## Discrete Topology Generation

## Triangle Intersection Tests

Two triangle-triangle intersection (TTI) tests were found to be the most widely used in relevant literature [2],[3] and the one implemented here will be discussed. For the purposes of describing TTI tests, let us denote the two triangles $T_0$ and $T_1$, and the nodes of the $T_0$ and $T_1$ as $N_{00}$, $N_{10}$, $N_{20}$, and $N_{01}$, $N_{11}$, $N_{21}$, respectively. Also let us state that for two triangles to intersect in three dimensions, the following two conditions must exist: two edges of each triangle must cross the plane of the other, and if so, then two edges must intersect the aforementioned planes within the boundaries of the triangles.

The TTI demonstrated by Aftosmis [3] involves a Boolean check for intersection that only involves multiplication and division and does not involve expensive operations like square roots and trigonometric functions. Once the triangles are found to be intersecting, the end points of the line segment defining the intersection can be calculated. The aforementioned Boolean test involves the calculation of the signed volume of a tetrahedron, $T_{abcd}$, where $a$, $b$, $c$, and $d$ are the nodes that define the tetrahedron and $a_i$, $b_i$, $c_i$, and $d_i$ are the node coordinates. This signed volume is calculated using Equation 2.

$$6V(T_{abcd}) = \begin{vmatrix} a_0 & a_1 & a_2 & 1 \\ b_0 & b_1 & b_2 & 1 \\ c_0 & c_1 & c_2 & 1 \\ d_0 & d_1 & d_2 & 1 \end{vmatrix} = \begin{vmatrix} a_0 - d_0 & a_1 - d_1 & a_2 - d_2 \\ b_0 - d_0 & b_1 - d_1 & b_2 - d_2 \\ c_0 - d_0 & c_1 - d_1 & c_2 - d_2 \end{vmatrix} \quad \text{Eq. 1}$$

The result is six times the volume of the tetrahedron used to construct the equation. The sign of the volume, $T_{abc}$, is negative when the triangle formed by nodes *abc* forms a clockwise circuit

when viewed from the observation point of node *d*. This Boolean test constitutes a topological primitive and is the fundamental building block for all TTI tests performed in this research.

Recall that for two triangles to intersect in three dimensions, the following two conditions must exist: two edges of each triangle must cross the plane of the other, and if so, then two edges must intersect the aforementioned planes within the boundaries of the triangles. To determine if two edges of each triangle cross the plane of the other, the following is done.

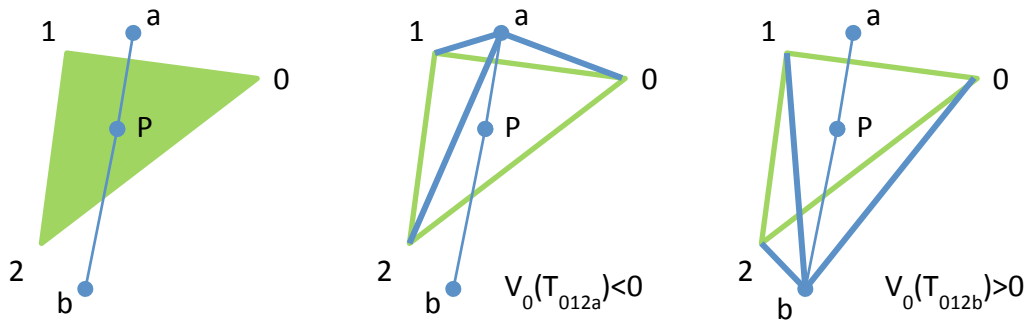


Figure 1.1 Triangle-Edge intersection test using topological primitive

Above in Figure 1.1, an example of using signed tetrahedral volumes to determine if a line segment pierces a plane is shown [3]. The signed volume defined by *(0,1,2,a)*, $V(T_{0,1,2,a})$, is compared to the signed volume *(0,1,2,b)*, $V(T_{0,1,2,b})$. If they are of opposite sign then the line segment pierces the plane of the triangle. This must be done for each edge to check to make sure that at least two edges from each triangle pass this check. Next, it must be determined if the line segment found to intersect the plane of the triangle pierces within the boundaries of the triangle.

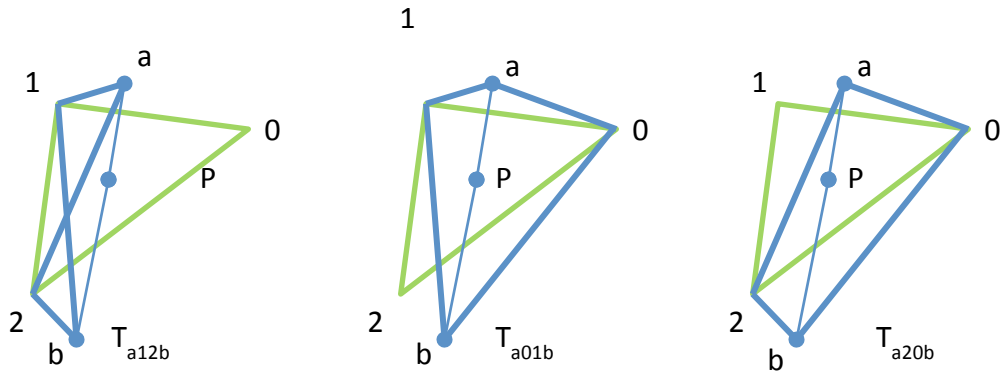


Figure 1.2 Triangle-Edge Intersection, Edge within boundary of Triangle

$$[V(T_{a,1,2,b}) < 0 \text{ and } V(T_{a,0,1,b}) < 0 \text{ and } V(T_{a,2,0,b}) < 0] \text{ or}$$
$$[V(T_{a,1,2,b}) > 0 \text{ and } V(T_{a,0,1,b}) > 0 \text{ and } V(T_{a,2,0,b}) > 0] \qquad \text{Eq. 3}$$

In Figure 1.2, an example of using signed tetrahedral volumes to determine if a line segment pierces a plane within the boundaries of a triangle is shown [3]. Three volumes must be checked to determine if the line segment pierces within the boundary of the triangle. The volumes are denoted $V(T_{a,1,2,b})$, $V(T_{a,0,1,b})$, $V(T_{a,2,0,b})$. If all of these volumes have the same sign, Equation 3, then the edge pierces within the boundaries of the triangle and the pair of triangles intersects.

A topological primitive is defined in [3] as, "an operation that tests an input and results in one of a constant number of cases." It is further stated that, "Such primitive can only classify, and constructed objects (like the actual locations of the pierce points…) cannot be determined without further processing. These primitives do, however, provide the intersections implicitly, and this information suffices…" In this case, the constant number of cases that can be returned from the volume calculation is three: positive (+), negative (-), or zero. Positive and negative results represent non-degenerate cases and zero represents some degeneracy involved with the geometry. By defining "zero" locally for each pair of triangles tested for intersection, this tool

becomes very robust and does not need computationally-expensive, exact-arithmetic routines. See the section on robustness later in this section for a more detailed explanation on how computational errors associated with degenerate geometries are handled.

## Neighbor Tracing

Lo and Wang [4] presented a method for further reducing the cost of repairing intersecting triangular meshes. The intersection between discrete surfaces is defined by a set of connected line segments. Each pair of triangles that intersect contributes one line segment to this set. Instead of relying solely on a spatial subdivision scheme to reduce the number of TTI tests performed, Lo and Wang [4] proposed that once a pair of intersecting triangles was found that the topology of the mesh be used to construct the set of line segments defining the intersection. They denoted this process "Tracing Neighbors of Intersecting Triangles (TNOIT)." TNOIT involves first finding a pair of intersecting triangles, and then the topological relations in the mesh can be used to move along the lines of intersection in the mesh—further reducing the number of TTI tests required to repair the mesh.

The determination of how to move through the mesh is determined on the type of intersection present. In Figure 1.3, three different types of intersections can be seen. Type 1 is a general intersection where one edge from each triangle intersects the other triangle. Type 2 is a special case of a general intersection where two edges from one triangle pierce the other triangle. Type 3 has only one edge that pierces. This means that of the two points that define the line segment that defines the intersection, one is a node in the existing geometry.

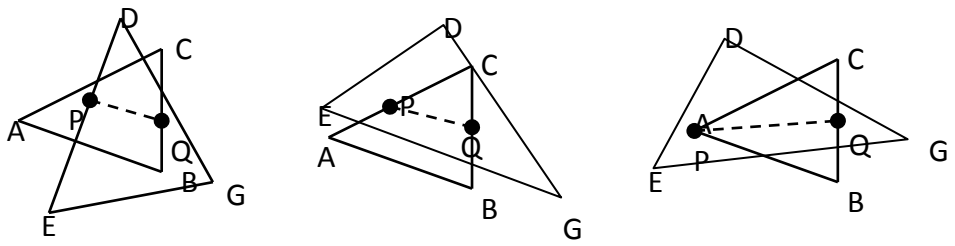


Figure 1.3 Three fundamental types of triangle intersections

In Figure 1.4 the three different types of intersections can be seen in place in a local mesh. This demonstrates how TNOIT can be used to construct the chain or loop of line segments that define an intersection. Starting with, as indicated in Figure 1.4a, triangles F1 and T1, if the intersection point, P, lies on the edge of F1, then the next pair to be tested for intersection should be T1 and F2—which is topologically adjacent to F1 across the edge. In Figure 1.4b, a similar process is used to move from the pair T1 and F1 to F1 and T2. However, in Figure 1.4c, the next intersection point is a node and therefore all of the topologically adjacent elements, T1-T5, must be tested for intersection with F1 before moving on.

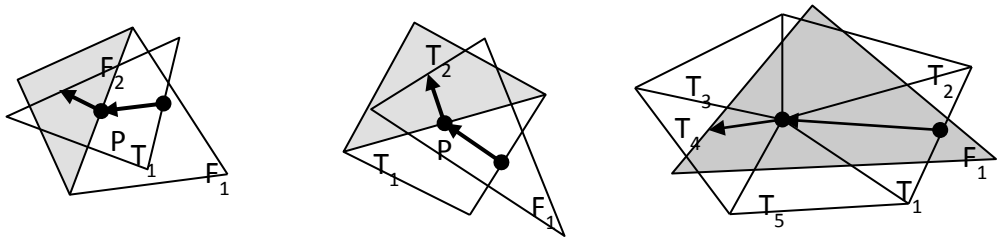


Figure 1.4 Three possibilities of how to move through a mesh using neighbor tracing

In addition to the above three intersection types, others which include degenerate geometries have been developed. As can be seen in Figure 1.5, an edge might not pierce within the boundaries of a triangle. If it does not, then it either must pierce an edge of the triangle, or an edge pierces a node of the triangle. Each of these requires different methods of moving to the

next pair of intersecting triangles. In Figure 1.5a, an edge of T1 intersects an edge of F1. This means that both edges would have to be traversed in order to move to the next pair of intersecting triangles. In Figure 1.5b, an edge of T1 intersects a node of F1. This means the element topologically adjacent to T1 would have to be tested against every element attached to the intersecting node, P, in order to move to the next pair of intersecting triangles.

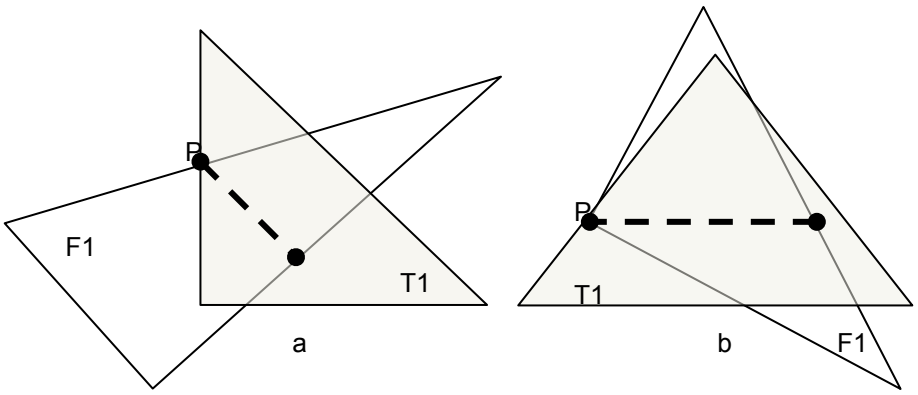


Figure 1.5 Degenerate possibilities of intersections

## Local Repair

While tracing the segments through the mesh, the segments that lie in each triangle are stored for later use. These line segments represent the intersection between the two surfaces. In order to remove the intersection, the line segments must be inserted into both surfaces. This would leave a set of non-manifold edges shared by the intersecting surfaces, but the intersection would be removed. The process of inserting these line segments into the surfaces is simplified by the realization that each triangle has a set of edges that need to be inserted locally. This means that instead of a global set of edges to insert into the mesh, the problem can be broken into many smaller sets of edges inserted into one triangle locally. Inserting edges in a triangle is strictly a two dimensional task and no attempt to make a three dimensional generalization of this

procedure is made here. A temporary, two-dimensional mesh is constructed out of the triangle and the nodes that define the edges. This local, two-dimensional transformation is accomplished by rotating the geometry into the x-y.

By rotating the geometry, instead of projecting it, or using any other means, the undistorted geometry is transformed into two-dimensional space. This is important because if the wrong geometry were created in two-dimensional space because of an incorrect transformation, the resulting three-dimensional geometry would also be incorrect. An example of the rotated geometry, including the triangle and the to-be-inserted edges can be seen in Figure 1.6a.

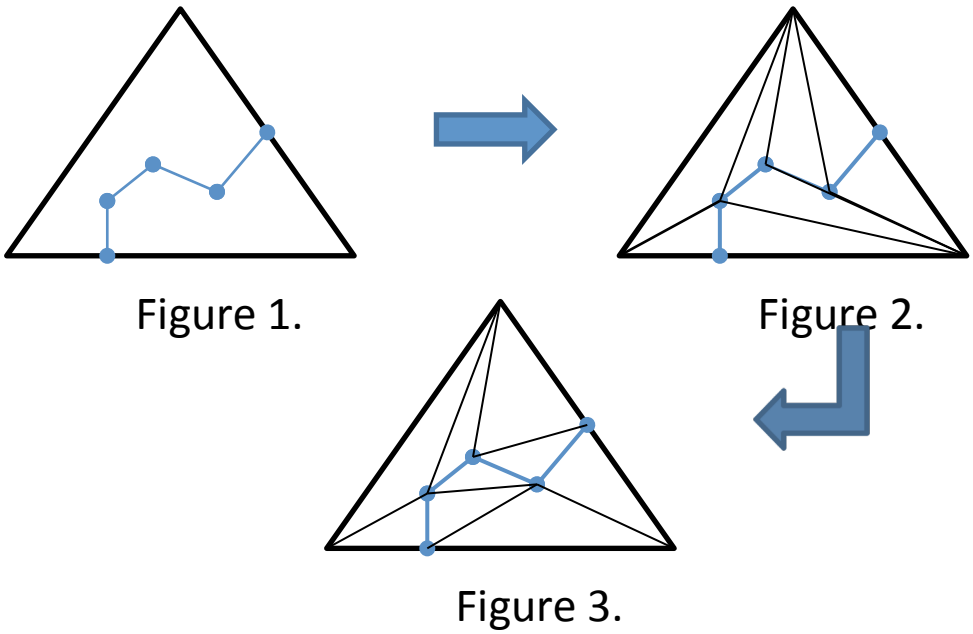


Figure 1.6 Local repair view edge insertion

The edges are inserted individually by first inserting the defining nodes and then recovering the edge, Figure 1.6b. A local min-max reconnection pass is then performed until no more edges fail

the min-max test, Figure 1.6c. Each of these steps, node insertion, edge recovery, and local reconnection will now be discussed in more detail.

## 1.1 Node Insertion

In order to put edges into the triangulation, the nodes that define the edges must first be inserted. The process of finding the triangle that contains the node involves another topological primitive. Let the vector from node $N_0$ to node $N_1$, and node $N_0$ to node $N_2$ be denoted as follows in Equation 4a and Equation 4b.

$$\overline{N_{01}} = \left[ N_{1x} - N_{0x}; N_{1y} - N_{0y}; N_{1z} - N_{0z} \right] \quad \text{a.}$$

$$\overline{N_{02}} = \left[ N_{2x} - N_{0x}; N_{2y} - N_{0y}; N_{2z} - N_{0z} \right] \quad \text{b.}$$

$$A_{0,1,2} = \frac{1}{2} \cdot \left( \left\| \overline{N_{01}} \otimes \overline{N_{02}} \right\| \right) \quad \text{c}$$

Eq. 4

This topological primitive in this case is the area of a triangle. Equation 4b is used to calculate the area of a triangle. In two dimensions, the absolute value is removed because the only non-zero component will be the $z$ component. The $z$ component, along with its sign, is taken to be the area of the two dimensional triangle. The calculated area is positive if the nodes form a counter-clockwise circuit, i.e., the resulting vector is in the positive (+) $z$ direction. In order to test if a node is within the boundaries of a triangle, three areas must be checked.

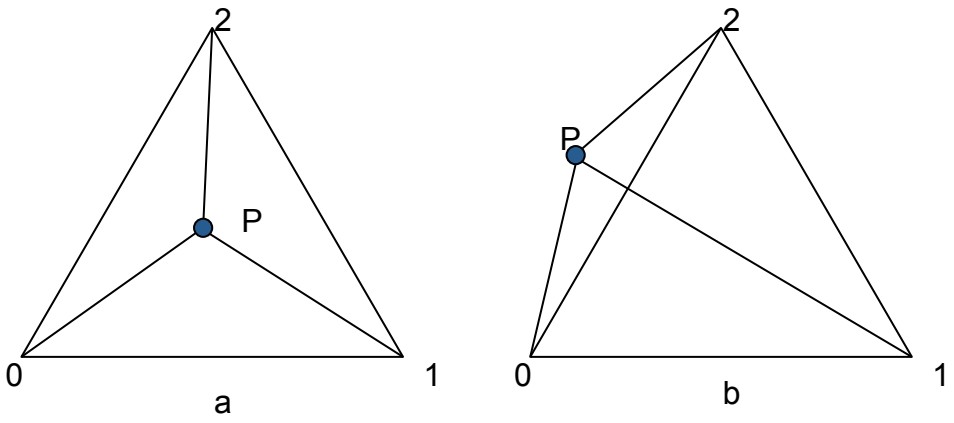


Figure 1.7 Containing triangle area check

In Figure 1.7a, the three areas that must be checked are the triangles formed by *(0,1,P)*, *(1,2,P)*, and *(2,0,P)*. If all of these areas are positive, the node, *P*, is in the interior of the triangle, *(01,2)*. However, in Figure 1.7b, all three areas are not positive. The area of triangle *(2,0,P)* is negative. Therefore the edge, *(02)* is considered to be "associated with" the negative area. Because of negative area formed by *(2,0,P)*, the node *P* is not in the interior of triangle *(0,1,2)*. The two dimensional node-in-triangle check is used as a path finding mechanism for finding the containing triangle of a node. Consider Figure 1.8, in which the search for the containing triangle begins in the seed face. Since the node does not reside in the seed face, the edge that is associated with the negative area is traversed. For example, in Figure 1.7b, the searching algorithm would go to the element that is topologically adjacent to edge *(02)*.

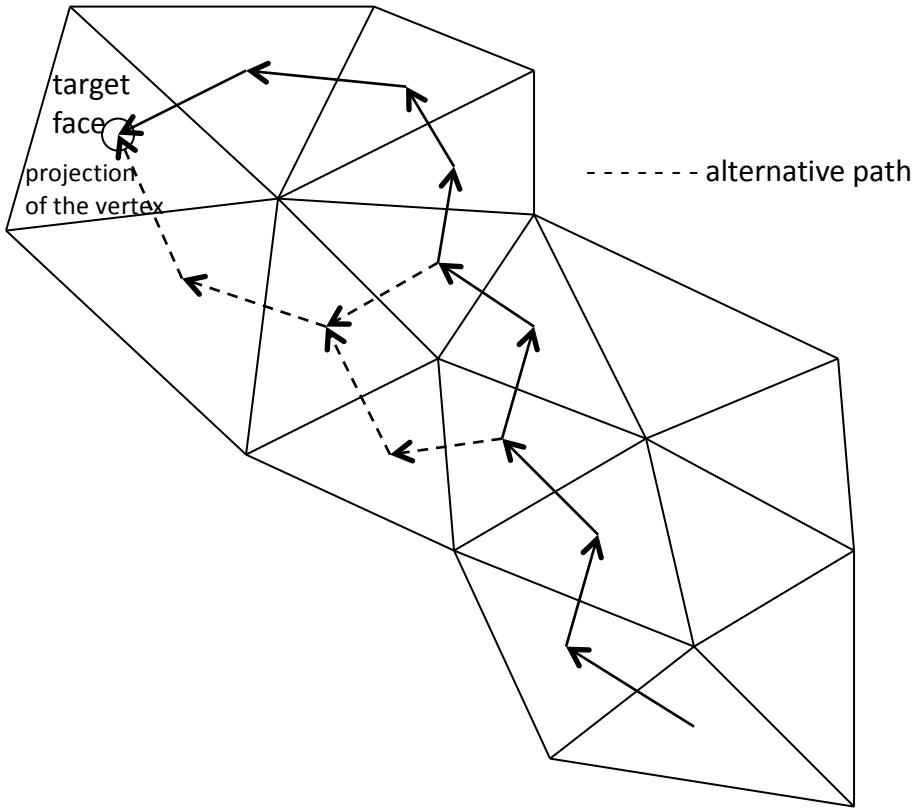


Figure 1.8 Two-dimensional, containing triangle search example

This process, calculating areas and traversing edges, is repeated until the containing triangle is found or the search fails because the node lies on an edge. If a containing triangle is found, it is split into three triangles, as seen in Figure 1.9a. If the node is found to lie on an edge, the edge is split as seen in Figure 1.9b.

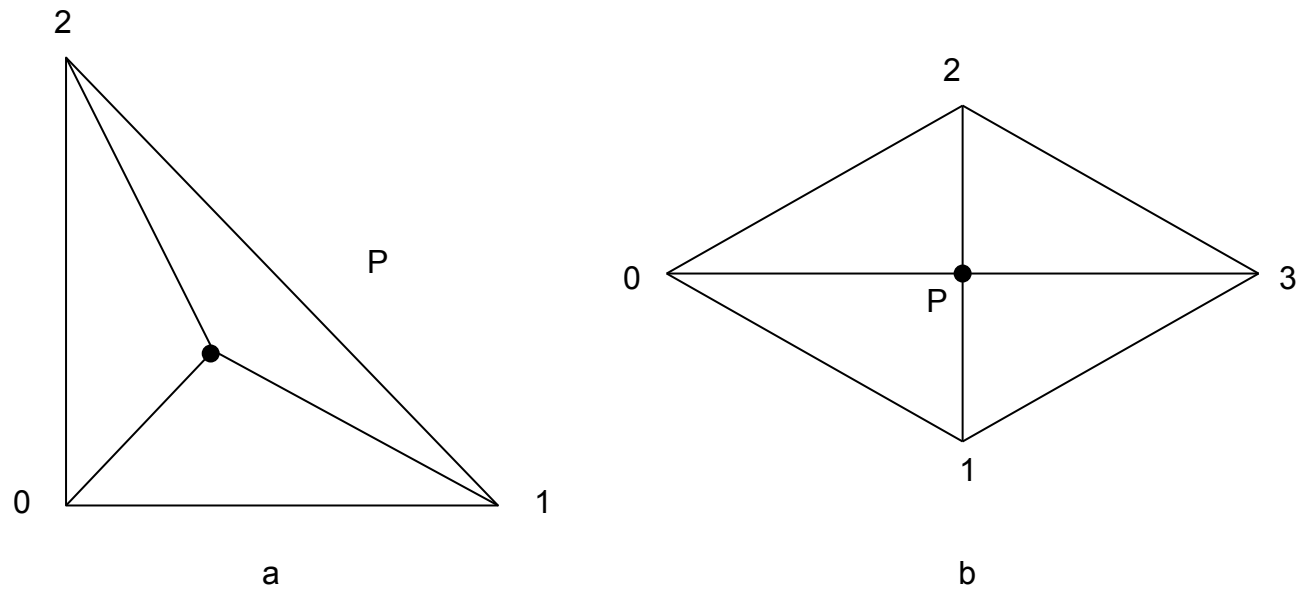


Figure 1.9 Triangle splitting and edge splitting example.

It is possible to not include the edge splitting option and rely on the local reconnection routine to improve mesh quality. However, the creation of nearly degenerate geometry by inserting nodes that are close to edges might cause the subsequent node insertions or containing-triangle searches to fail. Nearly degenerate geometry could also cause incorrect results from numerical inaccuracies. Therefore, the edge-split option was included.

## 1.2 Edge Recovery

Once the defining nodes of an edge are successfully inserted into the triangulation, the edge itself must be recovered. It has been proven that the recovery of an edge in two dimensions is always guaranteed through a topological operation called edge swapping (Figure 1.13) **Error! Reference source not found.**. In order to recover the edge, a list of edges that should be swapped needs to be constructed. It should be noted that the equations used to determine if two edges intersect within some tolerance do not calculate the point of intersection directly. They

instead calculate the closest point on an edge to the other edge. The list of edges that should be swapped contains only the edges that intersect the to-be-recovered edge. An example of this can be seen in Figure 1.10. Starting at the node on the left, each edge that intersects the to-be-recovered edge is traversed—and stored—until the node on the right is found.

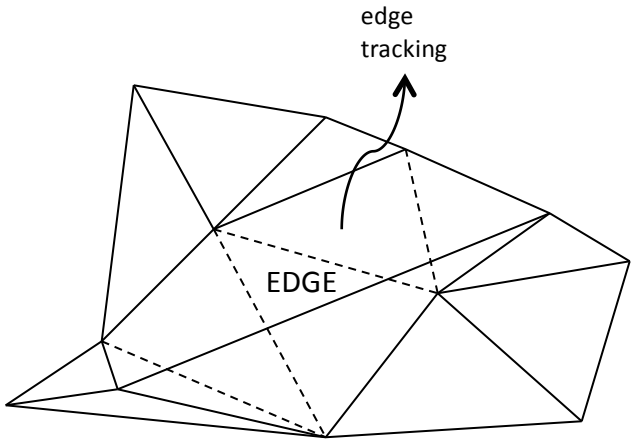


Figure 1.10 Finding edges that intersect the to-be-recovered edge using edge tracking

Once the list has been constructed, the following algorithm can be used to recover the desired edge [6].

1. Each edge of the set is swapped if
    a. *First Contraint*: its new swapped configuration does not create intersections
    b. *Second Contraint*: its new swapped configuration does not intersect the to-be-recovered edge.
2. If there are edge left unswapped in the list due to the constraints of 1(a) and 1(b) then the following strategy is performed.
    a. Relax the second constraint for the first unswapped edge and try to perform the swaps of the rest of unswapped edges. Flag the first relaxed edge still unswapped for the secong visit.
    b. A sweep of edges with both of the constraints being in effect is followed for swap.
    c. This trial scheme is continued until the edge is recovered or no swap could be performed due to geometrical validity, i.e. first constraint.

d. If the edge is not recovered due to the first constraint then it is replaced with a set of edges forming a path between its end vertices. These edge pieces are recursively tried to be recovered.

Figure 1.11 Edge swapping algorithm

This process is demonstrated in Figure 1.12. The list of swappable edges includes edge, 1, 2, 3, and 4. Looping through the edges on Figure 1.12 the first pass we see that edges 1 and 2 can be swapped. Edge 3 cannot be swapped because it violates condition 1a from Figure 1.11. Once edge 4 is swapped, then edge 3 can be swapped in the second pass to arrive at the desired geometry. It is worth noting again that this process is guaranteed to converge to the desired result in two dimensions.

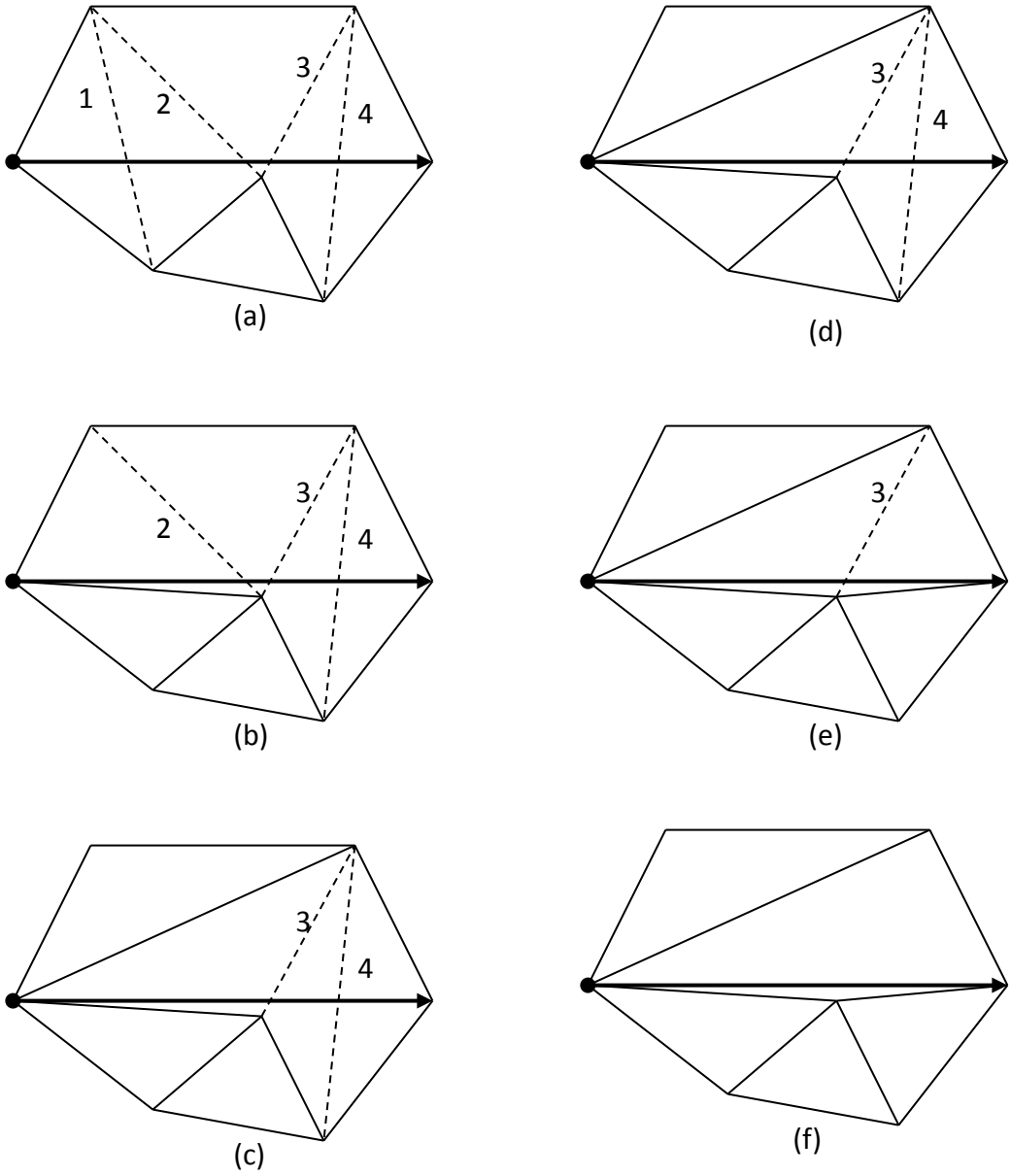


Figure 1.12 Edge recovery process via edge swapping

## 1.3 **Local Reconnection**

Once all of the required edges have been recovered in the temporary mesh, a constrained min-max (minimize the maximum angle) reconnection algorithm is used to improve the element quality is the mesh. The aforementioned constraints are the inserted edges. These edges must be present for the final geometry to repair the intersection. For a local edge to be reconnected, its

reconnected state must reduce the maximum angle of the current state. An example of this can be seen in Figure 1.13.

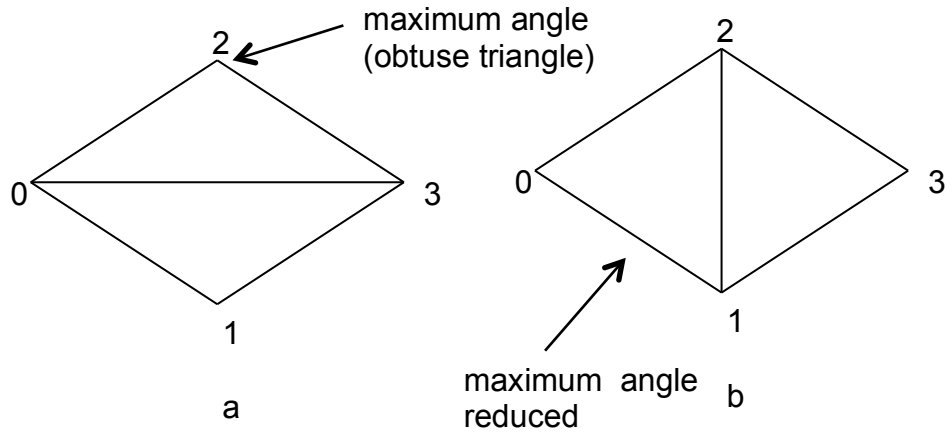


Figure 1.13 Local reconnection example using Min-Max criterion

In addition to the reconnection criterion, a stopping criterion was needed to ensure that the process did not reconnect geometry needlessly. If the maximum angle in the current or reconnected geometry is near ninety degrees, then the edge is left alone—since this can cause endless reconnections to be made while trying to improve the element quality. The loop that reconnects this geometry, since it avoids infinite reconnections, is guaranteed to converge [7].

## 1.4 **Translating Local to Global**

As stated previously, a two dimensional mesh was created for the purposes of simplifying the process of inserting nodes and subsequent edges into individual triangles. After all of the nodes have been inserted and edges recovered in the two-dimensional mesh, the topology of the two-dimensional mesh is used to update the topology of the three-dimensional mesh without any further transformations. This is accomplished through the use of "parent" nodes. The node class has a data member called a parent_GRX_NODE_ which is a pointer to a node. Since all of the

geometry exists in three dimensions and then is transformed to two dimensions, each of the "two-dimensional" nodes has a "parent" from which it is derived or created. Creating the three-dimensional topology from the two dimensional, temporary mesh is as simple as creating all of the triangles that exists in the two dimensional mesh using the "parent" nodes instead of the "child" nodes for the connectivity. No additional calculations are used to transform the temporary mesh back to three dimensions—only the connectivity from the two dimensional mesh.

## Post Processing Intersecting Mesh

The process of inserting the line segments, or edges, defining the intersection into all of the appropriate discrete surfaces necessarily creates non-manifold meshes. The purpose of this tool is to aid in the production of watertight, manifold meshes. Therefore, some way of removing these non-manifold meshes needed to be developed, otherwise the intersection has removed one problem, intersecting geometry, and created another, non-manifold edges. One solution is to use a surface painting algorithm. This post-processing step of surface painting would, if possible, "break-out" the surface defined in part or in whole by the non-manifold edges just created by the mesh intersection routines. These surfaces that have been "broken-out" can be removed or kept by the user based on the desired results.

## References


[1] Park, Sang C., "Triangular mesh intersection," Department of Information & Systems Engineering, Ajou University, August 2004.

[2] Möller, Tomas, "A Fast Triangle-Triangle Intersection Test," *Journal of Graphics Tools*, 2, no. 2 (1997).

[3] Aftosmis, M. J., Berger, M. J., and J. E. Melton, "Robust and Efficient Cartesian Mesh Generation for Component-Based Geometry," U.S. Air Force Wright Laboratory / NASA Ames, CA.

[4] Lo, S. H., and W. X. Wang, "Finite element mesh generation over intersecting curved surfaces by tracing of neighbors," *Finite Element in Analysis and Design*, 41 (2005) pp. 351-370.

[5] Gellert, W., Gottwald, S., Hellwich, M., Kästner, H, and H. Knstner (Eds), *VNR Concise Encyclopedia of Mathematics, 2nd ed*. New Yor, Van Nostrand Reinhold, pp. 541-543, 1989.

[6] Karamete, B. Kaan, Garimella, Rao V., and Mark S. Shepard, "Recovery of an arbitrary edge on an existing surface mesh using local mesh modifications," *International Journal for Numerical Methods in Engineering*, 50 (2001), pp. 1389-1409.

[7] Lawson, Charles L., "Properties of n-dimensional triangulations", *CAGD* 3, no. 4 (1986), pp. 231-246.

[8] Edelsbrunner, H. and Ernst Peter Mücke, "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms", University of Illinois at Urbana-Champain, 1990.