INTERSECTION OF LINE SEGMENTS IN A DISCRETE DOMAIN

Marcel Lourenço de Luna
André Kubagawa Sato
Marcos de Sales Guerra Tsuzuki
Rogério Yugo Takimoto
Thiago de Castro Martins
Computational Geometry Laboratory, Escola Politécnica da USP
marcel.luna@gmail.com, andre.kubagawa@gmail.com, mtsuzuki@usp.br, takimotoyugo@gmail.com, thiago@usp.br

Abstract. *The intersection of line segments has already been studied by several authors. Several problems occurs when using floating point representation, in which number comparison is performed using a tolerance value. One of the main problems is the non-transitive property, as it may lead to incorrect results. Thus, fixed precision is preferred because it solves several problems. However, fixed precision adoption has some complications. One of the issues is relates to the intersection determination between two or more line segments. Particularly, in this research, it is shown that current methods to determine intersection between line segments with fixed precision are not commutative; i.e., depending on which order the line segments are processed a different result may be obtained. The line segment partition problem must be solved beforehand. In order to approach these problems, two line segment representations are considered: geometric and set representation. To the best of our knowledge, this is the first work which discusses the non commutative problem of line segment intersection with finite precision.*
Keywords: *line segment intersection, discrete domain, floating point, Bresenham.*

1. INTRODUCTION

Intersection of line segments is one of the most fundamental tasks in computer graphics. Applications include windowing, clipping, hidden-line removal and Boolean operations. Windowing, clipping and hidden-line removal are cases in which high precision is not necessary. On the other hand, 2D Boolean operations require precision and repeatability. Sato *et al.* (2012) proposed an algorithm to solve irregular packing problems which requires millions of Boolean operations to find a solution.

Boolean operations over polygons have the problem of lacking robustness. They face numerical instability and theoretical difficulties when performing geometric computations. These difficulties occur in boundary evaluations involving ill-conditioned geometric intersections (Hoffmann, 1989). There is a great amount of research on robust geometrical representations and computations. In floating point arithmetics, a threshold $\epsilon > 0$ is required to compare two numbers. Hoffmann (1989) presented the incidence asymmetry problem, in which a vertex can be incident to another vertex but not the other way around, and the incidence intransitivity problem.

Hobby (1999) adopted finite precision to achieve robust algorithms for intersecting line segments. Agarwal *et al.* (2002) used CGAL to implement a Minkowski sum algorithm with exact rational numbers, and they reported execution times that range from a few seconds for shapes involving a small amount of concavities and up to twenty minutes for highly irregular shapes. Hu *et al.* (1996) employed interval arithmetics to ensure robustness. Wallner *et al.* (2000) showed that interval arithmetic is not geometric as it does not give exact error bounds. Several researchers used finite precision to implement Boolean operations over polygons (Sato *et al.*, 2013; Martínez *et al.*, 2009; Leonov and Nikitin, 1997).

The majority of line segment intersection implementations use the sweep line algorithm proposed by Bentley and

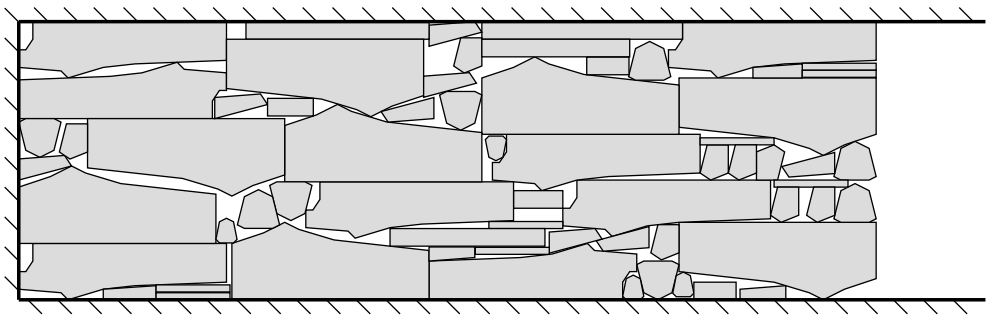

Figure 1. Packing problem example. Irregulars items are arranged inside a rectangular container such that the occupancy rate is maximized.

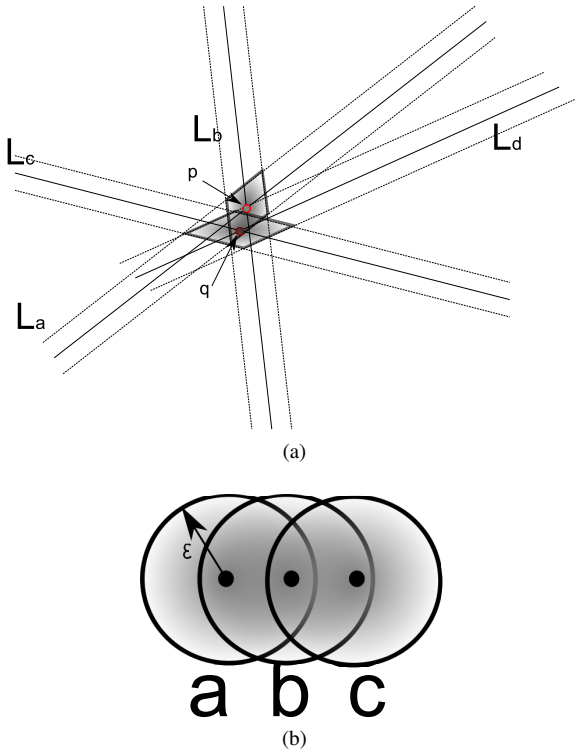

Figure 2. Floating point arithmetics problems. (a) Asymmetric incidence problem. (b) Incidence intransitivity.

Ottmann (1979). The sweep line algorithm determines the intersection points with the aid of an imaginary line sweeping the 2D space from left to right. However, if finite precision is adopted and a different direction is considered, from the right to the left for example, a different result is obtained. This is a consequence of the fact that line segment intersection determination with finite precision is not commutative. This implies that the order in which processes are performed impacts the end result, specially in the case where there are multiple intersections between line segments.

The classic problems of cutting and packing, in which the objective is to find a layout where a set of items is placed in a configuration which minimizes wasted material or unoccupied space (see Fig. 1), are examples of procedures that use methods for line segment intersection determination (Sato *et al.*, 2010, 2011). Due to non transitive property of the operation, layouts obtained for packing problems may change when the order of operations is modified. Consequently, it is possible that the best solution can be only obtained when a specific order of operations is employed. This adds an extra dimension to an already complex problem.

This work investigates the non commutative property of intersection determination operations with finite precision. It is shown that it is not easy to propose a solution and it might be necessary to modify some well assumed basic concepts. The problem is approached using the Bresenham line drawing algorithm, which is responsible for selecting pixels to represent a line segment. To the best of our knowledge, this is the first work which discusses the no commutative problem of line segment intersection with finite precision. This work is structured as follows. Section 2 explains the problems with discrete geometry. The proposed method to determine line segment intersections based in the Bresenham algorithm to draw lines is addressed in section 3. Section 4 discusses problems which are specific to the proposed approach and conclusions are drawn in section 5.

2. DISCRETE GEOMETRIC COMPUTATION PROBLEMS

In this section, problems with floating point computation are briefly explained and the robustness of discrete geometry is discussed. In the following section, the non transitive property of line segment intersection determination with finite precision is presented.

2.1 Why Discrete Geometry?

In B-Rep solid modelers, geometric entities such as point coordinates are considered mathematically ideal. They are represented by floating point numbers and are consequentially processed using floating point arithmetics. However, as demonstrated by Hu *et al.* (1996), data obtained using floating point arithmetics are usually approximated, thus making floating point representation for geometric entities imprecise. This problem is caused by the fact that the precision of

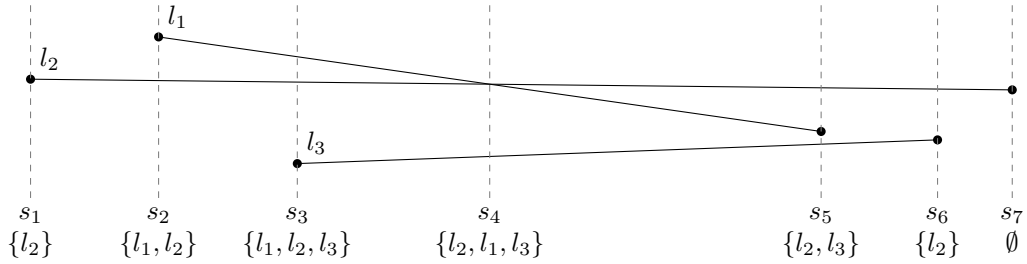

Figure 3. Line segment list example. s_1, s_2 ... s_7 represent the sweep line positioned at each event point. Below the sweep line the active list is displayed. The active list contains the line segments which crosses the sweep line immediately after the event point.

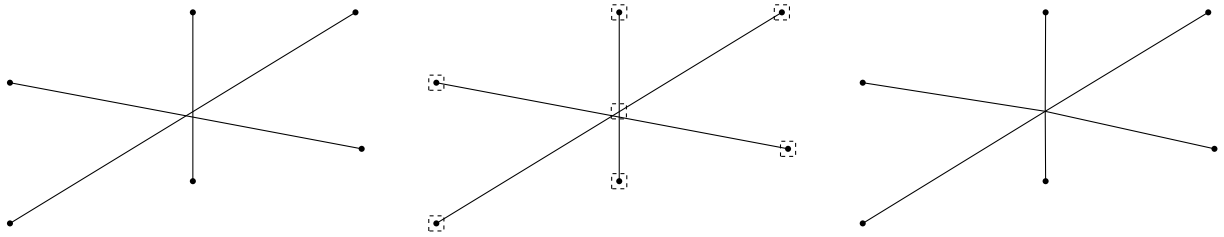

Figure 4. Snap rounding example. (a) Three line segments which intersect in three different positions. (b) Tolerance squares associated with each endpoint and the intersection point. (c) As the tolerance squares for all intersections are the same, a point with the same coordinates is inserted in each line segment.

floating point arithmetics is finite.

Imprecisions may cause the following problems: non-reliability of geometric operations and inconsistencies between the geometry and topology of geometric entities. Consider the problem of asymmetric incidence, shown in Fig. 2a. Four line segments are considered: L_a, L_b, L_c and L_d. Point p is the intersection between L_a and L_b and point q is the crossing point of line segments L_c e L_d. Although point q is incident to point p, point p in not incident to point q. This results from the adoption of a threshold $\epsilon > 0$ to determine if a floating point number equals zero. A point is considered to lay on a line segment if the distance between these entities is less than ϵ. Thus, the intersection can be any point located in the parallelogram shaped region generated by the intersection of the two regions with thickness of 2ϵ centered along each of the line segments L_a e L_b. The same procedure can be applied to line segments L_c and L_d. Point q is internal to both intersection regions of L_a–L_b and L_c–L_d, whereas point q is not internal to the intersection region of L_c and L_d.

Incidence intransitivity is another problem encountered when using floating point arithmetics. Fig 2b display three points, a, b and c, where $a = b$ as $\|a - b\| < \epsilon$, $b = c$ as $\|b - c\| < \epsilon$, but $a \neq c$, as $\|a - c\| > \epsilon$. Finite precision arithmetic can solve these and other problems. Nevertheless, as it will be clear in the following sections, additional problems may appear.

2.2 Sweep Line Algorithm

The geometric algorithm considered in this work is the sweep line algorithm. Originally proposed by Bentley and Ottmann (1979), its main feature is the reduction of line segment pairs candidates for the intersection determination. Hobby (1999) proposed an implementation of the sweep algorithm with fixed precision, which aims to avoid extraneous intersections by the use of a snap rounding technique.

In the sweep line algorithm, an imaginary vertical line moves from the leftmost to the rightmost vertex, stopping at event points in order to determine new intersections. When the sweep line reaches its leftmost position, all intersections are determined and the algorithm is finished. There are three different types of event points: left endpoint, right endpoint and crossing point. The endpoint events can be determined in a first step and the crossing events are determined as the line sweeps the space, i. e. moves from left to right.

The line segments which intersects the sweep line are stored in the active list. This list is sorted by the vertical coordinate of the intersection point of each line segment with the sweep line. If the list does not change when the sweep line moves in one direction, then there is no line segment intersection in the region traversed by the sweep line. Changes in the active list occurs only at event points. Thus, checking for intersection is only needed at event points. Moreover, a line segment can only intersect with its immediate superior or immediate inferior in the active list. Performing only these

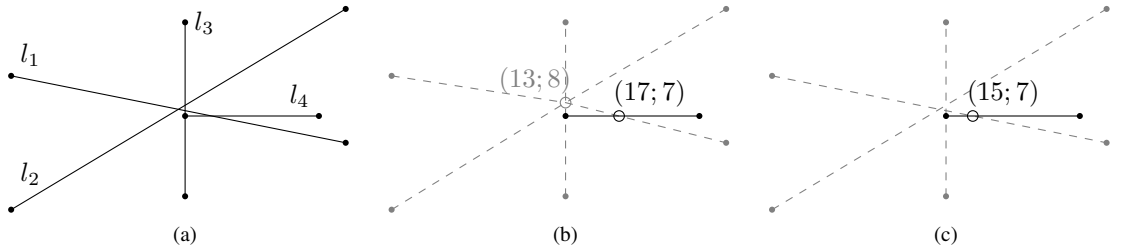

Figure 5. Example where the commutative property is not satisfied. (a) Coordinates for the endpoints are: $l_1 : (0; 10)$ and $(25; 5)$, $l_2 : (0; 0)$ and $(25; 15)$, $l_3 : (13; 14)$ and $(13; 1)$, $l_4 : (13; 7)$ and $(23; 7)$. (b) Result of the intersection of l_4 with the result of the intersection processing of l_1, l_2 and l_3. (c) Intersection of line segments l_1 and l_4.

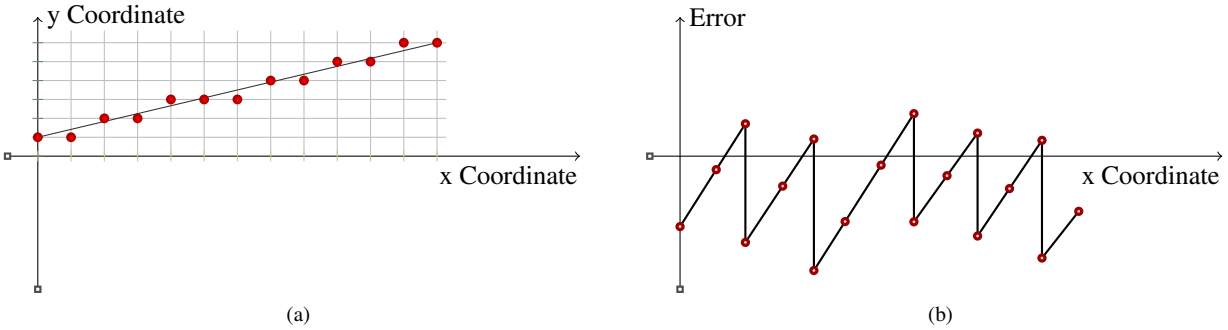

Figure 6. Example of the application of the Bresenham algorithm for line segment representation. (a) Line segment represented by Bresenham algorithm; (b) correspondent error representation.

checks reduces the complexity of the algorithm when compared with algorithms which checks all possible line segment pairs. Fig. 3 shows the active list corresponding to each event point.

Although he Bentley–Ottmann algorithm can be implemented directly using exact real geometry, the memory needed to store the exact coordinates may become too large as more operations are executed, thus a practical alternative must be adopted. One solution for this problem is to use finite precision. However, rounding the intersection coordinates to integer values without careful inspection can lead to extraneous intersections. The practical intersection algorithm proposed by Hobby (1999) solves this problem with snap rounding. At each intersection, the intersection tolerance square, a unitary square centered on a grid point, is determined and the point is inserted to the appropriated line segments. Before the snap rounding, all tolerance squares are created using the intersections determined by the Bentley–Ottmann algorithm. A second step of the algorithm is then performed, consisting of determining each line segment which crosses the tolerance squares and inserting a point in such line segments. Fig. 4 shows an example of the snap rounding strategy, which avoids extraneous intersection by bending the line segments which intersects the tolerance square. Using this algorithm, extraneous intersections are avoided (Hobby, 1999).

2.3 Non Commutative Property

Fig. 5 shows an example in which the commutative property is not satisfied. If line segments l_1, l_2 and l_3 are processed first, line segments l_1 and l_2 are bended. When intersecting the result with l_4, the crossing point is $(17; 7)$, as can be seen in Fig. 5b. If the input of the first operation are l_1 and l_4, then the intersection can be determined exactly at $(15; 7)$ (see Fig. 5c). As the crossing is exact, line segment l_1 remains a straight line segment and the intersections with l_2 and l_3 do not change. The intersection between l_1 and l_4 can result in different points depending on the evaluation order.

3. BRESENHAM ALGORITHM

The Bresenham algorithm can shed light on the non commutative property discussion. It was proposed by Bresenham (1965) and it is used as an algorithm for line segments representation in discrete domain, with fixed precision. As it determines coordinates for each point in the line segment, it was originally employed to determine which pixels of a screen should be set in order to create a visual representation of the line segment. From the initial point, one of the endpoints of the line segment, a main direction is defined and, at each iteration, the coordinate of the point is incremented in this direction. The perpendicular direction is determined by the distance from the real line segment. This distance is defined as the error of each chosen point and the next point is chosen so as to minimize the error.

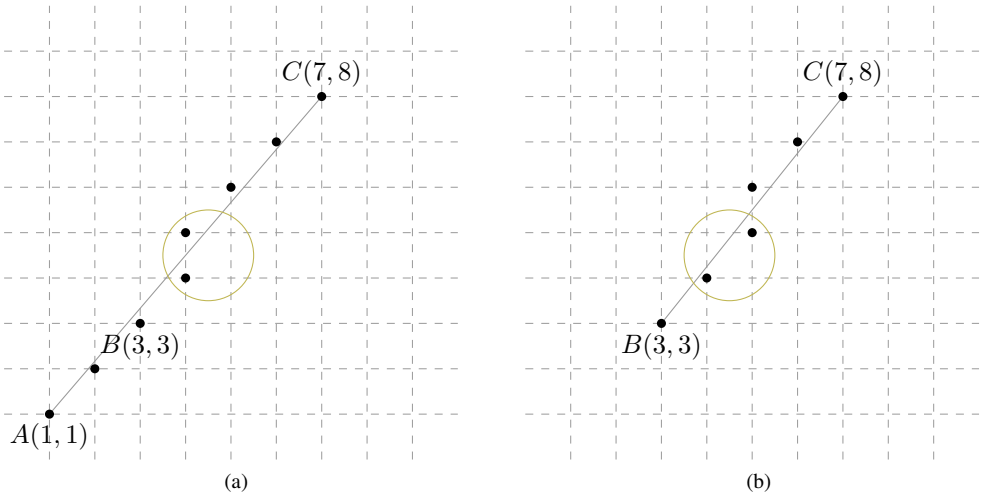

Figure 7. Bending problem using Bresenham algorithm for line segment representation. (a) Original line segment. (b) Partitioned line segment.

Algorithm BRESENHAMLINE is an implementation of the Bresenham algorithm for line segment representation, described in Algorithm 1. The described algorithm can only be applied to a line segment whose slope is between $0°$ and $45°$. Input are the endpoints P_1 and P_2, such that P_1 is the left endpoint and P_2 is the right endpoint and the output is a list of points. Variable err stores the approximation error at each point. The algorithm can be generalized for any slope by decrementing y instead of incrementing, when $\Delta y < 0$, and swapping the x and y dimensions when $\Delta x < |\Delta y|$. Fig. 6 shows an the result and errors for an example case.

Algorithm 1: BRESENHAMLINE(x_1, y_1, x_2, y_2)

$\Delta x \leftarrow x_2 - x_1$;
$\Delta y \leftarrow y_2 - y_1$;
$x \leftarrow x_1$;
$y \leftarrow y_1$;
$E_{err} \leftarrow 2\Delta y - \Delta x$;
while $x \leq x_2$ **do**
  <Store point (x, y)>;
  if $E_{err} > 0$ **then**
    $y \leftarrow y + 1$;
    $E_{err} \leftarrow E_{err} - 2\Delta x$;
  $x \leftarrow x + 1$;
  $E_{err} \leftarrow E_{err} + 2\Delta y$;
<Store point (x, y)>;

4. DISCUSSION

Two different representations for line segment are discussed. Both representations can be derived from the Bresenham algorithm. The Bresenham algorithm has as input two points in the 2D space and as output, the algorithm determines a set of points associated with the line segment. Then, a line segment can be represented by its endpoints (called geometric representation). And, the line segment can be represented by a set of points (called set representation). The geometric representation needs two points, thus memory requirement is low. For the set representation, on the other hand, the amount of memory necessary to store the line segment depends on its length.

Two problems are discussed in this section: line segment partition and line segment intersection. Both problems are correlated, as when an intersection is present, the intersecting line segments are partitioned.

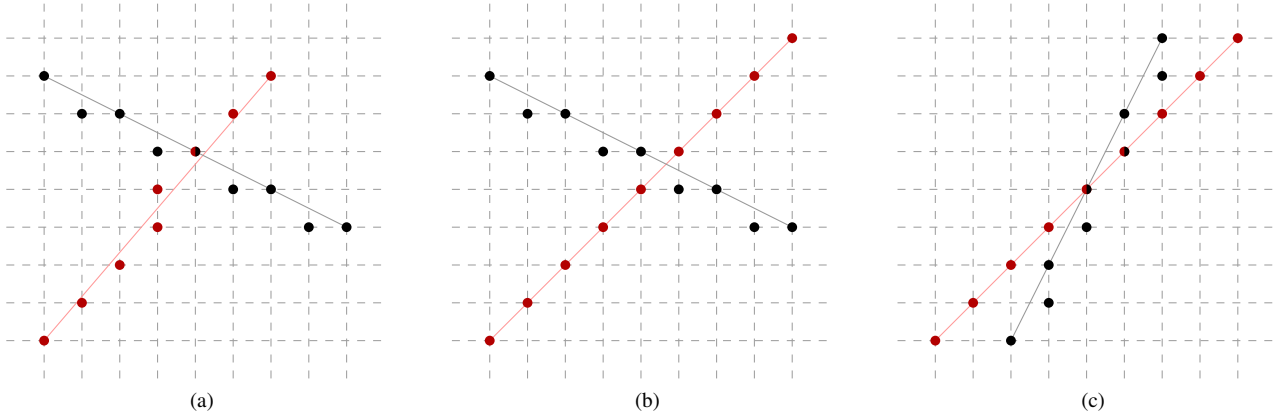

Figure 8. Bresenham generated lines intersection. (a) One point in common; (b) no common point between line segments; (b) multiple common points.

4.1 Line Segment Partitioning

Line segment subdivision occurs when a point is inserted into a line segment. When geometric algorithms for intersection detection are adopted, line segments are represented by their endpoints. Thus, node insertion in the discrete domain usually causes the line segment to bend as nodes are always inserted in grid coordinates. When line segments bend, the union of the partitioned line segments is not equal to the original line segment. If special techniques are not employed, bending may lead to extraneous intersections and non commutativity of the operation. Bresenham based line segment division also has the same problem if the partitioned line segments are reprocessed, as shown in Fig. 7.

The original line segment is from $A(1,1)$ to $C(7,8)$. Consider that the original line segment is divided at $B(3,3)$. The partitioned line segment BC has different points when compared to the original line segment. This way, the union of the set of points representing line segments AC and BC is different with the set of points representing AC. It should be better if the set of points is kept the same and defined by the original line segment.

The line segment partitioning problem can be stated as: the original line segment and both partitioned line segments must have the same set of points. Considering the set representation, the solution is easy. Just separate the point set in two appropriate sets. The geometric representation can have a solution if additional information is added to the Bresenham algorithm. For example, some variables that store the state of the Bresenham algorithm: E_{err} the initial error, Δx and Δy the slope of the line segment. The error associated with each point is used to generate the original vector of points.

Algorithm 2 describes the algorithm MODIFIEDBRESENHAMLINE. If the line segment is original, i.e. none of the endpoints were created by the intersection algorithm, the algorithm is equal to the algorithm BRESENHAMLINE. In this case, $flag$ is true. If the line segment is a partitioned line segment, then the algorithm state is retrieved from the input variables, replicating the original line segment.

Algorithm 2: MODIFIEDBRESENHAMLINE$(x_1, y_1, x_2, y_2, E_{err}, \Delta x, \Delta y, flag)$

if <*flag*> **then**
  $\Delta x \leftarrow x_2 - x_1$;
  $\Delta y \leftarrow y_2 - y_1$;
  $E_{err} \leftarrow 2\Delta y - \Delta x$;
$x \leftarrow x_1$;
$y \leftarrow y_1$;
while $x \leq x_2$ **do**
  <Store point (x, y) >;
  if $E_{err} > 0$ **then**
    $y \leftarrow y + 1$;
    $E_{err} \leftarrow E_{err} - 2\Delta x$;
  $E_{err} \leftarrow E_{err} + 2\Delta y$;
  $x \leftarrow x + 1$;
<Store point (x, y) >;

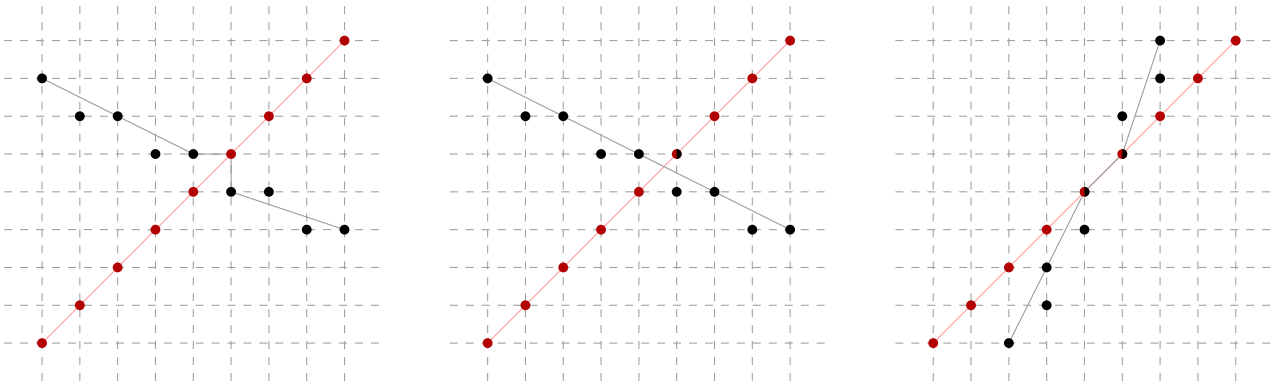

Figure 9. Dealing with intersection with no common point or more than one common points. (a) No common points, multiple division approach. (b) No common points, point addition proposal. (c) Multiple common points, subdivision approach.

4.2 Bresenham Based Line Intersection

Considering the proposed solution for the line segment partition problem, intersection determination can be performed. Using the Bresenham algorithm, line segments can be represented by a set of points, one for each coordinate in the main direction, in the interval between endpoints. Intersection can be determined by the intersection between sets. If two line segments cross, the intersection is a point. However, when using the proposed representation, three cases exist for crossing line segments:

1. the intersection is a single point, which can be considered the approximate intersection in the discrete domain (see Fig. 8a).
2. the intersection is empty (see Fig. 8b);
3. the intersection is a set of points, which represents a line segment (see Fig. 8c).

Fig 8 shows examples for each possibility. When determining intersection, case 1 is the ideal situation, as there is only one possibility for the crossing point. Both line segments can be partitioned using the algorithm 2, without any modification. The other two cases are more complex and needs further investigation.

In case 2, two line segments intersect, nevertheless their set representations shows no common point. Fig. 8b exhibits a typical example. The intersection occurs inside one grid square and points from both line segments occupy all four corners of such square. The approximated crossing point can be selected by finding the closest grid point, same as the geometric algorithm. In the case of Fig. 8b, the top right corner of the rectangle is selected. A point of the positive sloped line segment is located in this corner, so no changes are performed for such line segment. For the negative sloped line segment, two solutions are proposed. The first is to divide the line segment into four partitioned line segments, as shown in Fig. 9a. Bending only occurs on the square in which the line segments cross, limiting the impact. A second option is to simply add the intersection point to the line segment, the top left corner point in the example shown in Fig. 8b. In this case, more than one point exists on the same vertical, which is usually not considered when dealing with Bresenham generated algorithms (see Fig. 9b). In both cases, the set of points do not change.

The intersection point is also determined by the closest grid point when multiple common points exist. Consider the situation shown in Fig. 8c where the set intersection consists of two points. The intersection lies exactly on grid point $(5, 5)$. The simplest approach is to ignore the other common points and adopt $(5, 5)$ as the intersection point. Second option consists of partitioning the line segment. The result consists of three line segments, the first starts at the initial point and ends at the first common point. The following line segment ends at the last common point and the third and last line segment ends in the final endpoint. Fig. 9c shows this approach.

5. CONCLUSIONS

In this work, the non commutative property of line segment intersection with finite precision is discussed. It is shown that a more basic property should be the preservation of the point sets when a line segment is partitioned. The information of the original line segment must be kept. Two possibilities for line segment representation are considered: geometric and set representations. Both representations satisfies the commutative property if the point sets are preserved. Three possible line segment intersection situations were considered: single point, empty and multi-point. For empty intersection, some

micro line segments can be included to correctly represent the intersection. Another possibility is to modify the line segment representation to include some additional points. This work is a first attempt to understand and find the solutions to the problem of obtaining robust geometrical processing using finite precision.

6. ACKNOWLEDGEMENTS

Marcel Lourenço de Luna, André Kubagawa Sato and Rogério Yugo Takimoto are supported by FAPESP (grants 2012/19196-0, 2010/19646-0 and 2011/22402–8), Marcos de Sales Guerra Tsuzuki and Thiago de Castro Martins were partially supported by CNPq (grants 309570/2010-7 and 306415/2012-7). This work was supported by FAPESP (grant 2010/18913-4).

7. REFERENCES

Agarwal, P.K., Flato, E. and Halperin, D., 2002. "Polygon decomposition for efficient construction of minkowski sums". *Computational Geometry*, Vol. 21, pp. 39–61.

Bentley, J.L. and Ottmann, T.A., 1979. "Algorithms for reporting and counting geometric intersections". *IEEE Transactions on Computers*, Vol. C-28, pp. 643–647.

Bresenham, J., 1965. "Algorithm for computer control of a digital plotter". *IBM Systems Journal*, Vol. 4, No. 1, pp. 25–30.

Hobby, J.D., 1999. "Practical segment intersection with finite precision output". *Computational Geometry*, Vol. 13, pp. 199–214.

Hoffmann, C.M., 1989. "The problems of accuracy and robustness in geometric computation". *Computer*, Vol. 22, pp. 31–41.

Hu, C.Y., Patrikalakis, N.M. and Ye, X., 1996. "Robust interval solid modelling part I: representations". *Computer Aided Design*, Vol. 10, pp. 807–817.

Leonov, M.V. and Nikitin, A.G., 1997. "An efficient algorithm for a closed set of Boolean operations on polygonal regions in the plane". Technical report, A. P. Ershov Institute of Informatics Systems.

Martínez, F., Rueda, A.J. and Feito, F.R., 2009. "A new algorithm for computing Boolean operations on polygons". *Computers & Geosciences*, Vol. 35, pp. 1177–1185.

Sato, A.K., Martins, T.C. and Tsuzuki, M.S.G., 2010. "Rotational placement using simulated annealing and collision free region". In *Proceedings of the 10^{th} IFAC Workshop on Intelligent Manufacturing Systems*. pp. 253–258.

Sato, A.K., Martins, T.C. and Tsuzuki, M.S.G., 2011. "A simulated annealing based algorithm with collision free region for the irregular shape packing problem". In *Proceedings of the 18^{th} IFAC World Congress*. pp. 3968–3973.

Sato, A.K., Martins, T.C. and Tsuzuki, M.S.G., 2012. "An algorithm for the strip packing problem using collision free region and exact fitting placement". *Computer Aided Design*, Vol. 44, pp. 766–777.

Sato, A.K., Martins, T.C. and Tsuzuki, M.S.G., 2013. "Collision free region determination by modified polygonal boolean operations". *Computer-Aided Design*, Vol. 45, No. 7, pp. 1029 – 1041.

Wallner, J., Krasauskas, R. and Pottmann, H., 2000. "Error propagation in geometric constructions". *Computer Aided Design*, Vol. 32, pp. 631–641.

8. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.