IGI Global publications, (www.igi-global.com), pp.195-.235, 2016

A Generic Design for Implementing Intersection Between Triangles in Computer Vision and Spatial Reasoning, Innovative Research in Attention Modeling and Computer Vision Applications, IGI Global publishers, (www.igi-global.com) *2015*

Chaman L. Sabharwal and Jennifer L. Leopold
Missouri University of Science and Technology
Rolla, Missouri, USA – 65409
{chaman, leopoldj}@mst.edu

Table of Contents

A Generic Design for Implementing Intersection Between Triangles in Computer Vision and Spatial Reasoning

Chaman L. Sabharwal and Jennifer L. Leopold
Missouri University of Science and Technology
Rolla, Missouri, USA – 65409
{chaman, leopoldj}@mst.edu

Abstract

The intersection between 3D objects plays a prominent role in spatial reasoning, and computer vision. Detection of intersection between objects can be based on the triangulated boundaries of the objects, leading to computing triangle-triangle intersection. Traditionally there are separate algorithms for cross and coplanar intersection. For qualitative reasoning, intersection detection is sufficient, actual intersection is not necessary; in contrast, the precise intersection is required for geometric modeling. Herein we present a complete design and implementation of a single integrated algorithm independent of the type of intersection. Additionally, this algorithm first detects, then intersects and classifies the intersections using barycentric coordinates. This work is directly applicable to: (1) VRCC-3D+, which uses intersection detection between 3D objects as well as their 2D projections essential for occlusion detection; and (2) CAD/CAM geometric modeling where curves of intersection between a pair of surfaces are required for numerical control machines. Experimental results are provided.

Keywords: *Intersection Detection, Geometric Modeling, Classification Predicates, Spatial Reasoning, Triangle-Triangle Intersection*.

INTRODUCTION

Triangular mesh is used to represent the boundary of freeform 3D objects**.** The intersection between 3D objects plays a prominent role in spatial reasoning, geometric modeling, and computer vision. Detection of possible intersection between objects can be based on the objects' boundaries (approximate triangulations of objects) in region connection calculi (RCC8), leading to computing triangle-triangle intersection. Traditionally there are specialized algorithms for cross intersection and coplanar intersection. The intersection detection is a byproduct of actual intersection computations. Most of the time intersection detection is done prior to the determination of the actual intersection so that the non-intersecting objects can be discarded. For example, in qualitative spatial reasoning, intersection detection is sufficient, actual intersection is not required; however in geometric applications, it is desirable to have actual intersections.

For early intersection detection, we present a complete uniform integrated algorithm that is independent of cross and coplanar intersection. Herein we illustrate this with an algorithm, and with a Python implementation where the output is displayed with tables and figures.

The ability to detect the existence of possible intersection between pairs of objects is important in a variety of problem domains such as geographic information systems (Egenhofer & Golledge, 1998), real-time rendering (Tropp, Tal & Shimshoni, 2006), collision-detection, geology (Caumon, Collon, Le Carlier de Veslud, Viseur, & Sausse, 2009), surface modeling (Houghton, Emnett, Factor, & Sabharwal,1985), computer vision, networking and wireless computing. Typically, the boundary of each object is represented as a triangulated surface and a triangle-triangle intersection is the computational basis for determining intersection between objects. Since an object boundary may contain thousands of triangles, algorithms to speed up the intersection detection process are still being explored for various applications, sometimes with a focus on innovations in processor architecture (Elsheikh & Elsheikh, 2012).

For pairs of triangles, there are three types of intersections: zero dimensional (single point), one dimensional (line segment), and two dimensional (area) intersection. In the past, almost all attention has been devoted to determining the cross intersections, which resulted in an absence of analysis in two dimensional intersections. Coplanar triangle intersections are unique because an intersection may be any of the aforementioned three types. If the triangles cross intersect, only zero or one dimensional intersection is possible. If the planes are parallel and distinct, the triangles do not intersect. If the triangles are coplanar, then there is a possibility of intersection. Even when the cost of intersecting a triangle pair is constant, the cost of intersecting a pair of objects A and B is order $O(T_A * T_B)$ where T_A is the number of triangles in object A, and T_B is the number of triangles in object B.

In qualitative spatial reasoning, spatial relations between regions are defined axiomatically using first order logic (Randell, Cui, & Cohn, 1992), or the 9-Intersection model (Egenhofer & Franzosa, 1991). Using the latter model, the spatial relations are defined using the intersections of the interior, boundary, and exterior of one region with those of a second region. It has been shown in (Sabharwal & Leopold, 2011) that it is sufficient to define the spatial relations by computing 4-Intersection predicates, (namely, Interior–Interior (IntInt), Boundary–Boundary (BndBnd), Interior–Boundary (IntBnd), and Boundary–Interior (BndInt)) instead of 9-Intersections. Since IntBnd and BndInt are the converse of each other, only three algorithms are necessary for these predicates. In order to implement these algorithms, we must first implement the triangle-triangle intersection determination. The triangle-triangle intersection is a lower level problem that must be solved in order to determine the 4-Intersection predicates which, in turn, determine the qualitative spatial relation between two objects.

In geometric modeling, the surface-surface intersection problem occurs frequently. In CAD/CAM, Numerical Control (NC) machines use automated tooling for parts and automated machines use algorithms for cutting sheet metal parts. The parts are represented as 3D objects using the ANSI (Brep) model. Since the surface is represented as triangulated mesh, intersection between 3D objects amounts to detecting and computing intersection between 3D triangles. Herein the triangle-triangle intersection is required for such applications. The algorithm presented here is extremely useful for geometric modeling where intersection between objects occurs thousands of times. For geometric applications, cross intersection is the most often used approach to determine curves of intersection between surfaces (Houghton et al., 1985).

Most of the algorithms use code optimization of same or similar cross intersection algorithms. In order to realize this for cross or coplanar intersection, we must design a criteria that will work for all cases unambiguously. Thus here we present new algorithms that are different from previous approaches.

This chapter is organized as follows. First we briefly review the background: vector concepts and vector equation solutions, and related inequalities resulting from an intersection framework. Then we discuss the types of possible cross and coplanar intersections between a pair of triangles. We then describe the cross intersection, coplanar area intersection algorithm, and composite algorithms for general triangles. Finally we develop the overall generic algorithm for triangle-triangle intersection, and classify the intersections. Later we describe experiments, the timing results, and two of the applications where this approach is directly applicable.

BACKGROUND

Vector Notation And Terminology

Vector analysis is the foundation for the intersection problems. The relevant definitions can be found in any book on calculus. The terms such as *position* vector, *localized* vector, scalar multiplication, *dot* or *scalar* or *inner* product, *cross* or *vector* product, *unit* vector, *collinear* vectors, *coplanar* vectors, and vector equation of a line are standard.

We do review some of the terminology here. The vector A is a geometric entity defined by its components, $[a_1, a_2, a_3]$. A vector is conventionally denoted by bold letter, **A**, or with an arrow over it, $\vec{A}$, but for convenience, we will use the same symbol A for vector and point. The vector $A = [a_1, a_2, a_3]$ is a row vector; equivalently, a vector can be represented by a column vector as $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ or $[a_1, a_2, a_3]^t$, the transpose of row vector $[a_1, a_2, a_3]$. It is a *position* vector of a point (a_1, a_2, a_3) if the start point is the origin of the coordinate system and the terminal point is the position of point (a_1, a_2, a_3). The length of vector is $\sqrt{(a_1^2+a_2^2+a_3^2)}$, and the direction is from the origin to the point A. Conventionally i, j, and k denote unit vectors along the principal axes (i.e. x-,y-, and z-axis). If A and B are position vectors, then AB is a vector from position A to position B and $AB = \begin{bmatrix} b_1 - a_1 \\ b_2 - a_2 \\ b_3 - a_3 \end{bmatrix}$. The vector A is a *free* vector if it refers to any line segment with same length and direction, i.e. a free vector is free from the coordinates of the start point and the terminal point which may be different depending on the location of the vector.

The two operations on vectors are *dot* product and *cross* product. The *dot* product of two vectors A and B is denoted by as A•B and is defined as $A \bullet B \equiv a_1b_1 + a_2b_2 + a_3b_3$ or simply $A \bullet B \equiv A^tB$. The dot product is commutative (*symmetric*). The *cross* product of vector A with B is denoted by AxB and is defined as $A \times B = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}$ or simply $A \times B = A\ I \times B = A \times I\ B$ where I is the 3x3 identity matrix. The cross product is *anti-symmetric*. Geometrically, it is a vector perpendicular to A and B so that A, B, and AxB form a right-handed system.

Two vectors A and B are *orthogonal* if A•B = 0. Two vectors A and B are *parallel* if AxB = 0.

Two vectors A and B are *collinear* if they are parallel and have a point in common, i.e., they are located along the same line. The vector equation of a line through a point P along a direction U is given by R(u) = P + u U, $-\infty < u < \infty$. Two lines R(a) = P + a A and R(b) = Q + b B are *coplanar* if AxB•PQ = 0. The position vectors of points A and B are always coplanar, two localized vectors need not be coplanar.

We use the following three identities very frequently in our derivations. If A, B, C, and D are vectors, then: (1) AxB•C = A•BxC, (2) Ax(BxC)= A•C B – A•B C, and (3) (AxB)x(CxD) = AxB•D C - AxB•C D.
If A and B are coplanar and non-collinear vectors, any vector X in the plane of A and B can be expressed as X = a A + b B, where a, b are the components of the vector X with respect to vectors A and B.
If A, B and C are non-coplanar vectors in 3D, any vector X in the 3D space can be expressed as X = a A + b B + c B, where a, b, and c are the components of the vector X with respect to vectors A, B, and C.

Using Vectors To Solve Equations and Pitfalls

In this section, we present a method for solving the vector equations and ensuring the accuracy of the solution. This is the backbone of the generic algorithm we present in later sections.
Let U, V (non-collinear) and W be 3D vectors. We solve the following equation for u and v where

u U + v V = W

The general technique to solve this equation is to define two vectors Ux(UxV) and Vx(UxV) orthogonal to U and V, respectively. When we dot multiply this equation with Vx(UxV) and Ux(UxV), we quickly get u, v as

$$u = \frac{W \bullet V \times (U \times V)}{U \bullet V \times (U \times V)} = -\frac{W \times (U \times V)}{(U \times V) \bullet (U \times V)} \bullet V$$

and

$$v = \frac{W \bullet U \times (U \times V)}{V \bullet U \times (U \times V)} = \frac{W \times (U \times V)}{(U \times V) \bullet (U \times V)} \bullet U$$

We will encounter this vector equation frequently when we analyze the triangle-triangle intersection problem in later sections. This quick solution looks like a perfect solution, when in fact it is not robust. This innocent looking solution is treacherous. We need to be very careful. There is a short coming in this solution. This solution (u, v) may or may not be valid, i.e., if we plug it into the equation it may or may not satisfy the equation. If it does not satisfy the equation, then either the solution is inconsistent or the equation is inconsistent. For example, let x i + y j = k. Applying the above method, we get x = y = 0, which does not satisfy the equation because k is a unit vectors. This complicates overfitting because the equation is inconsistent. This equation is consistent if and only if UxV•W=0.
To make the implementation code robust, additional testing is necessary. A test UxV•W=0 has to be made to ascertain that the equation is consistent. We want a solution that satisfies the constraints of the given equation.

Using Vectors To Represent Triangles

A triangle (three angles) ABC is a shape (more accurately the boundary of a shape) defined with three non-collinear points A, B, and C. The shape consists of infinitely many points represented by (compressed into) three points. It is a polygon with three vertices, three sides, and three angles. More accurately, it is a trigon. The sum of the interior angles of a triangle is equal to 180 degrees, the angles may be acute, obtuse or right angles. The orientation of a triangle ABC is right handed if one moves along the sides, AB, BC, and CA so that the interior of the triangle is always to its left. More formally, if the direction of the head is represented by a normal vector N, then the triangle ABC is oriented counter clockwise if AB**x**AC•N > 0.

There are various types of triangles: equilateral, isosceles, and scalene. There are various properties associated with a triangle: interior, exterior, boundary, area, perimeter, centroid, orthocenter, incenter, etc. Interestingly triangles have been used as symbols to represent various phenomena. For example, Egyptians used triangles to design pyramids, the department of transportation uses triangles as a warning symbol on highways, a "yellow" triangle is used as a biological hazard symbol in hazardous environments and for weak WIFI wireless signals, and a "pink" triangle symbol was used as gay rights protest symbol on badges in Nazi concentration camps.

In this exposition, we are interested in the representation and intersection of a pair of triangles devoid of any other properties. Thus we represent the terms, *vertex, edge, edgeInterior, and triangleInterior* in terms of parametric coordinates. A triangle can be represented in Cartesian coordinates or parametric coordinates. The parametric representation is succinct and more useful in complex geometric computations. The parametric representation of a *plane* is determined by three non-collinear points A, B, C in 2D/3D, as

$R(b, c)= A +b (B – A) + c (C – A)$ where $-\infty < b, c < \infty$ or

$R(b, c)= (1-b-c) A +b B + c C$ where $-\infty < b, c < \infty$.

Since a triangle has infinitely many points represented by (or compressed into) three points A, B, C in 3D, any point in the triangle can be parametrically represented as

$R(a, b, c)= a A + b B + c C$ where $0 \le a, b, c \le 1, a + b + c =1$.

The parameters a, b and c are called the *barycentric* coordinates of a point in the triangle. Since a, b, c are not independent, the equation of a triangle alternately can be written as

$R(b, c)= (1- b – c) A +b B + c C$ where $0 \le b, c, b+c \le 1$ or

$R(b, c) \equiv A +b (B – A) + c (C – A)$ with $0 \le b, c, b+c \le 1$.

Associated with a triangle, the terms *vertex, edge, edgeInterior* and *triangleInterior* are parametrically defined here.

vertex: The parametric values (b, c) are the barycentric coordinates corresponding to a point in the triangle ABC. In particular, R(0, 0) represents the ***vertex*** *A*, R(1, 0) represents the *vertex B*, and R(0, 1) represents the *vertex C*.

edge: In addition, $R(b, 0) \equiv A +b (B – A)$ where $0 \le b \le 1$ is the *side* AB, $R(0,c) \equiv A + c (C – A)$ where $0 \le c \le 1$ is the *side* AC, and $R(b, c) \equiv A +b (B – A) + c (C – A)$ where $0 \le b, c \le 1$ and $b+c = 1$ is the *side* BC. The sides of the triangles are also referred to as the ***edge***s.

edgeInterior: R(b,0), R(c,0), with $0 < b, c < 1$ are ***interior***s of edges AB and AC, and where R(b, c) with $b+c =1$, $0 < b, c < 1$ is the ***interior*** of edge BC.

triangleInterior: If $0 < b, c, b+c < 1$, then R(b, c) is ***interior point*** of triangle ABC.

In particular if a triangle ABC is parameterized with $R_1(b,c)$, a triangle PQR is parameterized with $R_2(q,r)$ and X is a point such that $X= R_1(1,0)$ or $R_1(0,1)$ or $R_1(0,0)$, and $X= R_2(1,0)$ or $R_2(0,1)$ or $R_2(0,0)$, then X is a ***vertex*** common to both of the triangles ABC and PQR.

Solving Inequalities and Pitfalls

The inequalities of the form below appear as part of intersection algorithms.

$$m \le ax + by \le n \qquad M \le Ax + By \le N$$

The following example explains why one should beware of the pitfalls when solving these inequalities:

$$-1 \le x + y \le 1 \quad (a) \qquad -1 \le x - y \le 1 \quad (b)$$

Adding (a) and (b) yields $-1 \le x \le 1$, and subtracting (b) from (a) yields $-1 \le y \le 1$ which is the area enclosed by dotted boundary in Fig. 1(a). This is an inaccurate solution to the inequalities (a), and (b). But the accurate solution is in the shaded area in Fig. 1(a), which is $|x| \le 1$, and $|y| \le (1 - |x|)$.

If we modify the coefficients in (b) so that (b) becomes (b'): $0 \le x - y \le 0$, then on adding (a) and (b'), it yields $-1/2 \le x \le 1/2$, and (b') yields x=y, and the result is a straight line as seen in Fig. 1(b).

Further, if we further modify the coefficients in (a) so that (a) becomes (a'): $0 \le x + y \le 0$, and we have y=x from (b'), then (a') yields y=x=0, and the result is a single point as seen in Fig. 1(c).

Thus to accurately solve these inequalities $-1 \le x + y \le 1$, and $-1 \le x - y \le 1$, we first solve these for one variable x, then use that value to solve for the other variable y; otherwise, we run the risk of an inaccurate solution due to the additional extraneous part in the solution displayed in the box, but not included in the diamond in Fig. 1(a).

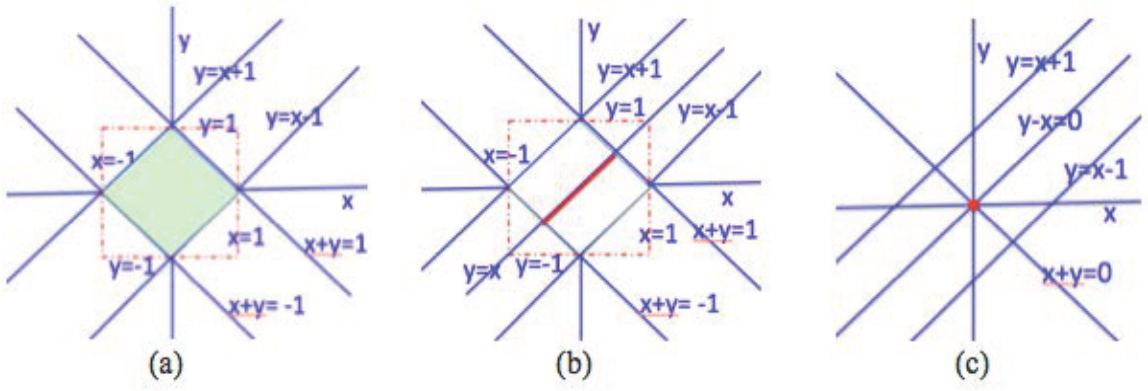

Fig. 1(a) Solution to inequalities: $-1 \le x + y \le 1$ and $-1 \le x - y \le 1$. Using brute force method of elimination of variables yields the area enclosed by the dotted boundary, but the accurate solution is enclosed by the shaded area. **(b)** Solution to inequalities: $-1 \le x + y \le 1$, and $0 \le x - y \le 0$. The solution results in a shaded line segment. **(c)** Solution to inequalities: $0 \le x + y \le 0$ and $0 \le x - y \le 0$. The solution results in a shaded single point.

Solving Two General Inequalities

The most general form of the two inequalities in x and y is:

$$m \le ax + by \le n \,(1) \qquad M \le Ax + By \le N \qquad (2)$$

Solving the inequalities of this form will be of paramount importance in the generic algorithm presented later. These inequalities may or may not have a solution. For example, the inequalities

$0 \le x+y \le 1$ and $4 \le 2x+2y \le 6$ do not have a solution. The following algorithm determines: (1) *existence*: whether a solution exists or not; and (2) if a *solution* exists, then it determines the range of x, and computes x_m, x_M such that for each x in $[x_m, x_M]$, the inequalities hold.

Algorithm for the existence and evaluation of x values

***Input*:** coefficients of the two inequalities *m, a, b, n, M, A, B, N*

Output: If a solution does not exist, return false; otherwise, return true and x_m, x_M such that for each x in $[x_m, x_M]$, the inequalities are satisfied.

Pseudocode:

boolean solve_x (m, a, b, n, M, A, B, N, x_m, x_M)

First assume b and B are non-negative. If not, multiply the inequalities by -1 to update them as non-negative. Now multiplying (1) by B and (2) by b, subtraction leads to

$$(mB - Nb) \le (aB - Ab)x \le (nB - Mb)$$

If $(mB - Nb) \le (nB - Mb)$, then there exists a solution in x, else the inequalities do not have a solution.

This inequality yields a range $[\mathbf{x_m, x_M}]$ for x values provided $aB-Ab \neq 0$, otherwise any arbitrary value x satisfies the inequality and we may assign $x_m = -\infty$, $x_M = \infty$, hence the solution in x is unbounded.

Assuming $aB-Ab \neq 0$, once x_m, x_M have been determined, for each x in $[x_m, x_M]$ in the inequalities, we similarly determine the range $[y_m(x), y_M(x)]$ for y(x). That is, after the range $[x_m, x_M]$ is determined, only then for each x in $[x_m, x_M]$, if a solution in y exists, the range for y is determined for the selected value x.

Algorithm for the existence and evaluation of y values

Input: coefficients of the two inequalities *m, a, b, n, M, A, B, N; the range for* x: $[x_m, x_M]$ is determined from the above algorithm.

Output: If a solution is not found, return false; otherwise, return true and a range for y $[y_m(x), y_M(x)]$ for each x in $[x_m, x_M]$, for which the inequalities hold. In the process, the values of $[x_m, x_M]$ may be updated to retain those x values for which the solution in y exists.

***Pseudocode*:**

boolean solve_y (m, a, b, n, M, A, B, N, x, y_m, y_M)

Given that $x_m \le x \le x_M$ are known, it solves the inequalities for y_m, y_M.

Now for an x such that $x_m \le x \le x_M$, the inequalities become

$$m - ax \le by \le n - ax \text{ and}$$

$$M - Ax \le By \le N - Ax.$$

If $m - ax > n - ax$ or $M - Ax > N - Ax$ the solution does not exist because the inequalities are inconsistent. Assume b and B are non-negative, otherwise multiply the inequalities by -1 and update b, and B to non-negative. The solution is determined as follows.

if b>0 and B>0 then

$$y_m(x) = \max\left(\frac{m-ax}{b}, \frac{M-Ax}{B}\right) \text{ and}$$

$$y_M(x) = \max\left(\frac{n-ax}{b}, \frac{N-Ax}{B}\right)$$

elseif b>0 then

$$y_m(x) = \frac{m-ax}{b} \text{ and}$$

$$y_M(x) = \frac{n - ax}{b}$$

elseif B>0 then

$$y_m(x) = \frac{M - Ax}{B} \text{ and}$$

$$y_M(x) = \frac{N - Ax}{B}$$

else y is arbitrary

$y_m(x) = -\infty$ and $y_M(x) = \infty$, hence the solution in y is unbounded.

These two algorithms solve a pair of inequalities completely. The result is either no solution, a single point, a line segment, or an area.

THE CONVENTIONAL TRIANGLE-TRIANGLE INTERSECTION STRATEGIES

There is an abundance of papers devoted to the intersection between a pair of triangles ((Mo¨ller, 1997), (Didier, 1990), (Guigue & Devillers, 2003), (Sabharwal & Leopold, 2013)). Interestingly, most of this work concentrates on *cross* intersection; the authors simply reinvent the algorithm and optimize the code to implement it slightly differently and more efficiently, with no innovation. In these approaches, attempts are made to compute the intersection immediately, rather than first trying to determine whether an intersection even exists. The paper (Sabharwal & Leopold, 2013) surveyed various approaches for determining the cross intersection detection, and developed a fast vector version of the cross intersection detection, as well as classification of the type of intersection. The papers (Mo¨ller, 1997) and (Guigue & Devillers, 2003) also compare hardware implementation of their algorithm based on the number of arithmetic +, -, *, / operations. Another paper (Guigue & Devillers, 2003) also compares the optimized intersection times. Recent papers (Didier, 1990) and (Guigue & Devillers, 2003) considered an approach for determining the intersection detection covering coplanar intersection. These approaches, (Didier, 1990), (Guigue & Devillers, 2003), and (Sabharwal & Leopold, 2013), lead to one composite algorithm that uses two separate algorithms, one for cross intersection and one for coplanar intersection. Our approach follows (Sabharwal, Leopold, & McGeehan, 2013) and is exhaustive and analytically more rigorous than the previous approaches of (Mo¨ller, 1997), (Held, 1997), (Didier, 1990), (Guigue & Devillers, 2003), and (Sabharwal & Leopold, 2013). It computes intersection in terms of barycentric coordinates. Logical rather than computational tests are used to detect whether the intersection is a single point, or a line, or an area. The algorithms in (Sabharwal, Leopold & McGeehan, 2013) are not robust completely as they are missing critical steps for (t_m(s), t_M(s)) yielding an approximate solution and there are exposition errors in the derivation of the triangle-triangle intersection algorithm. Herein we address those errors and provide the implementation for accurately solving the inequalities. Consequently, we give a robust algorithm and clear exposition in this paper overcoming all the shortcomings in the previous papers. Our approach is different from previous ones in that we depend on only one equation rather than different ones for each special case. We show that it is possible to detect the existence of intersection before the precise intersection computations are performed. Two triangles may not intersect (as seen in Fig. 2), or may cross intersect or may coplanar intersect.

It should be noted that the precise intersection of coplanar triangles is a little more complex because it can result in area intersection as well. For *coplanar* triangles, the intersection can be classified as: (1) single point intersection (*vertex-vertex, vertex-edgeInterior*)

shown in Fig. 3(a, b); (2) line segment Intersection (*edge-edgeCollinear*) shown in Fig. 4(a), and (3) area intersection bounded by 3 edges, shown in Fig. 5(a, b); bounded by 4, 5, 6 edges, shown in Fig. 6(a, b, c). A triangle may be entirely contained in the other triangle, see Fig. 6(d). Note that coplanar or crossing triangles in Fig. 3(a, b) and Fig. 4(b), look alike when projected in a plane. There are 4 versions of Fig. 6(c).

If the triangles *cross intersect*, the intersection can be classifies as: (1) single point intersection (*vertex-vertex, vertex-edgeInterior, edgeInterior-edgeInterior, vertex-triangleInterior*), shown in Fig. 3(a, b, c, d), or (2) line segment intersection (*edge-edge, edge-triangleInterior, triangleInterior-triangleInterior*), shown in Fig. 4(a, b, c). It is possible that two triangles cross intersect in a line segment even when a triangle is on one side of the other triangle. In that case, it may be desirable to know which side of the other triangle is occupied. In Fig. 4(b), the triangle PQR (except QR which is in ABC) is on the positive side of triangle ABC. So PQR does not intersect the interior of object of triangle ABC.

Categorizing of Triangle-Triangle Intersection

The intersection between a pair of triangles can be abstracted as: Cross (C) intersection or Parallel (P) coplanar triangles intersection. For taxonomy of cross and parallel coplanar triangles, the conceptual intersections are supported with figures presented here. The specific cases are as follows:

No intersection
disjoint (C, P) (see Fig. 2)

Single Point Intersection
vertex-vertex Intersection (C, P) (see Fig. 3(a))
vertex-edgeInterior Intersection (C, P) (see Fig. 3(b))
vertex-triangleInterior Intersection (C) (see Fig. 3(c))
edgeInterior-edgeInteriorCross Intersection (C) (Fig. 3(d))

Line intersection
edge-edgeCollinear Intersection (C, P) (see Fig. 4(a))
edge-triangleInterior Intersection (C) (see Fig. 4(b))
triangleInterior-triangleInterior Intersection (C) (Fig. 4(c))

Area Intersection
vertex-triangleInterior Intersection (P) (see Fig. 5(a,b), Fig. 6(a, b, d))
edgeInterior-edgeInterorCross Intersection (P) (Fig. 5(a,b), Fig. 6(a, b, c))
edge-triangleInterior Intersection (P) (see Fig. 5(a, b), Fig. 6(d))
triangleInterior-triangleInterior Intersection (P) (see Fig. 5(a,b), Fig. 6(a, b, c, d))

It is possible that two triangles cross intersect in a line segment even when a triangle is on one side of the other triangle. In that case, it may be desirable to know which side of the other triangle is occupied. In Fig. 4(b), the triangle PQR (except QR which is in ABC) is on the positive side of triangle ABC. So PQR does not intersect the interior of object of triangle ABC.
It should be noted that the vertex-edge intersection encompasses vertex-vertex and vertex-edgeInterior intersection, whereas the vertex-triangle intersection encompasses vertex-vertex,

vertex-edgeInterior, and vertex-triangleInterior. Thus 1D jointly exhaustive and pairwise disjoint (JEPD) cross intersection between ABC and PQR can be one of the three possibilities: (1) collinear along edges, (2) an edge of PQR lying in the plane of triangle ABC, or (3) triangles "pierce" through each other yielding an intersection segment.

Example Figures for Intersection Between a Pair of Triangles

In this paper, we present a detailed integrated analytical study of the intersection of coplanar as well as cross intersecting triangles, which previously has not been explored. We will use this concept development in the section where we discuss our generic algorithm. Here we display some representative figures for intersection between a pair of triangles.

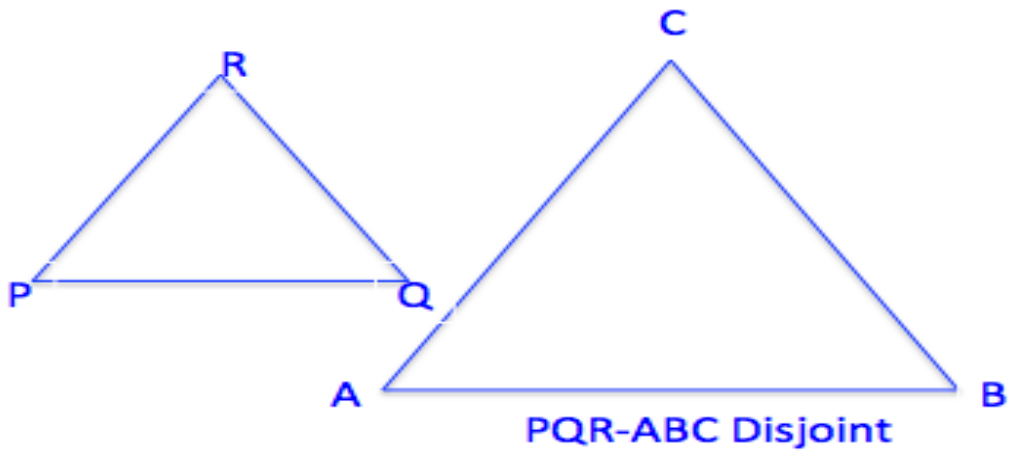

Fig. 2. Disjoint triangles: Planes supporting the triangles may be crossing or coplanar. The triangles do not have anything in common.

It should be noted that the vertex-edge intersection encompasses vertex-vertex and vertex-edgeInterior intersections, whereas the vertex-triangle intersection encompasses vertex-vertex, vertex-edgeInterior, and vertex-triangleInterior intersections. Thus 1D JEPD cross intersection between ABC and PQR can be one of the three possibilities: (1) collinear along edges, (2) an edge of PQR lying in the plane of triangle ABC, or (3) the triangles may "pierce" through each other yielding an intersection segment.

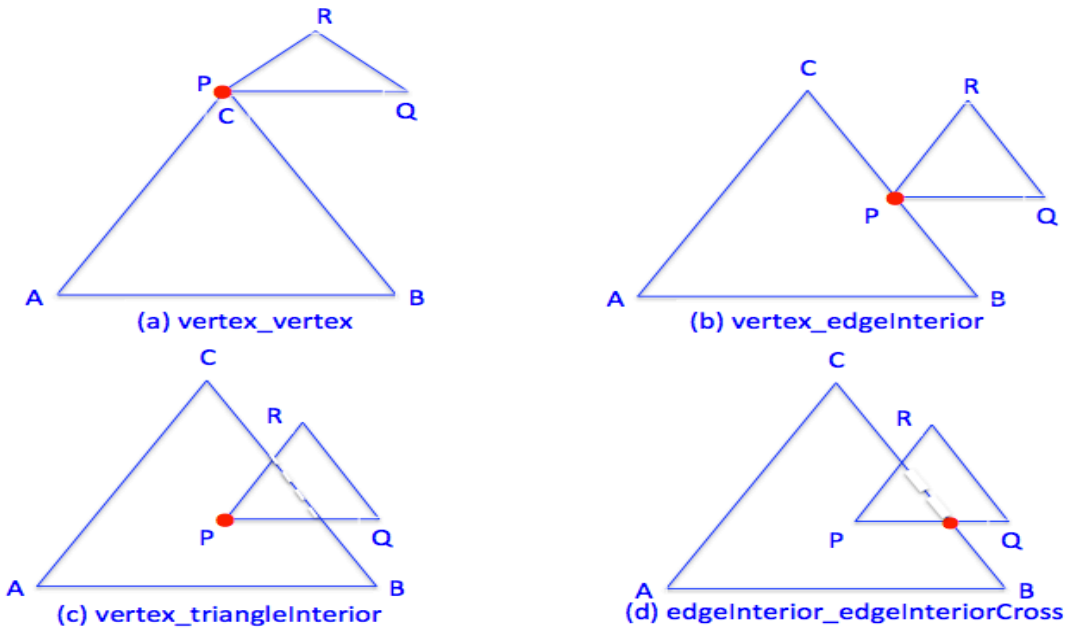

Fig. 3. Triangles intersect at a single point. The intersections between triangles ABC and PQR are JEPD (Jointly Exhaustive and Pairwise Distinct) cases of Single Point intersection between triangles. (a) vertex-vertex and (b) vertex-edgeInterior can occur in both cross and coplanar intersections. However, (c) vertex-triangleInterior and (d) edgeInterior-edgeInterior intersection point can occur in cross intersection only.

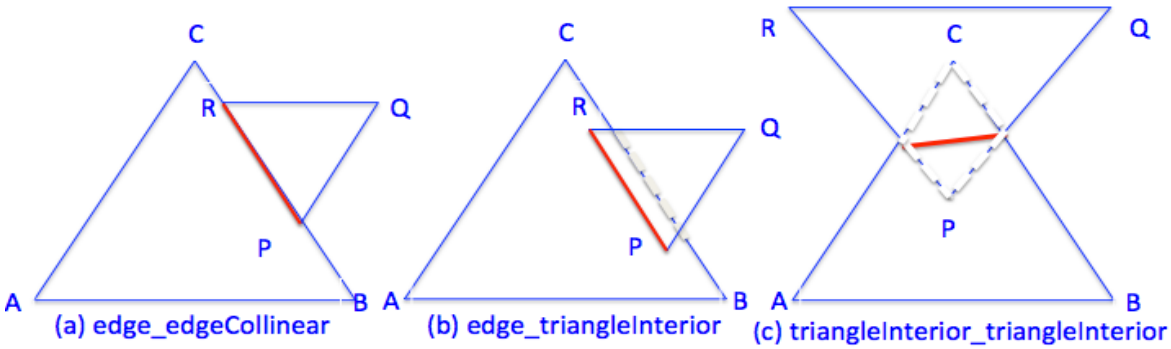

Fig. 4. Triangles intersect in a line segment. (a) edge-edgeCollinear intersection can occur in both cross and coplanar intersections. However, (b) edge-triangleInterior and (c) triangleInterior-triangleInterior intersection segment occur in cross intersection only.

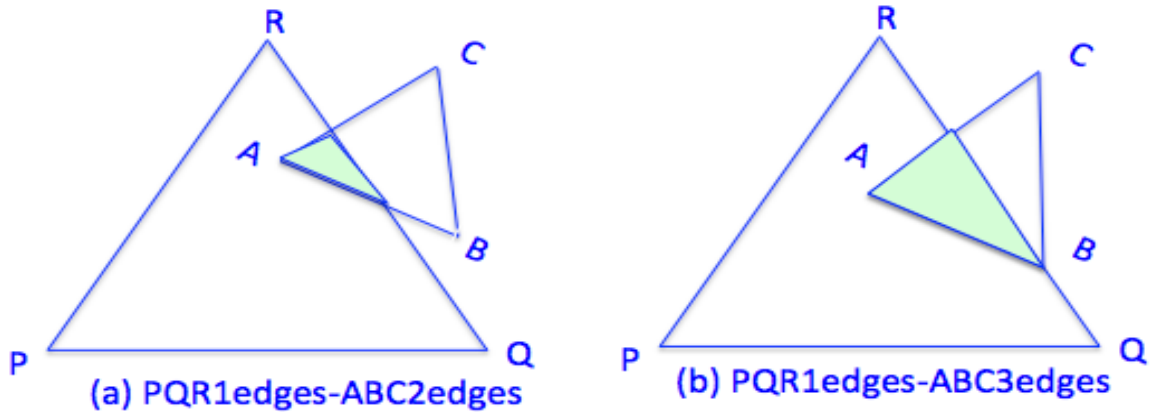

Fig. 5. Triangles intersect in an area (shaded). (a) One edge of triangle PQR and two edges AB and AC of triangle ABC intersect, vertex A is in the interior of PQR. (b) One edge of triangle PQR with three edges of ABC, and vertex A in the interior of PQR. The common area is bounded by three edges. The intersections vertex-triangleInterior, edge_triangle, and edgeInterior-triangleInterior hold.

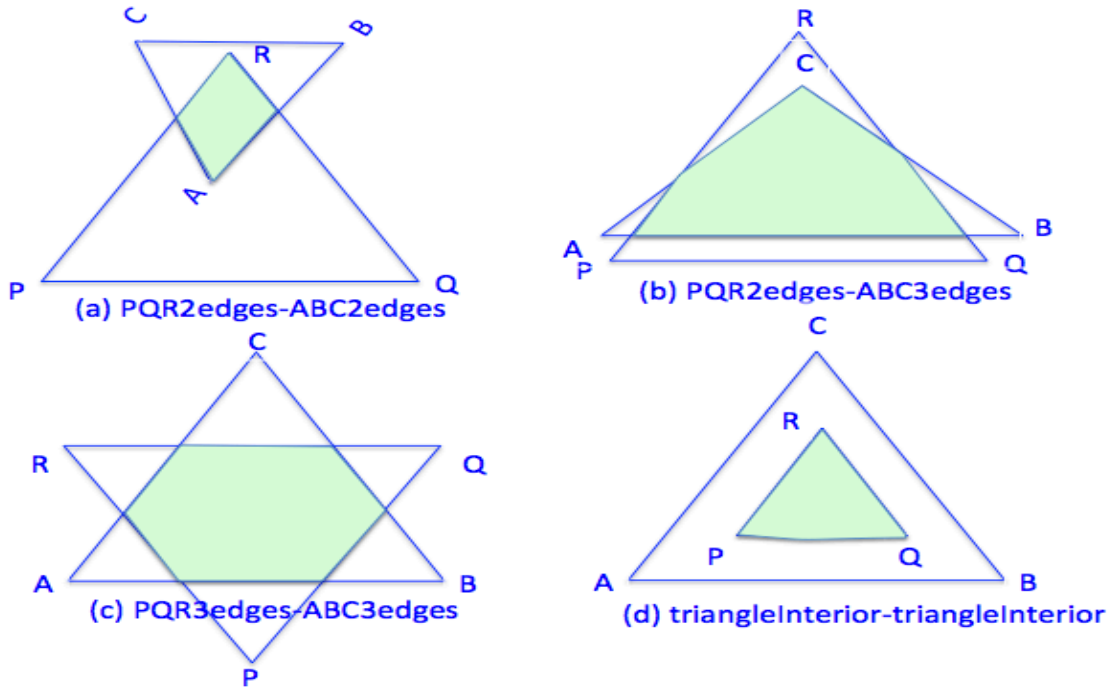

Fig. 6. Triangles intersect in an area (continued). The coplanar triangle intersections are bounded by four, five, and six edge segments. (a) Two edges of triangle PQR and two edges AB and AC of triangle ABC intersect, vertex A is in the interior of PQR, and vertex R is in the interior of triangle ABC. The intersection area is bounded by four edges. (b) Two edges of triangle PQR and three edges of triangle ABC intersect; vertex C is in the interior of PQR. The intersection area is bounded by five edges. (c) Three edges of triangle PQR and three edges of triangle ABC intersect; every vertex of one triangle is outside the other triangle. The intersection is bounded

by six edges. (d) No edge of triangle PQR intersects any edge of triangle ABC; vertices P, Q, R are in the interior of triangle ABC. The intersection area is the triangle PQR.

The examples Fig. 3–6 show that there can be various types of intersections for both cross intersection and coplanar intersection between the triangles. All other configurations are homeomorphic to the figures presented in this paper. For qualitative spatial reasoning, in some cases (when the knowledge of cross intersection is insufficient), we resort to coplanar intersection to distinguish the externally or tangentially connected objects.

SPECIALIZED INTERSECTION METHODS AND ALGORITHMS

Here we describe the conventional approach to triangle-triangle intersection in terms of two algorithms, one for cross intersection and one for coplanar intersection. Then we derive the composite algorithm for triangle-triangle intersection. First we give the supporting algorithms for special cases.

Single Point Intersection (0D)

We first analyze the vertices of the triangle PQR with respect to triangle ABC to determine if a vertex P or Q or R is common to the ABC triangle and conversely.

Input: A vertex X of triangle PQR and a triangle ABC
Output: return false if X lies outside ABC; otherwise, return true and in addition, classify if X is vertex, or edge interior or triangleInterior point.
PseudoCode:
vertex-triangleTest (X, tri = ABC)
To determine the relation of X ∈{P, Q, R} to the triangle ABC, we solve
A + u U + v V = X for $0 \le u, v, u + v \le 1$,
Rearranging the equation, we get
u U + v V = AX
Let N = U**x**V, then N**x**U and N**x**V are orthogonal to U and V. To eliminate one of the parameters u, v in this equation, we dot product the equation with vectors N**x**V and NxU .

Let $\gamma = \dfrac{AX \times (U \times V)}{(U \times V) \bullet (U \times V)}$

then u = - γ•V and v = γ•U
if $0 \le u, v, u + v \le 1$
    return true // X of PQR, intersects the triangle ABC.
else
    return false
/*end of algorithm*/

The vector $\dfrac{(U \times V)}{(U \times V) \bullet (U \times V)}$ is computed only once and used repeatedly. As a result $\gamma = \dfrac{AX \times (U \times V)}{(U \times V) \bullet (U \times V)}$ is calculated with one cross product, and u, v are calculated with one dot

product. The parameters u, v naturally lend themselves to classification of intersections. Similarly for $\gamma' = \frac{PX \times (S \times T)}{(S \times T) \bullet (S \times T)}$.

Classification of Intersection Point

In order to determine whether the vertex X of triangle PQR is a *vertex* of ABC, or on the *edge* of ABC, or an *interior point* of triangle ABC, no extra computational effort is required now. Logical tests are sufficient to establish the classification of this intersection. Since $0 \le u, v, u + v \le 1$, we can classify point X relative to ABC in terms of the following parametric coordinates:

***vertex* ((u, v)):** If $(u, v) \in \{ (0, 0), (0, 1), (1, 0)\}$, then X is one of the vertices of ABC. In particular, parameter (0,0) corresponds to parametric coordinates of vertex A, (1,0) corresponds to parametric coordinates of vertex B, and (0,1) corresponds to parametric coordinates of vertex C.

edge ((u, v))**:** If ($u = 0, 0 \le v \le 1$) or ($v = 0, 0 \le u \le 1$) or ($u + v = 1, 0 \le u \le 1$), then X is an edge of ABC. In particular ($u = 0, 0 \le v \le 1$) is the parametric representation of edge AC, ($v = 0, 0 \le u \le 1$) is the parametric representation of edge AB, ($u + v = 1, 0 \le v \le 1$) is the parametric representation of edge BC, and ($u + v = 1, 0 \le u \le 1$) is the parametric representation of edge CB.

edgeInterior ((u, v))**:** If ($u = 0, 0 < v < 1$) or ($v = 0, 0 < u < 1$) or ($u + v = 1, 0 < u < 1$), then X is on an edge of ABC, excluding vertices. In particular, ($u = 0, 0 < v < 1$) is the parametric representation of the interior of edge AC, ($v = 0, 0 < u < 1$) is the parametric representation of the interior of edge AB, ($u + v = 1, 0 \le v \le 1$) is the parametric representation of the interior of edge BC, and ($u + v = 1, 0 \le u \le 1$) is the parametric representation of the interior of edge CB.

***triangleInterior* ((u, v)):** If ($0 < u < 1$ and $0 < v < 1$ and $0 < u + v < 1$), then X is an interior point (excluding the boundary) of the triangle ABC.

Similarly, as above we can classify vertex X of triangle ABC as the *vertex, edgeInterior,* or *triangleInterior* point of triangle PQR. Single point intersection may result from cross intersection of edges as well. An edge point may be a vertex or an interior point of the edge.

The Edge-Edge Cross Intersection Point

If two triangles cross intersect across an edge, the edge-to-edge intersection results in a single point. The edge-edge cross intersection ***algorithm*** is presented below followed by the algorithm for edge-edge cross intersection with a single point classification.

Input: An edge of triangle ABC and an edge of triangle PQR

Output: Return false if the edges do not cross intersect; otherwise, return true and classify the intersection point.

***Pseudocode*:**

edge_edgeCrossIntersection (edge1, edge2)

Let the two edges be AB and PQ. Then the edges are represented with equations

$X = A + u\,U$ with $U = B - A$, $0 \le u \le 1$

$X = P + s\,S$ with $S = Q - P$, $0 \le s \le 1$

if $U \times S \bullet AP \neq 0$, return false //non - coplanar lines

elseif $U \times S = 0$, return false //lines are parallel

else UxS ≠0, // lines cross
/* solve for u_P, s_A values for the intersection point*/

$A + u_P U = P + s_A S$

$u_P U - s_A S = P - A$

$u_P U - s_A S = AP$

$$u_P = \frac{S \bullet PA \times (U \times S)}{(U \times S) \bullet (U \times S)}$$

$$s_A = \frac{U \bullet AP \times (U \times S)}{(U \times S) \bullet (U \times S)}$$

if ($u_P < 0$) or ($u_P > 1$), return false //no cross intersection,
elseif ($s_A < 0$) or ($s_A > 1$),
return false //no cross intersection,
else
return true and (u_P , s_A) //there is edge-edge cross intersection.
endif
/* end of algorithm*/

Edge-Edge Cross Intersection Single Point Classification

If u_m, s_m, be the pair of parametric coordinates of the 3D intersection of an edge of ABC and an edge of PQR.

If $u_m = 0$ or $u_m = 1$, the it is *vertex* of ABC
Elseif $0 < u_m < 1$, then it is edge interior of an edge of ABC
Else no edge-edge cross intersection.
If $s_m = 0$ or $s_m = 1$, the it is *vertex* of PQR
Elseif $0 < s_m < 1$, then it is *edgeInterior* of an edge of PQR
Else no edge-edge cross intersection.

Composite Classification Of Single Point Intersection

Here we represent the intersection point uniformly for both the algorithms. Let A_m, P_m, be the pair of bilinear parametric coordinates (u_m, v_m) and (s_m, t_m) of the 3D intersection points $R_1(u_m,v_m)$ and $R_2(s_m,t_m)$ with respect to triangles ABC and PQR, respectively. When there is no confusion, we will refer to the points as A_m and P_m instead of 3D points $R_1(u_m,v_m)$ and $R_2(s_m,t_m)$.
From vertex-triangle intersection, we have
P_m is a vertex of PQR, and $A_m = (u_m, v_m)$, where u_m and v_m are $u_m = -\gamma \bullet V$, $v_m = \gamma \bullet U$
or A_m is a vertex of ABC, and $P_m = (s_m, t_m)$, where s_m and t_m are $s_m = -\gamma' \bullet T$, $t_m = \gamma' \bullet S$
From edge-edge cross intersection, we have
$A_m = (u_m, v_m) = (0, u_P)$ or $(u_P, 0)$ or $(u_P, 1 - u_P)$ or $(1 - u_P, u_P)$
$P_m = (s_m, t_m) = (0, s_A)$ or $(s_A, 0)$ or $(s_A, 1 - s_A)$ or $(1 - s_A, s_A)$

If A_m is a vertex of ABC and P_m is vertex of PQR, then it is *vertex-vertex* intersection. If A_m is a vertex of ABC and P_m is an edgeInterior of PQR, then it is *vertex-edgeInterior* intersection. If A_m is an edgeInterior point of ABC and P_m is a vertex of PQR, then it is *edgeInterior-vertex* intersection. If A_m is an edgeInterior point of ABC and P_m is an edgeInterior point of PQR, then it is *edgeInterior-edgeInterior* intersection. If A_m is a vertex of ABC and P_m is an triangleInterior point of PQR, then it is *vertex-triangleInterior* intersection.

This completes the discussion of single point intersection classification and parameters for the corresponding 3D points.

Line Intersection (1D)

Besides edge-edge cross intersection, edge-edge collinear intersection is a possibility, independent of crossing or coplanar triangles. In this section we discuss algorithms that result in a segment (1D) intersection; see Fig. 4.

Intersection Algorithm And Parametric Coordinates

Here we derive an edge-edgeCollinear intersection algorithm. This algorithm is seamlessly applicable to both cross intersecting and coplanar triangles. The following algorithm implements intersection of edges of the triangles ABC and PQR.

Input: two line segments , edges from each triangle
Output: if they do not intersect, return false otherwise return true and the intersection segment, degenerate or non-degenerate
PseudoCode:
boolean edge-edgeCollinearTest (edge1, edge2)

First we compute the linear parameter coordinates u_P, u_Q, s_A, s_B for intersection of X = A + u (B - A), for X = P, Q and X = P + s (Q - P), for X = A, B. Similarly we can compute the intersection of other edges of triangle ABC with any edge of triangle PQR. Then we update the parameters for the common segment. This algorithm is standard, straightforward, follows quickly from the vertex-edge intersection algorithm, and hence is omitted.

Uniform Representation Of Edge-Edge Intersection

Again we first represent the intersection parameters uniformly as A_m= (u_m, v_m), A_M=(u_M, v_M) and P_m = (s_m, t_m), and P_M = (s_M, t_M). Now we have the linear coordinates for intersection points u_P, u_Q and s_A, s_B. We map the linear parameters for intersection points to bilinear parameter coordinates (u, v) and (s, t). If u_P, u_Q are known along an edge and the edge is AB, let $u_m = u_P$, $u_M = u_Q$, $v_m = 0$, $v_M = 0$. Similarly for AC, let $u_m = 0$, $u_M = 0$, $v_m = u_P$, $v_M = u_Q$; and for BC, let $u_m = u_P$, $u_M = u_Q$, $v_m = 1 - u_P$, $v_M = 1 - u_Q$.
Thus ABC triangle bilinear coordinates for the intersection points are:
A_m=(u_m, v_m), A_M=(u_M, v_M), where v_m=v_M = 0 or u_m=u_M =0 or u_m+v_m=u_M + v_M = 1
Similarly for the triangle PQR, the linear coordinates s_A, s_B of intersection translate into bilinear coordinates

$P_m = (s_m, t_m)$, $P_M = (s_M, t_M)$,where $t_m = t_M = 0$ or $s_m = s_M = 0$ or $s_m+t_m = s_M+t_M = 1$.
Now we have the bilinear parametric coordinates u, v, s, t for the intersection segment. The common 3D segment is denoted by $[R_1(A_m), R_1(A_M)]$ which is $[R_2(P_m), R_2(P_M)]$ or $[R_2(P_M), R_2(P_m)]$. It is possible that the intersection segment is equal to both edges, or it overlaps both edges, or it is entirely contained in one edge. Since the intersection is a part of the edges, it cannot properly contain any edge.

Composite Classification Of Line Intersection

For collinear edge intersection A_m and A_M are normally distinct and similarly P_m, P_M may be distinct. Though the intersection segment is given by $[R_1(A_m), R_1(A_M)] = [R_2(P_m), R_2(P_M))$ or $[R_1(A_m), R_1(A_M)] = [R_2(P_m), R_2(P_M)]$, it is not necessary that parameter coordinates $[A_m, A_M] = [P_m, P_M]$ or $[A_m, A_M] = [P_M, P_m]$. The predicate for edge-edge collinear intersection segment becomes
*edge-edgeCollinear (*edge1, edge2*)* = $[A_m, A_M] \subseteq$ edge1(ABC) and $[P_m, P_M] \subseteq$ edge2(PQR) and $[R_1(A_m), R_1(A_M)] == [R_2(P_m), R_2(P_M)]$ or $[R_1(A_m), R_1(A_M)] == [R_2(P_M), R_2(P_m)]$
Also it may be noted that for cross intersecting triangles, an *edge-triangleInterior* intersection may result in a segment intersection, see Fig. 4(b). For cross intersecting planes we have:
*edge-triangle (*edge, triangle*)* = $[A_m, A_M] \subseteq$ edge(ABC) and $[P_m, P_M] \subseteq$ triangle(PQR) and $[R_1(A_m), R_1(A_M)] == [R_2(P_m), R_2(P_M)]$ or $[R_1(A_m), R_1(A_M)] == [R_2(P_M), R_2(P_m)]$
This completes the classification of segment intersection (1D) in 3D for both *cross* and *coplanar* triangle intersections.

Philosophy Of Previous Cross Intersection Approaches

We briefly review three approaches for determining intersection detection that are relevant to our discussion herein, starting with Moller's algorithm (Mo¨ller, 1997). For two triangles T_1 and T_2, two planes P_1 and P_2 that support the triangles are determined. Then it is determined whether triangle T_1 and plane P_2 overlap, and triangle T_2 and plane P_1 overlap. If all vertices of triangle T_1 lie on the same side of plane P_2, and no vertex of triangle T_1 lies in plane P_2, then triangle T_1 and plane P_2 do not overlap; hence the triangles do not intersect. The same test is repeated for triangle T_2 and plane P_1. If the two tests succeed, then the algorithm computes the intersection segment S_{12} of triangle T_1 and plane P_2, as well as segment S_{21} of triangle T_2 and plane P_1. Further, a line-line 3D intersection algorithm is used to test whether the two segments S_{12} and S_{21} overlap; see Fig. 7(a). If they do intersect, the triangles are guaranteed to cross intersect. It should be noted that this algorithm does not address additional questions related to the classification of intersection points, information that would be useful for spatial reasoning.
The second approach (Held, 1997) reviewed here starts out similar to the above approach for solving the triangle-triangle intersection problem. If the above preliminary test succeeds, now instead of computing segment S_{21} of the intersection between triangle T_2 and plane P_1, the solution is approached slightly differently. It will be determined whether intersection segment S_{12} of triangle T_1 and plane P_2 overlap. If segment S_{12} and triangle T_2 overlap, the intersection algorithm succeeds, and the determination is made that the two triangles intersect; see Fig. 7(b). Unfortunately this approach has the same problems as the previous algorithm for degenerate cases.

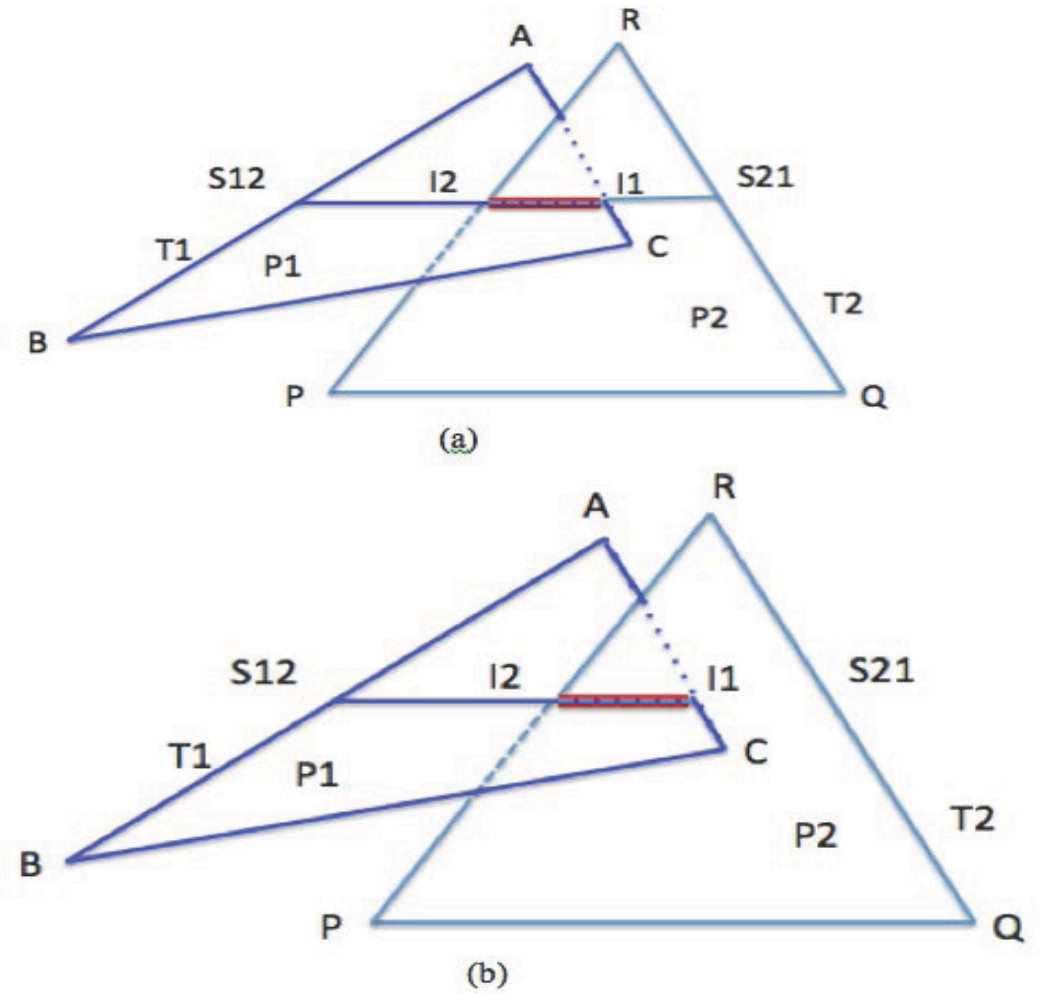

Figure 7. (a) S_{12} is the intersection of T_1 and P_2, S_{21} is the intersection of T_2 and P_1; intersection of S_{12} and S_{21} is the intersection of T_1 and T_2. (b) I_2 I_1 is the intersection S_{12} and P_2.

A third approach in (Didier, 1990) differs in its strategy from the previous two algorithms: if the edges of one triangle intersect the surface of the other triangle, it can be concluded that the triangles intersect. In this case, six such intersection tests are performed: three to test triangle ABC against the edges of triangle PQR, and three tests where the roles of ABC and PQR are reversed. This technique solves the six sets of linear equations associated with the problem and exploits the relations between these sets to speed up their solution. Like the other algorithms, it does not answer the related intersection classification questions. Notably, Badouel's algorithm (Didier, 1990) uses a vector approach in preference to Cartesian coordinates, yet still performs the same tests as the aforementioned algorithms.

Our intersection outcome is closer to that of (Held, 1997), but our approach is more direct and different from the previous strategies.

THE TRIANGLE-TRIANGLE INTERSECTION ALGORITHMS

Here we give the line intersection for cross intersection and area intersection for coplanar triangles separately and combine the two into one algorithm as is conventionally done. In the next section we present our new algorithm, which employs a more singular, seamless approach instead of combining the separate algorithms.

The Triangle-Triangle Line Intersection Algorithm

The cross intersection algorithm encompasses the single point and edge intersection cases. Here we give a solution which is *different* from previous approaches. The algorithm has been implemented in Python 3.3.3.

Algorithm for cross intersection, line intersection
Input: Two triangles ABC and PQR
Output: Determine if they cross intersect. Return true if they intersect, otherwise return false.
PseudoCode:
boolean crossInt (tr1 = ABC, tr2 = PQR)
The vector equations for two triangles ABC and PQR are

$R_1(u, v) = A + u\,U + v\,V,\ 0 \le u, v, u + v \le 1$

$R_2(s, t) = P + s\,S + t\,T,\ 0 \le s, t, s + t \le 1$

where U = B - A, V = C - A, and S = Q - P, T = R – P are direction vectors along the sides of triangles.
Let $N_1 = U \times V$, $N_2 = S \times T$ be normals to the planes supporting the triangles. The triangles intersect if there exist some barycentric coordinates (u, v) and (s, t) satisfying the equation

$A + u\,U + v\,V = P + s\,S + t\,T$

Since $N_1 \times N_2 \neq 0$ for cross intersecting triangles, and S and T are orthogonal to N_2, the dot product of this equation with N_2 eliminates S and T from the above equation to yield

$u\,U \bullet N_2 + v\,V \bullet N_2 = AP \bullet N_2$

This is the familiar equation of a line in the uv - plane for real variables u, v. The vector equation using real parameter l becomes

$$(u,v) = AP \bullet N_2 \frac{(U \bullet N_2, V \bullet N_2)}{U \bullet N_2^2 + V \bullet N_2^2} + \lambda (V \bullet N_2, -U \bullet N_2)$$

Then parameter values u, v are explicitly written as

$$u = AP \bullet N_2 \frac{U \bullet N_2}{U \bullet N_2^2 + V \bullet N_2^2} + \lambda V \bullet N_2$$

$$v = AP \bullet N_2 \frac{V \bullet N_2}{U \bullet N_2^2 + V \bullet N_2^2} - \lambda U \bullet N_2$$

$$u + v = AP \bullet N_2 \frac{(U \bullet N_2 + V \bullet N_2)}{U \bullet N_2^2 + V \bullet N_2^2} + \lambda (V \bullet N_2 - U \bullet N_2)$$

If there is a λ satisfying three equations such that $0 \le u, v, u + v \le 1$, then the triangles are ensured to intersect. The solution in λ is the range of values of λ satisfying $\lambda_m \le \lambda \le \lambda_M$ for some λ_m and λ_M. This detects whether the two triangles cross intersect only.

In fact, for precise intersection, using λ_m, λ_M, as parameter values, we compute (u_m, v_m) from λ_m and (u_M, v_M) from λ_M for the end points of the segment of intersection in triangle ABC. Similarly the values (s_m, t_m) and (s_M, t_M) represent the segment of intersection in triangle PQR.

There are two ways to solve this problem. The first approach is to compute (s_m, t_m) and (s_M, t_M) from scratch. The precise intersection between the two triangles is the common segment of these two segments (Mo¨ller, 1997), (Sabharwal & Leopold, 2013). The second approach is to compute (s_m, t_m) using (u_m, v_m), and (s_M, t_M) using (u_M, v_M) in the main equation. This can be simplified, and we can compute the sub-segment of the precomputed segment for triangle ABC (Held, 1997). If the segment degenerates into a single point, the parameter values also can be used to classify the intersection as a vertex, an *edgeInterior* point or *triangleInterior* point in the triangles ABC and PQR.

The Triangle-Triangle Area Intersection Algorithm

For coplanar triangles only, the area intersection is possible. The input, output, method prototype and pseudo code are given below.
Algorithm for area intersection
Input: Two triangles ABC and PQR coplanar
Output: If they do not intersect, then return false; otherwise, return true and the intersection area.
PseudoCode:
boolean coplanarInt (tr1 = ABC, tr2 = PQR)
For coplanar triangles, there may be no intersection (Fig. 2), a single point (Fig. 3(a, b)), a segment (Fig. 4(a)) or an area (Fig. 5, Fig. 6(a, b, c)), including one triangle contained in another, (Fig. 6(d)). An area can result from two edges of one triangle and one, two, or three edges of another triangle, or three edges from both triangles creating a star shaped figure. The resulting area is bounded by 3, 4, 5, or 6 edges. All other configurations are homeomorphic to the figures presented earlier in this paper. The derivation of the algorithm is based on extensive use of the intersections of the edges of one triangle with the edges of the second triangle and the collection of the relevant interaction points to form the area if any (Guigue & Devillers, 2003). If the triangles ABC and PQR are coplanar and a vertex of PQR is in the interior of ABC (or the converse is true), then an area intersection occurs, (Fig. 5(a, b), Fig. 6(a,b,c,d)). If no two edges intersect and *vertex_triangleInterior (*vertex, triangle = tr2) is true for every vertex of a triangle tr1, then the triangle tr1 is contained in tr2, and conversely. If no *edge-edge* intersection takes place and no vertex of one triangle is inside the other triangle (or the converse is true), then they do not intersect, and hence they are disjoint.

Some of the previous methods may use edge-oriented techniques to determine the area of intersection; however, those will be lacking classification capability (Guigue & Devillers, 2003).

The Triangle-Triangle Traditional Composite Intersection Algorithm

A classical approach is to first determine whether two triangles are cross or coplanar, and then apply an appropriate algorithm.
Algorithm: for intersection between coplanar intersetion
Input: Two triangles ABC and PQR
Output: If they do not intersect, then return false; otherwise, return true and the intersection.
PseudoCode:
boolean compositeInt (tr1 = ABC, tr2 = PQR)
The vector equations for two triangles ABC and PQR are

$R_1(u, v) = A + u\,U + v\,V,\ 0 \le u, v, u + v \le 1$
$R_2(s, t) = P + s\,S + t\,T,\ 0 \le s, t, s + t \le 1$

where U = B - A, V = C - A, and S = Q - P, T = R – P are direction vectors along the sides of triangles.

```
if (UxV )x(SxT) ≠ 0 // planes supporting triangles are not parallel
        if crossInt (tr1, tr2) // cross intersect the triangles
                return true
        else
                return false
elseif (UxV )x(SxT) = 0, // triangles planes are parallel
                if AP•(UxV ) = 0, //the triangles are coplanar
                        if coplanarInt (tr1, tr2)
                                return true
                        else
                                return false
                elseif AP•(UxV ) ≠ 0, //the triangles are not coplanar,
                        no Intersection
                        return false
                endif
endif
/*end of algorithm*/
```

GENERIC ALGORITHM FOR TRIANGLE-TRIANGLE INTERSECTION

We now present an algorithm that is more comprehensive, robust, and analytically rigorous; it is implicitly capable of handling any specific type of intersection, which may be a single point, a segment, or an area. The triangles may be coplanar or crossing. This single algorithm is not a modification of any previous algorithm, but rather a new approach different from the other strategies. Even existing methods that use somewhat similar alternate edge-oriented techniques to determine the area of intersection (Guigue & Devillers, 2003) are much more limited than what we present.

Algorithm **The generic algorithm for intersection between arbitrary triangles.**
Input: Two triangles ABC and PQR in 3D, triangles may be coplanar or crossing
Output: If the triangles do not intersect, return false; otherwise, return true and the intersection, which may be single point, a line segment, or an area.
PseudoCode:
boolean triTriIntersection (tr1 = ABC, tr2 = PQR)
The triangles ABC and PQR are
$X = A + u\,U + v\,V$ with $U = B - A$, $V = C - A$, $0 \le u, v, u + v \le 1$
$X = P + s\,S + t\,T$ with $S = Q - P$, $T = R - P$, $0 \le s, t, s + t \le 1$
The general set up for detecting intersections is to solve the equation
$A + u\,U + v\,V = P + s\,S + t\,T$
with $0 \le u, v, u + v, s, t, s + t \le 1$,
Rearranging the equation, we have
$u\,U + v\,V = AP + s\,S + t\,T$ (1)

This is an underdetermined system of equations involving four parameters u, v, s, t and three equations in x, y, and z- coordinates of 3D vectors. This means one of the parameters can be arbitrarily assigned. The other three parameters can be determined in terms one parameter provided the system is consistent; otherwise, no solution can be found due to inconsistency. We need an additional constraint to have a valid solution. Here the parameters u, v, s, t are bounded such that $0 \le u, v, u+v \le 1$ and $0 \le s, t, s+t \le 1$. In addition, the solution for u, v, s, and t must be consistent. This may be confirmed by checking that the (u, v)-intersection and the (s, t)-intersection correspond and satisfy the equation. It can be quickly determined that the point X = P + s S + t T lies in the uv-plane by checking the dot product AX•UxV=0, and vice versa.

For simplicity in solving (1), we use the following notation.

Let $\delta = (U\mathbf{x}V)\bullet(U\mathbf{x}V)$ and AP = P - A be a vector. The vectors α, β, γ are explicitly defined as

$$\alpha = \frac{S \times (U \times V)}{\delta}, \beta = \frac{T \times (U \times V)}{\delta}, \gamma = \frac{AP \times (U \times V)}{\delta}$$

For intersection between triangles ABC and PQR, dot equation (1) with $(U\mathbf{x}V)\mathbf{x}V$ and $(U\mathbf{x}V)\mathbf{x}U$, we quickly get u and v as

$u = -(\gamma\bullet V + s\, \alpha\bullet V + t\, \beta\bullet V)$

$v = \gamma\bullet U + s\, \alpha\bullet U + t\, \beta\bullet U$

Adding the two equations,

$u + v = \gamma\bullet (U - V) + s\, \alpha\bullet (U - V) + t\, \beta\bullet(U - V)$

The constraint $0 \le u, v, u + v \le 1$ yields three inequalities in parameters s and t

(a) $-\gamma\bullet U \le \alpha\bullet U\, s + \beta\bullet U\, t \le 1 - \gamma\bullet U$

(b) $-1 - \gamma\bullet V \le \alpha\bullet V\, s + \beta\bullet V\, t \le -\gamma\bullet V$

(c) $-\gamma\bullet (U - V) \le \alpha\bullet(U - V)\, s + \beta\bullet(U - V)\, t \le 1 - \gamma\bullet(U - V)$

These inequalities (a) - (c) are linear in s and t and are of the general form

$m \le ax + by \le n$

$M \le Ax + By \le N$

We developed a robust technique to solve a pair of inequalities in an earlier section. The robust solution to this system of inequalities is derived via two functions

solve_x (m,a,b,n,M,A,B,N, x_m, x_M) and

solve_y (m,a,b,n,M,A,B,N, x, y_m, y_M)

This method of solution can be also applied to three linear inequalities. We apply these methods here in solving the inequalities pairwise (a),(b); (a),(c); and (b),(c).

Let $s_m = 0$, $s_M = 1$

If solve_x ($-\gamma\bullet U, \alpha\bullet U, \beta\bullet U, 1 - \gamma\bullet U, -1 - \gamma\bullet V, \alpha\bullet V, \beta\bullet V, -\gamma\bullet V, x_m, x_M$) // (a), (b)

$s_m = \max(s_m, x_m)$, $s_M = \min(s_M, x_M)$

If solve_x ($-\gamma\bullet U, \alpha\bullet U, \beta\bullet U, 1 - \gamma\bullet U, -\gamma\bullet(U - V), \alpha\bullet (U - V), \beta\bullet(U - V), 1 - \gamma\bullet(U - V), x_m, x_M$) // (a), (c)

$s_m = \max(s_m, x_m)$, $s_M = \min(x_M, s_M)$

If solve_x ($-1 - \gamma\bullet V, \alpha\bullet V, \beta\bullet V, -\gamma\bullet V, -\gamma\bullet(U - V), \alpha\bullet(U - V), \beta\bullet(U - V), 1 - \gamma\bullet(U - V), x_m, x_M$) // (a), (c)

$s_m = \max(s_m, x_m)$, $s_M = \min(x_M, s_M)$

if $s_m > s_M$

return false

else

for $s \in [s_m, s_M]$ // we solve the (a)-(c) inequalities for t

Let $t_m(s) = 0$; $t_M(s) = 1$;

if solve_y($-\gamma \bullet U, \alpha \bullet U, \beta \bullet U, 1-\gamma \bullet U, -1-\gamma \bullet V, \alpha \bullet V, \beta \bullet V, -\gamma \bullet V, s, y_m, y_M$) //(a), (b)

$t_m(s) = \max(t_m(s), y_m)$, $t_M(s) = \min(t_M(s), y_M)$

if solve_y($-\gamma \bullet U, \alpha \bullet U, \beta \bullet U, 1-\gamma \bullet U, -\gamma \bullet (U-V), \alpha \bullet (U-V), \beta \bullet (U-V), 1-\gamma \bullet (U-V), s, y_m, y_M$) //(a), (c)

$t_m(s) = \max(t_m(s), y_m)$, $t_M(s) = \min(t_M(s), y_M)$

if solve_y($-1-\gamma \bullet V, \alpha \bullet V, \beta \bullet V, -\gamma \bullet V, -\gamma \bullet (U-V), \alpha \bullet (U-V), \beta \bullet (U-V), 1-\gamma \bullet (U-V), s, y_m, y_M$) //(b), (c)

$t_m(s) = \max(t_m(s), y_m)$, $t_M(s) = \min(t_M(s), y_M)$,

if $t_m(s) > t_M(s)$

return false, $s_m = s$;

else

$t_m(s) \le t \le t_M(s)$

return true

/* end of algorithm */

We first solved the three inequalities pairwise for a range of values for s, so that $s_m \le s \le s_M$ holds good simultaneously with three inequalities. Then from this range of s values, we solved for t(s) values as a function of s such that $t_m(s) \le t(s) \le t_M(s)$. There is no closed form function as such, it is a numerical solution to t(s) for each s. Thus if $s_m = s_M$ and if $t_m(s) = t_M(s)$, t(s) is constant, it results in a single point; otherwise, it is a line segment. Also if $s_m < s_M$ and $t_m(s) = t_M(s)$ for each s, it is a line segment. If $s_m < s_M$ and $t_m(s) < t_M(s)$ for some s, then it is an area intersection. The parametric bounding box for overall intersection is $[s_m, s_M] \times [t_m, t_M]$ where $t_m = \min\{ t_m(s): s_m \le s \le s_M\}$, and $t_M = \max\{ t_M(s): s_m \le s \le s_M\}$. The bilinear parameter coordinates for range of parameters are denoted by $p_m = (s_m, t_m)$, $p_M = (s_M, t_M)$ where the corresponding 3D points are denoted by $P_m = P + s_m S + t_m(s_m) T$, and $P_M = P + s_M S + t_M(s_M) T$. In general, $P([s_m, s_M], t(s)) = P + s S + t(s) T$. This discussion may be summarized and the intersection points can be classified as follows:

if the algorithm returns false,

No Intersection

elseif ($s_m = s_M$ and ($t_m(s) = t_M(s)$ for $s_m \le s \le s_M$)

Single Point Intersection

elseif ($s_m = s_M$ or ($t_m(s) = t_M(s)$ for $s_m \le s \le s_M$)

Line segment intersection common to two triangles

else

Area Intersection common to two triangles

This will implicitly cover the case when a triangle is inside the other triangle as well. If triangles do not intersect, then the triangles are declared *disjoint*. This completes the discussion of the generic algorithm for intersection between triangles.

If required, we similarly can determine (u, v)-parameter values corresponding to triangle ABC. This algorithm detects whether the triangles intersect regardless of crossing or coplanar triangles, and we classify the intersection as a *vertex, edge-interior,* or *triangle-interior* point.

This algorithm may be used with any application (e.g., qualitative spatial reasoning, surface modeling, image processing etc.). The algorithm determines whether intersection exists or not (i.e., it returns true or false). If true, the parameter coordinates of intersection are readily available. Now we can derive all the classification information from the parametric coordinates; only logical tests are required for classification of the intersections. It is not the intent of this

algorithm to determine whether the triangles are crossing or coplanar. This can be quickly determined as follows: if UxV•SxT ≠0, then the triangles cross; otherwise, the triangles are parallel, or if AP•UxV = 0 or AP•SxT = 0, then the triangles are coplanar.

In order to determine whether an intersection point X=(u,v) is a *vertex* of ABC, or on the *edge* of ABC, or an *interior point* of triangle ABC, no extra computational effort is required now. Logical tests are sufficient to establish the classification of this intersection.

EXPERIMENTAL RESULTS

There are four possibilities for triangle-triangle intersection: no intersection, single point intersection, line segment intersection, and area intersection. There are *three* types of no intersection: parallel noncoplanar, parallel coplanar, and non-parallel triangles. There are *six* types of single point intersection: vertex-vertex parallel, vertex-vertex crossing, vertex-edgeInterior parallel, vertex-edgeInterior crossing, vertex-triangleInterior crossing, and edgeInterior-edgeInterior crossing. There are *four* types of line intersection: edge-edge parallel, edge-edge crossing, edge-triangleInterior crossing, and triangleInterior-triangleInterior crossing. Area intersection occurs only when the triangles are coplanar. There are *nine* types of area intersection: no edges intersect (0edge-0edge intersection), edge-2edge intersection, edge-3edge, 2edge-2edge, 2edge-3edge, and 3edge-3edge intersection (four cases). The converse cases are considered because the converse can be obtained by simply interchanging the parameters. Thus in all there are twenty-two triangle-triangle versions that could be considered for experiments.

The algorithm is implemented in MacPython 3.3.3. The test time results are obtained via the Python *timeit* utility. Time tests were performed on Apple Macintosh OS X processor (2.2 GHz intel Core i7). The average time for no intersection, single point intersection(0D), a line segment intersection (1D) and an area intersection (2D) intersection were measured in seconds. The program was executed 100 times on each of the twenty two sample triangle pairs. The intersection times shown in Table 1 are neither optimized nor hardware embedded, they also include classification of intersections. Times were shown as proof of concept that this single integrated algorithm works reliably on all triangle pairs. The test data domain consists of synthetic triangles that have every possible type of intersection. For example, the times for no intersection were averaged over 100 runs with three samples. Similarly single point intersection six sample pairs were averaged over 100 runs, then four samples were used for line segment and averaged over 100 runs, and finally nine sample triangle pairs were used and averaged. The composite average intersection time was computed. The test time statistics are displayed in Table 1. We also give three sample run output figures for examples of single point intersection, a line intersection and an area intersection. The Matlab software is used to draw the figures, see Fig. 8-10. The Matlab user interface allows the user to select any of the possible triangle-triangle pairs for intersections and it displays the corresponding triangles and the intersection. For the sake of space consideration, one of the 0D (single point) intersections, (see Fig. 8), one of the 1D line segment intersections, (see Fig. 9), and one of 2D area intersections, (see Fig. 10), are displayed here.

TABLE 1. EXECUTION AVERAGE TIMES OF ALGORITHM IN SECOND

Type of Intersection	Time in Seconds
No Intersection	0.000073
Single Point	0.000531
Line Segment	0.001483
Area	0.001962
overall	0.001353

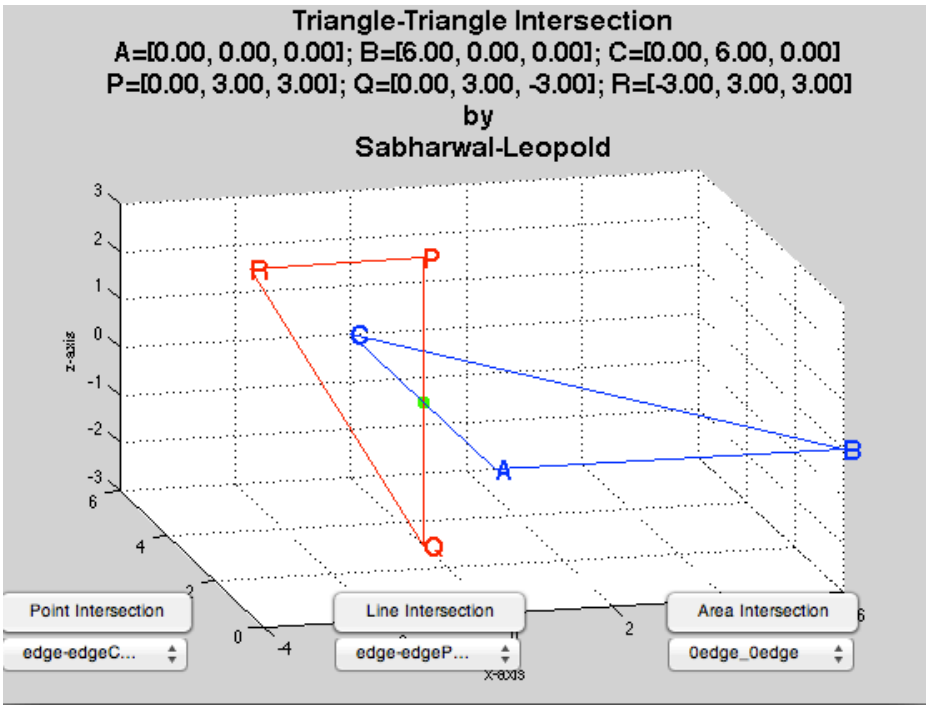

Fig. 8. Single Point Intersection

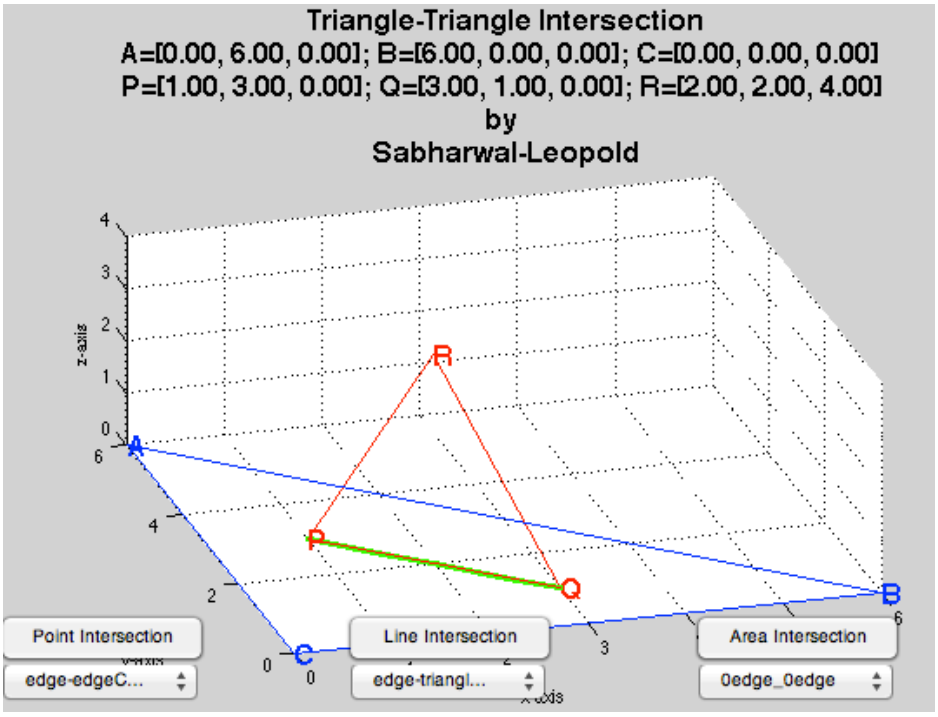

Fig. 9. Line Segment Intersection

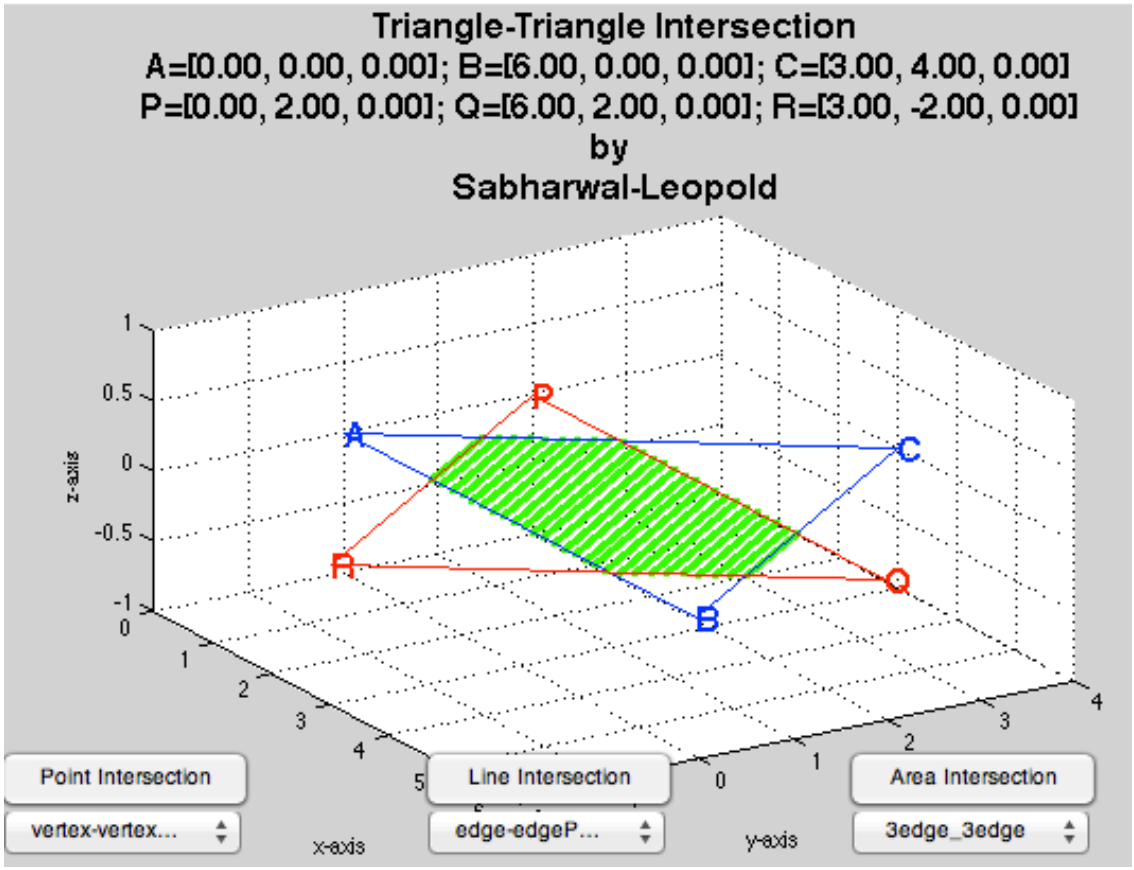

Fig. 10. Area Intersection

APPLICATIONS OF TRIANGLE-TRIANGLE INTERSECTION

Here we describe two of the applications where triangle-triangle intersection is used extensively: qualitative spatial reasoning and geometric modeling. It is not limited to these two applications; other applications include virtual reality, computer vision, and data mining (Lum, Singh, Lehman, Ishkanov, Vejdemo-Johansson, Alagappan, Carlsson & Carlsson, 2013).

Qualitative Spatial Reasoning

Qualitative Spatial Reasoning relies on intersections between objects whose boundaries are triangulated. The spatial relations are determined by the 9-Intersection/4-Intersection model (Egenhofer & Franzosa, 1991), (Sabharwal & Leopold, 2011). That is, for any pair of objects A and B, the interior-interior intersection predicate, IntInt(A, B), has true or false value depending on whether the interior of A and the interior of B intersect without regard to precise intersection. Similarly, IntBnd(A, B) represents the truth value for the intersection of the interior of A and the boundary of B, and BndBnd(A,B) represents the predicate for the intersection of the boundaries of A and B. These four qualitative spatial reasoning predicates are sufficient to define RCC8 spatial relations (Sabharwal & Leopold, 2011).

For spatial reasoning, we classify pairwise intersection based on the predicates IntInt(A,B) (intersection of Interior of object A and Interior of object B), IntBnd(A,B) (intersection of Interior of object A and Boundary of object B), BndInt(A,B), and BndBnd(A,B), without computing the precise extent of intersections. When cross intersection is insufficient to determine tangential intersection, some applications such as RCC8 and VRCC-3D$^+$ resort to coplanar intersection to support relations such as externally connected (EC) and tangentially connected (TPP, TPPconverse). The precise intersection of coplanar triangles is a little more complex because it can result in area intersection as well.

In the application VRCC-3D+ (Sabharwal, Leopold, and Eloe, 2011), the boundary of an object is already triangulated; that is, we will need to intersect pairs of only triangles. To reduce the computational complexity, the algorithm uses axis-aligned bounding boxes (AABB) to determine the closest triangles which may possibly intersect (Sabharwal, et al., 2011). For example, for objects A and B, if bounding boxes for triangles of A are disjoint from bounding boxes for triangles of B, either A is contained in B (BndInt, IntInt is true), or B is contained in A (IntBnd, IntInt is true), or A is disjoint from B. The test for such containment of objects can be designed by casting an infinite ray through the centroid of A. If the ray intersects B an odd number of times, then B is contained in A or A is contained in B. If A is not contained in B and B is not contained in A, then A and B are disjoint (i.e., IntInt(A, B), IntBnd(A, B), BndInt(A, B), and BndBnd(A, B) are all false).

Without the knowledge of BndBnd(A,B), BndInt(A,B), IntBnd(A,B), calculation of IntInt(A,B) is too costly. On the other hand, with this prior knowledge, it becomes quite inexpensive as the odd parity can be used for quick IntInt detection.

If the triangles cross intersect (e.g., *triangleInterior–triangleInterior* is true), then IntInt, IntBnd, BndInt, BndBnd will be true. However if the triangles are coplanar and intersect, only BndBnd(A, B) is true and IntInt(A, B), IntBnd(A, B), BndInt(A, B) are false for the objects; otherwise, BndBnd(A, B) is also false.

It is possible that two triangles cross intersect in a line segment even when a triangle is on one side of the other triangle, so *edgeInterior–triangleInterior* is true. In that case, it may be desirable to know which side of the other triangle is occupied. In Fig. 3(b), the triangle PQR is on the positive side of triangle ABC. For example, if triangle1 of object A cross intersects the negative side of triangle2 of object B, then BndInt(A, B) is true.

For qualitative spatial reasoning, in some cases (when the knowledge of cross intersection is insufficient), we resort to coplanar intersection to distinguish the externally or tangentially connected objects. Some applications may even require the area of coplanar triangle-triangle intersection. Since intersection area is enclosed by 3, 4, 5,or 6 sides, it can be triangulated into 1, 2, 3, or 4, triangles, the intersection area will be the sum of the areas of the composing triangles.

Table 2 is a characterization of the intersection predicates, which subsequently can be used to resolve the eight RCC8 relations. Here we assume all normals are oriented towards the outside of the object. Each characterization in Table 2 describes when the associated predicate is true. If the truth test fails, then other triangles need to be tested. If no pair of triangles results in a true value, then the result is false.

TABLE 2. CHARACTERIZATION OF INTERSECTION PREDICATES

BndBnd	At least one pair of triangles (cross or coplanar) intersects.
BndInt	At least one pair tr1 and tr2 intersect, at least one vertex of tr1 is on the negative side of triangles of object 2. Or object 1 is contained inside object2, i.e. every vertex of object1 is on the negative side of triangles of object 2.
IntBnd	At least one pair tr1 and tr2 intersect, at least one vertex of tr2 is on the negative side of triangles of object 1. Or object 2 is contained inside object1, i.e. every vertex of object2 is on the negative side of triangles of object 1.
IntInt	At least one pair of triangles cross intersects (triangleInterior-triangleInterior) Or an object is contained in the other.

This characterizes the intersection predicates that help in resolving the RCC8 relations.

Geometric Modeling

In geometric modeling, the surface-surface intersection problem occurs frequently. In CAD/CAM, the objects are represented with enclosing surface boundaries using the ANSI (Brep) model. Intersection between 3D surfaces amounts to detecting and computing intersection between 3D triangles. Briefly, a pair of surfaces is subdivided using axis-aligned bounding boxes (AABB) until the surfaces are reasonably planar and bounding boxes intersect. Then the plane surfaces are triangulated and the triangles are tested for cross-intersection and the intersection computed. The intersection segments are linked together to form curves of surface-surface intersection. The curves may be open curves, 'flares' or closed curves, 'loops' or even a combination of both (Houghton et al.,1985). The triangle-triangle intersection is required for such applications. The algorithm presented here is extremely useful for geometric modeling where intersection between objects occurs thousands of times. For geometric applications cross intersection is most often used to obtain the line segment of intersection. Detailed analysis and implementation of the most comprehensive surface/surface intersection algorithm may be found in (Houghton et al., 1985).

CONCLUSION

Triangle-Triangle intersection plays a prominent role in applications such as computer vision and spatial reasoning. Herein we presented a single algorithm for the complete design and a robust implementation of a complete framework for determining and characterizing the triangle-triangle intersections. In contrast to other combined algorithms, this approach is a generic technique to detect any type of cross or coplanar intersection using only one algorithm. The algorithm is independent of whether the triangles cross or are coplanar. The classification of

intersections is based on logical tests on parametric coordinates rather than computational arithmetic tests in Cartesian coordinates. Thus our algorithm not only detects whether or not an intersection exists, but also classifies and computes the intersections as a single point, a line segment, or an area, whichever the case may be. The algorithm provides more information than required by applications such as spatial reasoning systems. Consequently, we hope the new ideas and additional information including classification of 3D intersection presented herein will be useful for a variety of spatial-based applications.

REFERENCES

Caumon, G.; Collon - Drouaillet P, Le Carlier de Veslud C, Viseur S, Sausse J (2009) Surface - based 3D modeling of geological structures. *Math Geosci* 41:927–945.

Didier, Badouel,(1990), An Efficient Ray - Polygon Intersection, *Graphics Gems (Andrew S. Glassner, ed.)*, Academic Press, pp. 390 - 393.

Egenhofer, M. J.; R.G. Golledge, (1998), Spatial and Temporal Reasoning in Geographic Information Systems, *Oxford University Press*, USA.

Egenhofer,Max J.; R. Franzosa, (1991), Point-Set topological Relations, International Journal of *Geographical Information Systems* 5(2), pp. 161 - 174.

Eloe, N. , J.L. Leopold, C.L. Sabharwal, and Z. Yin, (2012), "Efficient Computation of Boundary Intersection and Error Tolerance in VRCC-3D+ ", *Proceedings of the 18h International Conference on Distributed Multimedia Systems (DMS'12), Miami, FL*, Aug. 9 - 11, 2012, pp. 67 – 70.

Elsheikh,Ahmed H.; Mustafa Elsheikh, (2012), A reliable triangular mesh intersection algorithm and its application in geological modeling, *Engineering with Computers*, pp.1 - 15.

Guigue, P; Devillers O, (2003), Fast and robust triangle - triangle overlap test using orientation predicates. *Journal of GraphicsTools* 2003; **8** (1): pp. 25–42.

Held M., (1997); ERIT a collection of efficient and reliable intersection tests. *Journal of Graphics Tools* 1997; 2(4): pp. 25–44.

Houghton, E.G. ; Emnett R.F., Factor J.D. and Sabharwal C.L.,(1985), Implementation of A Divide and Conquer Method For Surface Intersections*; Computer Aided Geometric Design* Vol.2, pp. 173 - 183.

Lum, P. Y. ;G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson & G. Carlsson,(2013) Extracting insights from the shape of complex data using topology, *SCIENTIFIC REPORTS* | 3 : 1236 | DOI: 10.1038/srep01236.

Mo¨ller, T.,(1997), A fast triangle - triangle intersection test. *Journal of Graphics Tools*, 1997; 2(2): 25–30.

Randell, D. A. ; Z. Cui, and A.G. Cohn, (1992), A Spatial Logic Based on Regions and Connection., *KR, 92*, pp. 165–176, 1992.

Sabharwal, C.L.; and J.L. Leopold,(2011), "Reducing 9-Intersection to 4-Intersection for Identifying Relations in Region Connection Calculus", *Proceedings of the 24th International Conference on Computer Applications in Industry and Engineering* (CAINE 2011), Honolulu, Hawaii, Nov. 16-18, 2011, pp. 118-123.

Sabharwal, C.L.; J.L. Leopold, and Nathan Eloe, (2011), "A More Expressive 3D Region Connection Calculus", *Proceedings of the 2011 International Workshop on Visual Languages and Computing (in conjunction with the 17th International Conference on*

Distributed Multimedia Systems (DMS'11))), Florence, Italy, Aug. 18-20, 2011, pp. 307-311.

Sabharwal, Chaman L; and Jennifer L Leopold,(2013), "A Fast Intersection Detection Algorithm For Qualitative Spatial Reasoning", *Proceedings of the 19h International Conference on Distributed Multimedia Systems* (DMS'13), Proceedings of the 2013 *International Workshop on Visual Languages and Computing* (VLC 2013), Brighton, United Kingdom, Aug. 8-10, 2013, pp. 145-149.

Sabharwal, Chaman; Jennifer Leopold, and Douglas McGeehan,(2013): Triangle-Triangle Intersection Determination and Classification to Support Qualitative Spatial Reasoning, Polibits, *Research Journal of Computer Science and Computer Engineering with Applications* Issue 48 (July–December 2013), pp. 13–22.

Tropp, Oren; Ayellet Tal, Ilan Shimshoni,(2006), A fast triangle to triangle intersection test for collision detection, *Computer Animation and Virtual Worlds*, Vol17 (50), pp.527 - 535.