A provably robust algorithm for triangle-triangle intersections in floating-point arithmetic

Mccoid, Conor Joseph; Gander, Martin Jakob

A provably robust algorithm for triangle-triangle intersections in floating-point arithmetic

CONOR MCCOID and MARTIN J. GANDER, University of Geneva, Switzerland

Motivated by the unexpected failure of the triangle intersection component of the Projection Algorithm for Nonmatching Grids (PANG), this article provides a robust version with proof of backward stability. The new triangle intersection algorithm ensures consistency and parsimony across three types of calculations. The set of intersections produced by the algorithm, called representations, is shown to match the set of geometric intersections, called models. The article concludes with a comparison between the old and new intersection algorithms for PANG using an example found to reliably generate failures in the former.

CCS Concepts: • **Mathematics of computing** → Graph enumeration; *Computation of transforms*; • **Software and its engineering** → **Consistency**; **Software reliability**; • **Computing methodologies** → **Mesh models**.

Additional Key Words and Phrases: mesh intersection, advancing front algorithms, non-matching grids, polygon clipping, floating-point arithmetic, robustness

ACM Reference Format:
Conor McCoid and Martin J. Gander. 2021. A provably robust algorithm for triangle-triangle intersections in floating-point arithmetic. 1, 1 (June 2021), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In 2013 Gander and Japhet [11, 12] presented an algorithm to resolve a mesh intersection problem (also known as intergrid communication problem or grid transfer problem [12]) for two triangular or tetrahedral meshes. The algorithm is named PANG (Projection Algorithm for Nonmatching Grids). An important component of this algorithm is the calculation of the polygon representing the intersection of two triangles in 2D, hereafter referred to as the polygon of intersection. This component is the focus of this article. The algorithm also makes use of an advancing front algorithm to reduce the cost of computing the intersection of arbitrary meshes to linear complexity. A similar advancing front algorithm was independently developed in [8].

Calculating the intersection of two polygons is well-studied as the polygon clipping problem, primarily in the field of computer graphics where one wishes to know when a given 'subject' polygon U is hidden from an observer by a second 'clipping' polygon V. Speed is valued over accuracy in these applications and glitches of varying severity are commonplace in video games and computer-generated imagery.

A number of algorithms have been proposed for the intersection of polygons. One may classify them by considering how they handle each of the three types of vertices found in the polygon of intersection: vertices of U lying inside V; the intersections between the edges of U and those of V; and the vertices of V lying inside U.

Authors' address: Conor McCoid, conor.mccoid@unige.ch; Martin J. Gander, martin.gander@unige.ch, University of Geneva, Rue du Conseil-Général 7-9, 1205 Geneva, Switzerland.

XXXX-XXXX/2021/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Some algorithms divide the plane into sections based on the edges of V [16, 18, 19, 22]. Each edge defines an infinite reference line of which the edge is a finite interval. This line then defines a parameter that is positive on one side and negative on the other. If all parameters associated with all edges are positive for a vertex of U then it lies inside V.

An alternative approach is to consider the projection of vectors between vertices and edges and the respective normal vectors [6, 21]. The goal is to find a vector projection whose sign indicates whether a vertex is in V. The calculation of an intersection would then reduce to finding where the projection is zero.

A third option is to trace out the polygon of intersection [13, 25]. One computes all intersections between all edges of U and V, then chooses an intersection and marches along the vertices of U, adding each to the polygon of intersection. Once this trace reaches another intersection between U and V it switches to march along the vertices of V. This is repeated until the first intersection is reached. Such a procedure encounters problems when U and V share vertices [12].

One can reduce the cost of calculating the intersections between U and V by first determining which edges intersect. This can be done on an edge-by-edge basis [16, 21, 22] or considering the polygons as a whole [13, 18].

In computer graphics, the polygon clipping problem often reduces to a line clipping problem [6, 16, 21]. As such, vertices of V lying inside U are often irrelevant. These can be dealt with by repeating the process for vertices of U inside V, swapping U and V. Algorithms using a trace procedure [13, 25] make no distinction between U and V and find such vertices in the same manner as those of U in V.

The Sutherland-Hodgman algorithm [22] has a unique method for finding vertices of V in U. The algorithm takes a given edge of V and finds the corresponding reference line, defining a positive side of the line which contains V and a negative side which does not. It then discards all vertices of U on the negative side of the line and calculates intersections with the line for each edge of U that had one of its vertices removed in this way. The result is a new polygon lying entirely on one side of the reference line. The process is repeated with a new edge of V until all edges of V have been used. In this way, a vertex of V in U is the last of a sequence of intersections of the edges of intermediary polygons with reference lines extending from edges of V.

The triangle intersection algorithm in PANG from 2013 [12] favours vector projections for finding each type of vertex of the polygon of intersection, similar to the Cyrus-Beck algorithm [6]. The authors in [12, Remark B.1] stated for the floating-point arithmetic implementation of the triangle-triangle intersection part of PANG that

> "[i]n all the tests over the past years, the routine above never failed to compute the correct intersection. There is however no proof using a detailed floating-point analysis of this."

On February 21st 2019, Jerónimo Rodriguez Garcia from the Universidade de Santiago de Compostela sent an example found by his PhD student Jorge Albella Martínez to the second author of the present article where the floating-point implementation of the intersection algorithm part of PANG failed.

The left of Figure 1 shows this example: The algorithm identifies two vertices of the blue triangle as being inside the red triangle (apex and left base). As well, PANG identifies the left base vertex of the red triangle as being inside the blue triangle, as the two left base vertices are coincident. However, PANG then calculates only one intersection: the left base vertex of both triangles. As only two points are found for the intersection, no volume is calculated. The right of Figure 1 shows that a slight perturbation in all vertices of the small triangle leads to the correct result.

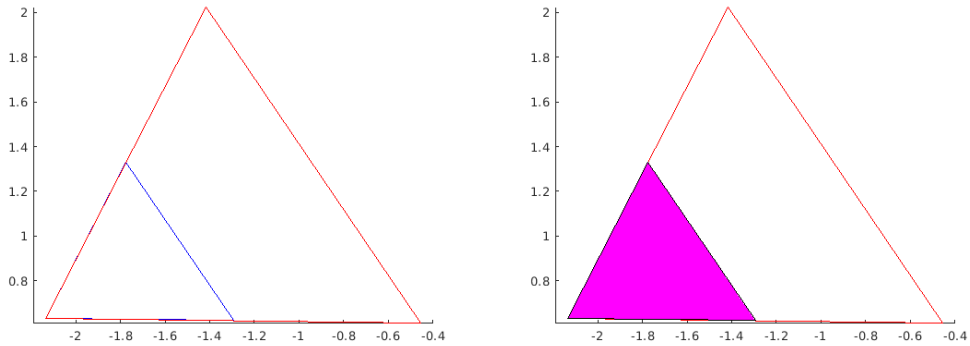


Fig. 1. Left: example of a triangle intersection where an intersection is not calculated despite one of substantial size clearly existing. Right: perturbing vertices of the smaller triangle allows the algorithm to find the correct result. Pink shading indicates the calculated intersection.

One can determine the point of failure in this example by examining how PANG finds vertices of the polygon of intersection. For this task PANG uses two sub-algorithms: PointsOfXInY (which determines if a vertex of triangle U is inside triangle V) and EdgeIntersections (which calculates the intersections between the edges of two triangles). These routines use different, mathematically equivalent vector projections that, due to round-off error from floating-point arithmetic, can produce inconsistent results.

Such an inconsistency may be found in the example of Figure 1. According to PointsOfXInY in floating-point arithmetic, the right base vertex of the blue triangle does not lie within the red triangle. According to EdgeIntersections in floating-point arithmetic, all three vertices of the blue triangle lie inside the red triangle and so no intersections can be calculated (excepting the vertex that is coincident with a vertex of the red triangle). In short, the algorithm fails because the subroutines do not agree on whether one vertex of one triangle lies inside the other. It is clear from this example that consistency is required amongst the subroutines of the algorithm: All components must produce consistent results even in the presence of round-off error [20].

The goal of this paper is to formulate a triangle intersection algorithm that provides this level of robustness to the calculation of the intersection. That is, the error in this calculation is bounded and continuous with respect to perturbations. To achieve this the algorithm enforces consistency across its numerical calculations. This culminates in Theorem 1 that follows. We assume that we encounter neither overflow nor underflow.

Theorem 1. *Let $f(U,V)$ be the area of the intersection between triangles U and V calculated by the algorithm described in Section 4, where V is transformed into a reference triangle. Let L be the ratio of the diameters of U and V, α the smallest angle in V, $\alpha \leq \pi/3$, and C the ratio of the lengths of the sides of V that form this angle, $C \leq 1$. Let ΔU and ΔV be perturbations in the positions of the vertices of U and V, respectively. If the relative change in V is smaller than $C\sin(\alpha/2)$ then*

$$|f(U+\Delta U, V+\Delta V) - f(U,V)| = \frac{1+L}{C\sin(\alpha/2) - O(\Delta V/V)} O(\Delta U + L\Delta V) + O(\Delta V).$$

The algorithm found in Section 4 is a successor to PANG, and therefore named PANG2. This algorithm makes use of reference lines to determine which vertices of U lie inside V, like some of the algorithms cited above [16, 18, 19, 22]. This information is then used to identify which edges intersect, similar to [16, 21, 22]. The intersection points then indicate which vertices of V lie inside

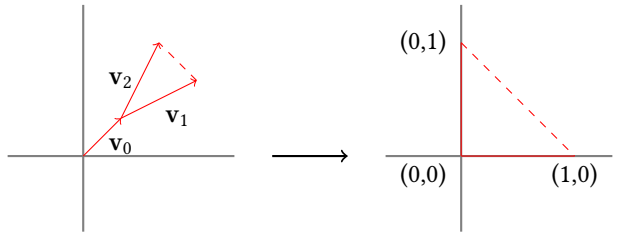


Fig. 2. Coordinate transformation.

U. As each step uses all available data from previous steps the algorithm is parsimonious, which is sufficient for robustness [10].

The mesh intersection problem and the need to project between non-nested meshes occurs in a diverse range of areas, including contact problems [26] and the Arlequin method [1]. PANG has been applied in a variety of contexts [3, 5, 14, 17]. Alternatives to this algorithm include supermesh construction [8, 9] and Plücker coordinates [2]. More alternatives may be found in [7].

2 CHANGE OF COORDINATES

The triangle intersection algorithm in PANG [12] implicitly uses a change of coordinates to test whether a vertex of U, the 'subject' triangle, is inside V, the 'clipping' triangle. Effectively, the coordinates of all vertices are changed so that all calculations involving the clipping triangle are as simplified as possible. For PANG2, this change of coordinates is made explicit and used much more extensively.

We first codify the position and shape of the triangle V. Select a vertex of V. Its position is labelled by the vector $\mathbf{v}_0$. The vectors pointing from this vertex to the other two vertices of V are represented by $\mathbf{v}_1$ and $\mathbf{v}_2$, so that the positions of the vertices of V may be summarized by the matrix $\mathbf{v}_0\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}$, where $\mathbf{1}$ is a column vector containing three ones. The same is done for U, resulting in $\mathbf{u}_0$, $\mathbf{u}_1$ and $\mathbf{u}_2$ and the positions $\mathbf{u}_0\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{u}_1 & \mathbf{u}_2\end{bmatrix}$.

We perform a change of coordinates through an affine transformation on both V and U. The goal is to transform V into a right angle triangle aligned with the coordinate axes, called the reference triangle Y. This reference triangle has vertices at $(0,0)$, $(0,1)$ and $(1,0)$. It is clear then that the affine transformation $A\mathbf{v} + \mathbf{b}$ satisfies the equation

$$A\left(\mathbf{v}_0\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}\right) + \mathbf{b}\mathbf{1}^\top = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}.$$

It can be deduced that $\mathbf{b} = -A\mathbf{v}_0$ and $A\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix} = I$, the identity matrix. Thus, A is the inverse of $\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}$.

The transformation of U, called X, may then be obtained by calculating $A(\mathbf{u}_0\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{u}_1 & \mathbf{u}_2\end{bmatrix} - \mathbf{v}_0\mathbf{1}^\top)$. Alternatively, since A is used exclusively in this step, one may solve the following system for the coordinates of the vertices of X:

$$\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}\begin{bmatrix}x_1 & x_2 & x_3\\ y_1 & y_2 & y_3\end{bmatrix} = \mathbf{u}_0\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{u}_1 & \mathbf{u}_2\end{bmatrix} - \mathbf{v}_0\mathbf{1}^\top.$$

Affine transformations are numerically stable if the matrix involved in the transformation is well-conditioned. The condition number of A is proportional to the aspect ratio of the triangle V: As the columns of A^{-1}, $\mathbf{v}_1$ and $\mathbf{v}_2$, become parallel the angle between them shrinks, thus making V thinner [15]. There are two affine transformations used in this algorithm which are inverses of each other.

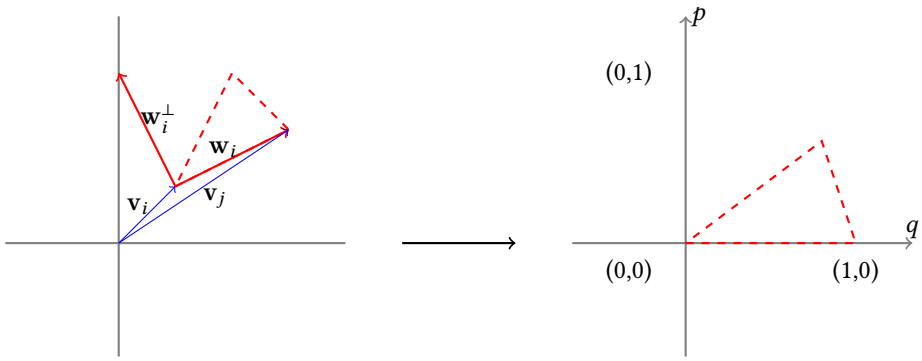


Fig. 3. Reference-free parametrization.

The original PANG intersection algorithm performs this procedure twice: once to find vertices of U lying inside V and a second time for vertices of V lying inside U, resulting in two different affine transformations. An unrelated procedure is used to calculate the intersections. To achieve our goal of a robust algorithm, PANG2 keeps the underlying geometry of all calculations consistent: The coordinates created at this step to describe U and V will be used exclusively in the remainder of the algorithm to calculate the intersections between the edges of U and those of V; which vertices of U lie inside V and; which vertices of V lie inside U. This idea of consistency is key for the robustness of PANG2.

2.1 Reference-free parametrizations

The change of coordinates described above is asymmetric over the vertices of V. The selection of a vertex of V as having position $\mathbf{v}_0$ is not unique, and depending on which of the three vertices are chosen three different geometries can result. However, this change of coordinates is not necessary. The parametrizations of the reference lines of Y may be found directly from the original coordinate system, without any transformation of V to Y.

To proceed, we first re-codify the positions of the vertices of V and U. Let the i–th vertex of V lie at the position defined by the vector $\mathbf{v}_i$, and let $\mathbf{u}_i$ represent the same for the i–th vertex of U. Let $\mathbf{w}_i$ be the vector running between the i–th and j–th vertices of V, where $j = i \pmod 3 + 1$.

As before, we seek an affine transformation that maps the i–th vertex to $(0, 0)$ and the j–th vertex to $(1, 0)$. Given that we are not concerned with the final position of the third vertex of V we ask only that the transformation avoids shearing to minimize possible error. Ultimately, the affine transformation for the i–th vertex of V satisfies

$$A\left(\mathbf{v}_i\mathbf{1}^\top + \begin{bmatrix}0 & \mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}\right) + \mathbf{b}\mathbf{1}^\top = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix},$$

where $\mathbf{w}_i^\perp$ is a vector orthogonal to $\mathbf{w}_i$. The transformation of $\mathbf{u}_k$, represented by (q_k, p_k), is found by solving

$$\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}\begin{bmatrix}q_k\\ p_k\end{bmatrix} = \mathbf{u}_k - \mathbf{v}_i.$$

There is a degree of freedom in the choice of $\mathbf{w}_i^\perp$. It is natural to choose the triangle V to lie on the positive side ($p \geq 0$) of the line $p = 0$. To ensure this, we require that $\mathbf{w}_j^\top \mathbf{w}_i^\perp > 0$. This reduces to checking if the vertices of V are listed clockwise or counterclockwise.

An intersection $(q_0, 0)$ between this edge of V and a given edge of X must be transformed back into the original coordinates. This position is found using the formula $\mathbf{v}_i + q_0\mathbf{w}_i$.

2.1.1 FLOP count comparison. This reference-free parametrization of the edges of V is not recommended as it takes more floating-point operations (FLOPs) to run in practice. In this parametrization, each edge is treated equally, meaning no calculations from the other edges are used to speed up computing time. The change of coordinates described in Section 2 takes advantage of the degree of freedom used here to prevent shearing to combine two of the parametrizations and makes the third considerably cheaper to implement.

Under the reference-free parametrization, for each edge one must solve: a system of the form $A\hat{X} = B$ where A is a matrix of size 2×2 and $\hat{X}$ and B are of size 2×3; three checks of the sign bits of the p–coordinate of the vertices of X; up to two intersections between the line $p = 0$ and the edges of X; up to two transformations using the formula $\mathbf{v}_i + q_0\mathbf{w}_i$ and; two tests of the vertices of V. These last four steps will be explained in Section 3. Additionally, there is a check of the value of $\mathbf{w}_j^\top \mathbf{w}_i^\perp$, which constitutes 2 multiplications and 1 addition.

By comparison, using the change of coordinates only the last four of these steps need to be taken for each edge. This saves two system solves of the form $A\hat{X} = B$ and the check of $\mathbf{w}_j^\top \mathbf{w}_i^\perp$. However, the edge $x + y = 1$ needs special attention. As will be explained in Section 3 this edge has a pre-computed affine transformation of 4 additions and 1 multiplication for each vertex of X. Moreover, the transformations of the intersections along this edge use the form $\mathbf{v}_0 + (1 - q_0)\mathbf{v}_1 + q_0\mathbf{v}_2$. This represents a further 3 additions and 2 multiplications for each such intersection, of which there are at most two. The difference in computations therefore totals 2 fewer system solves, 17 more additions and 5 extra multiplications.

If the number of FLOPs to solve the system $A\hat{X} = B$ is more than 11, which is true of most methods to solve such systems, then the change of coordinates is a more efficient subroutine. This depends strongly on implementation. The programs presented here are written in a compact form and do not represent the most efficient implementation, which would not rely as strongly on subroutines and indexing.

Regardless of efficiency, the reference-free parametrization may prove more accurate. This parametrization presents two clear advantages to this point. Firstly, it is independent of the choice of vertex of V which is necessary for the change of coordinates. Secondly, the matrices used in the three affine transformations are shear-free.

3 COMPUTATION OF THE POLYGON OF INTERSECTION

As explained in the introduction, the polygon of intersection for the triangles X and Y has three types of vertices: intersections between the edges of X and Y; vertices of X lying inside Y and; vertices of Y lying inside X.

The edge intersections of X and Y are the affine transformations of those of U and V and so may be reverse transformed to provide the latter. The locations of the vertices of U are known without reverse transformation, and so once the indices of which vertices of X lying inside Y are found they index also those of U that lie inside V. The same is true of the vertices of V lying inside U, using the indices of the vertices of Y lying inside X.

3.1 Edge Intersections

The reference triangle Y (the transformation of V) has two edges of length 1 aligned with the lines $x = 0$ and $y = 0$. The third edge runs along the line $x + y = 1$. These three lines will be called reference lines.

Parameters	$y = 0$	$x = 0$	$x + y = 1$
$p(x, y)$	y	x	$1 - x - y$
$q(x, y)$	x	y	$(1 - x + y)/2$

Table 1. Parameters $p(x, y)$ and $q(x, y)$ for each of the three reference lines of Y.

To standardize the calculations of the intersections and simplify the discussion, we construct parameters $p(x, y)$ and $q(x, y)$ for each reference line so that the reference line lies on $p = 0$, q increases orthogonally to p, and the edge of Y lies between $q = 0$ and $q = 1$. For the lines $x = 0$ and $y = 0$ this parametrization is trivial. Table 1 lists these parameters for each of the reference lines.

This transform from (x, y) to (q, p) represents another affine transformation. As opposed to the transformation in the previous section these transformations are known *a priori*. The most computationally intensive of these, for the line $x + y = 1$, is a rotation and translation. The condition number of a rotation is always 1 and so these affine transformations are stable.

We consider one edge of Y, its reference line, and the corresponding parameters $p(x, y)$ and $q(x, y)$. Let $p_i = p(x_i, y_i)$ and $q_i = q(x_i, y_i)$ be the values of the parameters for the i–th vertex of X. An intersection occurs between the reference line and the edge connecting the i–th vertex of X and its j–th vertex if p_i is positive and p_j is negative, or vice versa. We can then enforce the following condition: We will only calculate an intersection for the pair of vertices (i, j) if p_i and p_j have different signs.

The degenerate case where a vertex of X lies on the reference line needs to be considered. Without loss of generality, we suppose the first vertex of X lies on the reference line. By most conventions the sign of such a value of p_1 would be zero. If $p_2 > 0$ and $p_3 < 0$ then both pairs (1,2) and (1,3) have intersections with the reference line. As both of these are in fact the same point there is redundancy in these calculations. To avoid this, any vertex lying on the reference line will be considered to be on the positive side of the line. That is, we use the following binary-valued sign function:

$$\operatorname{sign}(p) = \begin{cases} 1 & p \geq 0, \\ 0 & p < 0, \end{cases} \tag{1}$$

as opposed to the ternary-valued sign function that allows a third value at 0. The condition $\operatorname{sign}(p_i) \neq \operatorname{sign}(p_j)$ is both necessary and sufficient for an intersection to exist, excepting the case where $p_i = p_j = 0$.

For the degenerate case where $p_i = p_j = 0$ there is an infinite number of intersections. However, to construct the polygon of intersection we need only the corners of the polygon. Therefore, there is no distinction between this degenerate case and the same configuration with p_i and p_j shifted an imperceptible distance away from the line in the positive p direction, as the polygon of intersection retains the same shape and size. Thus, the previous equivalence statement may ignore the case of an infinite number of intersections.

Suppose $\operatorname{sign}(p_i) \neq \operatorname{sign}(p_j)$. Then an intersection exists for the pair (i, j). To find the intersection it suffices to find the q–intercept of the line running through the two sets of parameters. There are a number of formulas to produce this result. For example, one may use the following:

$$q_0 = \frac{p_j q_i - p_i q_j}{p_j - p_i}. \tag{2}$$

This particular formula has the advantage of being symmetric in i and j. To test if the intersection is on the edge of the reference triangle, and not merely on the reference line, one tests if $q_0 \in [0, 1]$.

Proposition 1. *At most six intersections will be calculated under any triangle intersection algorithm that checks if* $\text{sign}(p_i) \neq \text{sign}(p_j)$. *The number of intersections calculated is even.*

Proof. Consider the condition $\text{sign}(p_i) \neq \text{sign}(p_j)$. The function $\text{sign}(p)$ has only two possible values: 0 and 1. There are only two ways to partition three objects (p_1, p_2, p_3) into two sets (either 0 or 1): 3-0 and 2-1. No intersections are calculated for the first of these. For the other, there are two intersections. Thus, for a given reference line there is either 0 or 2 intersections calculated.

This proof may be applied to each of the three reference lines. If each condition produces the maximum two intersections, six intersections will be calculated. □

Given the simple nature of the edges of the reference triangle Y the equations for the intersections are straightforward to write down:

$y = 0$**:** $(q_0, 0)$;
$x = 0$**:** $(0, q_0)$;
$x + y = 1$**:** $(1 - q_0, q_0)$.

This provides the transformation from (q, p) coordinates to (x, y) coordinates. To retrieve the coordinates of the intersections in the original coordinates, i.e. the coordinates of U and V, one multiplies the coordinate vector $\begin{bmatrix} x & y \end{bmatrix}^\top$ by the matrix $\begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}$. For example, to retrieve the original coordinates of an intersection with the line $y = 0$, one performs the calculation

$$\begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} \begin{bmatrix} q_0 \\ 0 \end{bmatrix}.$$

Note that the possible error of the position of the intersections with regard to the reference triangle are limited. The calculated intersection with the line $y = 0$ must lie on the line $y = 0$ and so there is no vertical error. The same is true of horizontal error for the intersection with the line $x = 0$ and of error along the line $x + y = 1$ for the intersection along this line.

3.2 Vertices of Y in X

To maintain consistency, we use the information from the floating-point calculations of the intersections to decide if the vertices of Y lie within X. First note that triangles are convex. Therefore it is both necessary and sufficient that a vertex of Y lies on a line between the boundaries of X for this vertex to lie inside of X.

By Proposition 1 there are at most two intersections for each line. Suppose a given line has two such intersections. One may differentiate between the two by using q_0^1 and q_0^2. Which pairs of i and j correspond to q_0^1 and q_0^2 is inconsequential.

Without loss of generality, suppose $q_0^1 < q_0^2$. The interval $[q_0^1, q_0^2]$ along the line $p = 0$ lies within the triangle X. Moreover, no points outside this interval and along this line lie within X. Two vertices of the triangle Y lie along this line, at $q = 0$ and $q = 1$. Therefore, a vertex of Y lies inside X if and only if $0 \in [q_0^1, q_0^2]$ or $1 \in [q_0^1, q_0^2]$. Figure 4 shows this process graphically.

If a given line has zero intersections, then it is impossible for a vertex of Y that lies along this line to be within X, as no part of this line lies within X. In this case the triangle X lies entirely on one side of the line or the other.

The tests of $q_0^1, q_0^2 \in [0, 1]$ and $0, 1 \in [q_0^1, q_0^2]$ simplify to checking whether $q_0 < 0$ and $1 - q_0 < 0$ for both q_0^1 and q_0^2. For intersections along the same edge of X there is correspondence between these tests.

Proposition 2. *The number of intersections lying on* Y *is even.*

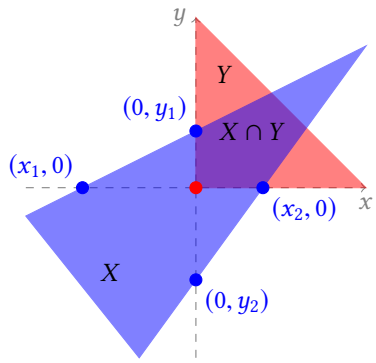


Fig. 4. Intersections between the edges of X and the lines $y = 0$ and $x = 0$ surrounding the vertex of Y at the origin. For the line $y = 0$ the parameters q and p are x and y, respectively, and $q_0^1 = x_1$ and $q_0^2 = x_2$. For the line $x = 0$ $q = y, p = x$, $q_0^1 = y_1$ and $q_0^2 = y_2$.

Reference line	Numerator of q_0	Numerator of $1 - q_0$
$y = 0$	$x_i y_j - x_j y_i$	$(1 - x_i)y_j - (1 - x_j)y_i$
$x = 0$	$y_i x_j - y_j x_i$	$(1 - y_i)x_j - (1 - y_j)x_i$
$x + y = 1$	$(1 - x_j)y_i - (1 - x_i)y_j$	$(1 - y_j)x_i - (1 - y_i)x_j$

Table 2. Numerators of q_0 and $1 - q_0$ for the three reference lines of Y. These have been simplified and use the change of coordinates.

Proof. If all calculated intersections for a given reference line lie entirely inside or entirely outside $[0, 1]$ then Proposition 1 provides the result. However, if $q_0^1 \in [0, 1]$ and $q_0^2 \notin [0, 1]$ then we must prove that the same occurs for another reference line.

Without loss of generality, suppose $q_0^2 < 0$. Then $0 \in [q_0^2, q_0^1]$ and there must be two intersections on the second reference line running through the vertex at 0 such that they bound the same vertex, see Figure 4. Let these intersections be denoted $\hat{q}_0^1$ and $\hat{q}_0^2$, and let the vertex they bound be 1 such that $1 - \hat{q}_0^1 < 0$ and $1 - \hat{q}_0^2 \geq 0$. If $\hat{q}_0^2 \geq 0$ it forms a pair with q_0^2.

If instead $\hat{q}_0^2 < 0$ both vertices are bound by the intersections along this reference line. Thus, along the third reference line the second vertex is found between the two intersections, i.e. $1 - \tilde{q}_0^1 < 0$ and $1 - \tilde{q}_0^2 \geq 0$. Since $1 \notin [q_0^2, q_0^1]$ the third vertex is not found between the intersections of the third reference line and $\tilde{q}_0^2 \geq 0$. This intersection forms the necessary pair with q_0^2. □

Consider the numerators of q_0 and $1 - q_0$ under the change of coordinates. These are presented in Table 2. Each is divided by their respective $p_j - p_i$. The values of the numerators are shared over calculations for the same edges of X. Thus, by ensuring the calculations agree on the signs of these numerators the algorithm can make a consistent determination on which side of a vertex of Y the edge of X falls.

While the value of q_0 is used to calculate the intersections, we need only the sign of $1 - q_0$ and compare it with the sign of p_j. Each of these numerators, of which there are nine in total, can be computed as a determinant. The numerators for the entire triangle X can be computed as cross products.

Under the reference-free parametrization the correspondence is less obvious but is still present. Let $\mathbf{w}_i$, $\mathbf{v}_i$, $\mathbf{u}_k$ and j be as defined in Section 2.1, and let q_k^i and p_k^i be the transformation of $\mathbf{u}_k$ for

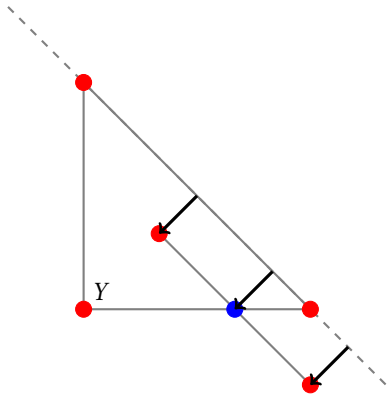


Fig. 5. Test of sign of $1 - q_0$ without computation. In this example, both vertices of the edge of X lie on the positive side of the reference line $x + y = 1$. The intersection must also, implying $1 - q_0 > 0$.

the i–th vertex of V. Then

$$\begin{aligned}
\begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix} \begin{bmatrix} 1 - q_k^i \\ p_k^i \end{bmatrix} &= \mathbf{w}_i + \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_k^i \\ p_k^i \end{bmatrix} \\
&= \mathbf{w}_i + \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix}^{-1} (\mathbf{u}_k - \mathbf{v}_i) \\
&= \mathbf{w}_i + \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix}^{-1} (\mathbf{u}_k - \mathbf{v}_j + \mathbf{w}_i) \\
&= \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{w}_j & \mathbf{w}_j^\perp \end{bmatrix} \begin{bmatrix} q_k^j \\ p_k^j \end{bmatrix}.
\end{aligned}$$

Note that the choice of $\mathbf{w}_j^\top \mathbf{w}_i^\perp > 0$ made in Section 2.1 forces the determinant of the product of $\begin{bmatrix} \mathbf{w}_i & \mathbf{w}_i^\perp \end{bmatrix}^{-1}$ and $\begin{bmatrix} \mathbf{w}_j & \mathbf{w}_j^\perp \end{bmatrix}$ to be positive, see Lemma 6 in Appendix C. Therefore, for a given edge of X the numerator of $1 - q_0$ for the i–th vertex of V has the opposite sign of the numerator of q_0 for the j–th vertex of V. Both numerators measure the distance from the j–th vertex of V.

As a protective measure, we test whether the edge of X intersects neighbouring reference lines. Let p_i' and p_j' be the p–coordinates of the i–th and j–th vertices of X, respectively, for the neighbouring reference line. If the edge between these vertices does not intersect the neighbouring reference line then $\text{sign}(p_i') = \text{sign}(p_j')$. Moreover, they are both equal to $\text{sign}(q_0)$ or $\text{sign}(1 - q_0)$, depending on the line, see Figure 5. By performing this check we prevent intersection errors involving edges of X that intersect only one reference line of Y, see Sections 4.1 and 5.2.

3.3 Vertices of X in Y

Vertices of X in Y cannot be found using the same approach as the two problems are not symmetric: We do not have the same information from the intersection calculations. However, there is sufficient information to make consistent determinations using a different approach.

The parameters $p(x, y)$ have been set up such that the reference triangle Y lies on the positive side of the line $p = 0$. That is, if $(x, y) \in Y$ then $p(x, y) \geq 0$ for each of the three reference lines. Conversely, if $p(x, y) \geq 0$ for all of the reference lines then $(x, y) \in Y$. Therefore, the i–th vertex of X lies inside Y if and only if $\text{sign}(p_i) = 1$ for all three reference lines. One can keep track of this at each edge by creating a dummy variable, s, originally set to 1 and multiplied by $\text{sign}(p_i)$ during

```
function [Q,ind,indR]=EdgeIntersect(p,sp,edge,D)
% EDGEINTERSECT intersection of edges of X with reference line of Y
%   [Q,ind,indR]=EdgeIntersect(p,sp,edge,D) calculates the intersections
%   between all edges of X and the reference line of Y indicated by edge
%   and stores the values in Q. It also calculates Boolean indicators for
%   neighbours of X intersecting Y (ind) and vertices of Y inside X (indR).
%   p and sp store the p-coordinates and their signs, respectively, and D
%   stores the determinants used to calculate the intersections.
if edge==1                                  % v1
  a=[1;0]; b=[0;0];                         % inversion of parametrization
  R1=1; R2=2;                               % vertices of V on edge
  D1=D(R1,:); D2=D(R2,:);                   % relevant numerators
  E0=2; E1=3;                               % adjacent ref lines
elseif edge==2                              % v2
  a=[0;1]; b=[0;0]; R1=1; R2=3; D1=-D(R1,:); D2=-D(R2,:); E0=1; E1=3;
elseif edge==3                              % line connecting v1 and v2
  a=[-1;1]; b=[1;0]; R1=2; R2=3; D1=-D(R1,:); D2=D(R2,:); E0=1; E1=2;
end
Q=zeros(2,3); ind=zeros(1,3); indR=ind; D2=sign(D2)>=0;
for i=1:3
  j=mod(i,3)+1;
  if sp(edge,i)~=sp(edge,j)                 % vertices on opposite sides of edge?
    q0=D1(i)/(p(edge,j)-p(edge,i));         % calculate intersection, eqn (2)
    if (q0<0 && sp(E0,i)~=sp(E0,j)) || (~sp(E0,i) && ~sp(E0,j)) % see Sec. 4.1
      indR([R1,R2])=indR([R1,R2])+[0.25,0.75];
    elseif (D2(i)~=sp(edge,j) && sp(E1,i)~=sp(E1,j)) || (~sp(E1,i) && ~sp(E1,j))
      indR([R1,R2])=indR([R1,R2])+[0.75,0.25];
    else
      indR([R1,R2])=indR([R1,R2])+[0.75,0.75];
      Q(:,i)=q0*a+b; ind(i)=1;
    end
  end
end
```

Program 1. Intersection algorithm of a triangle X with an edge of the reference triangle Y. Note that `indR(R1)` equals 1 if and only if the vertex indexed by `R1` lies between the two calculated values of `q0`. The same is true for `R2`.

each edge parametrization. Since $\text{sign}(p_i) = 0$ if $p_i < 0$ the final result of s will be 0 if any $p_i < 0$ and will be 1 if and only if $p_i \geq 0$ for every parameter $p(x, y)$.

Because the values of p_i are also used in the floating-point calculations of the intersections, this step is consistent with these calculations and, by extension, the determination of the vertices of Y in X. The three types of points in the polygon of intersection make use of the same information, and a numerical error in part of this information affects each type in a consistent way.

4 ROBUST ALGORITHM FOR 2D TRIANGLE INTERSECTIONS

We will now write the intersection algorithm of PANG2 in full.

Step 1: Change of coordinates. An affine transformation is used to change the coordinates of the vertices of X into a reference frame whereby Y is the reference triangle described in Section 2.

```
function [P,Q,R,n]=TriIntersect(U,V)
% TRIINTERSECT intersection of two triangles
%   [P,Q,R,n]=TriIntersect(U,V) gives the vertices of U inside V (P),
%   intersections between the edges of U and those of V (Q), the vertices
%   of V inside U (R), and the neighbours of U that also intersect V (n).
v1=V(:,2)-V(:,1);               % vector between V1 and V2
v2=V(:,3)-V(:,1);               % vector between V1 and V3
X=[v1,v2]\(U-V(:,1));           % Step 1: change of coordinates
p=[X([2,1],:);1-X(1,:)-X(2,:)]; % values of the parameters p
sp=sign(p)>=0;                  % sign of the p values
D=cross([X(1,:);1-X(1,:);X(1,:)],[X(2,:);X(2,:);1-X(2,:)],2); % num.s of q0 and 1-q0
Q=zeros(2,9); indQ=zeros(1,9); n=zeros(1,3); indR=ones(3);
for i=1:3                       % Step 2: select ref. line
    ind=(1:3) + 3*(i-1);        % indices of Q for this ref. line
    [Q(:,ind),indQ(ind),indR(i,:)]=EdgeIntersect(p,sp,i,D(:,[3,1,2]));
    n=max(n,indQ(ind));         % update nhbr index
end
P=U(:,prod(sp)==1);             % vertices of U in V
Q=V(:,1) + [v1,v2]*Q(:,indQ==1);% Step 4: reverse change of coordinates
R=V(:,max(indR==1));            % vertices of V in U
```

Program 2. Robust algorithm for the intersection of 2D triangles, written in MATLAB. The affine transformation of the change of coordinates (see Section 2) is done using MATLAB's backslash operator.

Step 2: Select reference line. Choose a reference line of the reference triangle. Apply the correct functions for the parameters and make note of which vertices of Y lie on this line.

2a: Intersections. Test if $\text{sign}(p_i) \neq \text{sign}(p_j)$. If so, calculate the intersection with the reference line and test $q_0 < 0$ and $1 - q_0 < 0$. If both are false, the intersection lies on the edge of Y. Repeat this step for all three pairs of vertices of X. By Proposition 1 this results in at most two unique intersections. One may remove duplicates at this stage but it is not necessary.

2b: Vertices of Y in X. Use the tests of $q_0 < 0$ to determine if $0 \in [q_0^1, q_0^2]$. If it does, the corresponding vertex of Y lies within X. Repeat for the vertex at $q = 1$.

Repeat step 2 for each of the three reference lines.

Step 3: Vertices of X in Y. Multiply the three values of $\text{sign}(p_i)$ together for each of the three vertices of X. The result will either be 0 or 1. If it is 1, the vertex lies inside Y.

Step 4: Reverse change of coordinates. The vertices of X and Y in the standard coordinates are already known and so one can take those vertices determined to be in Y and X, respectively, without additional calculations. For the intersections one must apply the inverse affine transformation of the first step of the algorithm.

4.1 Test of $q_0 < 0$

We outline the implementation of step 2a of the algorithm, where we test if $q_0 < 0$ and $1 - q_0 < 0$. Without loss of generality we consider only the test of $q_0 < 0$. As explained in Section 3.2 if the signs of p_i' and p_j', the $p-$coordinates of the vertices of X for the neighbouring reference line, are equal then $\text{sign}(q_0) = \text{sign}(p_i') = \text{sign}(p_j')$. The first test is then to check whether $\text{sign}(p_i') = \text{sign}(p_j')$. Depending on this result we either use the value of $\text{sign}(p_i')$ to determine if $q_0 < 0$ or we test directly $\text{sign}(q_0)$.

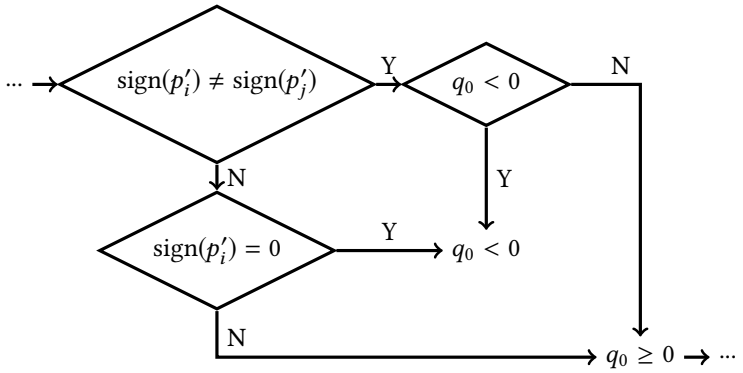


Fig. 6. Decision tree for the test of $q_0 < 0$. The test of $1 - q_0 < 0$ is identical.

Figure 6 shows the decision tree of the test of $q_0 < 0$. As outlined above, there are three determinations to be made: whether $\text{sign}(p'_i) \neq \text{sign}(p'_j)$; $\text{sign}(q_0)$ and; $\text{sign}(p'_i)$. To make the code compact these three decisions can be combined into a single logical test:

$$[\text{sign}(p'_i) \neq \text{sign}(p'_j) \wedge q_0 < 0] \vee [\text{sign}(p'_i) = 0 \wedge \text{sign}(p'_j) = 0] \implies q_0 < 0.$$

If this test returns that $q_0 < 0$ then this information must be used to determine if the vertex of Y at $q = 0$ lies in X. We store this information and await the result of the test of the other intersection on this reference line. If this second test returns $q_0 \geq 0$ then the vertex of Y lies in X. In Programs 1 and 3 the value 0.25 is added to the indicator for the vertex of Y if $q_0 < 0$ and the value 0.75 is added if $q_0 \geq 0$. In this way the indicator is equal to 1 if the vertex of Y lies inside X and either 0, 0.5 or 1.5 otherwise. The values 0.25 and 0.75 are chosen for their exact representation in binary.

5 CONSISTENCY ERRORS

The triangle intersection algorithm presented in Section 3 calculates the polygon of intersection for a right angled reference triangle Y and an arbitrary triangle X using a single consistent geometry. Combined with the change of coordinates of Section 2 it may be used to find the polygon of intersection for any two triangles U and V, again with a consistent geometry for all calculations. It remains to show that any numerical errors incurred throughout the calculations maintain the consistency of the algorithm, namely that the algorithm produces an intersection of similar size and shape to the one that exists between U and V.

There are three errors related to consistency that can occur in this algorithm. Firstly, a vertex of X may be found to lie inside (resp. outside) of Y when it is meant to lie outside (inside) (X-in-Y error). Secondly, an intersection of an edge of X with a reference line of Y may be found to lie on (off) the edge of Y when it is meant to lie off (on) (intersection error). Thirdly, a vertex of Y may be found to lie inside (outside) of X when it is meant to lie outside (inside) (Y-in-X error). We examine each of the three errors independently, while noting where applicable when one error can cause another.

5.1 X-in-Y errors

It is important that when an X-in-Y error occurs an appropriate number of intersections is calculated. Whether the intersections are deleted or created, there should be the correct number associated with the intersection of two triangles, albeit slightly altered. The positions of the intersections may

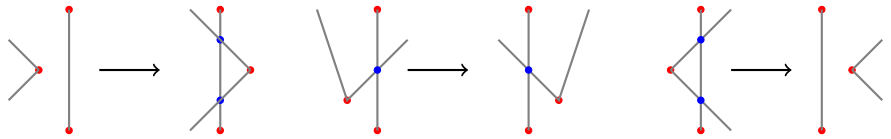
```
function [P,Q,R,n]=TriIntersectRF(U,V)
% TRIINTERSECTRF intersection of two triangles
%   [P,Q,R,n] = TriIntersectRF(U,V) produces the vertices of U inside V (P), the
%   intersections between the edges of U and V (Q), the vertices of V
%   inside U (R) and the neighbours of U that also intersect V (n).
%   The result is the same as TriIntersect(U,V) up to error on the order of
%   machine epsilon. The calculations are performed without change of
%   coordinates to a reference triangle.
W=V(:,[2,3,1])-V;                       % vectors between vertices of V
T=[0,-1;1,0]*W; T=sign(T(:,1)'*W(:,2))*T; % vectors perp. to W
n=zeros(1,3); indR=zeros(3); indP=indR; p=indR;
Q=zeros(2,9); indQ=zeros(1,9); sq=indR; q0=indR;
for i=1:3                               % for each edge of V
  qp=[W(:,i),T(:,i)]\(U-V(:,i));        % parametrization
  p(i,:)=qp(2,:); indP(i,:)=sign(p(i,:))>=0;  % p-coordinates
  q0(i,[2,3,1])=cross(qp(1,:),qp(2,:)); % num. of q0
  sq(i,:)=sign(q0(i,:))>=0;             % sign of num. of q0
end
for i=1:3
  j=mod(i,3)+1; k=mod(i+1,3)+1;         % nhbring ref. lines
  for a=1:3
    b=mod(a,3)+1; m=a+3*(i-1);
    if indP(i,a)~=indP(i,b)             % see Sec. 4.1
      if (sq(i,a)~=indP(i,b) && indP(k,a)~=indP(k,b)) || (~indP(k,a) && ~indP(k,b))
        indR(i,[i,j])=indR(i,[i,j])+[0.25,0.75];
      elseif (sq(j,a)==indP(i,b) && indP(j,a)~=indP(j,b)) || (~indP(j,a) && ~indP(j,b))
        indR(i,[i,j])=indR(i,[i,j])+[0.75,0.25];
      else
        indR(i,[i,j])=indR(i,[i,j])+[0.75,0.75];
        Q(:,m)=V(:,i) + (q0(i,a)/(p(i,b)-p(i,a)))*W(:,i); % eqn (2)
        indQ(m)=1; n(a)=1;              % nhbrs of U intersect V
      end
    end
  end
end
P=U(:,prod(indP)==1); Q=Q(:,indQ==1); R=V(:,max(indR==1));
```

Program 3. Alternative to `TriIntersect(U,V)` using the reference-free parametrization.

be in error, but this will be considered later as a separate error. The following lemma provides that if the algorithm suffers an X-in-Y error then there will be a corresponding change in the number of intersections calculated.

Lemma 1. *Suppose the i–th vertex of X lies outside the reference triangle Y but is determined to be within Y by the algorithm described in Section 4. There is at least one reference line between the correct and calculated positions of the i–th vertex. An intersection is calculated by the algorithm between this reference line and the edge of X between the i–th and j–th vertices if and only if this reference line and edge do not intersect.*

Proof. There are either one or two reference lines between the correct and calculated positions of the i–th vertex of X. One of these reference lines has parameters $p(x, y)$ and $q(x, y)$, see Table 1.



(a) No edges intersect to both edges intersect. (b) One edge intersects to the other edge intersects. (c) Both edges intersect to no edges intersect.

Fig. 7. Results of X-in-Y errors.

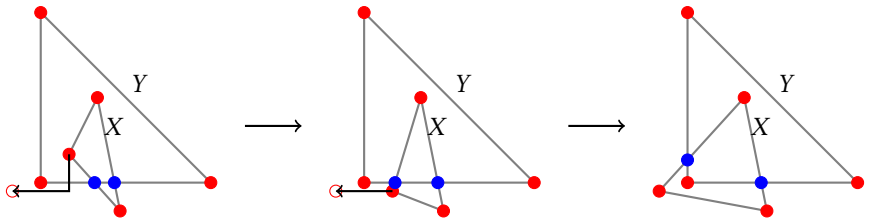


Fig. 8. The cascade of errors used to describe when an X-in-Y error alters intersections on two separate reference lines. An intermediate position between the correct and calculated positions of the i–th vertices is presented in the centre of the figure.

The correct value of $\text{sign}(p_i)$ is 0 but the algorithm has returned $\text{sign}(p_i) = 1$. The value of $\text{sign}(p_j)$ must equal one of these. If $\text{sign}(p_j) = 1$ then the edge of X intersects the reference line but no intersection is calculated. If $\text{sign}(p_j) = 0$ then the edge does not intersect the reference line but an intersection is calculated. □

Lemma 1 remains true if the correct and calculated positions of the i–th vertex are reversed. This lemma does not give an indication as to whether these intersections lie on the edges of Y. Geometrically, such an intersection must lie on an edge of Y. However, the algorithm allows the possibility of error. Such intersection errors are considered separately in Section 5.2.

For a vertex of X that suffers an X-in-Y error there are two other vertices of X. The effects of Lemma 1 therefore occur twice for each error of this type. Figure 7 shows the possible combinations of two applications of Lemma 1 when the intersections occur over the same edge of Y. In each of these figures, the triangle vertex of X moves to the right to cross the reference line of Y. As it does so, two intersections are created and move further apart (Figure 7a); the intersection slides along the edges of the triangles, crossing over the vertex (Figure 7b) or; the two intersections move towards each other until they meet at the moment the vertex crosses the reference line, disappearing (Figure 7c).

If the intersections occur over separate edges of Y then we must consider the result to be a cascade of errors. We consider an intermediate position of the i–th vertex of X such that the movement between the correct and intermediate positions creates intersections along a single edge of Y and the movement between the intermediate and calculated positions causes two intersection errors that result in a Y-in-X error, see Figure 8.

5.2 Intersection errors and Y-in-X errors

A small change in the position of an intersection can only cause an error in consistency if it passes over a vertex of Y. In such a case, the intersection either enters or leaves the polygon of intersection.

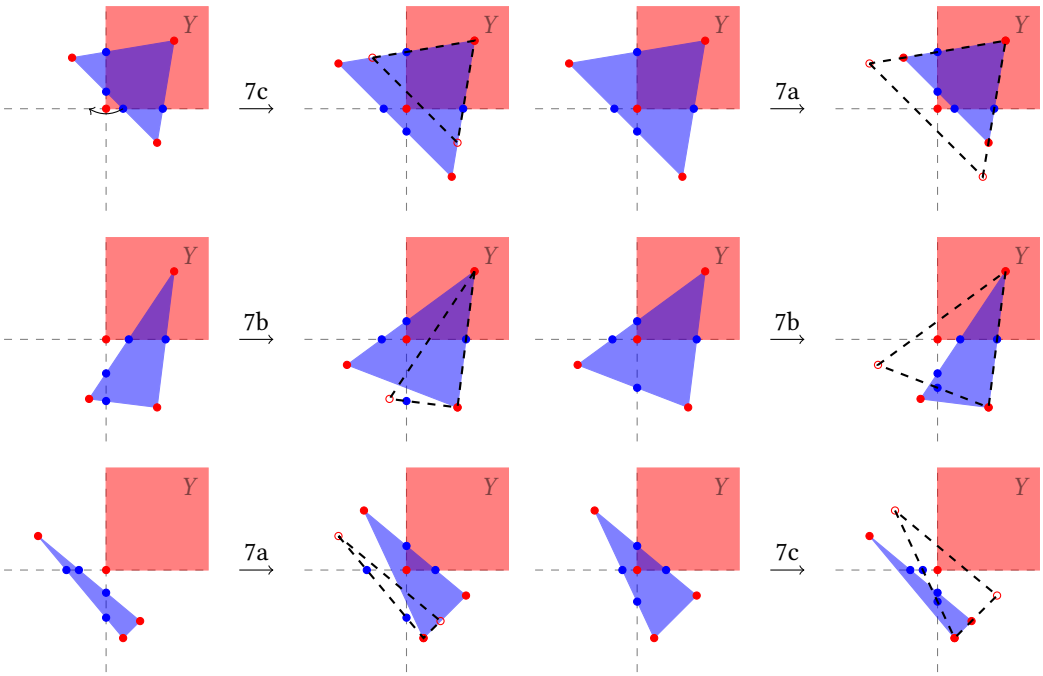


Fig. 9. Paired intersection errors. These errors have the same results as X-in-Y errors. The corresponding results of Figure 7 are referenced for each configuration.

We call this an intersection error. Since this algorithm uses the intersections to determine which vertices of Y lie in X, this can cause Y-in-X errors.

We begin examining intersection errors by dividing them into two categories, paired and unpaired. A paired intersection error occurs when two intersections along the same edge of X both suffer intersection errors over the same vertex of Y. An unpaired intersection error is a single intersection error that occurs on its own. We prove that paired intersection errors preserve consistency and that unpaired intersection errors are impossible.

There are 6 configurations of paired intersection errors, see Figure 9. An intersection error represents a reversal in sign in either q_0 or $1 - q_0$ for a given edge of X. Without loss of generality, we suppose it is in q_0. Recall from Table 2 that the numerator of q_0 is shared with that of another intersection of the same edge of X. Thus, if the sign of q_0 is in error then the sign of this second intersection along this edge of X is also reversed. This pairs the intersection errors.

An unpaired intersection error over a vertex of Y can only occur if the edge of X does not intersect the other reference line that extends from the vertex. Then, in the parametrization for this other reference line, the p–coordinates of the two vertices of X attached to this edge are either all positive or all negative. The algorithm tests for this to prevent the error, see Section 4.1. In this way, unpaired intersection errors cannot occur.

Figure 9 indicates that even under intersection errors a vertex of Y lies within X if and only if it is surrounded on all sides by four intersections. Tests of Y vertices along neighbouring reference lines return consistent results. Proposition 2 is therefore unaffected by errors.

Lemma 2. *Let q_0 be the position of the intersection between the i-th and j-th vertices of X along a given reference line of Y. Without loss of generality, let $p_j \geq 0$ and $p_i < 0$. Suppose the numerator of*

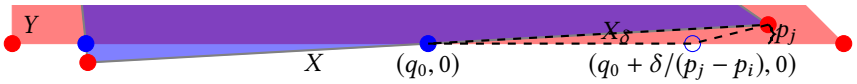


Fig. 10. Error of an intersection amplified in equation (2) is mitigated by the position of a neighbouring node.

q_0 has floating-point error δ from its calculation due to equation (2). Then the error in the area of the polygon of intersection due to equation (2) is $O(\delta)$.

Proof. Since the error in the numerator of q_0 is δ, the error in q_0 is $\delta/(p_j - p_i)$. Given that $\text{sign}(p_j) \neq \text{sign}(p_i)$ there is no cancellation error in $p_j - p_i$ and the error in the denominator is negligible.

The coordinates $(q_0, 0)$, $(q_0 + \delta/(p_j - p_i), 0)$ and (q_j, p_j) form a triangle, denoted X_δ. This triangle has a base of $\delta/(p_j - p_i)$ and a height of at most p_j and so has an area of at most $\delta p_j/2(p_j - p_i) < \delta/2$. It suffices then to show that this triangle represents the largest possible change in the polygon.

Consider the polygon of intersection before the effects of the error. In particular, consider the two neighbours of the intersection $(q_0, 0)$. One of these neighbours lies along the same reference line and is either another intersection or a vertex of Y. The other lies along the edge of the triangle X between its i-th and j-th vertices. Therefore, it is either the j-th vertex of X or an intersection between this edge of X and another reference line. Such an intersection would have a height less than p_j.

After the effects of error, the intersection has shifted. One of the edges of the polygon that extend from this intersection has shrunk while the other has grown. We seek a triangulation of the polygon such that its change is represented by a single triangle. Since one of the neighbouring corners of the polygon lies along the reference line containing both $(q_0, 0)$ and $(q_0 + \delta/(p_j - p_i), 0)$ this single triangle is defined by these nodes and the other neighbour. As previously stated this other neighbour lies on the same edge of X as $(q_0, 0)$ and the j-th vertex of X. The triangle representing the change in the polygon then lies within the triangle X_δ and so has an area of at most $O(\delta)$. □

While $p_j - p_i$ is not subject to cancellation error, the numerator of q_0 is. The magnitude of δ is then $O\left(L^2\epsilon_m\right)$, where L is the longest edge of X and ϵ_m is machine epsilon. The value of L is a measure of the relative sizes of the original triangles U and V. If the two triangles are of roughly equal diameter then this magnitude is 1.

6 GRAPHS OF TRIANGLE-TRIANGLE INTERSECTIONS IN 2D

Section 5 indicates that each error of the algorithm can cause a change in the shape of the polygon of intersection. It must be shown that these changes in shape can cause no significant changes to the area of the intersection. That is, a change in the shape of the polygon must be in some way continuous.

Our strategy to prove this is to construct the set of graphs that represent the possible intersections of two triangles and show that the errors transform these graphs between each other and not outside the set. In this way we will show that at worst the calculated intersection corresponds to two triangles that are of slightly different size, shape and position.

We first note that PANG2 is valid: Given a pair of triangles and assuming no numerical error in any of the calculations the algorithm will correctly identify the intersection between them. Therefore, we need only consider the set of possible intersections and how the errors affect this set. Thus, the first step is to determine this set.

While the intersection of two triangles is a purely geometric concept, the calculation of such an intersection is subject to numerical error. This error can cause vertices and intersections to move.

The resulting polygon of intersection may not lie precisely within the edges of the triangles. We therefore construct graphs of the triangle intersections. In this way the intersections need not be geometric, allowing for the possibility of error.

As these graphs represent geometric objects in the 2D plane the graph edges are straight. We distinguish between two types of graphs: those of geometric intersections and those of intersections calculated by the algorithm. Hoffmann [15] refers to the first as models and the second as representations. The algorithm is robust if the models are close to the representations.

Both types of graphs satisfy a number of conditions.

Vertex condition: triangle vertices (of which there are exactly 6 and are coloured red) are nodes with two neighbours.

Intersection condition: edge intersections (which may number 0, 2, 4 or 6, see Proposition 2, and are coloured blue) are nodes with four neighbours.

Triangle condition: the triangle vertices are divided into two disjoint 3-cycles, each of which corresponds to one of the triangles. The points of overlap of these cycles are the edge intersections.

The triangle condition is so named because it ensures that the graph represents the intersection of triangles rather than some other pair of shapes.

In addition, when degenerate cases are removed (as may be done with this algorithm, to be discussed later) the models possess the following property.

Sheltered polygon property: no graph edge between two intersection nodes touches the exterior of the graph. That is, in the dual graph there is no graph edge between the node representing the polygon of intersection and the node representing the exterior of the graph.

The only way for the sheltered polygon property to be violated is for two edges of the triangles to be coincident, a degenerate case considered in the previous sections. Representations also satisfy the sheltered polygon property.

Lemma 3. *Any representation of this algorithm has the sheltered polygon property.*

Proof. By construction, the nodes of the polygon of intersection as calculated by the algorithm lie interior to or on the boundary of Y. As such, a graph edge between two intersection nodes is either interior to Y or lies on a reference line of Y. In the former case, the interior of Y shelters the graph edge from the exterior.

In the latter case, these intersection nodes were calculated because two edges of X intersect this reference line, such that $\text{sign}(p_i) = \text{sign}(p_j) \neq \text{sign}(p_k)$. Without loss of generality, suppose $\text{sign}(p_k) = 0$ and the k–th vertex of X lies outside the triangle Y. Then there is a graph edge between this vertex and each of the intersection nodes. This shelters the graph edge between the intersection nodes from the exterior of the graph. □

This ignores a number of degenerate cases where edge intersections and triangle vertices overlap. It will be shown later that all degenerate cases may be represented by non-degenerate cases, all of which satisfy these three conditions and the sheltered polygon property. Figure 11 gives graphical examples of the triangle condition and the sheltered polygon property. It also illustrates the proof of Lemma 3. The graph has one graph edge between intersection nodes. If the blue triangle is Y then this graph edge is interior to Y and therefore sheltered from the exterior. If the blue triangle is X then the portion of X that does not intersect Y shelters the graph edge from the exterior. Numerically this portion of X must be there or the intersections would not be calculated.

Figure 12 shows all graphs representing two intersecting triangles. Not shown is the graph of two non-intersecting triangles, which is two disjoint graphs of 3-cycles of red vertices.

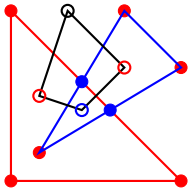


Fig. 11. Graphic representation of the triangle condition and sheltered polygon property for a given graph. The two cycles of the triangle condition are shown with red and blue edges, and the dual graph is shown with black edges.

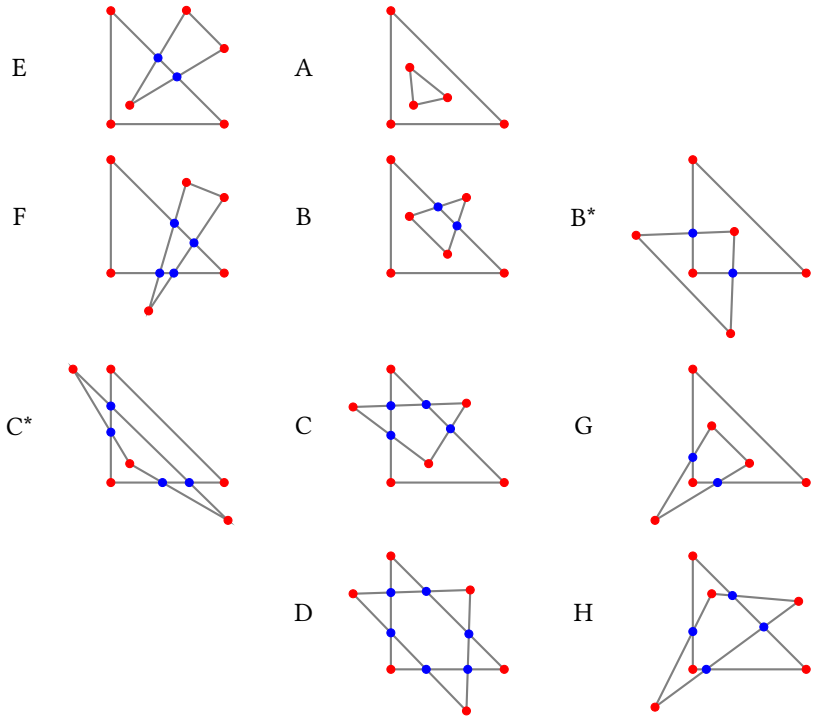


Fig. 12. The models arising from all possible triangle-triangle intersections, degenerate cases removed.

Lemma 4. *Figure 12 is an exhaustive list of all possible models.*

Proof. We begin by proving a restriction on the number of triangle vertices in the polygon of intersection: There can be at most three triangle vertices in the polygon, split into at most two groups, e.g. A and G.

Any collection of X vertices in the triangle Y necessarily connect to one another without any intersections between them, as there are no reference lines of Y inside of Y. As such, in the graph of the polygon of intersection, they are adjacent to one another. The same is true of Y vertices in the triangle X. Therefore, triangle vertices are divided into at most two groups.

Suppose two vertices of X lie inside Y. Without loss of generality, Y is the reference triangle. Choose a reference line of Y, with parameters p and q. The q–coordinates of the vertices of X that lie inside Y are necessarily between 0 and 1. Let q_0^1 and q_0^2 be the q–coordinates of the edges of X

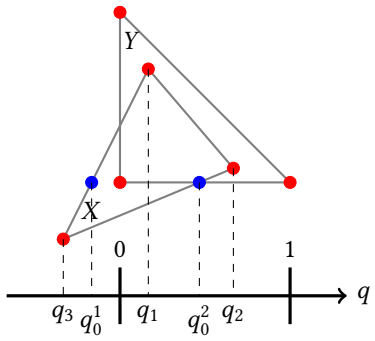


Fig. 13. If two vertices of X lie inside Y then at most one vertex of Y can lie inside X.

with this reference line. If q_0^1 is less than 0 then q_0^2 is less than 1, see Figure 13. Therefore, only one vertex of Y can lie inside X. This is symmetric: If there are two vertices of Y inside X then at most one vertex of X can lie inside Y.

Next, we prove that the number of intersections in the polygon is less than or equal to the number of triangle vertices outside of the polygon. Consider an intersection in the polygon and suppose one traverses the two cycles we know to exist by the triangle condition in a clockwise direction. At this starting intersection one cycle exits the polygon. At the next intersection the cycle enters the polygon. For the cycle to do so it must meet a corner between these intersections. Thus, there must be at least one vertex between the intersection of exit and the intersection of entry. Since each intersection is an intersection of exit for one of the two cycles, each intersection represents at least one triangle vertex outside of the polygon.

The vertex and intersection conditions and the two previous results may be combined in the following list of restrictions on the number of intersections n and the number of triangle vertices m in the polygon.

(i) $n \in \{0, 2, 4, 6\}$ (intersection condition);
(ii) $m \leq 3$ (previous result);
(iii) $3 \leq n + m$ (polygons have at least three vertices);
(iv) $n + m \leq 6$ (triangle condition and previous result).

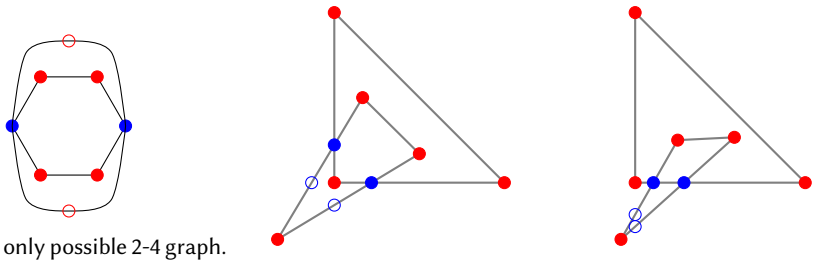
We can then ascertain that only the following n-m are permitted: 0-3, 2-1, 2-2, 2-3, 4-0, 4-1, 4-2, and 6-0.

To determine which, if any, graphs correspond to these combinations, we present the following algorithm to construct a triangle–triangle intersection graph from a polygon of intersection.

(1) Draw the polygon of intersection using one of the eight combinations provided. Only three combinations provide multiple possible polygons: 2-2, 2-3 and 4-2. One of the options for the combination 2-3 may be immediately discarded as it contains three triangle vertices in a row, which is an entire triangle and thus does not have any intersections, see Figure 20a in Appendix B.
(2) Connect each adjacent pair of intersections by a graph edge exterior to the polygon. A pair of intersections is adjacent if they lie next to each other or if they are separated only by triangle vertices.
(3) Following the cycles defined by the triangle condition, add either one or two triangle vertices to each of the graph edges created by the previous step such that each cycle contains the necessary three triangle vertices. This will exclude two of the 4-2 combinations, as some

m	n			
	0	2	4	6
3	A	G, G†		
2		B, B*	H, H†, H†*	
1		E	C, C*	
0			F	D

Table 3. Graphs corresponding to each n-m pairing. The graphs indicated by the † symbol do not satisfy the necessary conditions and are found in Figure 20 in Appendix B. The remainder are found in Figure 12.



(a) I†, the only possible 2-4 graph. (b) The graph G. (c) The graph B.

Fig. 14. The 2-4 pairing is shown to be invalid for representations.

cycles will have at least four triangle vertices, see Figures 20b and 20c in Appendix B. Only one combination, 4-1, has two possible ways of satisfying all conditions of these graphs.

In this way the eight permitted combinations result in the ten graphs of Figure 12. Table 3 identifies each n-m pairing with either one of the graphs of Figure 12 or one of the invalid configurations of Figure 20 found in Appendix B. □

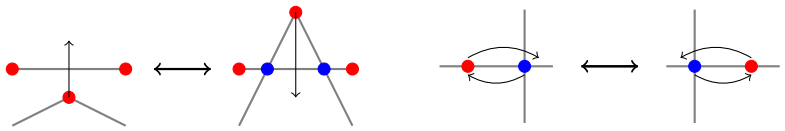
Lemma 4 does not immediately apply to intersections calculated by the algorithm because it relies on geometric arguments. In particular, the argument used to fix restriction (ii) $m \leq 3$ only applies when points of intersection lie strictly on the edges of the triangles. Numerical error can warp the positions of intersections to include or exclude triangle vertices in the polygon of intersection.

Lemma 5. *Restriction (ii) applies to representations.*

Proof. An intersection node in the polygon of intersection lies between two vertices of the triangle Y. The pair of intersections along the reference line that passes through these two vertices then does not enclose at least one of these vertices. If there are two intersections on the reference line neither vertex is enclosed. Each pair of intersections then excludes at least one vertex of Y from the polygon of intersection.

Therefore, if there are two vertices of Y in the polygon then there are at most two intersection nodes. Since the only way to violate restriction (ii) is to have $m = 4$ with two vertices of Y and two of X, the only new n-m to consider is 2-4.

By the same argument that eliminated graph G† there is only one possible graph with a 2-4 pairing, I†, see Figure 14a. This graph does not violate any of the required conditions. Two of the triangle vertices in the polygon of intersection are necessarily those of X, while the other two are those of Y. There are only two graphs of Figure 12 with exactly two vertices of X inside Y: graphs B and G.



(a) First type of graph rewrite. (b) Second type of graph rewrite.

Fig. 15. The two types of graph rewrites codified by the algorithm.

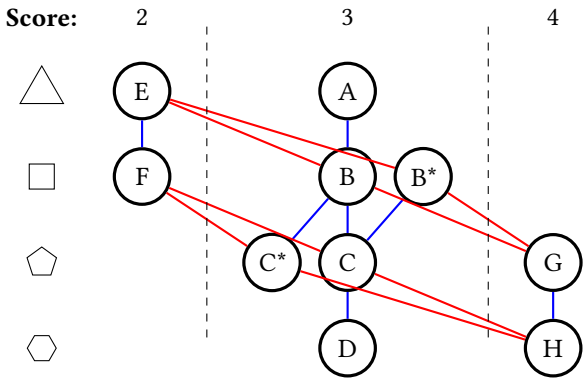


Fig. 16. Graph of intersection connectivity: A blue edge between two graphs indicates one may be achieved through a first type rewrite of the other, while a red edge indicates a second type rewrite is required.

In graph G any intersection error is paired. Refer to the middle row of Figure 9 for the results of one or two of these errors. Under one paired intersection error the graph G becomes the graph B, see Figure 14c. Under a second error the graph B reverts to the graph G, see Figure 14b.

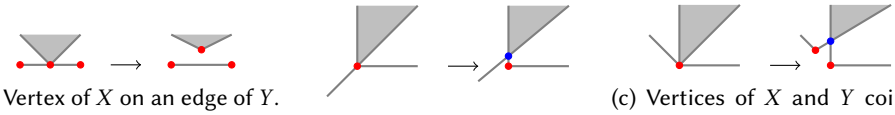
In graph B either the two intersections are both the only intersections along their respective edges of X or the edges of X both intersect another reference line. In the first case, any intersection errors are unpaired and thus impossible. In the second case, a paired intersection error results in graph G, see Figures 14b and 14c. A second error results in a different graph B, with the intersections on the other reference line. □

Corollary 1. *Figure 12 is an exhaustive list of all possible representations.*

Proof. It suffices to prove the four restrictions of the proof of Lemma 4 apply to representations. Restriction (i) applies by the intersection condition. By Lemma 5 restriction (ii) applies. Restriction (iii) applies naturally. The sheltered polygon property ensures the reasoning behind restriction (iv) applies. □

Figure 15 shows the two graph rewrites codified by errors in the algorithm. Each rewrite possesses an inverse, as indicated by the double headed arrows in the figure. Figure 15a corresponds to Figures 7a and 7c, while Figure 15b corresponds to Figure 7b. Recall these figures detail the results of X-in-Y errors and intersection errors. The correspondence between the intersection errors and their respective rewrites is indicated in Figure 9.

Figure 16 shows how the graphs of Figure 12 are connected through the rewrites of Figure 15. The graphs are organized in two ways. Each row has the polygon of intersection increasing in number of edges: graphs E and A have triangular polygons, graphs F, B and B* have quadrilateral polygons, and so on.



(a) Vertex of X on an edge of Y.

(b) Vertex of Y on an edge of X.

(c) Vertices of X and Y coincident.

Fig. 17. Degenerate cases. The shaded area represents the intersection between X and Y.

The columns are more complicated. One may assign point values to the vertices of the polygon of intersection: one point for each triangle vertex and half a point for each edge intersection. The columns of the graph of Figure 16 then have increasing point value sums or 'scores', with graph E having two points and graph H having four points.

The organization of this graph indicates important features of the two graph rewrites: Both rewrites add or subtract a graph edge to or from the polygon of intersection; and the second type rewrite changes the score of the polygon, adding or subtracting one. A score increase is coincident with a graph edge addition, so that when the score increases so does the number of graph edges of the polygon. The polygon of intersection therefore determines the position of the graphs of Figure 12 in the graph of Figure 16.

6.1 Degenerate cases

There are three major degenerate cases to consider: a vertex of X lying on an edge of Y; a vertex of Y lying on an edge of X and; a vertex of X coincident with a vertex of Y.

Should a vertex of X lie on an edge of Y it is considered to be inside of Y by the algorithm. An intersection is only calculated if another vertex lies on the negative side of this edge. For such an intersection the algorithm will calculate it as the vertex. When the copies of the vertex are removed, we will be left with a single point, a vertex of X in Y. Thus, this degeneracy reduces to the vertex of X being just inside the triangle Y. If the calculated intersection does not align with the vertex then the degeneracy is completely removed as the algorithm has created a numerical distance between the vertex and the edge by introducing an intersection between them.

If a vertex of Y lies on an edge of X, the algorithm considers an edge intersection to be coincident with the vertex of Y. This is ultimately equivalent to the same configuration with the edge intersection moved slightly along one of the two reference lines extending from the vertex of Y, such that the vertex is not considered to be inside of X. This keeps the same number of edges of the polygon of intersection and the same overall shape of the two triangles.

If a vertex of X is coincident with a vertex of Y we can consider both degenerate cases described above to have occurred. The degeneracy can be considered as a standard configuration with the vertex of X lying just outside of Y and the edge intersection moved in the same manner as when a vertex of Y lies on an edge of X.

7 PROOF OF THEOREM 1

Recall that the matrix A is the inverse of $\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}$. The vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ run along the edges of V, see Section 2. The condition number of the matrix A is less than $1/(C\sin(\alpha/2))$ where α is the angle between $\mathbf{v}_1$ and $\mathbf{v}_2$ and C is the ratio between their lengths, $C \leq 1$ [15]. Theorem 1 may then be restated in terms of the matrix A. To simplify notation, let $\hat{U}$, $\hat{V}$ and $\hat{X}$ denote the matrices containing the positions of the vertices of the triangles U, V and X, respectively.

Theorem 2. *If* $\|A\| \left\|\Delta\hat{V}\right\| < 1$ *then*

$$|f(U+\Delta U, V+\Delta V) - f(U,V)| = \frac{\kappa(A)(1+L)}{1-\kappa(A)\mathcal{O}(\Delta V/V)}\mathcal{O}(L\Delta V + \Delta U) + \mathcal{O}(\Delta V).$$

Proof. As stated in Section 2 the positions of the vertices of X are determined by solving the linear system

$$A^{-1}\hat{X} = \hat{U} - \mathbf{v}_0\mathbf{1}^\top,$$

where $\mathbf{v}_0$ is a vertex of V. Without loss of generality assume $\mathbf{v}_0 = 0$. Under the perturbation this system becomes

$$\left(A^{-1} + \Delta A^{-1}\right)\left(\hat{X} + \Delta\hat{X}\right) = \hat{U} + \Delta\hat{U}.$$

It is well known that if $\|A\|\left\|\Delta A^{-1}\right\| < 1$ then the change in $\hat{X}$ satisfies [24]

$$\left\|\Delta\hat{X}\right\| \leq \frac{\|A\|\left(\left\|\Delta A^{-1}\right\|\left\|\hat{X}\right\| + \left\|\Delta\hat{U}\right\|\right)}{1-\kappa(A)\left\|\Delta A^{-1}\right\|/\|A^{-1}\|}.$$

Lemma 2 provides that the change in area due to shifts in intersections is of the same magnitude as the change in the numerator of q_0. This is proportional to $L\left\|\Delta\hat{X}\right\|$. The change in area due to shifts in vertices of X is equal to the magnitude of the shifts times the diameter of Y which is $\mathcal{O}(1)$. Thus, the change in area of the intersection is $\mathcal{O}\left((1+L)\left\|\Delta\hat{X}\right\|\right)$.

Corollary 1 gives the list of possible representations. The shape of the polygon of intersection is continuous with respect to change in any given vertex or intersection since the graph rewrites of Figure 15 map these representations to themselves. Any shift in position is accounted for in Figure 16. Therefore, the change in area of the intersection remains $\mathcal{O}\left((1+L)\left\|\Delta\hat{X}\right\|\right)$.

The portions of $\hat{X}$ and $\Delta\hat{X}$ that lie inside Y must be transformed back into the coordinate system of U and V. The area of the polygon due to $\hat{X}$ is at most $\mathcal{O}(1)$. The perturbation of A^{-1} introduces a multiplication of this area by $\left\|\Delta A^{-1}\right\|$. This leads to the added term $\mathcal{O}(\Delta V)$ in the statement of the theorem.

The area due to $\Delta\hat{X}$ is multiplied by a constant proportional to $\left\|A^{-1}\right\|$. The change in area is then on the order of

$$\left\|A^{-1}\right\|(1+L)\left\|\Delta\hat{X}\right\| \leq \frac{\left\|A^{-1}\right\|\|A\|(1+L)}{1-\kappa(A)\left\|\Delta A^{-1}\right\|/\|A^{-1}\|}\left(\left\|\Delta A^{-1}\right\|\left\|\hat{X}\right\| + \left\|\Delta\hat{U}\right\|\right).$$

Given that $\left\|A^{-1}\right\|\|A\| = \kappa(A)$, $\left\|\hat{X}\right\| \leq L$, $\left\|A^{-1}\right\| = \mathcal{O}(V)$ and $\left\|\Delta A^{-1}\right\| = \mathcal{O}(\Delta V)$ one arrives at the equation in the statement of the theorem. □

If L, $\kappa(A)$ and $\mathcal{O}(\Delta V/V)$ are sufficiently small then PANG2 is backward stable. This is true if U is not significantly larger than V, the lengths of the edges of V differ by no more than a few orders of magnitude and the relative error in V is reasonable.

8 COMPARISON OF ALGORITHMS

We present another example problem to show that replacing the triangle intersection algorithm in PANG with the proposal presented here makes it robust. See Appendix A for how to make this replacement. This example was motivated by discussions with Frédéric Hecht at the CIRM Workshop on Parallel Solution Methods for Systems Arising from PDEs, September 16-20, 2019. It systematically generates failures of the triangle intersection subroutines in PANG to successfully compute all intersections to within an error on the order of machine precision.

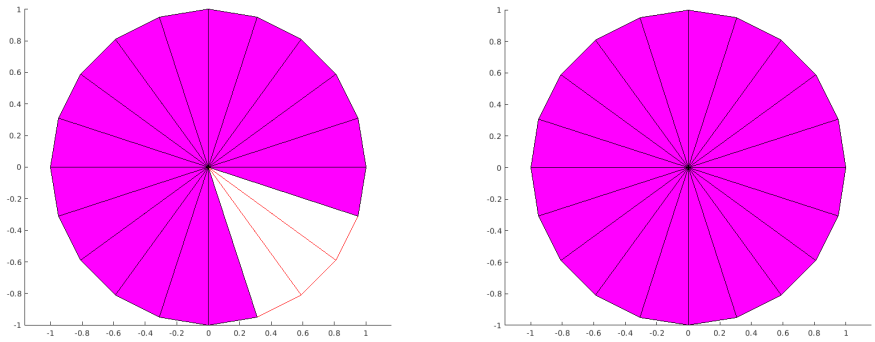


Fig. 18. Intersection of $\mathbb{T}$ and $\mathbb{T}_\epsilon$ for $\epsilon = 1e-16$ and $n = 20$. The magenta triangles represent calculated intersections.

The example considers a triangulation $\mathbb{T}$ with n identical triangles arranged radially around the origin. To define $\mathbb{T}$ we must define the grid points t_i for $i = 0, ..., n$ and the triangles T_i for $i = 0, ..., n-1$:

$$t_i = \left(\cos\left(\frac{i2\pi}{n}\right), \sin\left(\frac{i2\pi}{n}\right)\right), \; T_i = (0, t_i, t_{i+1}) \implies \mathbb{T} = \{T_i\}_{i=0}^{n-1}.$$

This grid is overlaid with a perturbation of itself, $\mathbb{T}_\epsilon$. The point at the origin is shifted by ϵ in a random direction and the radial points are collectively rotated by $O(\epsilon)$ radians:

$$\tilde{t}_i = \left(\cos\left(\frac{i2\pi}{n} + r_1\right), \sin\left(\frac{i2\pi}{n} + r_1\right)\right), \; \tilde{T}_i = (\epsilon r_2, \tilde{t}_i, \tilde{t}_{i+1}) \implies \mathbb{T}_\epsilon = \left\{\tilde{T}_i\right\}_{i=0}^{n-1},$$

where $|r_1| \leq \epsilon 2\pi/n$ and r_2 is a unit vector in a random direction.

For large enough ϵ errors in consistency are avoided: All subroutines of PANG agree on the underlying geometry of the problem. As ϵ becomes smaller, the triangulations become nearly identical. A small discrepancy between two steps of the triangle intersection algorithm in PANG has already been shown to cause large errors. The overlap of these triangulations presents several such discrepancies.

Figure 18 shows the resulting intersections between $\mathbb{T}$ and $\mathbb{T}_\epsilon$ for $\epsilon = 1e-16$ and $n = 20$. The original triangle intersection algorithm in PANG fails to find the intersection between T_i and $\tilde{T}_i$ for $i = 16$, 17 and 18. The proposed algorithm finds all intersections between T_i and $\tilde{T}_i$. Any remaining intersections (which cannot be independently verified) have area $O(\epsilon)$.

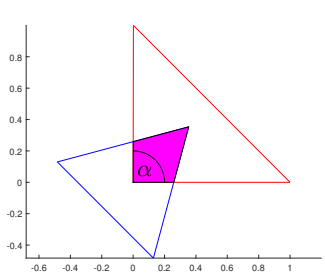
8.1 Comparison of accuracy and computation time

While comparisons of robustness with other polygon clipping algorithms proves challenging and their inclusion in projection algorithms complicated, we can easily compare accuracy and computation time on a single pair of triangles. Let V be the clipping triangle with two sides equal to 1 and the angle between these sides equal to α. Let U, the subject triangle, be an equilateral triangle whose vertices are all 0.5 from the vertex of V with angle α, one of which is halfway between the edges of V with equal length. The coordinates of the vertices as functions of α are presented in Table 4.

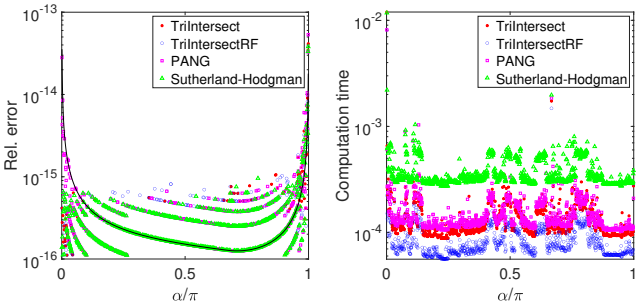
We compare four triangle-triangle intersection algorithms: the original PANG [12]; PANG2, with both change of coordinates and reference-free parametrization and; Sutherland-Hodgman,

V			U		
0	0	$\cos(\alpha)$	$\cos(\alpha/2)$	$\cos(\alpha/2+2\pi/3)$	$\cos(\alpha/2+4\pi/3)$
0	1	$\sin(\alpha)$	$\sin(\alpha/2)$	$\sin(\alpha/2+2\pi/3)$	$\sin(\alpha/2+4\pi/3)$

Table 4. Coordinates of the clipping and subject triangles.



(a) V (red) and U (blue) for $\alpha = \pi/2$.



(b) Relative error (left) and computation time (right) as a function of α/π.

Fig. 19. Example comparing accuracy and computation time of four triangle-triangle intersection algorithms.

an established polygon clipping algorithm [4]. The implementations of PANG2 are identical to Programs 1, 2 and 3.

Figure 19 shows the results for α between 0 and π. In terms of accuracy, PANG appears to have trouble with small angles, while the reference-free parametrization is nominally worse for midrange angles. In terms of computation time, this implementation of the reference-free parametrization is marginally faster than the change of coordinates. Both versions of PANG2 are faster than their competitors, in particular by nearly an order of magnitude over the more established Sutherland-Hodgman.

ACKNOWLEDGMENTS

The authors would like to thank Jerónimo Rodriguez Garcia for providing the example depicted in Figure 1 and Frédéric Hecht for the example used in Section 8.

A ADDITIONAL CODE

The algorithm PANG may be found in its entirety in [12]. To make use of the triangle intersection algorithm proposed here one needs the primitives found in Programs 1 and 2. PANG may be updated to use these primitives by using the patch file [23] found in Program 4. The line `+[P1,Q,R,n] = TriIntersect(X,Y);` may be replaced by `+[P1,Q,R,n] = TriIntersectRF(X,Y);` to use Program 3 instead.

Testing of the example in Figure 18 using the reference-free parametrization revealed that consistency over a single triangle-triangle intersection does not provide robustness in the advancing front algorithm used in PANG. While the calculation of an intersection may indicate a pair of intersecting triangles, this pair may have an intersection on the order of machine epsilon and can therefore be empty in floating-point arithmetic. The original advancing front algorithm of PANG kept only one intersecting pair when transitioning to another triangle of the second mesh. If the error described above occurs, the search terminates.

```
@@ -8,13 +8,11 @@
 %   element Y. The numerical challenges are handled by including
 %   points on the boundary and removing duplicates at the end.

-[P,n]=EdgeIntersections(X,Y);
-P1=PointsOfXInY(X,Y);
+[P1,Q,R,n] = TriIntersect(X,Y);
 if size(P1,2)>1                      % if two or more interior points
   n=[1 1 1];                         % the triangle is candidate for all
 end                                  % neighbors
-P=[P P1];
-P=[P PointsOfXInY(Y,X)];
+P=[P1 Q R];
 P=SortAndRemoveDoubles(P);           % sort counter clock wise
 M=zeros(3,3);
 if size(P,2)>0
```

Program 4. Patch file to update PANG. The primitive `TriIntersect(X,Y)` may be replaced with the primitive `TriIntersectRF(X,Y)` to use the reference-free parametrization.

Na	-2.134346793664016	-1.294592514989478	-1.774411380685791
	0.633829593520260	0.622483665789736	1.328933577453528
Nb	-2.134346793664016	-0.454838236314940	-1.414475967707566
	0.633829593520260	0.611137738059212	2.024037561386796

Table 5. Positions of the triangle vertices of Figure 1.

To avoid this, one can modify the algorithm so as to keep all intersecting pairs found over the course of the algorithm so far. The intersecting pairs are collected as intersections are calculated. This improved advancing front algorithm replaces the first part of PANG and contains calls to the other primitives. This replacement may be found in Program 5.

To allow replication of the example of Figure 1 we present in Table 5 the exact positions in double precision of the triangle vertices. The triangulations to pass to `InterfaceMatrix` are `Ta=Tb=[1,2,3,2,2,2]`.

B INVALID GRAPH CONFIGURATIONS

The proof of Lemma 4 requires drawing thirteen graphs to determine which satisfy all conditions of an intersection graph. Ten of these graphs are found in Figure 12. The three that are found to be invalid are presented in Figure 20.

The first of these, G†, from the pairing 2-3, contains an entire triangle in the polygon of intersection. As such, there are no intersections between this triangle and any other. The graph is therefore invalid.

The second two both come from the pairing 4-2. In both, one of the cycles identified by the triangle condition has four triangle vertices. Thus, the cycle does not correspond to a triangle at all, invalidating both graphs.

C ADDITIONAL PROOFS

Lemma 6. *The determinant of* $\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^{\perp}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{w}_j & \mathbf{w}_j^{\perp}\end{bmatrix}$ *is positive.*

```
function M=InterfaceMatrix(Na,Ta,Nb,Tb)
% INTERFACEMATRIX projection matrix for nonmatching triangular grids
%   M=InterfaceMatrix(Na,Ta,Nb,Tb); takes two triangular meshes Ta
%   and Tb with associated nodal coordinates in Na and Nb and
%   computes the interface projection matrix M

na=size(Ta,1); nb=size(Tb,1);    % number of triangles in each list
cand=[1;1];                      % init. candidate pair
bl=1;                            % list of triangles in Tb to treat
bd=zeros(nb+1,1); bd([1,end])=1; % flag triangles already treated
M=sparse(size(Nb,2),size(Na,2));
while ~isempty(bl)
  bc=bl(1); bl=bl(2:end);        % pick out 1st tri. in bl
  al=cand(1,(cand(2,:)==bc));    % list of candidates intersecting bc
  al=unique(al);                 % avoid duplicates
  ad=zeros(na+1,1);
  ad(end)=1; ad(al)=1;           % flag triangles of Ta already treated
  while ~isempty(al)
    ac=al(1); al=al(2:end);
    [P,nc,Mc]=Intersect(Nb(:,Tb(bc,1:3)),Na(:,Ta(ac,1:3)));
    if ~isempty(P)
      M(Tb(bc,1:3),Ta(ac,1:3))=M(Tb(bc,1:3),Ta(ac,1:3))+Mc;
      ta=Ta(ac,4:6);             % nhbrs of ac
      al=[al ta(ad(ta)==0)];     % add those nhbrs not treated to al
      ad(ta)=1;                  % flag new entries of al
      tb=[ac,ac,ac;Tb(bc,4:6)];  % nhbrs of bc (and index for ac)
      tb=tb(:,nc==1);            % nhbrs of bc which intersect ac
      bl=[bl tb(2,bd(tb(2,:))==0)]; % add those nhbrs not treated to bl
      bd(tb(2,:))=1;             % flag new entries of bl
      cand=[cand tb];            % ac is candidate for new bl entries
    end
  end
end
end
```

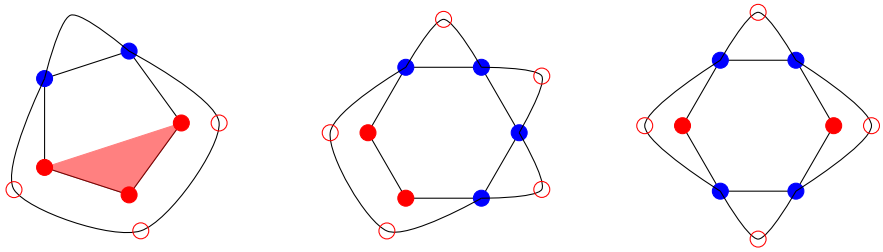
Program 5. Replacement of InterfaceMatrix.m, the driving file for PANG. The advancing front algorithm has been altered to use a larger list of candidate pairs.

Proof. First note that the determinant of $\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}^{-1}$ has the same sign as that of $\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}^\top$. Thus, we are concerned with the determinant of

$$\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}^\top \begin{bmatrix}\mathbf{w}_j & \mathbf{w}_j^\perp\end{bmatrix} = \begin{bmatrix}\mathbf{w}_i^\top\mathbf{w}_j & \mathbf{w}_i^\top\mathbf{w}_j^\perp \\ (\mathbf{w}_i^\perp)^\top\mathbf{w}_j & (\mathbf{w}_i^\perp)^\top\mathbf{w}_j^\perp\end{bmatrix}.$$

The value of $(\mathbf{w}_i^\perp)^\top\mathbf{w}_j$ is positive by the choice of $\mathbf{w}_i^\perp$ made in Section 2.1. The value of $\mathbf{w}_i^\top\mathbf{w}_j^\perp$ is

$$\mathbf{w}_i^\top\mathbf{w}_j^\perp = -(\mathbf{w}_j + \mathbf{w}_k)^\top\mathbf{w}_j^\perp = -\mathbf{w}_k^\top\mathbf{w}_j^\perp$$



(a) G† (b) H† (c) H†*

Fig. 20. The three graphs that do not meet the conditions required of a graph of intersection, see Lemma 4.

which is negative by the same choice. Since $\mathbf{w}_i^\perp$ and $\mathbf{w}_j^\perp$ are the vectors $\mathbf{w}_i$ and $\mathbf{w}_j$ under the same rotation their scalar products are the same. The determinant is then

$$\begin{aligned}\det\left(\begin{bmatrix}\mathbf{w}_i & \mathbf{w}_i^\perp\end{bmatrix}^\top\begin{bmatrix}\mathbf{w}_j & \mathbf{w}_j^\perp\end{bmatrix}\right) &= \begin{vmatrix}\mathbf{w}_i^\top\mathbf{w}_j & \mathbf{w}_i^\top\mathbf{w}_j^\perp\\ (\mathbf{w}_i^\perp)^\top\mathbf{w}_j & (\mathbf{w}_i^\perp)^\top\mathbf{w}_j^\perp\end{vmatrix}\\ &= \left(\mathbf{w}_i^\top\mathbf{w}_j\right)^2 + \mathbf{w}_k^\top\mathbf{w}_j^\perp\mathbf{w}_j^\top\mathbf{w}_i^\perp\end{aligned}$$

which is strictly positive. □

We use a bound on $\kappa(A)$ given by Hoffmann [15]. This bound requires modification for matrices where $C < 1$. For completeness we prove the form of the bound and a slightly tighter version.

Lemma 7. *Let $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^2$ such that α, the angle between them, is less than $\pi/2$, then*

$$\kappa\left(\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}\right) < \frac{1}{C\sin(\alpha/2)} \tag{3}$$

where $C \le 1$ is the ratio of $\|\mathbf{v}_1\|$ to $\|\mathbf{v}_2\|$.

Proof. The matrix $\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}$ may be expressed as

$$\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix} = cR\begin{bmatrix}1 & \cos(\alpha)\\ 0 & \sin(\alpha)\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & C\end{bmatrix}$$

where R is a rotation matrix and c is a scalar. The condition number is invariant under rotation and scalar multiplication, and so

$$\kappa\left(\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}\right) = \kappa\left(\begin{bmatrix}1 & \cos(\alpha)\\ 0 & \sin(\alpha)\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & C\end{bmatrix}\right) \le \kappa\left(\begin{bmatrix}1 & \cos(\alpha)\\ 0 & \sin(\alpha)\end{bmatrix}\right)\kappa\left(\begin{bmatrix}1 & 0\\ 0 & C\end{bmatrix}\right) = \frac{\cot(\alpha/2)}{C} < \frac{1}{C\sin(\alpha/2)}.$$

□

REFERENCES

[1] Jorge Albella, Hachmi Ben Dhia, Sebastien Imperiale, and Jerónimo Rodríguez. 2019. Mathematical and numerical study of transient wave scattering by obstacles with a new class of Arlequin coupling. *SIAM J. Numer. Anal.* 57, 5 (oct 2019), 2436–2468. https://doi.org/10.1137/19M1263959

[2] Jan Březina and Pavel Exner. 2017. Fast algorithms for intersection of non-matching grids using Plücker coordinates. *Computers and Mathematics with Applications* 74, 1 (2017), 174–187. https://doi.org/10.1016/j.camwa.2017.01.028

[3] Patrick Ciarlet, Erell Jamelot, and Félix D. Kpadonou. 2017. Domain decomposition methods for the diffusion equation with low-regularity solution. *Computers & Mathematics with Applications* 74, 10 (nov 2017), 2369–2384. https://doi.org/10.1016/J.CAMWA.2017.07.017

[4] Rosetta Code. 2021. *Sutherland-Hodgman polygon clipping.* Rosetta Code. Retrieved February 23, 2021 from https://rosettacode.org/wiki/Sutherland-Hodgman_polygon_clipping#MATLAB_.2F_Octave

[5] Simone Coniglio, Christian Gogu, and Joseph Morlier. 2019. Weighted average continuity approach and moment correction: New strategies for non-consistent mesh projection in structural mechanics. *Archives of Computational Methods in Engineering* 26, 5 (nov 2019), 1415–1443. https://doi.org/10.1007/s11831-018-9285-0

[6] Mike Cyrus and Jay Beck. 1978. Generalized two-and three-dimensional clipping. *Computers & Graphics* 3, 1 (1978), 23–28.

[7] Thomas Dickopf and Rolf Krause. 2014. Evaluating local approximations of the L 2-orthogonal projection between non-nested finite element spaces. *Numerical Mathematics: Theory, Methods and Applications* 7, 3 (2014), 288–316.

[8] Patrick E. Farrell. 2009. *Galerkin projection of discrete fields via supermesh construction.* Ph.D. Dissertation. Imperial College London.

[9] Patrick E. Farrell, Matthew D. Piggott, Christopher C. Pain, Gerard J. Gorman, and Cian R. Wilson. 2009. Conservative interpolation between unstructured meshes via supermesh construction. *Computer methods in applied mechanics and engineering* 198, 33-36 (2009), 2632–2642.

[10] Steven Fortune. 1989. Stable maintenance of point set triangulations in two dimensions. In *Annual Symposium on Foundations of Computer Science (Proceedings).* IEEE Computer Society, Los Alamitos, CA, USA, 494–499. https://doi.org/10.1109/sfcs.1989.63524

[11] Martin J. Gander and Caroline Japhet. 2009. An algorithm for non-matching grid projections with linear complexity. In *Domain decomposition methods in science and engineering XVIII.* Springer, Berlin, Heidelberg, 185–192.

[12] Martin J. Gander and Caroline Japhet. 2013. Algorithm 932: PANG: software for nonmatching grid projections in 2D and 3D with linear complexity. *ACM Trans. Math. Software* 40, 1 (2013), 6.

[13] Günther Greiner and Kai Hormann. 1998. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics* 17, 2 (1998), 71–83.

[14] Stéphane Guinard, Robin Bouclier, Mateus Toniolli, and Jean-Charles Passieux. 2018. Multiscale analysis of complex aeronautical structures using robust non-intrusive coupling. *Advanced Modeling and Simulation in Engineering Sciences* 5, 1 (dec 2018), 1. https://doi.org/10.1186/s40323-017-0094-z

[15] Christoph M. Hoffmann. 1989. The problems of accuracy and robustness in geometric computation. *Computer* 22, 3 (1989), 31–39.

[16] You-Dong Liang and Brian A. Barsky. 1984. A new concept and method for line clipping. *ACM Transactions on Graphics* 3, 1 (Jan. 1984), 1–22. https://doi.org/10.1145/357332.357333

[17] Federico Moro and Massimo Guarnieri. 2015. Efficient 3-D domain decomposition with dual basis functions. *IEEE Transactions on Magnetics* 51, 3 (mar 2015), 1–4. https://doi.org/10.1109/TMAG.2014.2352034

[18] William M. Newman and Robert F. Sproull. 1979. *Principles of Interactive Computer Graphics.* McGraw-Hill, Inc., New York.

[19] Ari Rappoport. 1991. An efficient algorithm for line and polygon clipping. *The Visual Computer* 7, 1 (1991), 19–28.

[20] Jonathan Richard Shewchuk. 1997. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry* 18, 3 (1997), 305–363.

[21] Václav Skala. 1993. An efficient algorithm for line clipping by convex polygon. *Computers & Graphics* 17, 4 (1993), 417–421.

[22] Ivan E. Sutherland and Gary W. Hodgman. 1974. Reentrant polygon clipping. *Commun. ACM* 17, 1 (1974), 32–42.

[23] Larry Wall, Paul Eggert, Wayne Davison, David MacKenzie, and Andreas Grünbacher. 2009. *PATCH(1) Linux User Manual.* GNU patch.

[24] David S. Watkins. 2004. *Fundamentals of matrix computations.* Vol. 64. John Wiley & Sons, New York.

[25] Kevin Weiler and Peter Atherton. 1977. Hidden surface removal using polygon area sorting. In *Computer Graphics*, Vol. 11. ACM, New York, 214–222. Issue 2.

[26] Barbara I. Wohlmuth and Rolf H. Krause. 2003. Monotone multigrid methods on nonmatching grids for nonlinear multibody contact problems. *SIAM Journal on Scientific Computing* 25, 1 (sep 2003), 324–347. https://doi.org/10.1137/S1064827502405318