller, Pearce 89]; instead we need to develop a set of intersection tests between spherical particles' trajectories and polygons.

Ignoring the particles' dimensions does not only affect physical simulations but can also lead to non-realistic images, especially when the spatial variance of the scenery is finer than the particle size. For instance, the volume of a quantity of snow cannot be ignored when trying to force it past a small opening. If we consider the trajectories of particles as rays, some snow would pass through the opening when it should not. We overcome this inconsistency by allowing the elementary quantity of the deformable material to have a specific volume.

In our model, we consider each particle's trajectory between two time steps as composed of linear segments, as is the case in many particle system simulators. In this circumstance, we may take advantage of this linear motion to provide a quick intersection test. To take into account the particle's volume, we model each segment of the trajectory as a cylinder with spherical caps, which is the trace of a moving sphere (Figure 1). Collisions are thus handled as cylinder-with-spherical-caps/triangle intersections (Figure 2). To simplify our method we ignore collisions between particles.

2. Main Algorithm

The main idea in our method is to find in each time step the collisions between a cylinder with spherical caps, representing a particle's trajectory, and the triangles that form the environment. If the trajectory intersects with more than one triangle, then only the



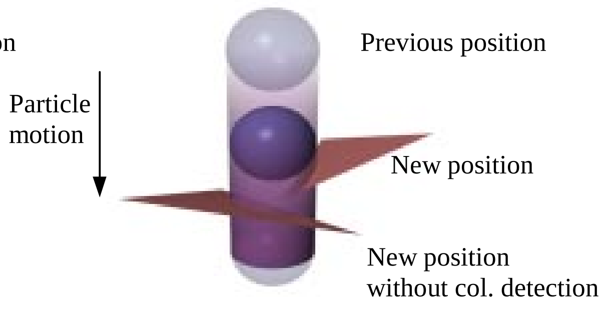


Figure1. Particle movement between 2 frames.   **Figure2.** Collision detection.

intersection closer to the beginning of the trajectory (i.e. is the particle's previous position) is of interest (Figure 2). The triangles are either found in a list or we may consider a space subdivision for efficiency. The main loop of the algorithm is, therefore, shaped as follows :

```
• for each spherical particle B do :
    • compute the new location I of B, assuming no collisions
    • for each triangle T in the triangle hierarchy :
        • check if there is a collision between B's trajectory and  T
        ( cylinder/triangle intersection )
        • if not
            check if there is a collision between B's final position and T
            ( spherical-cap/triangle intersection)
        • if a collision point Q exists
            • compare Q to previous collision point I
            • if Q is closer to B's previous position
                set I=Q
    • return I
```

There is no need to check for intersections between triangles and the spherical cap representing the initial position of the particle, as that position represents the final position of the previous time step.

The method described above consists of two distinct parts: spherical-caps/triangle and cylinder/triangle intersection tests. The combination of these two algorithms forms a cylinder-with-spherical-caps/triangle intersection test, which corresponds to finding collisions between a particle's trajectory and the environment. We should note here that the two tests (cylinder/triangle and sphere/triangle) are designed and optimized to work in conjunction. Thus, for example, some cylinder-with-spherical-caps/triangle intersections will be detected by the sphere/triangle test and not by the cylinder/triangle test, although the triangle may intersect both constituent objects, because the sphere/triangle test is computationally cheaper.

To achieve computational efficiency, we calculate a close estimate to the exact intersection point on the particle's trajectory in the case where the latter is expensive to compute. The approximation accuracy decreases as the length of the time step (cylinder height) increases in relation to the particle size and is acceptable in the usual case of deformable particles.

In the following tests, we assume that each particle is represented by a 3-dimensional point *C*, its radius *R*, and R^2 (for speed up reasons). Each triangle *T* is defined by its vertices ($T.P_1$, $T.P_2$, $T.P_3$), its normal vector $T.N=(\alpha,\beta,\gamma)$ and a constant $T.\delta$ (from the triangle's plane equation $\alpha x+\beta y+\gamma z+\delta = 0$).

3. Spherical - Cap / Triangle Intersection Test

As already mentioned, a sphere/triangle intersection test is required, for detecting collisions once the particle reaches its final position. Such an algorithm is also necessary if we want to detect particle collisions with moving objects. Although there are many references in the graphics bibliography to sphere/ray [Watt 92] or ray/triangle intersection tests [Voorhies, Kirk 91], we have not found any specialized method for detecting intersections between a sphere and a triangle. A general collision detection algorithm for convex polyhedra [Ponamgi et al 97] could be applied, but the performance would be poor.

Here we present a method based on tests of incremental complexity, which gives very satisfactory results. In parentheses we note the number of additions/subtractions (+/-) and multiplications (*) required.

Step1. Check whether the triangle's plane intersects with the sphere. If not, there is no sphere-triangle intersection(Figure 3a). Else go to step 2.

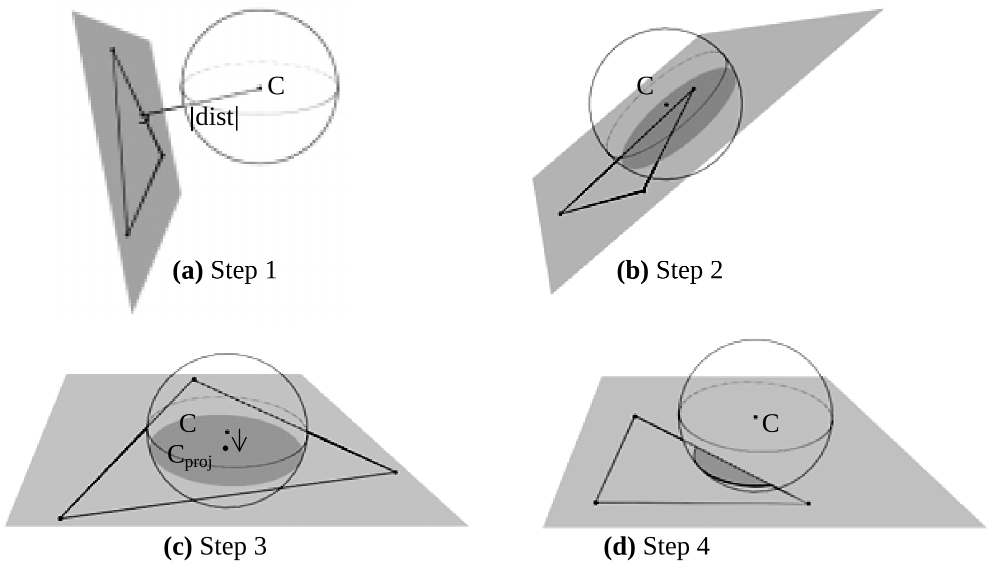


Figure 3. Sphere/triangle intersection test

This is done by calculating the plane's distance *dist* from the sphere's center (using the plane equation : $dist = T.\mathrm{N} \cdot C + T.\delta$) and comparing *abs(dist)* to R . (3+/-, 3*)

Step2. Check whether any of the triangle vertices is inside the sphere. If yes, the sphere and the triangle intersect (Figure 3b). Else go to step 3.
The test is done by calculating each vertex's distance *d* from the sphere's center and comparing d^2 to R^2 (best case : 5+/-, 3*, worst case : 15+/-, 9 *).

Step3. Project the sphere onto the triangle's plane : $C_{proj} = C - dist \cdot T.\boldsymbol{N}$ (3+/-, 3*).
Check if the projected center C_{proj} lies inside the triangle (15+/-, 14*). If yes, there is an intersection (Figure 3c). Else go to step 4.

Step4. Check whether the sphere intersects with a triangle edge (Section 3.1) (18 +/-, 16 *). If yes, the sphere and the triangle intersect (Figure 3d). Else there is no intersection.

3.1 Sphere / line intersection test (step 4)

In the algorithm presented above, we need to further analyze the sphere/edge intersection test (Figure 4). Below, we use bold symbols for position vectors and plain symbols for points.

Given a line segment defined by two points P_1, P_2 and a sphere defined by its center C and radius R, we want to determine whether the two objects intersect. If the line through C which is perpendicular to P_1P_2, intersects with P_1P_2 at Q, then our problem is equivalent to determining whether Q lies between P_1 and P_2 and comparing R^2 with the square of the distance d between C and Q.

Let $\boldsymbol{N}$ be P_2-P_1. Using the parametric equation of the line segment P_1P_2,

$$Q = P_1 + t \cdot (P_2 - P_1) = P_1 + t \cdot \boldsymbol{N} \tag{1}$$

the parameter t for point Q can be calculated by :

$$t = \frac{\boldsymbol{N} \cdot \boldsymbol{C} - \boldsymbol{N} \cdot \boldsymbol{P_1}}{\boldsymbol{N} \cdot \boldsymbol{N}} \tag{2}$$

Let the square of the distance between Q and C be d^2. If $d^2 > R^2$, there is no intersection. If $d^2 = R^2$ the line is tangential to the circle and the contact point Q is between P_1 and P_2 if $0 \le t \le 1$.

If $d^2 < R^2$, the line defined by P_1 and P_2 intersects with the sphere. If $0 \le t \le 1$ then Q lies between P_1 and P_2 and the line segment P_1P_2 intersects with the sphere. Otherwise, an

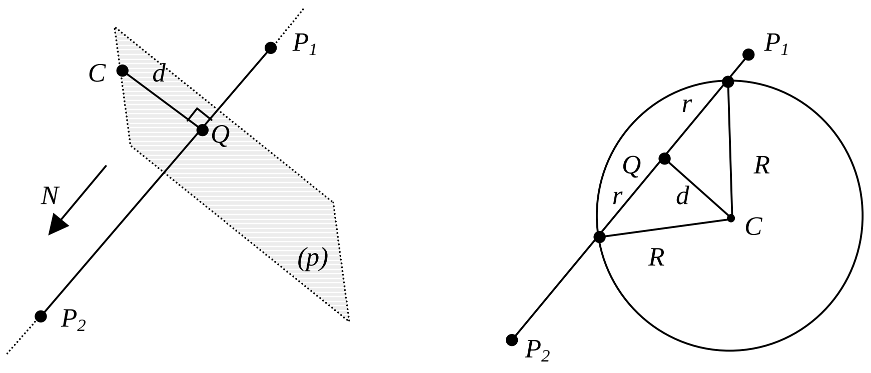


Figure 4. Line/sphere intersection.

intersection occurs if the closest end point to Q lies inside the sphere. The closest point is P_1 if $t < 0$ or P_2 if $t > 1$. However we have already excluded that possibility in Step 2. Therefore if $t < 0$ or $t > 1$, P_1P_2 does not intersect with the sphere.

The algorithm described above is of use in a variety of applications where edge/sphere or ray/sphere intersections are involved, such as ray-tracing. In fact this method is comparable to the one in [Held 97], and 10% faster than the widely used one suggested in [Watt 92]. The results have been calculated over a large number of test sets and are independent of the percentage of intersections found.

3.2 Sphere final placement

The sphere/triangle intersection test does not calculate the exact points of intersection. This task would be too complicated and time-consuming, without being essential to the application. Once we determine that the two objects intersect, we only need to leave the sphere at its current position so it will not pass through the polyhedron, whose surface contains the triangle. This is a valid assumption for particles which are not large relative to the triangles, or particles representing deformable objects.

In case of solid spheres, the algorithm could be easily modified to add a displacement depending on the triangle's normal and the sphere's radius, so that the two objects would merely be in touch.

4. Cylinder / Triangle Intersection Test

The cylinder/triangle intersection test is rather complicated for two reasons :

- it must take into account a variety of possible relative positions of the two objects
- the calculation of the intersection point cannot be omitted.

Since the cylinder represents a particle's movement in one time step, the objective is to find a good estimate of the point where this movement will be interrupted. The cylinder axis corresponds to the particle center's trajectory, so we shall seek a point on the axis that will represent the particle's position at the moment of collision. Each cylinder is represented by two points, C_1 and C_2, and a radius R. In the case of particle motion, C_1 corresponds to the previous position of the particle's center, while C_2 represents the position that the particle's center will have if there are no collisions in this path.

Step1. Define the normalized vector ***A***, which indicates the cylinder axis direction

($\boldsymbol{A} = \dfrac{C_1 - C_2}{\|C_1 - C_2\|}$) .

Step2. Calculate the plane equation values p_{1a}, p_{2a}, p_{3a}, p_{1b}, p_{2b}, p_{3b} for each triangle vertex ($T.P_1$, $T.P_2$, $T.P_3$), with regard to planes (π_a), (π_b), which mark the cylinder caps. Check the sign of the above values, to determine on which side of each plane the vertices lie. If all vertices are outside the caps and on the same side, there is no intersection.

Step3. Check whether the cylinder axis intersects with the triangle's plane (π). If the axis is parallel to (π), that is $\boldsymbol{A} \cdot T.\boldsymbol{N} = 0$, go to Par4, else go to Int4.

Axis is parallel to the triangle plane :

Par4. Find the distance between the axis C_1C_2 and (π): $dist = T.\boldsymbol{N} \cdot C_1 + T.\delta$. If $abs(dist) > R$ there is no intersection (Figure 5a).

Par5. If any of the triangle vertices lies inside the cylinder, set as intersection point Q the projection on C_1C_2 of the vertex inside the cylinder that is closer to C_1 (Figure 5b).

Par6. Project C_1C_2 on (π).Let the projection be $C_1'C_2'$. If C_1' lies inside T, then return C_1 as the intersection point.

Par7. Calculate the intersections between $C_1'C_2'$ and the edges of T (if any) and find the intersection Q' closer to C_1. Return as intersection point the projection Q of Q' on the cylinder axis (Figure 5c).

Par8. If no intersection is found, possible intersections of T should be detected by the spherical-cap/triangle test.

Axis intersects with the triangle plane :

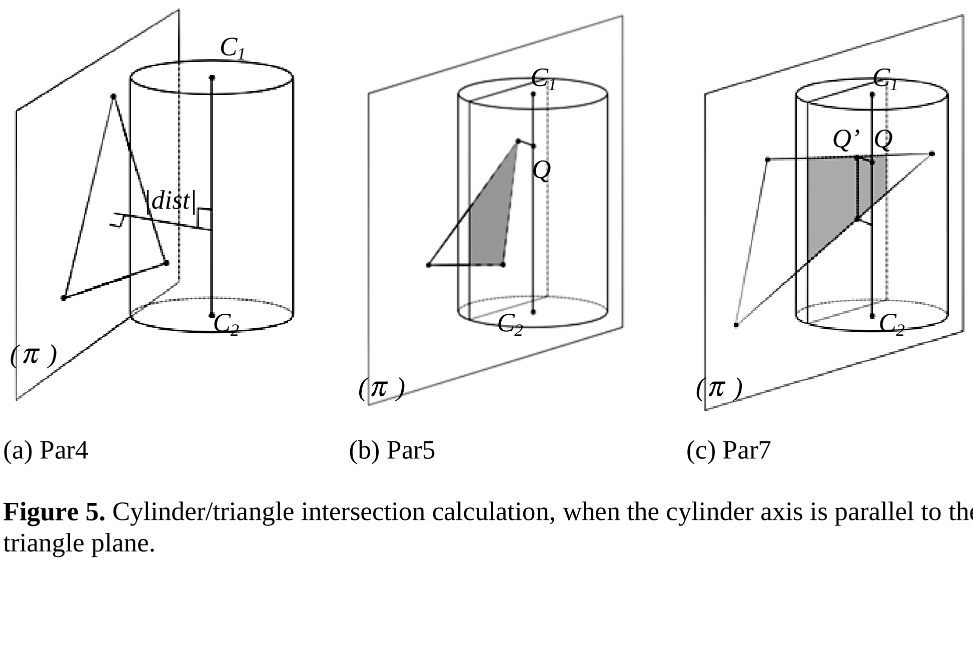
Int4. Find the intersection point Q.

($Q = C_1 + t \cdot (C_2 - C_1)$ where $t = -\dfrac{T.\boldsymbol{N} \cdot (C_1 - T.P_1)}{T.\boldsymbol{N} \cdot (C_2 - C_1)}$).

If Q lies between C_1 and C_2 ($0 \le t \le 1$) and inside T, then return Q as the intersection point (Figure 6a).

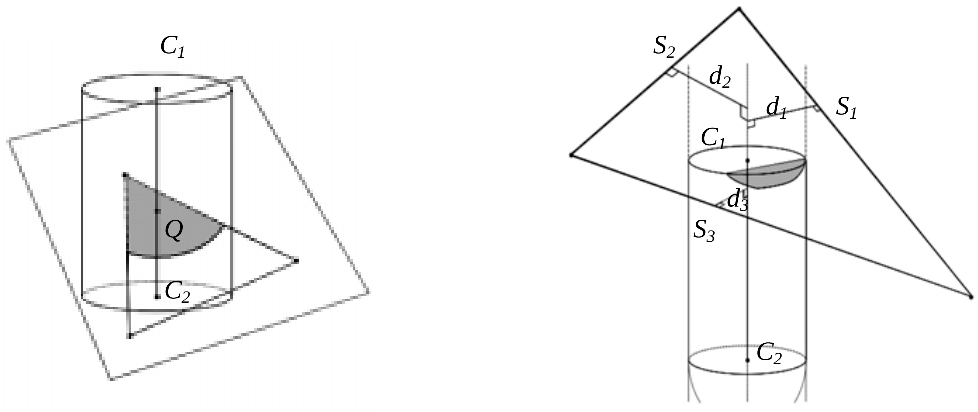
Int5. Find the closest points (S_1, S_2, S_3) between each triangle edge ($T.P_1\,T.P_2$, $T.P_2\,T.P_3$, $T.P_3\,T.P_1$ respectively) and the infinite axis C_1C_2 (see 4.1 for details). Let the corresponding distances be d_1, d_2, d_3 . If $d_1, d_2, d_3 > R$, then possible intersections (Figure 6b) will be detected by the spherical-caps/triangle test.

Int6. Check if any of S_1, S_2, S_3 lies inside the cylinder (Figure 6c). In this case, return as intersection point Q the projection on C_1C_2 of the point (among S_1, S_2, S_3) that is

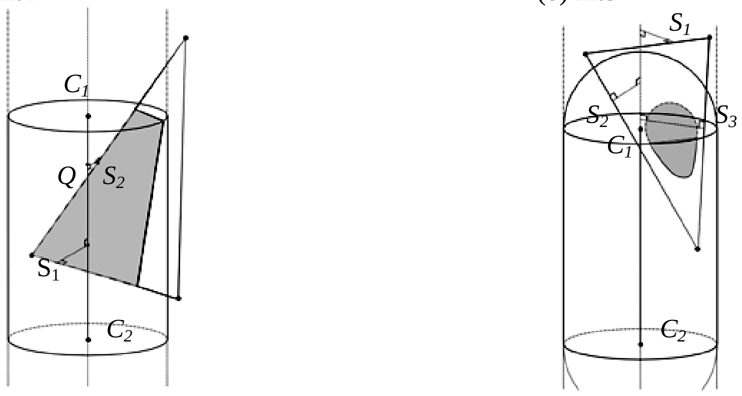


(a) Par4 (b) Par5 (c) Par7

Figure 5. Cylinder/triangle intersection calculation, when the cylinder axis is parallel to the triangle plane.



(a) Int4 (b) Int5



(c) Int6 (d) Int6

Figure 6. Cylinder/triangle intersection calculation, when the cylinder axis intersects with the triangle plane.

closer to C_1. Else intersections would be detected by the spherical-cap/triangle test (Figure 6d).

As already mentioned, this method does not only detect intersections but also calculates an intersection point. In most cases, the intersection point is exact. For case Int6, however, the exact point would be expensive to compute, and the algorithm as described returns an approximate intersection, which is sufficient for our application.

If the particles were always very small or the movement restricted to certain axes, the algorithm and the individual tests it uses in each step could be further simplified. However, this method has been developed to work regardless of particle size and the results have been satisfactory, even for particles whose dimensions were similar to those of the triangles.

4.1 Closest point between a line and a line - segment

In step Int5 of the above algorithm, we need to find the point P of an edge P_aP_b which has the minimum distance d from a line *(l)* defined by two points C_1 and C_2 (Figure 7).

Let the distances between P_a and *(l)* and P_b and *(l)* be d_a and d_b respectively. We wish to define the location of point $P = P_a + t \cdot (P_b - P_a)$. If we project P_aP_b, P_aQ_a and P_bQ_b onto a plane (π) that is perpendicular to *(l),* the projections will form a triangle (Figure 7). If we choose (π) so as to contain P_b, the triangle is $Q_bP_bP_a'$. Let the projection of P on (π) be $\mathbf{P' = P_a' + }t'\mathbf{(P_b - P_a')}$. Since distance proportions are maintained in parallel projections, $t = t' = \frac{s_a}{s}$. Using the Pythagorean theorem : $s_a = \frac{d_a^2 - d_b^2 + s^2}{2s}$.

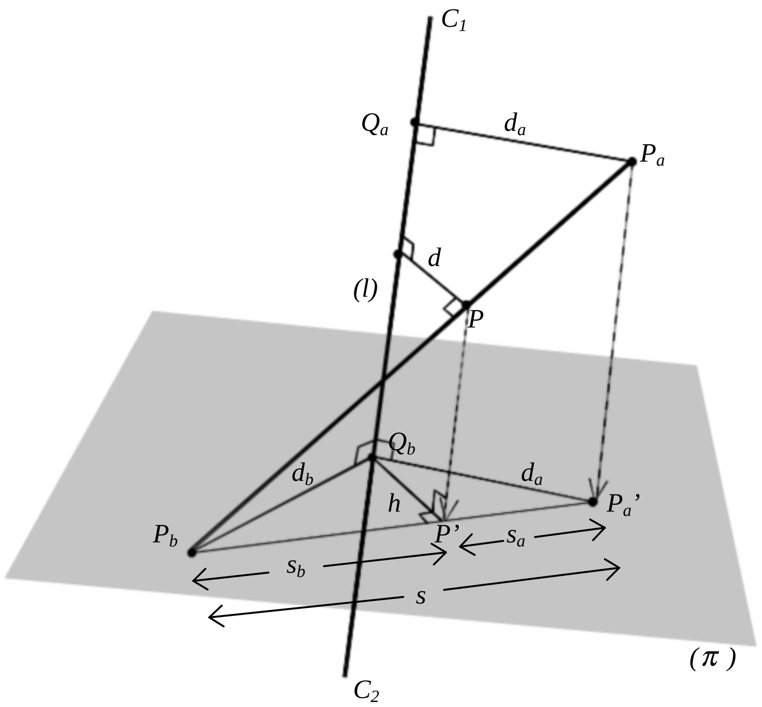


Figure 7. Calculation of the closest point *P* of segment P_aP_b to line *(l)*.

Therefore : $t = \dfrac{d_a^2 - d_b^2 + s^2}{2s^2}$ (3)

In the above equation, we need to compute s^2, which is the square distance between P_b and $\boldsymbol{P_a'} = \boldsymbol{P_a} + (\boldsymbol{Q_b} - \boldsymbol{Q_a}) = \boldsymbol{P_a} + (t_b - t_a)(\boldsymbol{C_2} - \boldsymbol{C_1})$ where t_a, t_b are the normalized distances from C_1 to Q_a and Q_b respectively (for the calculation of t_b, t_a see equation (2) in 3.1).

If the parameter calculated in equation (3) lies between 0 and 1, then the closest point P is found between P_a and P_b. If $t \leq 0$ then we should set $P = P_a$, while if $t \geq 1$ then P should be set equal to P_b.

5. Results

The cylinder/triangle and spherical-cap/triangle intersection algorithms, as well as the cylinder-with-spherical-caps/triangle combined algorithm were tested on a Pentium II PC with a 400MHz processor, running Windows 98. The code was compiled using Microsoft Visual C++ 6.0. Several sets of 10,000,000 tests were conducted for each case. Each test was done between a triangle and a cylinder, sphere or cylinder-with-spherical-caps, all randomly placed. The results are presented in Table 1 in comparison to the ERIT library [Held 97], which have been run on the same platform. The second and third columns give the average execution times for our algorithm and ERIT respectively. As can be observed, the sphere/triangle intersection tests have identical execution times, while our algorithm clearly outperforms the ERIT library in the case of the cylinder/triangle intersection test, despite the fact that our algorithm also computes an estimate to the intersection point. Of course, it must be taken into account that the ERIT test is a general one, whereas the one presented here cannot be used autonomously but only in conjunction with the spherical-cap/triangle intersection test.

Intersection test	**Execution time (sec) Our algorithms**	**Execution time (sec) ERIT**
Sphere/Triangle	16	16
Cylinder/Triangle	29	38
Combined	40	-

Table 1. Execution time (in sec) for 10,000,000 tests.

The above tests were used in a particle simulator which produced output for the Persistence of Vision Ray Tracer (POVray). Figures 8b and 8c show a room (Figure 8a) covered with a "blobby" mass. 2,912 particles where used. Figure 8b shows the results obtained with our method, while Figure 8c was produced using a standard ray/triangle intersection algorithm, which ignores each particle's volume. As can be seen, our approach produces better results, especially near edges and small objects. Figure 9 shows a grid covered with slime. 121,200 blobs have been used. Figure 10 shows a snow covered house. 153,600 blobs have been used. Both images have been created using one time-step; we have assumed particle movement on only one axis. The results of a more complex motion, could be easily modeled using more time steps.

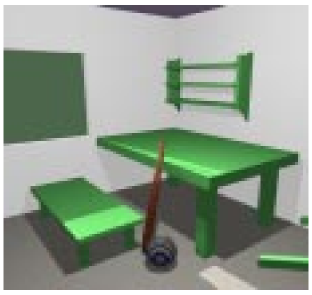


Figure 8a. Room

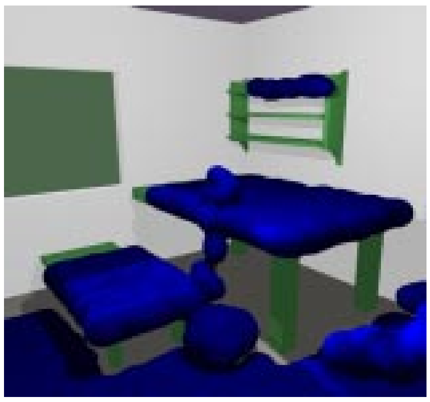
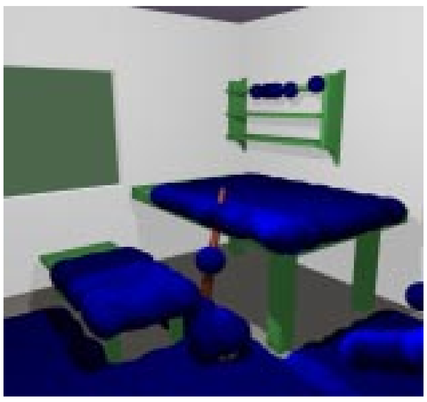



Figure 8b **Figure 8c**
Room covered with blobs. Left image is created using the algorithm proposed in this paper.
Right image is created using a ray/triangle intersection algorithm.

Acknowledgements

We would like to thank Prof. John Hughes for his constructive comments which helped to correct and improve an earlier version of the algorithm. Financial support from the University of Athens (project 70/4/3241) is acknowledged.

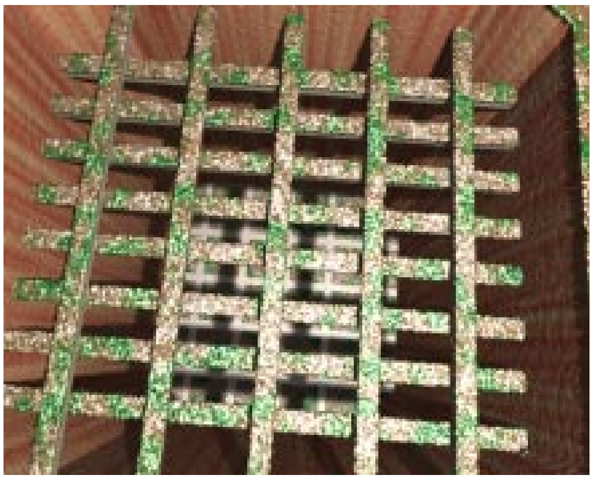


Figure 9. A grid covered with slime

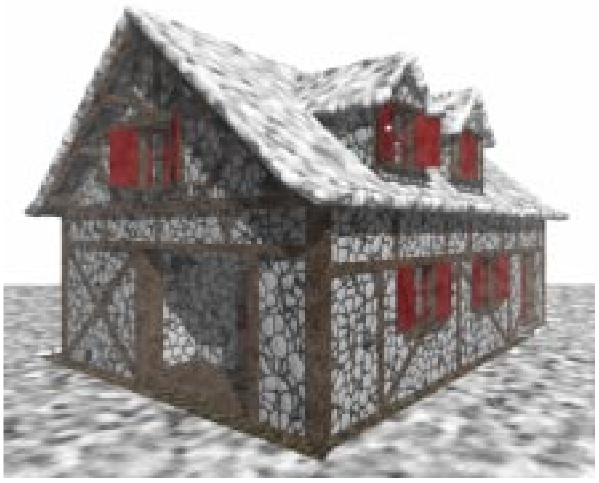


Figure 10. House covered with snow.

References

[Held 97] M. Held. “ERIT - a collection of efficient and reliable intersection tests.” Journal of Graphics Tools, 2(4):25-44 (1997).

[Miller, Pearce 89] G. Miller and A. Pearce. “Globular dynamics – A connected particle system for animating viscous fluids.” *Computer Graphics in Canada*, 13(3):305-309 (1989).

[Ponamgi et al 97] M. Ponamgi, D. Manocha and M. Ling. “Incremental algorithms for collision detection between polygonal models.” *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51-64 (Jan-Mar 1997)

[Reeves 83] W.T. Reeves. “Particle systems – A technique for modeling a class of fuzzy objects.” *Computer Graphics (Proc. SIGGRAPH 83)*, 17(3):359-376 (1983).

[Voorhies, Kirk 91] D. Voorhies and D. Kirk. “Ray-triangle intersection using binary recursive subdivision.” In *Graphic Gems II,* edited by J. Arvo, pp. 257-263. San Diego: Academic Press, 1991.

[Watt 92] A.Watt and M.Watt. “Advanced animation and rendering techniques.” New York: ACM Press, 1992.