Algorithms for Computing Intersection and Union of Toleranced Polygons with Applications

F. Cazals, G.D. Ramkumar *

Robotics Laboratory, Department of Computer Science, Stanford University,
Stanford, CA 94396 USA

E-MAIL: {cazals,ramkumar}@flamingo.stanford.edu

Abstract

Since mechanical operations are performed only up to a certain precision, the geometry of parts involved in real life products is never known precisely. Nevertheless, operations on toleranced objects have not been studied extensively. In this paper, we initiate a study of the analysis of the union and intersection of toleranced simple polygon. We provide a practical and efficient algorithm that stores in an implicit data structure the information necessary to answer a request for specific values of the tolerances without performing a computation from scratch. If the polygons are of sizes m and n, and s is the number of intersections between edges occuring for all the combinations of tolerance values, the pre-processed data structure takes $O(s)$ space and the algorithm that computes a union/intersection from it takes $O((n+m)\log s + k^{'} + k \log k)$ time where k is the number of vertices of the union/intersection and $k \leq k^{'} \leq s$. Although the algorithm is not output sensitive, we show that the expectations of k and $k^{'}$ remain within a constant factor τ, a function of the input geometry. Finally, we list interesting applications of the algorithms related to feasibility of assembly and assembly sequencing of real assemblies.

1 Introduction

In the life cycle of a part in a manufactured product, say an engine, a piece of furniture, or a toy, etc., its exact geometry can never be described exactly. In fact, mechanical operations are performed only up to as much precision as to ensure that a feature of the part (for example, a vertex) lies within a zone called its *tolerance* zone. Models for toleracing and their role in product design have received substantial interest in the literature [16, 17, 15, 18, 11, 6]. However, their geometric properties have not been looked at extensively. Intersection volumes and surface areas for cylinders with tolerances are described in [5]. Planning the assembly of toleranced products is addressed in [9]. But basic operations such as intersection, union, convolution, and Minkowski sum of toleranced polygons have not yet been studied. The basic operations among these are the union and the intersection. We adress their computation in this paper: given two simple polygons whose geometry (but not topology) can vary, pre-process the set of all possible intersections between edges of the two polygons so that for a particular set of tolerance values their intersection or union can be computed efficiently.

*F. Cazals is supported by Matra Datavision, G. D. Ramkumar is supported by NSF. Part of this work has also been supported by NSF/ARPA grant IRI-9306544.

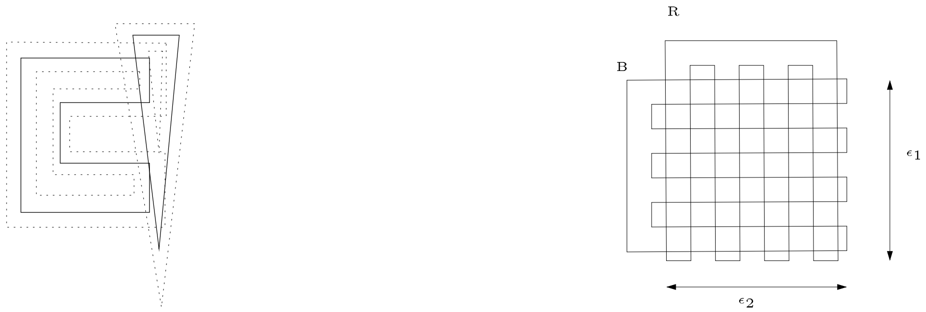

Figure 1: a)Intersection of two toleranced polygons and (b)Worst-case example of intersection

Figure 1(a) shows an example of two toleranced polygons intersecting in a configuration that may produce zero, one, or two components in the intersection depending on the exact tolerance values. Figure 1(b) shows such an example of two intersecting comb-like polygons R, and B — one vertical and one horizontal with tolerances ϵ_1 and ϵ_2 on their horizontal and vertical edges respectively. In this example, the number of components in the intersection can vary from 0 to $\Theta(m * n)$. In addition, the components are determined by different tolerance values, causing an exponential number of topologies. Hence it is infeasible to represent the topologies explicitly.

The input consists of two simple polygons, referred to as a red polygon of n edges and a blue polygon of m edges. Each edge has an associated tolerance parameter t defined with respect to a reference frame attached to its polygon. The relative position of the two polygons is determined by those of their respective frames. The configuration space $\mathcal{T}$ has dimension $n + m$ and an instantiation of the two polygons is fully specified by a set of tolerance values $T = \{t_1, t_2, \ldots, t_{n+m}\}$. Let S refer to the set of intersections between pairs of edges occurring for all combinations of tolerance values and by S_T those valid only for the input T.

Our results are three-fold. Firstly, we pre-process the toleranced polygons to enable efficient computation of the union or intersection for a query instance of tolerance values. The results of pre-processing are stored in a segment-tree like data structure. An instance of the intersection or union is computed in time $O((n + m) \log |S| + k' + k \log k)$ where k is the number of vertices of the union or intersection and $k \leq k' \leq |S|$. The term k' causes the complexity to be not output sensitive; however, this aspect seems necessary because the space of intersection of two toleranced edges only admits a complicated definition, namely as as a semi-algebraic set in dimension six defined by four inequalities involving polynomials of degree two. But we show that the amortized running time of our algorithm is output-sensitive by proving that the expectations of k and k' remain within a constant factor τ, a function of the input geometry. Secondly, we use our data structure to report the set of all possible "local" topologies of the union and intersection of the polygons. Thirdly, we present several straightforward applications of our algorithms to feasibility of assembly and assembly sequencing.

There is substantial literature related to computing union and intersection of polygons without tolerances. The work is motivated by theoretical as well as practical interest. The intersection algorithm of [2] can be used to compute the trapezoidal decomposition of an arrangement of edges. This provides a union/intersection algorithm of simple polygons that runs in $O((m + n) \log(m + n) + k)$. Our method is simpler and thus easier to implement and also is more efficient if $|S| = o(n + m)$, that is the tolerance spaces do not admit a super-linear number of intersections; we believe this is the case in practical situations. Other previous work in the area of union/intersection computation is an integration of the linear time triangulation algorithm of [1] with the linear time map merging of [7], giving a space-time optimal solution.

This paper is organized as follows. In section 1, we introduce the toleranced model and in section 2 we study the intersection of toleranced edges, the basic primitive. Section 3 presents the algorithm that outputs S_T from S; section 4 shows how to perform the intersection/union operation of the set S_T. In section 5 we discuss the stability of intersection features. Section 6 considers three applications of the toleranced intersection and section 7 list some oustanding problems and concludes the paper.

2 Tolerance model

2.1 Toleranced polygon

By toleranced polygon, we mean a simple polygon with an attached coordinate system $(p, \vec{i}, \vec{j})$ and each edge of which is defined by a triple $(\theta, [a, b])$ as follows: the normal to the line supporting this edge makes an angle $\theta \in [0, 2\pi)$ with the frame attached to the reference point of the polygon; the distance between the edge and the polygon reference point is allowed to span the interval $[a, b]$ where $a, b > 0$. The model assumes that the parameter θ is constant; in more general models θ can have a tolerance range as well. An edge is therefore characterized in $(p, \vec{i}, \vec{j})$ by the equation $x \cos\theta + y \sin\theta - d = 0$ with $d \in [a, b]$.

For a given polygon, these tolerance zones should be small enough so that all instances of the polygon have the same topology. A sufficient condition is that no vertex falls into the intersection of more than two stripes spanned by the edges.

Figure 2: (a)Toleranced polygon (b)Edge domain

In this model, the domain spanned by each edge is a trapezoid, as indicated in grey on figure 2(b). In fact, the lines supporting the two edges that have extremal values determine the parallel sides of the trapezoid, and the two other sides correspond to the maximal values of the two connected edges. More precisely, this trapezoid can be subdivided into three regions as indicated on figure 2(b):
-the $C-domain$: whatever value $d_i \in [a_i, b_i]$ may have, the i^{th} edge crosses this domain all the way (C stands for compulsory).
-the $O-domains$ O_l and O_r : a point lying on the i^{th} edge might not be attained because of the tolerances values of the adjacent edges (O stands for optional). It is for example the case of the point P in O_r on figure 2(b). This trapezoid is called the *toleranced edge* in the rest of this paper. It is important to observe that it depends on three variational parameters: the *main* parameter d_i, and the two *connected* parameters d_{i-1} and d_{i+1}. The tolerance model we use was introduced in [9].

2.2 Intersection of toleranced edges

2.2.1 Problem statement

In dealing with geometric operations between toleranced polygons, a crucial step consists of intersecting toleranced edges. Given two toleranced edges, the status of their intersection is one of the following: (i)they never intersect (ii)they always intersect (iii)they intersect conditionally to the variational parameters involved. Edges in cases (ii) and (iii) are referred to as *compulsory* and *optional* edges respectively.

As already observed, each toleranced edge depends on three parameters. The intersection between edges i and j therefore depends on the six parameters of indices $\{i-1, i, i+1, j-1, j, j+1\}$. We note ν_{ij} the corresponding subspace of $\mathcal{T}$. If an intersection exists, it belongs to the polygon I_{ij} intersection of the two toleranced edges. This polygon is shown in grey on figures 3(a) and 5. Compulsory intersections require that I_{ij} lies within the intersection of the $C-domain$ of the toleranced edges and that any of its section along the direction of a main parameter spans the whole domain of this parameter. This is the case on figure 3(a) for edges L_2 and M_2. On the other hand, finding a simple description of optional intersection is more laborious. Consider figure 3(b) where the three edges L_2, L_3 and M_2 have non null tolerances zones. The three dashed lines correspond to instances of these edges for the tolerance values x, y and z. In this configuration, edges L_2 and M_2 intersects at point I. But in addition to x and z, I also depends on y. Indeed for any value of y smaller than the one displayed, M_2 intersects L_3 but not L_2 anymore. It turns out that the condition under which the intersection occurs can no more be found easily in the primal space, which accounts for an algebraic computation of ν_{ij}.

Figure 3: Edges intersections

2.2.2 Algebraic specification of toleranced intersection vertex

Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be the two edges. They intersect iff the line supporting e_1 divides the edge e_2 and viceversa (see figure 4). Let $\mathrm{Det}(u, v, w)$ represent the signed area of the triangle uvw defined by vertices $u = (u_x, u_y), v = (v_x, v_y)$, and $w = (w_x, w_y)$:

$$\begin{vmatrix} 1 & 1 & 1 \\ u_x & v_x & w_x \\ u_y & v_y & w_y \end{vmatrix}$$

The condition $\mathrm{Det}(u, v, w) > 0$ requires that u, v, w are in counterclockwise order on the plane. The line supporting edge e_1 divides the edge e_2 iff u_2 and v_2 are on the opposite sides of the line supporting e_1, so that the conditions sought are

$$\mathrm{Det}(u_2, u_1, v_1)\,\mathrm{Det}(v_2, u_1, v_1) \;<\; 0 \text{ and } \mathrm{Det}(u_1, u_2, v_2)\,\mathrm{Det}(v_1, u_2, v_2) \;<\; 0 \quad (\mathcal{I})$$

Each of the vertices u_1, v_1, u_2, v_2 is a linear function of the tolerance parameters of the two edges intersecting to produce it. The set of constraints defines a volume bounded by quadratic surfaces in the space of 6 tolerance values. Since the edges preserve their orientation under variation of tolerance parameters, requiring the two edges to intersect is tantamount to fixing the sign of each determinant (a stronger condition than above).

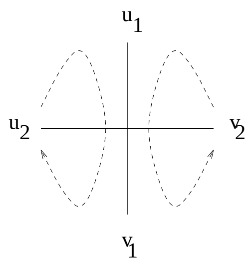

Figure 4: Intersection conditions

To illustrate the previous computation, we implemented condition $\mathcal{I}$ under Maple to run some experiments. For example, the configuration of figure 3, is defined precisely as follows

$$\begin{cases} L_1 := (Pi, a1 \in [5., 5.]);\ L_2 := (Pi/2, a2 \in [5., 11.]);\ L_3 := (Pi/4, a3 \in [4.5 * sqrt(2), 7.5 * sqrt(2)]); \\ M_1 := (Pi/2, b1 \in [3., 3.]);\ M_2 := (0, b2 \in [1., 6.]);\ M_3 := (Pi/2, b3 \in [14., 14.]); \end{cases}$$

leads to the two conditions

$$(b1 - a2)\left(-a2 + \sqrt{2}a3 + a1\right)^2(b3 - a2) < 0 \text{ and } \left(a2 - \sqrt{2}a3 + b2\right)(-b3 + b1)^2(b2 + a1) < 0$$

As this example shows, this space is not linear and actually not even convex. Indeed, it is easily checked that the two following points $P_1 = [5.;\ 6.;\ 8, 48.;\ 3.;\ 6.;\ 14.]$ and $P_2 = [5.;\ 7.;\ 9.19;\ 3.;\ 6.;\ 14.]$ belong to ν_{ij} but not their middle. Computing the volume p_{ij} of ν_{ij} cannot be performed using simple techniques as those of [10] for convex polytopes and this issue will be adressed in section 5.1 in dealing with the stability of the intersection components. However, storing ν_{ij} in a data structure to efficiently select the relevant intersections with respect to a particular input of tolerances remains an open question.

2.2.3 Approximate representation

For these reasons, we address the problem of finding an approximate representation of ν_{ij} as an hyper-rectangle ν'_{ij} of volume p'_{ij} such that all the points located in this hyper-rectangle are candidates with a high probability to be a true intersection between the two edges.

As intersection of two four sided polygons each depending on at least three parameters, I_{ij} is at most eight-sided and depends on at most six parameters. Moreover it is convex since the toleranced edges are convex. If k is one of the six parameters involved in the intersection of edges i and j, then let $stripe_k$ be the minimal closed stripe of the plane with sides parallel to the k^{th} parameter direction and containing I_{ij}. The two lines defining a stripe correspond to two values in the variational space of the corresponding parameter, possibly out of the range $[a_k, b_k]$ so that it is possible to reduce any stripe to its intersection with the tolerance zone of its parameter. Let ds_k the bounds of the reduced stripe in the variational zone of parameter k.It is also possible that $ds_k = [a_k, b_k]$ in which case the corresponding parameter is called *useless* since an intersection can occur for any of its value. If the opposite holds, it is called $useful$.

An obvious necessary condition on the *main* parameters for an intersection to exist is that $d_i \in ds_i$ and $d_j \in ds_j$. See e.g. figures 5(a)(b). For the *optional* parameters, the situation is more involved but we can also come up with a condition of the same type.

In summary, each of the six parameters involved in the intersection of edges i and j produces a range in its tolerance zone. Such a range might be useless if it spans the whole tolerance zone, and in this case we discard it for a connected parameter. This leads to a representation of I_{ij} as a hyper-rectangle ν'_{ij} of dimension at least two and at most six that partitions ν_{ij} in two: the set of points that fulfil a necessary condition for the edges to intersect, and its complement.

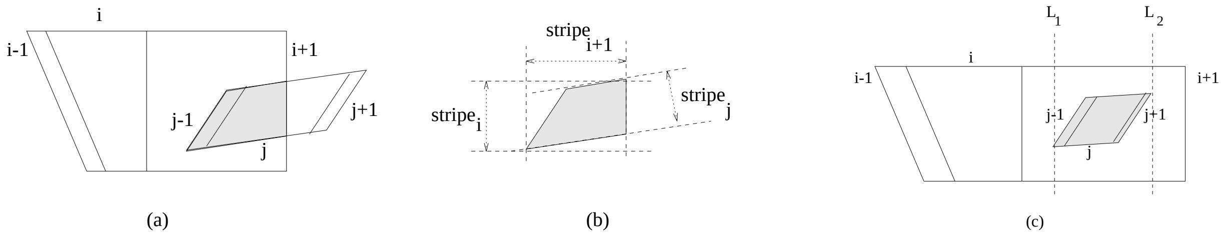

Figure 5: Intersection and stripes

2.3 Computing the intersection of toleranced edges

We use a traditional Bentley-Ottman style algorithm to compute the intersection of toleranced edges. It runs in $O((m + n)\log(m + n) + s)$ time.

3 Selection of relevant intersections

3.1 Selecting a subset of events

Let $T = \{t_1, \dots, t_{n+m}\}$ be an instantiation of the two input polygons. Before we can get the topology of the intersection/union, we must select the subset E_T of intersections that are valid for the particular input T. The problem we have to face here is that each I_{ij} depends on at most six parameters while the whole space has dimension $m+n$. Devising an efficient data structure to retrieve these small dimensional sub-spaces is a challenging problem. An easy way to get around it is therefore to allow the selection process to operate in two steps as follows: first select $E_{AT} \subset E$ and then derive E_T from E_{AT}.

This strategy allows us to represent I_{ij} using the hyper-rectangle ν'_{ij} described above. For a given t_i, finding all the ν'_{ij} containing it reduces to a point-in-segment enclosure test. A suitable data structure to perform this is a segment tree for each parameter leading to a forest of segment trees for the $n+m$ parameters.

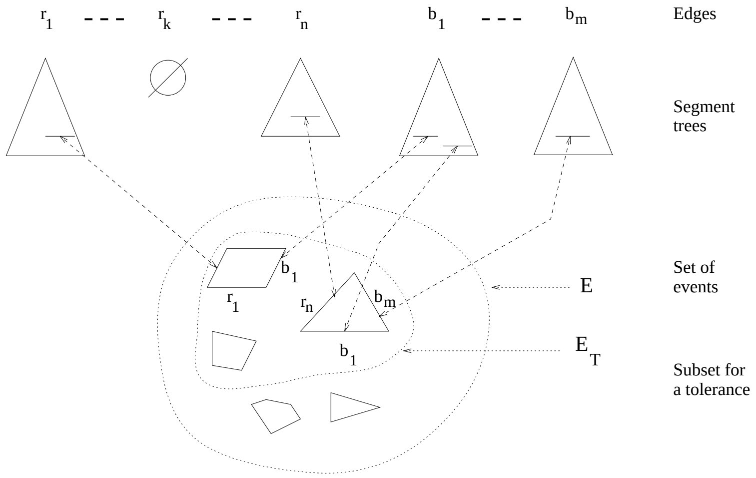

Figure 6: Selecting active edges

More precisely, each *main* parameter k involved in the description of any ν'_{ij} is stored as follows:

- If it is useful, it is stored in its segment tree ST_k. In addition, each such range is given a pointer to the ν'_{ij} it comes from.
- If it is useless but is involved in a compulsory intersection, it is stored in the linked list CL_k with a pointer to the ν'_{ij} it comes from.
- If it is useless and is involved in an optional intersection, it is stored in the linked list OL_k with a pointer to the ν'_{ij} it comes from.

An example configuration of the forest of segment trees is shown in figure 6. Observe that we have not stored the connected parameters, but they are accessed through the pointer from the main parameters to the ν'_{ij} they belong to. At last, we require that within each list or node the pointers are sorted according to the labels of the edges of the second polygon. This constraint will be used in section 3.2.

We now describe below the selection process that leads from E to E_T.

<u>Selection algorithm</u>

for $i := 1$ **to** $n + m$ **do**

(1)Add to E_T the intersections referenced by CL_i still not accessed

(2)Add to E_T the *relevant* intersections referenced by OL_i still not checked

(3)Add to E_T the intersections referenced by the segments of ST_i containing t_i , not accessed and relevant

(4)Merge all the valid intersections according to the labels of the edges of the second polygon

od

First observe that any ν'_{ij} involves two main ranges stored either in a list or a segment tree. Each intersection or hyper-rectangle can thus be accessed by either of the two pointers. We assume that the first access sets a flag.
Now, there are two kinds of intersections: compulsory and the optional, of which the latter have to be checked. If more than two parameters are involved in the description of its ν'_{ij}, the first step is to check that the connected parameters fulfill the required conditions. If so, the intersection has to be computed and a check has to be performed to see if it belongs to the two line segments supported by the main parameters. These tests should be applied in this order because of their respective computational costs.

3.2 Analysis

Let s_i be the cumulated size of the i^{th} segment tree and the associated lists. Selecting and reporting all the segments containing t_i in these structures costs $O(\log s_i + k'_i)$ with k'_i the output size. We handle separately the analysis of these two terms when summed over the forest.

The first term sums to $\sum_{i=1}^{n+m} O(\log s_i)$ which is easily seen to be smaller than $(n+m)\log s$ with $s = \sum_{i=1}^{n+m} s_i$. The second sums to $k_i \le k'_i$ relevant intersections by tree and lists. But remember that for a given index we require the ranges to be sorted with respect to the label of the second polygon. Let l_i be the number of different lists that the k_i ranges come from. A heap can be used for k-way merging of the l_i lists in $O(k_i \log l_i)$. The total merging time is thus $\sum_{i=1}^{n+m} k_i \log l_i < k \log k$, with $k = \sum k_i$.

An important task is to compare the relative values of k and k'. Since our data structure is intended to process several queries, it would be nice to have an amortization phenomenon over these queries. Let r be the number of queries. For any query i in $1..r$ and any intersection j in $1..s$, let ε'_{ij} be the random variable defined as 1 if the intersection j is selected at stage i through the ST data structure and 0 otherwise. Also, let ε_{ij} be a random variable defined as 1 if the previous intersection is not discarded and 0 otherwise. A measure of the acceptance rate over the queries is

$$\tau = \frac{E(\sum_{i,j} \varepsilon_{ij})}{E(\sum_{i,j} \varepsilon'_{ij})}$$

(where E is the expectation). But for any j, $\sum_i \varepsilon'_{ij}$ and $\sum_i \varepsilon_{ij}$ are distributed as binomial random variables of parameters $B(r, p'_j)$ and $B(r, p_j)$ respectively, so that

$$\tau = \frac{\sum_j p_j}{\sum_j p'_j}$$

It is difficult to state precisely which value τ might have. But since the only case where $p_j/p'_j = 0$ is when the edges involved in that intersection are parallel, while configurations such as the one figure 3(a)(b) respectively correspond to ratios of 1 and .549 (see section 5.1), it is reasonable to say that k' and k are within a constant factor.

Another interesting problem is whether or not it is possible to apply the paradigm of divide-and-conquer to this selection process. We skip its discussion for space reasons.

4 Intersection/Union

4.1 Computing the intersection structure from the events

Figure 7 shows an example configuration of the two polygons (after the tolerance values have been applied to the edges of the base polygons).

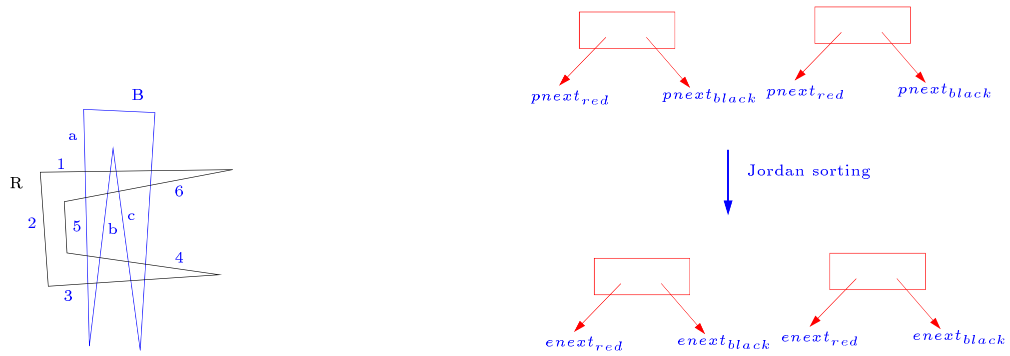

Figure 7: a) Computing intersection structure and b) events from the structure

Along with each intersection vertex $< e_R, e_B >$ found from the segment tree data structures, we also obtain red and blue pnext pointers. The red pnext pointer corresponds to the intersection of e_B with the next counter-clockwise edge of the blue polygon (and similarly for the blue pnext pointer). For example, consider the edge a of the polygon B in Figure 7. The red pnext pointer for the intersection $< a, 1 >$ stores the next intersection vertex along the red polygon R, namely the intersection vertex $< b, 1 >$. Similarly, the blue pnext pointer stores the vertex $< a, 3 >$. This chaining of intersection vertices is obtained as a result of sorting of the vertices at the nodes of the segment tree at which they are stored. We emphasize that the pnext pointer stores the next vertex along the polygon edge sequence, rather than the more obvious enext vertex along the sequence of intersections at a given edge (we are eventually looking for this next vertex). We refer to this latter pointer as the enext pointer and we show how to use *Jordan sorting* to derive the enext pointers from the pnext pointer.

4.2 Computing the intersections of the Jordan arc with a line segment

For each edge of each polygon, we use the Jordan sorting algorithm described in [8], to compute the sorted sequence of intersections *along the edge* starting from the sorted sequence of intersections *along the intersecting polygon* (the Jordan arc in the Jordan sorting algorithm). The sorting can be done in time linear in the number of intersections (independent of the size of the Jordan arc itself). We now obtain the (respectively red and blue) *enext* pointers for each intersection vertex, referring to the next intersection along the edge of the (respectively red and blue) polygon.

Note that in this step, we are making use of the assumption that each of the polygons is non-self intersecting for all possible values of the tolerance. If the polygon could be self-intersecting, it would not be possible to use Jordan sorting at all, since the polygon would not be a Jordan arc.

It is also possible to sort the intersection vertices for each edge directly using traditional sorting methods. The advantage of the Jordan sorting lies in the fact that we make use of the pre-sorted order given the pnext pointer.

4.3 Computing the intersection structure from the results of the Jordan sorting

Given the results of the Jordan sorting algorithm it is easy to compute the intersection structure by following up the edges of the intersection in sequence. At any vertex of the intersection it is clear from the local structure which edge must be pursued next, as in Figure 8. The enext pointers tell us which vertex appears first along the edge that is being currently pursued. We have finished with the current component when we come back to the vertex from which we started. It is then time to visit the next intersection vertex and enumerating the next component. This lets us iteratively obtain all of the components of the intersection in an output-sensitive time manner.

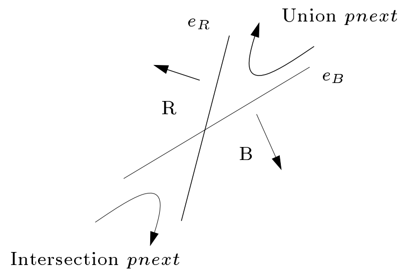

Figure 8: Computing the union and intersection of events

4.4 Computing the union

The union computation proceeds along very similar lines as the intersection. Figure 8 shows the local behavior of the union and the intersection of polygons R and B with polygon interiors depicted by the straight arrows. The only difference between the two cases is the behavior of the enext pointer: the two cases are shown using curved arrows. In both cases, we use the convention that the interior of the polygon is always to the left of the enext direction. The union has a single outer face and possibly multiple number of holes. Our convention for the enext results in a counter-clockwise outer face and clockwise holes.

5 Stability of Intersection Features

In this section, we study the stability of features of the intersection. Features of the union can be handled similarly. There are obviously two kinds of features: vertices, and components that can involve several vertices. For each of these, *stability* refers the probability of occurrence, with the distinction of *stable features* that always exist, and *optional features* otherwise.

5.1 Vertices

Checking if a vertex is compulsory is easily done when computing the approximation $\nu_{ij}^{'}$ of ν_{ij}. If the vertex is optional, a measure of its stability is the volume p_{ij} of ν_{ij}. Computing the exact value of this volume seems difficult since we do not have a description of its boundary. Getting an (ε, δ) approximation $\tilde{p_{ij}}$ of p_{ij} using a Monte Carlo method (see [12, 4]) is also an open problem since ν_{ij} might in general be non-convex. However, from a practical point of view, the following boot-strapping algorithm may give satisfactory results:

1. First, get a rough estimate $\tilde{p_{ij}}$ of p_{ij} as the fraction of points satisfying condition $\mathcal{I}$ over a sample of 'reasonable' size uniformly drawn in $[0, 1]^6$
2. Use $\tilde{p_{ij}}$ in the estimator theorem (theorem 11.1) of [12] which in turn gives the sample size such that $\tilde{p_{ij}}$ is accurate within a factor ϵ with a probability greater than $1 - \delta$.

For example, on the configuration of figure 3(b), an estimate on a sample of size 1000 gives $\tilde{p_{ij}} = .543$ while the corrected value according to a sample of size 11000 given by the estimator theorem for a (.05, .05) approximation is $\tilde{p_{ij}} = .549$.

5.2 Components

We define a sequence of red, blue and purple vertices forming a simple polygon intersection of the original polygons as a component. Extending this definition is tricky. Indeed, as depicted on figure 9(a), a component can be stable while none of its vertices are. In a similar way, a component can be stable while none of its edges are so: on figure 10(a)(b), by continuously transforming the leftmost configuration to the rightmost one, we go from a configuration where the intersection consists of the leftmost edge of the vertical rectangle together with the rightmost edges of the rotated square, to a configuration involving the rightmost edge of the rectangle and the leftmost edges of the square. For these reasons, we constrain the previous definition to a fixed sequence of edges from the two polygons. For example on figure 9, there are four different components, $\{2, b, 1\}, \{2, b, c, 1\}, \{2, a, b, 1\}$ and $\{2, a, b, c, 1\}$.

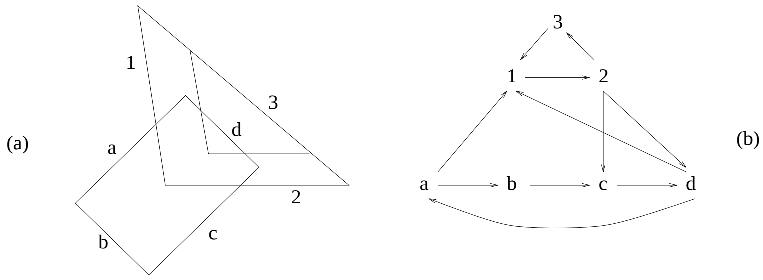

Figure 9: Components, stable vertices and $\mathcal{IVG}$ graph

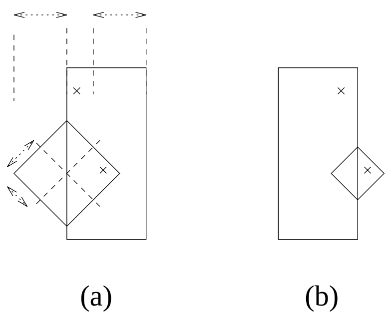

Figure 10: Components and stable edges

Enumerating these components is different from enumerating the set of all topologies of the intersection. Indeed, the later reflects the local interference between red and blue edges, while the former is the cartesian product of these. We therefore focus our attention on the computation of components involving a given intersecting edge. We do not claim any bound on the running time of the algorithm but just sketch it here. Let $\mathcal{IVG}$ be the intersection vertices graph that is the oriented graph defining the possible intersections between edges of the two polygons as well as the connectivity between the red and blue edges. Figure 9(b) depicts the $\mathcal{IVG}$ for figure 9(a). The convention used to assign directions to edges of $\mathcal{IVG}$ is the following: edge u points to edge v if a turn from u into v that has interior to the left. The basic idea of the algorithm consists in considering each a cycle of $\mathcal{IVG}$ and checking if it corresponds to a valid component. But any cycle is not a good candidate, since the following constraints need to be satisfied:

- Each turn must have the interior to its left.
- If an edge e of a polygon, say the blue edge, appears several times in the labelling, the red intersecting edges must respect the order output by the Jordan sorting algorithm.

- The labelling must respect as much as possible, the 'locality' of the intersection sought. For example on figure 9(b), the word $2abc$ can be expanded as $2abc1$ or $2abcd$. But $2abcd$ is obviously not valid. It is not clear however how to use this geometric information while generating good candidates cycles.

6 Applications

We briefly examine in this section two applications of the computations described in this paper and list another one that requires Minkowski operations to be defined for toleranced polygons. For space requirements, we omit the details and the reader is requested to refer to the cited papers.

6.1 Feasibility of assembly of two parts

A simple but instructive real life example on how useful operations between toleranced objects is the following from [5]: a fly fishing reel consists of a spool mounted on high quality bearings spinning around a centre pin assembly. The hole in the centre of the reel has to be machined smaller than the bearing so that when the reel is pressed in place interference is observed. Performing the intersection operations between the different parts of the reel (on a crosssection so that cylinders are represented as rectangles) therefore turns out to be very useful: it gives the topology of the interference zones, which provides feedback to the designer on how precisely the parts have to be milled depending on the type of material used (aluminum in this case).

6.2 Assembly sequencing

As already mentioned in the introduction, results were obtained recently for assembly sequencing of toleranced assemblies in [9] that defines the tolerance model we use. Without mentioning the details, the authors propose two algorithms:
-one that lists all the assembly sequences that are *always* feasible, whatever the tolerance values are
-another that lists the sequences that may be feasible *only* for some combinations of the parameters.

A failure of the second algorithm means that no instance of the product is assemblable, which in turns implies that there is a collision between two parts, or that the product is 'intrinsically' infeasible for assembly. In the first case, applying our intersection algorithm to the toleranced polygons gives the parameters involved in the intersection. This is important because paying more attention to these parts in the manufacturing process –that is reducing the tolerance zones, might solve the problem. Handling the second case is much more difficult and goes beyond the scope of this paper.

For the above applications, using polygons to model the input for union and intersection operations is not very restrictive since most of the contacts between parts are either cylindrical or between flat surfaces.

6.3 Collision detection

It is well known that one of the most elementary operations in robotics consists in computing the *Minkowski difference* between robots and an obstacle that encodes the space of intersection free translations of the robot. In particular in configurations that contain degenerate input such as contacts, tangencies, etc, collision checking on the toleranced objects may give additional information.

7 Conclusion

In this paper we presented data structures and algorithms for computing the intersection and union of simple polygons with tolerances on edges. Given two polygons of sizes n and m whose edges give rise to s intersections for all the combinations of the tolerances values, our algorithm pre-computes in time $O((m+n)\log(m+n)+s)$ a search structure that takes $O(s)$ space. Given specific values for the tolerances, we use the structure to output the specific intersection or union in time $O((m+n)\log s + k^{'} + k \log k)$ where k is the output size and $k \leq k^{'} \leq s$. Although the algorithm is not output sensitive, we show that the expected values of k and $k^{'}$ remain within a constant factor τ, a function of the input geometry. The algorithm is easy to implement, practical, and we believe it works well for realistic input instances. Also, several straightforward applications to feasibility of assembly and assembly sequencing are described.

Many difficult issues still remain. Firstly, directly storing the semi-algebraic set describing the intersection of toleranced edges may enable more efficient processing of union and intersection queries. Secondly, getting an (ϵ, δ) approximation for the volume of the semi-algebraic set is an open question.

At last, the problems remain unaddressed for the case when the tolerance model is extended to include angles on the polygons. There is also scope for future work on computing convolutions and Minkowski sums of toleranced polygonal objects.

ACKNOWLEDGMENTS: The authors wishes to thank Cyprien Godard and Danny Halperin for helpful discussions.

References

[1] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

[2] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. of the ACM*, 39(1-54), 1992.

[3] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte Carlo estimation (extended abstract). In *36th Annual Symposium on Foundations of Computer Science*, pages 142–149, Milwaukee, Wisconsin, 1995. IEEE.

[4] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. In *ACM STOC*, pages 375–381, 1989.

[5] W. H. ElMaraghy et al. Intersection volumes and surface areas of cylinders for geometrical modelling and tolerancing. *C.A.D*, 26(1):29–45, 1994.

[6] Shiaofen Fang and Beat Brüderlin. Robustness in geometric modeling — tolerance-based methods. In *Computational Geometry — Methods, Algorithms and Applications: Proc. Internat. Workshop Comput. Geom. CG '91*, volume 553 of *Lecture Notes in Computer Science*, pages 85–101. Springer-Verlag, 1991.

[7] U. Finke and K. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In 11^{th} *ACM Symposium on Computational Geometry*, Vancouver, 1995.

[8] K.Y Fung et al. Simplified linear-time jordan sorting and polygon clipping. *IPL*, 35, 1990.

[9] JC. Latombe and R.H. Wilson. Assembly sequencing with toleranced parts. In 3^{rd} *ACM Symposium on Solid Modeling and Applications*, Salt Lake City, 1995.

[10] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–71, 1991.

[11] B. Moller. Tolerances in product modelling. *Informatik, Informationen Reporte*, 1991(5):4–52, 1991.

[12] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[13] J. Nievergelt and F. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the A.C.M*, 25(10), 1982.

[14] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[15] U. Roy, C.R. Liu, and T.C. Woo. Review of dimensioning and tolerancing: representation and processing. *Computer Aided Design*, 23(7):466–83, 1991.

[16] V. Srinivasan. Recent efforts in mathematization of asme/ansi y14.5m standard. In 3^{rd} *CIRP seminars on computer aided tolerancing*, Ed. Eyrolles, Paris, 1993.

[17] R.K. Walker and V. Srinivasan. Creation and evolution of the asme y14.5.1 standard. *Manufacturing Review*, 7(1), 1994.

[18] N. Wang and T.M. Ozsoy. A scheme to represent features, dimensions, and tolerances in geometric modeling. *Journal of Manufacturing Systems*, 10(3):233–40, 1991.