INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Faster Triangle-Triangle Intersection Tests

Olivier Devillers — Philippe Guigue

N° 4488

Juin 2002

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4488--FR+ENG

Faster Triangle-Triangle Intersection Tests

Olivier Devillers , Philippe Guigue

Thème 2 — Génie logiciel
et calcul symbolique
Projets Prisme

Rapport de recherche n° 4488 — Juin 2002 — 17 pages

Abstract: This paper presents a new method for computing whether or not two triangles in three dimensions intersect. The code is very efficient and requires minimum arithmetic precision. Indeed, all branching decisions are carried out by evaluating the signs of degree three polynomials. In addition, an efficient test is proposed for the two-dimensional case.

Key-words: geometric predicates, low degree predicate, collision detection

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Tests Rapides d'Intersection de Triangles

Résumé : Ce document présente une nouvelle méthode pour déterminer si deux triangles dans l'espace s'intersectent. Le code est trés efficace et nécessite une précision arithmétique minimale. Toutes les décisions de branchement consistent, en effet, à évaluer le signe de polynômes de degré trois. En outre, un test efficace est proposé pour le cas planaire.

Mots-clés : prédicats géométriques, prédicat de bas degré, détection de collision

Contents

1 Introduction

Robustness issues in geometric algorithms have been widely studied in the recent years [9]. It appears that a good solution consists in a design that strictly separates the geometric tests (also called *predicates*) from the combinatorial part of the algorithm. In that way, the robustness issues due to numerical accuracies are isolated in the predicates and can be solved using relevant arithmetic tools.

From this point of view, a predicate is a piece of code which answers a particular basic geometric question. This code usually has two independent aspects, the first aspect is an algebraic formulation of the problem, most often, an algebraic expression whose sign gives the answer to the predicate, the second aspect is the arithmetic used to evaluate the value of that expression, it can be rounded arithmetic which is fast but inexact, or a different kind of arithmetic which allows to certify the exactness of the result.

Several quantities can be used to evaluate the quality of a predicate: the degree of the algebraic expression, the number of operations or the running time. The arithmetic degree of the predicate [5] is the highest degree of the polynomials that appear in its algebraic description. A low degree predicate has two advantages : used with an approximate arithmetic it has a better precision and used with an exact arithmetic less bits are required to represent numbers.

In this paper, we propose a new formulation of the predicate of intersection of two triangles in three dimensions (resp. in two dimensions). This formulation can be decomposed in a small number of three dimensional (resp. two dimensional) orientation tests and thus has only degree three (resp. degree two). This is a big improvement compared to the most popular implementation of this predicate due to Möller [6] and Held [4]. Using floating point computation, this new formulation improves the running time by around 20% while improving the stability of the algorithm. Using exact computation, it is in most cases only slightly faster but it can benefit from speedup techniques to minimize the overhead of exact arithmetic.

The structure of this paper is as follows: in the next section we introduce notations and describe the algorithms proposed by Möller and Held. Section 3 presents the new formulation of the three-dimensional predicate and compares the performance of the three algorithms. Finally, in section 4, the two-dimensional case is described.

2 Notations and Related work

2.1 Definitions and Notations

Definition 2.1 *Given two-dimensional points $a = (a_x, a_y)$, $b = (b_x, b_y)$, and $c = (c_x, c_y)$, we define the determinant*

$$[a,b,c] \;:=\; \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix}.$$

It gives twice the signed area of the triangle abc and is strictly positive if and only if a, b, c appear in counterclockwise order on the boundary of abc. In other words, $[a, b, c]$ is positive if the triplet a, b, c forms a left turn (i.e. c is strictly to the left of the line directed from a to b), negative if it forms a right turn and zero if the points are aligned (see Figure 1.a).

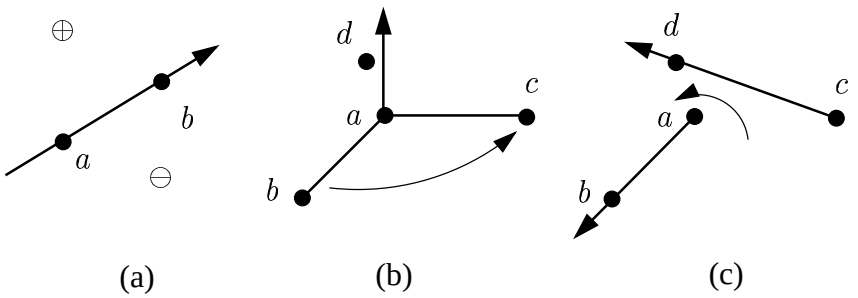

Figure 1: (a) : $[a, b, c]$ is positive if the triplet a, b, c forms a left turn. (b) and (c) : $[a, b, c, d]$ is positive according to a right-hand rule for planes or for lines.

Definition 2.2 *Given four three-dimensional points* $a = (a_x, a_y, a_z)$, $b = (b_x, b_y, b_z)$, $c = (c_x, c_y, c_z)$, *and* $d = (d_x, d_y, d_z)$, *we define the determinant*

$$[a, b, c, d] := \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix}$$

$$= \begin{vmatrix} a_x - d_x & a_y - d_y & a_z - d_z \\ b_x - d_x & b_y - d_y & b_z - d_z \\ c_x - d_x & c_y - d_y & c_z - d_z \end{vmatrix}.$$

The sign of $[a, b, c, d]$ has two geometric interpretations, each corresponding to a right-hand rule. It tells whether vertex d is above, below, or on a plane through a, b, and c, where *above* is the direction of a right-handed screw at a that turns from b toward c. Equivalently, it tells whether a screw directed along the ray $\overrightarrow{ab}$ turns in the direction of $\overrightarrow{cd}$ (see Figure 1.b and 1.c). In either interpretation, the result is zero iff the four points are coplanar.

In the sequel, T_1 and T_2 denote triangles with vertices p_1, q_1, and r_1, and p_2, q_2, r_2 respectively. π_1 and π_2 denote their respective supporting planes (see Figure 2). The three-dimensional triangle-triangle intersection test described in the sequel returns a boolean value which is true if the closed triangles (i.e. the triangles including their boundary) intersect. We also describe a variant of the algorithm that returns a symbolic description of the intersection.

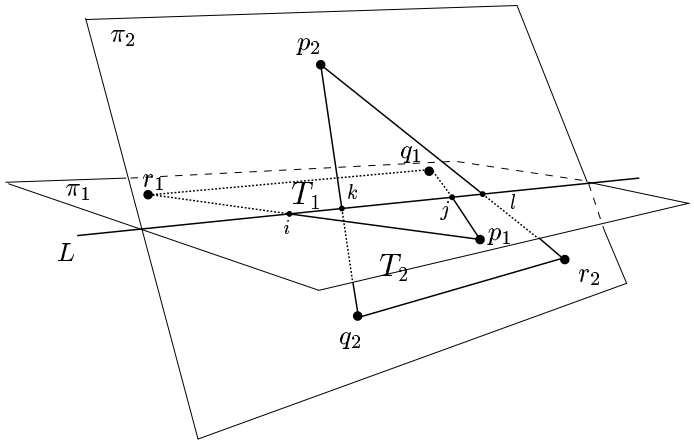

Figure 2: Triangles and the planes in which they lie.

2.2 Related work

Several solutions exist to test the intersection between three-dimensional triangles, although the most important ones, due to their efficiency, are the algorithms proposed by Möller [6] and Held [4]. Both are based on a constructive multistep process. In the following, we describe these two approaches.

2.2.1 General remarks

Each triangle is a subset of the plane it lies in, so for two triangles to intersect they must overlap along the line of intersection of their planes. Hence, a necessary condition for intersection is that each triangle must intersect the plane of the other. Based on this remark, quite some configurations of non intersecting triangle pairs can be quickly detected by simply checking if one triangle lies entirely in one of the open halfspaces induced by the supporting plane of the other.

2.2.2 Möller's algorithm

Möller's method begins by checking the mutual intersection of each triangle with the plane of the other. To do so, it determines for each triangle on which side of the other triangle 's supporting plane its vertices lie. Now, if all vertices of one triangle lie on the same side and no vertex is on the plane, the intersection is rejected. Otherwise, the input triangles are guaranteed to intersect the line of intersection of the two planes. Furthermore, these intersections form intervals on this line, and the triangles overlap iff these intervals overlap as well. Hence, the last part of the algorithm computes a parametric equation $L(t)$ of the line of intersection of the two planes, finds the intervals (i.e. scalar intervals on $L(t)$) for which the line lies inside each triangle and performs a one-dimensional interval overlap test (see. Figure 3).

2.2.3 Held's algorithm

Held's code begins similarly by determining on which side of the supporting plane π_1 the vertices p_2, q_2, r_2 lie. If T_2's vertices do not all lie in one of the closed halfspaces induced by π_1 the second part directly constructs the line segment $s = T_2 \cap \pi_1$ and the problem is reduced to testing whether the segment s intersects T_1. The constructed segment is coplanar with T_1 so this intersection is solved by a a two-dimensional triangle/line-segment test after projecting to a convenient plane (see Figure 4.).

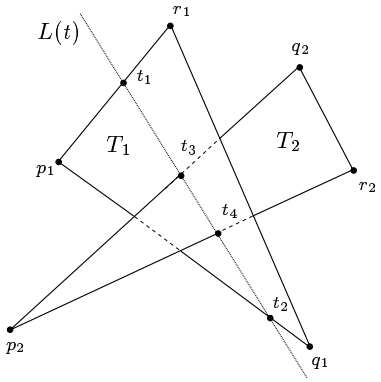

Figure 3: Möller's algorithm computes interval bounds (parameter values) on L and tests if the intervals overlap.

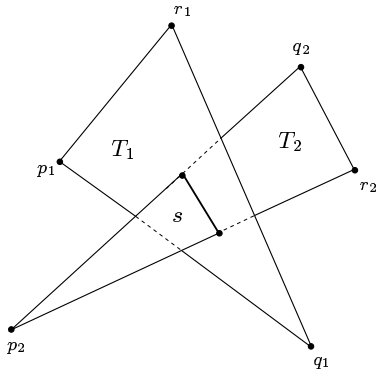

Figure 4: Held's algorithm constructs $s = T_2 \cap \pi_1$, and tests if $s \cap T_1$.

Despite their efficiency, these multistep processes have two disavantages. First, degenerate cases at each step of the process need to be handled specifically. Second, using the derived values from the initial constructions increases the required arithmetic precision and results on a prone to error implementation. The triangle intersection test presented in this paper does not need any intermediate construction but relies exclusively on orientation predicates. In consequence, its implementation has only degree three and is more reliable than Möller and Held code. Section 3.2 shows that it is also more efficient.

3 Three dimensional Triangle-Triangle Intersection Test

3.1 Description

To begin, we give an overview of our predicate: each vertex of each triangle is classified with respect to the other triangle using six orientation predicates. Then, the vertices are permuted in some canonical form and the conclusion is reached by two further orientation predicates. The special case

of two coplanar triangles is handled as the two dimensional case described in the next section. We now detail the algorithm.

Like Möller, we begin by checking the mutual intersection of each triangle with the plane of the other. Triangle T_1 is first tested for intersection with the plane π_2. To do so, the algorithm classifies the vertices of T_1 with respect to the plane π_2 by simply comparing the signs of the three determinants $[p_2, q_2, r_2, p_1]$, $[p_2, q_2, r_2, q_1]$ and $[p_2, q_2, r_2, r_1]$. Three distinct situations are possible: a) all three determinants evaluate to the same sign and no determinant equals zero, b) all three determinants equal zero, and c) the determinants have different signs. Case a) occurs when all three vertices of T_1 lie on the same open halfspace induced by π_2. In this case no further computation is needed since there can be no intersection. Case b) occurs when both triangles are coplanar and define the same plane. This special case is solved by a two-dimensional triangle-triangle intersection test (cf. section 4) after projecting the three-dimensional input vertices to a convenient plane. Finally, case c) happens when the vertices of T_1 lie on different sides of the plane π_2 and T_1 surely intersects π_2. In this case, the algorithm then checks in the next step whether T_2 intersects π_1 in a similar manner.

Indeed, if the input triangle pair passes these tests and if it is assumed to be in general position (i.e. if no vertex of one triangle is coplanar with the other triangle), each triangle has exactly one of its vertices lying on one side of the other triangle's supporting plane with the two other vertices on the other side. The algorithm then applies a circular permutation to the vertices of each triangle such that p_1 (resp. p_2) is the only vertex of its triangle that lies on its side. An additional transposition operation (i.e. a swap operation) is performed at the same time on vertices q_2 and r_2 (resp. q_1 and r_1) to map vertex p_1 (resp. p_2) on the positive open halfspace induced by π_2 (resp. π_1). Notice that such a reordering exists for all possible configurations except when two of the triangle's vertices lie on one side and the other is on the plane. Such cases are handled by permuting the vertices of the triangle such that p_1 (resp. p_2) is the only vertex of its triangle that does not lie on the negative open halfspace induced by π_2 (resp. π_1).

Due to our previous rejections and permutations, each incident edge of vertices p_1 and p_2 is now guaranteed to intersect L at a unique point. Let i, j, k, and l, be the intersection points of L with edges p_1r_1, p_1q_1, p_2q_2, and p_2r_2 respectively. These intersection points form closed intervals I_1 and I_2 on L that correspond to the intersection between the two input triangles and L. Furthermore, at this step, there is enough information to know a consistent order of the bounds of each interval. Precisely, $I_1 = [i, j]$ and $I_2 = [k, l]$ if L is oriented in the direction of $N = N_1 \times N_2$ where $N_1 = (q_1 - p_1) \times (r_1 - p_1)$ and $N_2 = (q_2 - p_2) \times (r_2 - p_2)$. Thus, it is only necessary to check a min/max condition to determine whether or not the two intervals overlap. By min/max condition, we mean that the minimum of either interval must be smaller than the maximum of the other, namely, $k \leq j$ and $i \leq l$.

Since the edges p_1r_1, p_1q_1, p_2q_2, and p_2r_2 intersect L, this condition reduces to check the predicate

$$[p_1, q_1, p_2, q_2] \leq 0 \ \wedge [p_1, r_1, r_2, p_2] \leq 0. \tag{1}$$

This can be easily seen by looking at Figure 2 and recalling the second interpretation of the three-dimensional orientation test. Indeed, this conjunction amounts to test whether a screw directed along

the ray $\overrightarrow{p_1q_1}$ (resp. $\overrightarrow{p_1r_1}$) turns in the direction of $\overrightarrow{q_2p_2}$ (resp. $\overrightarrow{p_2r_2}$).

To summarize, the steps of the entire algorithm are as follows. The first step determines the relative position of T_1's vertices w.r.t T_2's supporting plane and trivially rejects the intersection if all T_1's vertices lie on the same side of π_2. The second step does the same with T_2's vertices and π_1. The third step compares all signs to determine a convenient permutation of the vertices of each triangle and checks successively the two inequalities of conjunction (1).

Getting a symbolic description of the intersection
If one is interested in a symbolic description of the actual intersection it can be done either by replacing the two last orientation tests by three other tests, or by performing two more orientation tests at the end (cf. Figure 5).

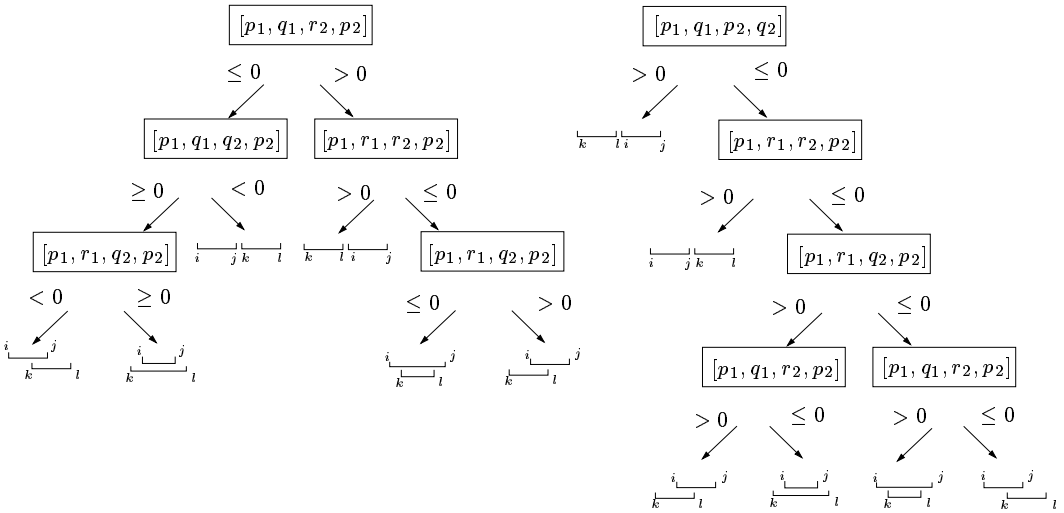

Figure 5: Symbolic description of the intersection.

3.2 Analysis

In this section, we give the arithmetic degrees and the number of arithmetic operations of the different algorithms. Then, we perform benchmarks for different kinds of data in general and degenerate position. The benchmarks are performed with exact and rounded arithmetics. For rounded arithmetic we count the number of wrong answers due to rounding.

The algorithm has been implemented using C++ language. The code does not handle degenerate triangles (i.e., line segments and points). However, if one whishes to handle those cases, they must be detected (all tests involving a degenerate triangle equal zero) and handled as special cases. Note

that if triangles are assumed to be non degenerate, coplanarity of the triangles can be detected as soon as the first three orientation tests are computed (all three signs equal zero).

The computation of the determinants at step 1. (resp. at step 2) can be optimized by computing the invariant parts consisting of 2×2 sub-determinants only once. For $[a, b, c, X]$, expansion by 2×2 sub-determinants gives the normal vector of the plane passing through a, b, and c, and reduces the determinants to a dot product of 3-tuples.

General case						
Code	ADD	MUL	CMP	DIV	ABS	=
Held						
$T_2 \cap \pi_1$?	24	15	13/21			21
$(T_2 \cap \pi_1) \cap T_2$?	50/70	20/30	20/29	1	3	30
	74/94	35/45	33/50	1	3	51
Möller						
$T_1 \cap \pi_2$?	21	20	2/5		0/3	15/18
$T_2 \cap \pi_1$?	21	20	2/5		0/3	15/18
$I_1 \cap I_2$?	15	10/12	10/20	4	3	16/26
	57	50/52	14/30	4	3/9	46/62
Möller no div.						
$T_1 \cap \pi_2$?	21	20	2/5		0/3	15/18
$T_2 \cap \pi_1$?	21	20	2/5		0/3	15/18
$I_1 \cap I_2$?	12	17	8/18		3	39
	54	57	12/28		3/9	69/75
Our code						
$T_1 \cap \pi_2$?	24	17	2			21
$T_2 \cap \pi_1$?	24	17	2			21
$T_1 \cap T_2$?	14/28	9/18	5/12			0/20
	62/76	43/52	9/16	0	0	42/62

Table 1: min/max number of operations of each types in each section of code for the general case.

We first compare our code in terms of arithmetic requirements to the two fastest existing methods due to Möller and Held[1] In our algorithm, all branching decisions are sign evaluation of degree three polynomial. Therefore, the algorithm has degree three. In Möller's approach, To compare this against Möller approach, we could parametrize the line of intersection of the two supporting planes as $L(t) = O + tD$, where D is the cross-product of the normals to the plane and O is some point on it, and solve for the values of t (i.e scalar intervals on L) that correspond to the intersection between the triangles and L. The algorithm then has to compare these t values that are rational polynomial with numerator of degree eight and denominator of degree three. Held's algorithm computes the line segment of intersection of T_2 with the plane π_1 by evaluating rational polynomial with numerator four and denominator of degree three. It then solves a two-dimensional segment-triangle that involves degree two polynomials in the input variables. Hence, both algorithms have degree eight. Consequently, our algorithm allows to limit the arithmetic requirements of a robust implementation. When operations are implemented in IEEE754 floating point arithmetic (as originally done in Möller and Held codes), then fewer bits are lost due to rounding and all branching decisions are proved to

[1]Möller's implementation was taken from the web site [7], the code for ERIT version 1.1. was obtained from its author.

be correct with high probability [3].

While the emphasis of this algorithm is on reducing arithmetic requirements, it can be also proved to be computationally efficient. We now evaluate our code in terms of number of operations and running times. Each code is divided into discrete parts according to the branching structure of its rejection steps. For the purpose of this evaluation, the operations within each code are divided into five types: addition/substraction, multiplication, comparisons, divisions, absolute values and assignments (see Table 1). Most simple FP listed operations (except division) take a fixed number of cycles. Floating-point division tie up the pipeline for a significantly longer time and can take from four to eight times as long as the other operations.

	IEEE double arithmetic		IEEE double arithmetic + ϵ check		GMP rational
Code	Execution times in micro-seconds	Percentage of wrong answers	Execution times in micro-seconds	Percentage of wrong answers	Execution times in micro-seconds
Pseudo Random intersecting triangle pairs					
Our algo	0.343	ϵ.	0.398	ϵ.	771.535
Möller	0.488	ϵ.	0.541	ϵ.	760.247
Möller (no div)	0.414	ϵ.	0.468	ϵ.	858.496
Held	0.471	ϵ.	0.484	ϵ.	772.960
Pseudo Random non intersecting triangle pairs					
Our algo	0.192	ϵ.	0.235	ϵ.	425.421
Möller	0.248	ϵ.	0.288	ϵ.	472.976
Möller (no div)	0.229	ϵ.	0.270	ϵ.	516.255
Held	0.304	ϵ.	0.300	ϵ.	500.388
Degenerate intersecting triangle pairs					
Our algo	0.338	7.0300	0.396	7.0300	819.960
Möller	0.483	17.8588	0.550	17.8592	867.540
Möller (no div)	0.411	20.6421	0.476	20.6427	967.800
Held	0.444	30.6704	0.440	30.6704	986.780
Degenerate nearly intersecting triangle pairs					
Our algo	0.326	19.6519	0.380	19.6519	740.340
Möller	0.481	50.5537	0.543	50.5538	862.710
Möller (no div)	0.406	50.0739	0.470	50.0739	960.000
Held	0.445	16.0344	0.440	16.0344	991.100
Intersecting triangle pairs with coplanar vertex					
Our algo	0.335	7.5640	0.421	ϵ.	815.530
Möller	0.427	14.3534	0.507	ϵ.	867.420
Möller (no div)	0.376	14.3534	0.482	ϵ.	968.600
Held	0.165	100.-ϵ	0.169	100.-ϵ	316.550
Nearly intersecting triangle pairs with coplanar vertex					
Our algo	0.196	1.0769	0.275	3.9650	448.460
Möller	0.266	1.8233	0.396	3.9648	521.590
Möller (no div)	0.248	1.8233	0.375	3.9648	565.940
Held	0.168	0.0002	0.167	0.0002	316.420

Table 2: Running times of the algorithms. ϵ denotes a statistically infinitesimal magnitude.

Running times obtained on *pseudo-random*, *degenerate* and *coplanar vertex* input are presented in Table 2. For *pseudo-random* input, we use points whose coordinate values are chosen pseudo-

randomly. The *degenerate* input are triangle pairs with grazing contacts obtained by generating two coplanar edges that intersect and two points on each side of the plane and converting exact coordinates to double precision. Finally, *coplanar vertex* input are triangle pairs with vertex contact. Notice that, vertex contact are not detected by Held's algorithm since it only reports interior intersections, for which at least one point is common to the relative interiors of both triangles.

Values are obtained by averaging running times on about 1000 executions of 10^6 predicates on points whose coordinate values are generated using the `drand48` system call. Ratios obtained from different kinds of coordinate values as 24-bit and 53-bit pseudo-random integers are similar. Implementations has been done using the GNU 2.95 compiler with optimization flag `-O2` on a 1GHz Pentium III. Times have been obtained using the `clock` command. Compared with Möller's and Held's implementation, our code improves the execution time by around 20% when using IEEE double precision floating point computation. Used with exact arithmetic, it is in most cases slightly faster. Moreover, our algorithm can benefit from speedup techniques [8, 2, 1] to minimize the overhead of exact arithmetic. This is not possible in construction based approaches.

4 Two dimensional Triangle-Triangle Intersection Test

We now look at the problem for two planar triangles. This section proposes a new implementation of the two-dimensional triangle-triangle intersection test.

4.1 Description

If no assumption is made on the orientation of the triangles, the first step of the algorithm tests their orientation and might performs a swap operation of two of their vertices so that both triangles $p_1q_1r_1$ and $p_2q_2r_2$ are counterclockwise oriented. The algorithm then classifies p_1 w.r.t to T_2 by computing the signs of the three areas determined by p_1 with each of the three edges of T_2 (see Figure 2.). If all three are positive, p_1 is strictly interior to T_2. If two are zero, then p_1 lies on a vertex. Case where all three are zero means that p_1 must be collinear with all three edges and is impossible since triangles are assumed not to be flat. Finally, if a single area is zero and the other two are nonzero, only when the other two have positive signs does p_1 lie on the interior of an edge of T_2. In all these cases an intersection between the input triangles occurs.

In all other cases, a circular permutation can be applied to the vertices of T_2 so that p_1 either belongs to the interior of the R_1 labelled region or belongs to the interior or to the boundary of the R_2 labelled region as defined in Figure 6.

Assume first that this permutation maps p_1 to region R_1. We can classify q_1 w.r.t the four regions depicted in Figure 7 in a manner similar to how we classified p_1 w.r.t T_2 and determine how p_1q_1 meets T_2 (see decision tree Figure 9). To do so, we firstly calculate the orientation of q_1 w.r.t the line directed from r_2 to p_2 (Test I). If the sequence r_2, p_2, q_1 is counterclockwise (i.e. $[r_2, p_2, q_1]$ is strictly positive) then q_1 belongs to R_{11}. Therefore, edge p_1q_1 is guaranteed not to intersect the line passing through p_2 and r_2 and the input triangles intersection depends on the location of vertex r_1.

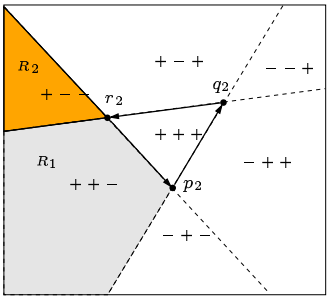

Figure 6: Assuming the edges of T_2 are counterclockwise, the sign pattern of the areas determined by p_1 and each edge are as shown. The boundary line between each $+$ and $-$ has "sign" zero.

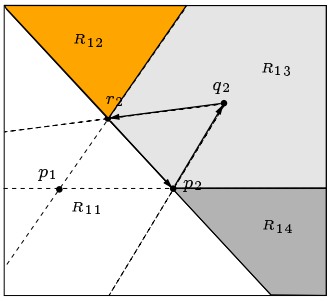

Figure 7: If vertex p_1 belongs to the ++ - labelled region, the plane is decomposed in four regions as shown.

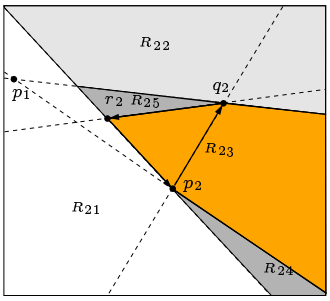

Figure 8: If vertex p_1 belongs to the + - - labelled region, the plane is decomposed in five regions as shown.

We then check $r_2p_2r_1$'s orientation (Test II.b), if it is clockwise, all three vertices of T_1 lie on same side of the line passing through r_2 and p_2 and the input triangles are disjoint, if not, we conclude with at most two more orientation calculations since an intersection takes place if and only if both triangles $q_1r_1r_2$ and $p_1p_2r_1$ are counterclockwise or flat (Tests III.b and IV.b).

Now, if the three points r_2, p_2, q_1, are counterclockwise or aligned, the orientation of triangle $r_2p_1q_1$ is computed (Test II.a). If $r_2p_1q_1$ is clockwise, q_1 belongs to R_{12} and there can be no intersection between the two triangles since T_1 is known to be counterclockwise oriented. If not, the calculation of $p_1p_2q_1$'s orientation is performed (Test III.a), if it is counterclockwise or zero then q_1 belongs to R_{13} and p_1q_1 must intersect r_2p_2. Otherwise q_1 belongs to R_{14} and the input triangles do intersect if and only if none of the two sequences p_1, p_2, r_1, and q_1, r_1, p_2, are clockwise oriented (Tests IV.a and V).

In the case when p_1 is mapped to region R_2, vertex q_1 is classified w.r.t. the five regions depicted in Figure 5 (see decision tree Figure 10). First the orientation of q_1 w.r.t the line directed from r_2 to p_2 is calculated (Test I). If the triplet r_2, p_2, q_1, is clockwise, $r_2p_2r_1$'s orientation is checked (Test II.b). If it is clockwise too, q_1 and r_1 lie on same side of the line passing through r_2 and p_2 and there can be no intersection between the input triangles. If not, $q_1r_1r_2$'s orientation is calculated (Test III.c). If it is clockwise only when the triangle T_1 intersects the edge q_2r_2 does an intersection occur. This leads to check that both triangles $q_1r_1q_2$ and $q_2r_2r_1$ are not clockwise oriented (Test IV.e and V.c). Finally, if $q_1r_1r_2$ is counterclockwise or flat, the triangles only intersect if r_1 does not belong to R_{24} (Test IV.d), which is true if and only if $r_1p_1p_2$ is as well counterclockwise oriented or flat. That leaves only the case when p_1 and q_1 lie on different side of the line passing through p_2 and r_2. We distinguish four different configurations depending to which region q_1 belongs to. The first two occur when the sequence q_2, r_2, q_1, is clockwise (Test II.a). In this case, we check whether q_1 belongs to R_{22} or to R_{25} by computing the orientation of $p_1q_2q_1$ (Test III.b). Now, if q_1 is in the interior of R_{22} there can be no intersection between the input triangles since $p_1q_1r_1$ is counterclockwise oriented. If q_1 lies in R_{25} (i.e $[p_1, q_2, q_1] \geq 0$), only when the edge q_2r_2 intersects the edge q_1r_1 do the input triangles intersect. In this last case, $q_2r_2r_1$ and $q_1r_1q_2$ should both be counterclockwise oriented or flat (Test IV.c and V.b). Now, if q_1 is left or on the line directed from q_2 to r_2, we then check whether q_1 belongs to the closed region R_{23} or to the interior of R_{24} or R_{22}. If $p_1p_2q_1$ is clockwise oriented (Test III.a), then q_1 belongs to R_{24} and the input triangles can only intersect when r_1 is left or on the lines directed from p_1 to p_2 and from r_2 to p_2 (Tests IV.b and V.a). If $[p_1, p_2, q_1] \geq 0$, $p_1q_2q_1$'s orientation is computed (Test IV.a). If it is clockwise, then q_1 lies in R_{22} and the input triangles are disjoint, otherwise, q_1 is in R_{23} and p_1q_1 surely intersects T_2.

4.2 Analysis

To summarize, the steps of the algorithm are as follows. The first step makes sure that the input triangles are counterclockwise oriented by calculating their orientation. If they are not, a swap operation of two of their vertices is performed. The second step classifies p_1 w.r.t. T_2's edges and applies a possibly circular permutation to T_2's vertices so that vertex p_1 lies in a region corresponding to one of the two generic cases. Finally, for each of these two possible configurations, the last step, concludes whether the input triangles intersect by calculating at most five successive orientation tests. In the worst case, the entire algorithm then has to perform two swap operations, a single circular permutation and ten degree two polynomials sign evaluations (eight if the orientation of the input triangles is known).

In comparison, the standard solution tests if at least one pair of edges of the input triangles (an edge of the first and an edge of the second triangle) intersect, or if one of the triangles is contained in the other. This leads to perform 18 orientation tests (six edge-edge intersection tests and two point-in-triangle containment tests) in the worst case when the triangles do not intersect.

The code of the two-dimensional and the three-dimensional intersection predicates presented in this paper will be included in a future release of CGAL software library (`www.cgal.org`).

References

[1] Francis Avnaim, Jean-Daniel Boissonnat, Olivier Devillers, Franco P. Preparata, and Mariette Yvinec. Evaluation of a new method to compute signs of determinants. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C16–C17, 1995.

[2] Hervé Brönnimann, Ioannis Emiris, Victor Pan, and Sylvain Pion. Sign determination in Residue Number Systems. *Theoret. Comput. Sci.*, 210(1):173–197, 1999. Special Issue on Real Numbers and Computers.

[3] O. Devillers and F. P. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete and Computational Geometry*, 20:523–547, 1998.

[4] Martin Held. ERIT – A collection of efficient and reliable intersection tests. Technical Report, University at Stony Brook, 1996.

[5] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1998.

[6] Tomas Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

[7] Tomas Möller. A fast triangle-triangle intersection test. *Web page, June 2002.* http://www.acm.org/jgt/papers/Moller97/.

[8] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997.

[9] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.

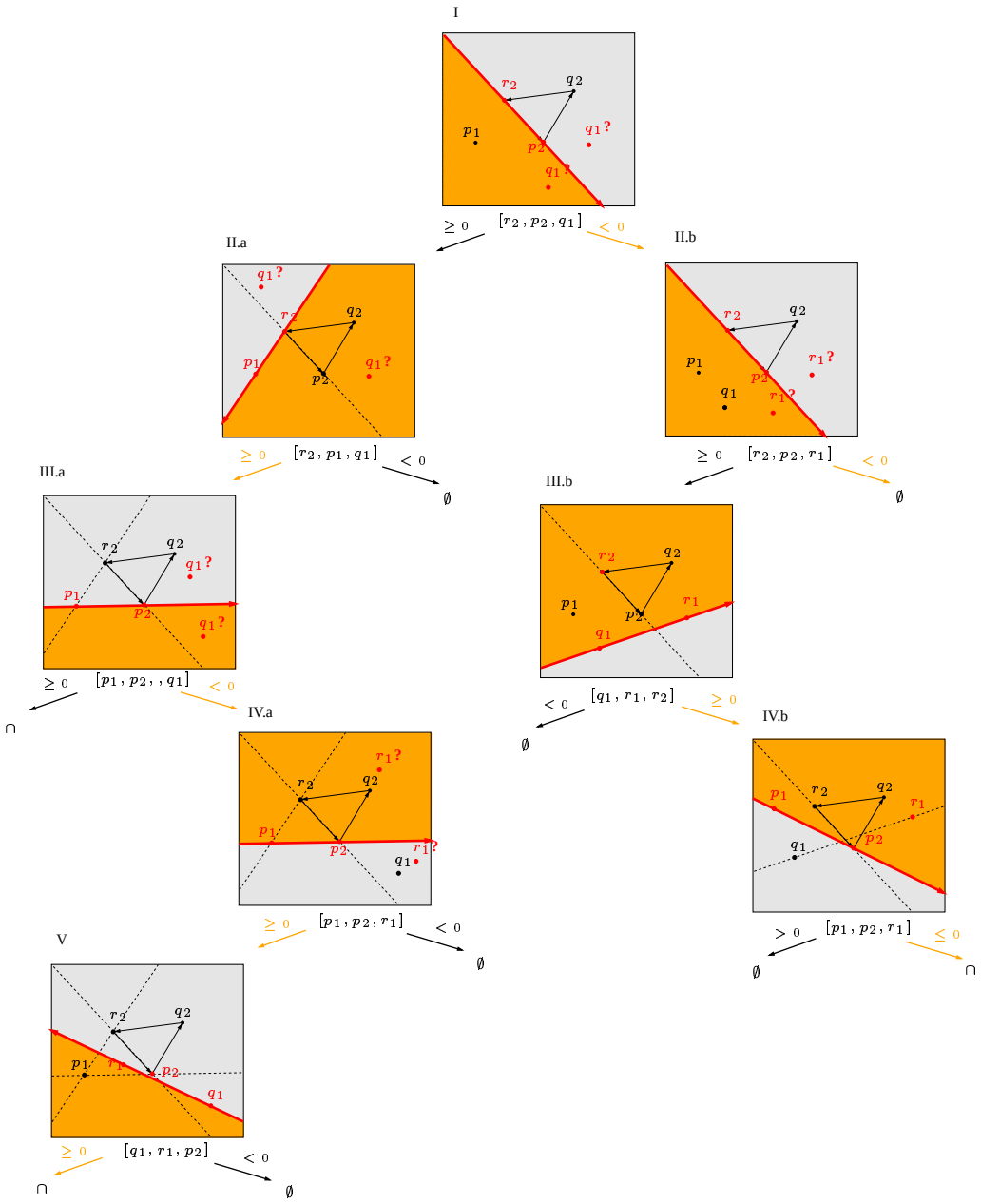

Figure 9: Decision tree when vertex p_1 belongs to region R_1.

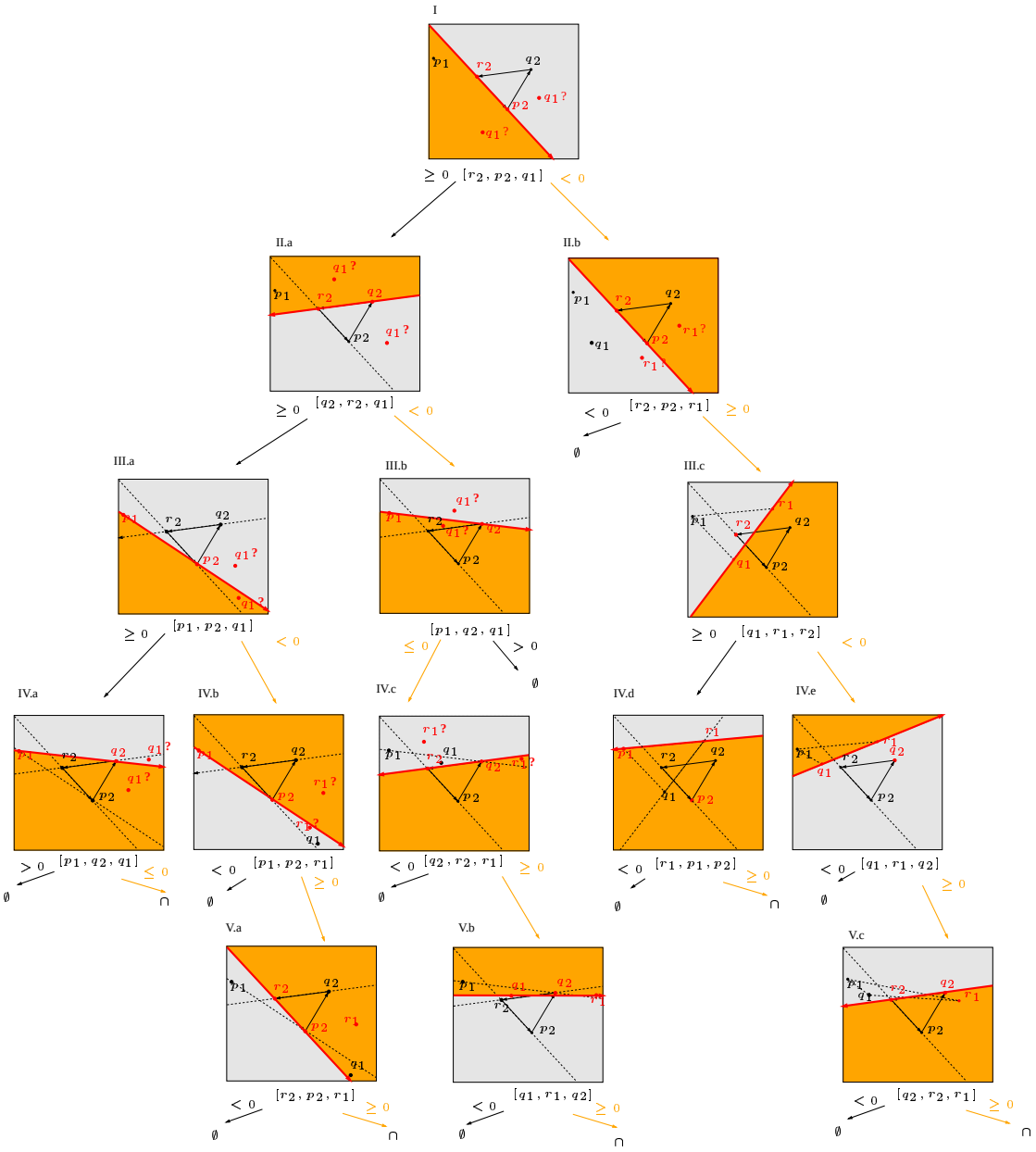

Figure 10: Decision tree when vertex p_1 belongs to region R_2.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399