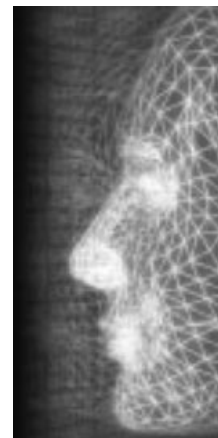# A fast triangle to triangle intersection test for collision detection

Oren Tropp, Ayellet Tal* and Ilan Shimshoni

The triangle-to-triangle intersection test is a basic component of all collision detection data structures and algorithms. This paper presents a fast method for testing whether two triangles embedded in three dimensions intersect. Our technique solves the basic sets of linear equations associated with the problem and exploits the strong relations between these sets to speed up their solution. Moreover, unlike previous techniques, with very little additional cost, the exact intersection coordinates can be determined. Finally, our technique uses general principles that can be applied to similar problems such as rectangle-to-rectangle intersection tests, and generally to problems where several equation sets are strongly related. We show that our algorithm saves about 20% of the mathematical operations used by the best previous triangle-to-triangle intersection algorithm. Our experiments also show that it runs 18.9% faster than the fastest previous algorithm on average for typical scenarios of collision detection (on Pentium 4). Copyright © 2006 John Wiley & Sons, Ltd.

Introduction

Collision detection is a fundamental problem in many disciplines, including computer animation, virtual reality, robotics, computer simulations, solid modeling, computational geometry and molecular modeling.[1–5] Given two objects, in particular—two meshes, the goal is to determine whether they intersect or not.

The naive approach of testing all the primitives (i.e., triangles) of one model against all the primitives of the other, requires an immense number of *triangle-to-triangle* intersection tests. Thus, many algorithms have been devised to reduce this number by using hierarchical data structures. See References 6 and 7 for a good overview.

All these data structures reduce the number of triangle-to-triangle intersection tests considerably. However, at the bottom of the hierarchy, triangle-to-triangle collision tests must still be performed. Moreover, these tests are usually performed in close proximity or collision. Thus, performing these tests rapidly is especially important in worst case scenarios. This paper presents a new algorithm for determining triangle-to-triangle intersection.

The brute force method for determining whether two triangles embedded in three dimensions intersect requires the solution of six sets of linear equations, each corresponding to an intersection of one triangle's edge with the surface of the other triangle. A few faster algorithms, which make use of the line of intersection between the planes of the two triangles, have been suggested.[8–10] In Reference 8, Möller proposes an algorithm that relies on the scalar projections of the triangle's vertices on this line. In Reference 9, Held discusses a technique that first calculates the line segment intersection of one triangle with this line and then checks for intersection between this segment and the other triangle's edges. Both methods have been further developed to achieve faster and more reliable variants. The fastest previous algorithm, which is an improvement of Reference 8, has recently been published by Guigue and Devillers.[10] Their technique relies solely on evaluating the sign of orientation predicates (4×4 determinants) and does not require any intermediate constructions.

While the above algorithms look at the problem from a geometric point of view, our viewpoint is algebraic.

*Correspondence to: A. Tal, Department of Electrical Engineering, Technion—Israel Institute of Technology, Israel.
E-mail: ayellet@ee.technion.ac.il

Our method starts from the linear equations used in the brute force approach. Our key observation is that the set of equations are strongly related to each other. Thus, common elements of the different equations can be re-used to speed up the solution, exploiting the linearity of the matrix operations involved.

Although our algorithm focuses on triangle-to-triangle collision tests, our technique is general and can be applied to other problems of similar nature. Specifically, our technique addresses problems which require the solution of a group of equation sets of type $Ax = b$, where the columns of A and b are linearly dependent in the different sets. For instance, rectangle-to-rectangle intersection test can utilize our method as well.

In the case of triangle-to-triangle intersection, we show that our algorithm performs less arithmetic operations than other known algorithms. In particular, it performs only 95–97 additions, multiplications and comparisons, compared to 114–144 in Reference 10 and 126–148 in the no-division version of Reference 8. Moreover, our algorithm typically runs 18.9% faster than the fastest previous method[10] for a Pentium 4 and 11% for a Celeron. Finally, our algorithm can find the exact intersection coordinates with a much lower cost than previous algorithms.

The rest of the paper is organized as follows. The next section presents the algorithm and analyzes the number of operations it performs in comparison to previous algorithms. The following section describes our experimental results. We conclude in the last section.

Algorithm

Let A and B be two triangles in three dimensions. If A and B intersect, then edges of one triangle intersect the surface of the other. Naively, one could determine intersection by checking separately all possible edge-triangle intersections. Without loss of generality, let p_1 and p_2 be edges of triangle B having a common vertex P and q_i ($1 \leq i \leq 3$) be the edges of triangle A emanating from vertices Q_i (Figure 1). To find the intersection point between the plane defined by p_1 and p_2 and edge q_i, the following set of equations needs to be solved:

$$P + \alpha_1 \times p_1 + \alpha_2 \times p_2 = Q_i + \beta_i \times q_i \qquad (1)$$

In order for the intersection point to reside inside the triangle, the solution to this equation set should satisfy the following constraints: $0 \leq \beta_i \leq 1$, $\alpha_1, \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 \leq 1$.

Obviously, six such intersection tests need to be performed: three to check triangle A against the edges of B, and three where the roles of A and B are reversed. Thus, there are six equation sets.

The key idea of our algorithm is to save arithmetic operations by reusing common elements and making use of the linearity of matrix operations. Moreover, this is done only for three equation sets and the results obtained are used to complete the intersection test differently.

The algorithm starts by partially solving the three sets of equations that determine β_i, $1 \leq i \leq 3$. These sets correspond to the intersections of the three edges of triangle A with the plane of triangle B (Equation 1). The choice of the roles of A and B is arbitrary. However, these sets are not solved completely as only β_i is required at this stage. The values of β_i can lead to a fast rejection (i.e., no intersection). Otherwise, these β_i's are used to construct the line segment of intersection between A and the plane of B ($\vec{t}$ in Figure 1). Obviously, any intersection between the triangles must lie on this
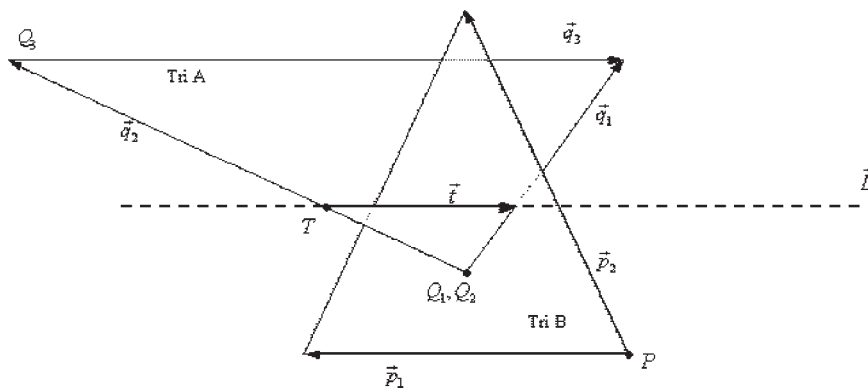

Figure 1. Problem setting.

line. The problem is therefore reduced to a planar intersection test between this segment and triangle B.

More precisely, the algorithm consists of the following five stages. We elaborate below on the way each of these stages is performed.

Algorithm 1 triangle-to-triangle intersection

1. Find the parameters β_i, $1 \leq i \leq 3$ using determinants.
2. If no legal β_i's exist, conclude that there is no intersection and exit.
3. Construct the segment of intersection between triangle A and the plane of B.
4. If this segment intersects triangle B, the triangles intersect.
5. If desired, construct the segment of intersection between the two triangles as linear combinations of the parameters found.

Stage 1

Before we show how to perform the first stage efficiently, let us re-write Equation 1 in its matrix form. Let $r_i = Q_i - P$, then Equation 1 becomes:

$$(p_1|p_2|q_i)\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ -\beta_i \end{pmatrix} = (r_i) \qquad (2)$$

Notice that the two first columns of the 3×3 matrix in Equation 2 are the same for $1 \leq i \leq 3$. Defining the matrix $A(v) = (p_1|p_2|v)$, our equations can be written in the standard form

$$A(q_i)x_i = r_i \qquad (3)$$

where $x = (\alpha_1, \alpha_2, -\beta_i)$.

In order for the intersection point to reside inside the edge q_i itself, the solution to this equation set should satisfy $0 \leq \beta_i \leq 1$. When this condition holds, β_i is called a *legal* β_i.

The equation set is solved using determinants. (An algorithm using Gaussian elimination is also possible but is somewhat less elegant.) That is, β_i is computed by $\beta_i = -\frac{|A(q_i)|}{|A(r_i)|}$. We describe later how to avoid divisions.

The determinants of matrices $A(q_i)$ and $A(r_i)$ are calculated through expansion by minors, by eliminating the third column. In this case, the minors are the same in all the determinants. This is so because p_1 and p_2 are the first two columns in all the matrices. Therefore, the

minors need only be computed once. In addition, for $i = 3$,

$$\begin{aligned} q_3 &= q_2 - q_1 \\ Q_3 &= Q_1 + q_1 \\ r_3 &= P - Q_3 = P - (Q_1 + q_1) = r_1 + q_1 \end{aligned} \qquad (4)$$

Since the process of computing a determinant through a column is linear, the determinants for $i = 3$ can be found from:

$$\begin{aligned} |A(q_3)| &= |A(q_2 - q_1)| = |A(q_2)| - |A(q_1)| \\ |A(r_3)| &= |A(r_1)| + |(A(q_1)| \end{aligned} \qquad (5)$$

Since all the determinants on the right hand side have been computed for $i = 1, 2$, very little work is performed for $i = 3$.

In summary, in Stage 1 only two sets of equations, rather than three, are used. Only one variable is computed for each set. The minors are computed only once and the third equation is solved using trivial linear combinations.

Stage 2

If no legal β_i exists, then all the vertices of triangle A are on the same side of the plane B and thus, the triangles are disjoint. This test is equivalent to half of the fast rejection test of Reference 8–10.

If the determinants of all $A(q_i)$ are zero, then all the edges of both triangles lie on the same plane and a coplanar intersection procedure is called Reference 8. We continue with the common case of non singular matrices.

Stage 3

The goal of this stage is to construct the segment of intersection between triangle A and the plane of B. Since the segment of intersection lies on the line adjacent to both triangle planes, the edges of triangle A cut the plane of B (and the line of intersection) in exactly two points. These two points are given by two legal β_i's and lie at $Q_i + \beta_i \times q_i$.

Consider the segment of intersection between triangle A and the plane of B. Denote T the point of intersection between an edge of triangle A and the plane of B and $\overrightarrow{t}$ the vector connecting T to the second point of intersection (Figure 1). $\overrightarrow{t}$ and T can be constructed as linear combinations of the two legal β_i's, q_i's, and Q_i's. For

(a) $\vec{t}$ intersects an edge of B at X_1

(b) $\vec{t}$ is fully contained in B.
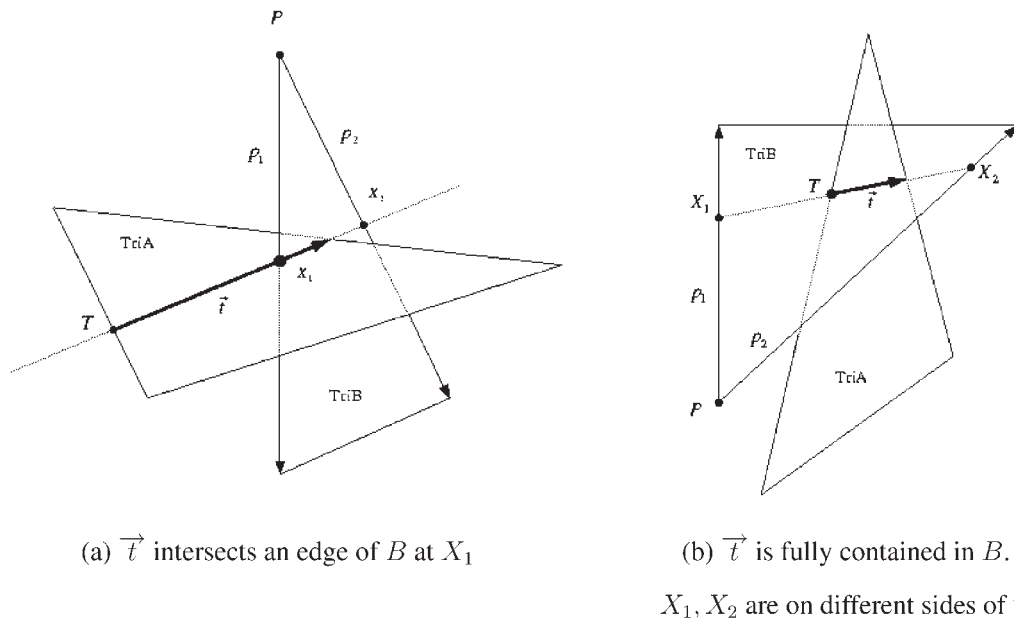
X_1, X_2 are on different sides of T

Figure 2. Intersection configurations.

example, if β_1 and β_2 are legal, then T and $\vec{t}$ are given by:

$$T = Q_1 + \beta_1 q_1$$
$$\vec{t} = \beta_2 q_2 - \beta_1 q_1 \qquad (6)$$

Stage 4

There are two possibilities for an intersection between the two triangles. Either the line segment $(T, T + \vec{t})$ intersects at least one of the three edges of B (Figure 2(a)) or it is completely contained in B (Figure 2(b)). In order to find the intersections between the line segment and the edges of triangle B the following three equations sets need to be solved:

$$P + \delta_1 p_1 = T + \gamma_1 t$$
$$P + \delta_2 p_2 = T + \gamma_2 t \qquad (7)$$
$$(P + p_1) + \delta_3(p_2 - p_1) = T + \gamma_3 t$$

The third Equation is for edge $p_3 = p_2 - p_1$ which originates from vertex $P + p_1$ (Figure 1). Observe that though the vectors in Equation 7 are 3×1, they all lie on the same plane and thus 2×2 equations sets are solved. Applying determinants to solve the problem and exploiting the linearity of the determinants, computations

can be saved in a similar fashion to the technique described in Equation 5.

Solving these sets of equations, it should now be verified whether the solutions are legal. We first check whether $\vec{t}$ intersects one of the edges of B. A solution to a set of equations finds the point of intersection along the lines defined by $\vec{t}$ and p_i. The intersection lies within the edge p_i only if the solution satisfies $0 \leq \delta_i \leq 1$. Moreover, this intersection lies within the line segment $\vec{t}$, only if $0 \leq \gamma_i \leq 1$. Only when both conditions are satisfied, it can be concluded that $\vec{t}$ intersects p_i. Let us define the point of intersection between the lines of $\vec{t}$ and p_i as X_i. It follows that X_i is given by

$$X_i = P_i + \delta_i p_i \qquad (8)$$

where P_i is the origin vertex of edge p_i. For example, in Figure 2(a) X_1 is a valid intersection of $\vec{t}$ with p_1, while X_2, the intersection of $\vec{t}$ with p_2, is illegal because it occurs outside of the segment $[T, T + \vec{t}]$ ($\gamma_2 > 1$).

It remains to check the second case where $\vec{t}$ is fully contained within triangle B (Figure 2(b)). This case occurs when the edges of triangle B intersect the line defined by $\vec{t}$ on both sides of T. Hence, it is detected by checking whether the last stage has found two legal δ_i's having γ_i's with different signs. For example, in Figure 2(b), X_1 and X_2 are on both sides of segment $\vec{t}$ and the segment of intersection is fully contained in B.

Stage 5

Finally, the optional last stage of the algorithm finds the exact points of intersection from linear combinations of the vectors and the parameters that have already been computed. The end-points of the segment of intersection between the two triangles lie along the segment defined by T and $\vec{t}$. In Figure 2(b), the points T and $T + \vec{t}$ are themselves the end points of the desired segment of intersection and in Figure 2(a) X_1 and $T + \vec{t}$ are the two desired endpoints. Generally, the two endpoints are a subset of $\{T, T + \vec{t}, X_i\}$, and can be computed from the linear combinations described in Equations 6 and 8.

Avoiding Divisions

Divisions are typically considered to be 4–8 times more expensive than other arithmetic operations.[11] We describe below how divisions are avoided in our algorithm.

The algorithm, as described, obtains β_i by $\beta_i = -\frac{|A(q_i)|}{|A(r_i)|}$ where the minus sign results from Equation 2. In order for the intersection to be within edge q_i, β_i should satisfy $0 \le \beta_i \le 1$. This condition is mathematically equivalent to the condition $0 \le \beta_i \cdot |A(r_i)|^2 \le |A(r_i)|^2$. Divisions can be avoided by checking directly whether $0 \le -|A(q_i)| \cdot |A(r_i)| \le |A(r_i)|^2$.

A similar trick of checking the constraint on the product rather than on the quotient is used when checking the constraints on γ_i and δ_i in Stage 3 of the algorithm.

β_i is used for calculating T and $\vec{t}$. If β_i is legal, then edge q_i cuts the other triangle's plane at

$$T = Q_i + \beta_i q_i. \qquad (9)$$

Since $\beta_i \cdot |A(r_i)|^2$ is computed, rather than β_i, all the other elements need to be multiplied by the same scaling factor $|A(r_i)|^2$. Thus instead of Equation 9 we get:

$$|A(r_i)|^2 T = |A(r_i)|^2 Q_i + (|A(r_i)|^2 \beta_i) \cdot q_i. \qquad (10)$$

The result is that in the last stage of the algorithm, the matrices are multiplied by a known constant s. Since $Ax = b$ and $sAx = sb$ have the same solution, the solution of this scaled set of equations is a valid solution to the original set.

If the exact points of intersection are necessary, they are obtained by linear combinations. In this case two divisions are necessary to take care of the above scaling factor.

Arithmetic Operations

Our triangle-to-triangle intersection test algorithm performs 95–97 additions, multiplications and comparisons and does not perform any division. This is compared to 114–144 operations in Reference 10 and 126–148 in the no division version of Möller[8] (where absolute values are counted as well). Table 1 summarizes the number of arithmetic operations performed by our algorithm and by other state-of-art algorithms.[8,9,10,12]

It is common to consider the time to compute multiplications, additions/subtractions and comparisons as equal, since today's computers perform all these operations in roughly the same time. Divisions, however, are more expensive and are estimated to take as much as 4–8 times more than the other arithmetic operations.[11]

Experimental Results

Since branching decisions in the different algorithms greatly change the number of operations actually used during run time, Table 1 does not directly predict the efficiency of the algorithms. It is therefore necessary to run the different algorithms on typical scenarios and measure their running times. In this case, not only the effects of the different branchings in the codes are taken into account, but also computer-dependent effects such as pipelining and optimizations.

Typical scenarios mean that triangles should not be generated randomly. After all, most systems use

Algorithm	$+/-$	MUL	CMP	DIV	ABS	$=$
Held[9]	74/94	35/45	33/50	I	3	5I
Möller—no div[11]	54	57	12/28		3/9	69/75
Guigue et al.[10]	62/76	43/52	9/16	—	—	42/62
Ours	26/27	56/57	13	—	—	3I/35

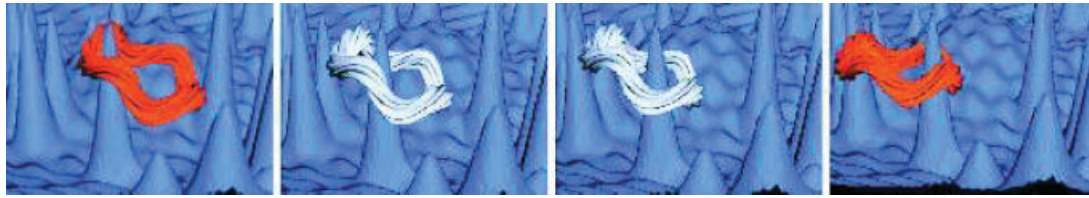**Table I. Comparison of arithmetic operations—A Summary**

Figure 3. The Torus model.

triangle-to-triangle intersection tests within hierarchical data structures. Therefore, our goal is to apply and compare these tests within a common data structure.

We chose to use OBBTrees[4] as our data structure and RAPID as our software package.[13] This is because of the availability of RAPID and because it is widely used. It is important to notice that for our own algorithm we slightly changed the representation of data in RAPID. While the original version maintains three vertices for each triangle, our algorithm maintains one vertex and two edges. The two representations require the same storage space and are easily interchangeable.

Within RAPID, each time a triangle-to-triangle test was called, the six triangle vertices were collected into a file. We then ran the different triangle-to-triangle algorithms on this set of triangles 10^5 times and running times were measured.

The models on which we ran our tests are a torus model moving on a spiky surface (Figure 3) and a pipes model tumbling within a version of itself enlarged fifteen times (Figure 4). Both models have been used before to evaluate collision detection algorithms.[4] The running times were measured for several paths of these complex and irregular models. Thus, the collection of positions and orientations used is general and represents real-life situations.

The experiments were run on two configurations: Pentium 4, 1.8GHz, Windows 2000 and the Microsoft compiler and a Celeron 1.2Ghz, Windows XP, and the Microsoft compiler. Double precision was used for the variables.

Tables 2 and 3 show the results for *path2* for the torus and *path1* for the pipes, both available online and described in.[13] As can be clearly seen from the tables, the best previous algorithm is [10], which relies on orientation predicates. Note that our algorithm outperforms [10] for all scenarios. For the Pentium 4 processor the average improvement is 18.9% and for the Celeron processor 12.6%.

It is interesting to note that our algorithm is faster both for tests that result in intersections and for tests that result in separations. However, its advantage is typically larger for intersections (except for the Torus model on the Pentium).

This difference between intersection and separation stems from a fundamental distinction between the algorithms. The algorithm of [10] tries to find a separation and decides on intersection only if all separation attempts
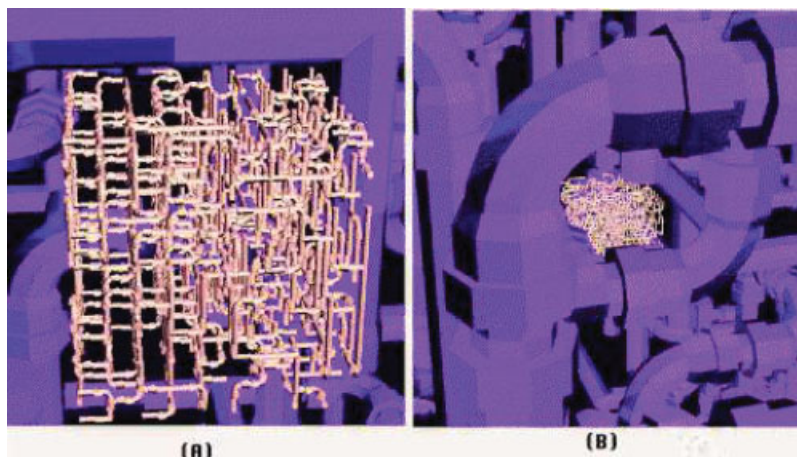

Figure 4. The Pipes model.

Torus model					
	19566 tests		14244 misses		5322 intersections
Algorithm	Average test time		Average Miss time		Average Intersect time
Pentium 4					
Möller-NoDiv[14]	0.24 μs		0.214 μs		0.307 μs
Guigue *et al.*[10]	0.238 μs		0.229 μs		0.264 μs
Shen *et al.*[12]	0.332 μs		0.312 μs		0.387 μs
Ours	0.181 μs		0.169 μs		0.214 μs
Celeron					
Möller-NoDiv[14]	0.588 μs		0.56 μs		0.662 μs
Guigue *et al.*[10]	0.54 μs		0.522 μs		0.59 μs
Shen *et al.*[12]	0.69 μs		0.66 μs		0.77 μs
Ours	0.474 μs		0.458 μs		0.519 μs

Table 2. Comparison of timings for different triangle-to-triangle collision tests—Torus model

fail, which is the worst-case scenario for this algorithm. Our algorithm, though, can exit earlier than the worst-case scenario both for intersection and for separation. This makes our algorithm more attractive in situations where there are many intersections.

Degeneracies. Below we evaluate the performance of the different algorithms for a couple of degenerate cases: a vertex of one triangle is adjacent to the plane of the other, and an edge one triangle is adjacent to the plane of the other. These cases become harder when there is an intersection or almost an intersection.

In our experiments, we generated 10^6 triangle pairs within the unit cube for each degenerate case. We then added to the coplanar vertices a vector in the direction of the normal of the plane of the second triangle, multiplied by a value distributed uniformly in the range $\pm\varepsilon$. Our choice of $\varepsilon = 10^{-14}$ makes sure that all the algorithms still make mistakes for a noise of $\pm\varepsilon$, but that the error rate is significantly below 50%. The sign of ε determined the correct answer. The results are shown in Table 4.

Our algorithm is slightly less robust than the other algorithms for the case of coplanar edges. This is due to

Pipes model					
	97873 tests		63124 misses		34749 intersections
Algorithm	Average test time		Average Miss time		Average Intersect time
Pentium 4					
Möller-NoDiv[14]	0.414 μs		0.392 μs		0.453 μs
Guigue *et al.*[10]	0.281 μs		0.268 μs		0.304 μs
Shen *et al.*[12]	0.402 μs		0.381 μs		0.442 μs
Ours	0.242 μs		0.236 μs		0.252 μs
Celeron					
Möller-NoDiv[14]	0.749 μs		0.724 μs		0.795 μs
Guigue *et al.*[10]	0.603 μs		0.579 μs		0.647 μs
Shen *et al.*[12]	0.799 μs		0.768 μs		0.856 μs
Ours	0.567 μs		0.559 μs		0.582 μs

Table 3. Comparison of timings for different triangle-to-triangle collision tests—Pipes model

Algorithm	vertex on plane	Edge on plane
Möller-NoDiv[11]	26.7%	14.1%
Guigue and Devillers[10]	24.3%	16.4%
Shen et al.[12]	26.7%	14.9%
Ours	26.6%	18.8%

Table 4. Degenerate cases error rate

the fact that our algorithm is basically a construction method. Results from the first stages of the algorithm are the input to the last stage, so numerical errors accumulate. However, note that the few percents difference between the error rates is caused by a noise of $\varepsilon = 10^{-14}$. This level of accuracy is completely sufficient within normal collision detection scenarios.

Conclusion

We have presented a method for determining the intersection between two triangles in three-dimensional space, and if desired, finding the exact segment of intersection with a minimal additional cost. Unlike previous algorithms which are based on geometrical methods, our viewpoint is algebraic.

Through the use of the linearity of the arithmetic operations on the matrix defining the problem and the linear relations between the columns of this matrix, the process of detecting the intersection is greatly accelerated. The principle of accelerating the solution of strongly related equation sets is novel and can be applied to other problems in geometry, such as 3D polygon-polygon intersection test.

Our algorithm performs roughly 20% less arithmetic operations than the fastest previous algorithm. It also runs 18.9% faster on Pentium 4 and 12.6% faster on Celeron compared to the fastest previous method. In fact, the algorithm usually excels in the case of intersection. This suggests that as hierarchical data structures keep improving by bounding the geometric primitives more tightly, our algorithm will become more advantageous, because a larger percentage of the calls will result in intersection.

This advantage of the algorithm is also its drawback. For applications other than collision detection, where the triangles are in general position and not necessarily in close proximity, it is preferable to use an algorithm with a fast rejection test. Another drawback of our algorithm is that it has a higher error rate in degenerate cases, though the difference is marginal.

References

1. Barequet G, Chazelle B, Guibas LJ, Mitchell J, Tal A. BOX-TREE: a hierarchical representation for surfaces in 3D. In *Proceedings of Eurographics*, 1996; pp. 387–396.
2. Cohen J, Lin M, Manocha D, Ponamgi K. I-COLLIDE: an interactive and exact collision detection system for large-scaled environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*; pp. 189–196.
3. Dobkin DP, Kirkpatrick DG. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 1983; **27**: 241–253.
4. Gottschalk S, Lin MC, Manocha D. OBBTree: a hierarchical structure for rapid interference detection. *SIGGRAPH'96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques ACM SIGGRAPH* 1996; **30**: 171–180.
5. Klosowski JT, Held M, Mitchell JSB. Evaluation of collision detection methods for virtual reality fly-throughs. *The 7th Canad. Conf. Computat. Geometry* 1995; **14**: 36–43(2): 205–210.
6. Gottschalk S, Lin M. Collision detection between geometric models: a survey. In *Proceedings of IMA Conference on Mathematics of Surfaces* 1998; pp. 3–15.
7. Floriani De L, Puppo E, Magillo P. *Applications of Computational Geometry to Geographic Information Systems*, chapter 7, Elsevier Science & Technology: 1999; 333–388.
8. Möller T. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 1997; **2**(2): 25–30.
9. Held M. ERIT a collection of efficient and reliable intersection tests. *Journal of Graphics Tools* 1997; **2**(4): 25–44.
10. Guigue P, Devillers O. Fast and robust triangle-triangle overlap test using orientation predicates. *Journal of Graphics Tools* 2003; **8**(1): 25–42.
11. Devillers O, Guigue P. Faster triangle-triangle intersection tests. *Technical Report 4488, INRIA*, 2002.
12. Shen H, Heng PA, Tang Z. A fast triangle-triangle overlap test using signed distances. *Journal of Graphics Tools* 2003; **8**(1): 3–15.
13. http://www.cs.unc.edu/?geom/obb/obbt.html.

Authors' biographies:

Oren Tropp is an MSc student at the department of Electrical Engineering, Technion. He received the BSc degree in Electrical Engineering (Summa cum Laude) from the department of Electrical Engineering, Technion in 2001.

Ayellet Tal is a faculty member at the department of Electrical Engineering, Technion. She holds a PhD in Computer Science from Princeton University, an MSc degree (Summa cum Laude) in Computer Science from Tel-Aviv University and a BSc degree (Summa cum Laude) in Mathematics and Computer Science from Tel-Aviv University.

Ilan Shimshoni is a faculty member of the Department of Management Information Systems at Haifa University. He holds a PhD in computer science from the University of Illinois at Urbana Champaign, an MSc in computer science from the Weizmann Institute of Science and a BSc in Mathematics and Computer Science from the Hebrew University in Jerusalem. His research interests are in the fields of Computer Vision, Robotics, and Computer Graphics specializing mainly in applications of statistical methods in these fields.