Inner Approximation of Polygons and Polyhedra by Unions of Boxes

Christian Spielberger* Martin Held†

Abstract

Given a multiply-connected polygonal area $\mathcal{P}$ in the plane and a point set $S \subset \mathbb{R}^2$, where some points of S may lie inside of $\mathcal{P}$, we present a fast approximation method for finding a largest axis-aligned or oriented rectangle contained in $\mathcal{P}$ which does not contain any points of S. All standard meanings of "largest" are supported, such as maximum area and maximum perimeter. This heuristic is extended to finding k rectangles whose union is largest. Furthermore, we present an extension of our method to 3D, i.e., to computing inner approximations of polyhedra (possibly with holes, voids and cavities) by unions of (oriented) boxes.

Our 2D algorithm is based on a discretization of space by means of a regular mesh of size $w \times h$ and on a discretization of the rotation angles. Let n be the sum of the number of vertices of $\mathcal{P}$ and the number of points in S. Then an inner approximation by k rectangles is found in $O\left(mn(w+h) + k2^k(mwh)^k\right)$ time and $O(wh+n)$ space, where m denotes the number of rotation angles tested. A similar bound is obtained for the 3D case. Several algorithmic improvements help to decrease the time complexity in practice considerably, thus making it quite feasible to determine inner approximations of complex objects by several boxes within a few seconds of CPU time. Extensive practical tests have yielded a formula for predicting a mesh resolution suitable for achieving the approximation quality sought by a user.

1 Introduction

Motivation. A problem that often arises in metal or textile industry is to cut a rectangle as large as possible out of an arbitrarily shaped piece of sheet metal or cloth [3]. Typically, the term "large" means "maximum area" but other measures of size (such as maximum perimeter) may also be of interest. Sometimes the situation is further complicated if discrete spots of material imperfectness are to be excluded from the rectangle sought.

Finding one or more maximal boxes that are located inside of a shape can also be regarded as an *inner approximation* (or *inner cover*) of that shape. For instance, for visibility tests we are asked whether an object $\mathcal{O}$ in the foreground occludes other objects in the background. Since visibility tests in 3D scenes are very time consuming for complex objects $\mathcal{O}$, researchers in graphics have long been interested in inner approximations of $\mathcal{O}$ by one (or more) convex shapes, such boxes, spheres and ellipsoids: rather than testing a 3D scene for occlusion against $\mathcal{O}$, it may be significantly faster to test the scene against an inner approximation of $\mathcal{O}$. Similarly, simple pre-tests for path planning are based on inner approximations.

Overview of Results. We study the following problem: Given an integer k, a multiply-connected planar area $\mathcal{A}$ and a set S of n points (in $\mathbb{R}^2$), find k rectangles inside of $\mathcal{A}$ such that the rectangles do not contain any point of S and such that their union has maximum size according to a measure μ. Typically, μ will denote the area of a rectangle but our algorithm will be able to deal with any measure μ provided that $\mu(A) \leq \mu(B)$ if $A \subseteq B$ for two planar areas A and B. In particular, μ could also measure the perimeter or the length of a rectangle. Depending on the application, the rectangles will have sides parallel to the coordinate axes (*axis-aligned*) or they will be *oriented* rectangles, i.e., they will have arbitrary orientations.

A natural extension to 3D asks to determine k cuboids inside of a polyhedron $\mathcal{P}$ such that the size of their union is maximum with respect to μ and such that no point of a set S of points of $\mathbb{R}^3$ is contained in a cuboid. As for the 2D polygons, the polyhedron $\mathcal{P}$ may contain voids, holes and cavities. Again, we are interested in both axis-aligned and oriented cuboids.

In the sequel we present a fast heuristic for finding such a set of k large boxes in 2D and 3D. Our algorithm is based on a discretization of space by means of a regular mesh of size $w \times h$. Furthermore, we apply a straightforward discretization of the rotation angles. Let n be the sum of the number of vertices of $\mathcal{P}$ and the number of points in S. Then the worst-case time complexity of the 2D algorithm is $O\left(mn(w+h) + k2^k(mwh)^k\right)$, where m denotes the number of rotation angles tested; its space complexity is $O(wh+n)$. A similar bound is obtained for the 3D case. Several algorithmic improvements help to decrease the time complexity in practice considerably, thus making it quite feasible to determine inner approximations of complex objects by several axis-

*Department of Scientific Computing, University of Salzburg, Salzburg, Austria; christian.spielberger@aon.at

†Department of Scientific Computing, University of Salzburg, Salzburg, Austria; held@cosy.sbg.ac.at

aligned or 2–3 oriented boxes within a few seconds.

Since our algorithm depends on the resolution of the mesh it cannot guarantee a constant-factor approximation of the true maximum size of an optimum set of k boxes. Furthermore, there is an obvious trade-off between the quality of the inner approximation and the resolution of the mesh. Based on extensive practical tests we came up with a formula that allows to predict a mesh resolution such that the approximation quality sought by a user can be expected to be achieved with a user-specified probability.

Our algorithms for inner approximation by k boxes have been implemented in C++ for both the 2D and the 3D case. In the sequel, we will mostly focus on the 2D case, though.

Related Work. Restricted versions of our problem have received considerable interest in the past. For the 2D case, if $\mathcal{P}$ is restricted to a rectangle A, the problem of finding the largest rectangle inside of A whose sides are parallel with those of A and which does not contain a point of S was discussed in a variety of papers, see [8] for a survey. (In most papers, "largest" means "largest area".) The fastest algorithm that solves this problem runs in $O(n \log n + s)$ time, where s is the number of axis-aligned rectangles, whose edges contain a point of S or are contained in the border of A [8]; its expected time complexity is $O(n \log n)$. It is notable that this algorithm uses only $O(n)$ memory. A more general problem is to find the largest empty oriented rectangle bounded by a point of S on each of its four sides. This problem can be solved in $O(n^3)$ time [2, 6].

A largest-area axis-aligned rectangle (LAR) inside of a multiply-connected polygonal area can be found in $O(n \log^2 n)$ time [3]. For polygons without islands an $O(n \log n)$ algorithm is presented in [1]. Note that none of those papers considers additional point constraints within $\mathcal{P}$. (That is, the set S is empty.) Also, nothing is known for finding k (oriented) rectangles (LORs) such that their union has maximum size.

For the 3D case of a set S of n points inside of a bounding cuboid, a largest empty cuboid can be computed in $O(n^3)$ time and $O(n^2 \log n)$ space in the worst case, see [4]. In [7], it is claimed that all locally maximal cuboids can be reported in $O(c + n^2 \log n)$ time and $O(n)$ space, where c is bound by $O(n^3)$.

No algorithms are known for finding largest axis-aligned and largest oriented cuboids inside a polyhedron. As in 2D, an inner approximation by $k > 1$ axis-aligned or oriented boxes also is an open problem.

2 Finding a LAR Inside a Polygon

Our heuristic for computing a maximum axis-aligned rectangle (relative to the measure μ) uses a regular mesh for discretizing the plane. A rectangle that consists entirely of mesh cells is called a *mesh rectangle*. Our algorithm finds the largest mesh rectangle which lies completely in the given polygon $\mathcal{P}$ and does not contain any point of S.

Consider a regular mesh $\mathcal{M} := \{0, 1, \ldots, w-1\} \times \{0, 1, \ldots, h-1\}$ with resolution $w \times h$. In the first step, all cells $\mathcal{M}(a, b)$ that are intersected by the boundary $\partial\mathcal{P}$ of the polygon are determined and their cell values are set to zero. Then all cells that include at least one point of S are set to zero. Both can be done in $O(n(w + h))$ time. Then all outer cells are set to -1 in $O(wh)$ time. The inner cells are set to the values specified by the *chessboard distance*. Several algorithms are known for computing the chessboard distance on a $w \times h$ mesh in $O(wh)$ time, see, for instance, [5]. We use a modified flood-fill algorithm: in the first step, all eight neighbor cells of zero cells that currently have undefined values get the value one. Then the neighbor cells of 1-cells get the value 2, and so on. (In Fig. 1, the shaded cells depict the zero cells occupied by $\partial\mathcal{P}$ and the points of S). The resulting value obtained for a cell $\mathcal{M}(a, b)$ is denoted by $d(a, b)$.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	-1	0	0	0	0	-1	-1	-1	-1	-1
-1	-1	0	0	1	1	0	0	0	0	-1	-1
-1	0	0	1	1	1	1	1	1	0	-1	-1
-1	0	1	1	2	2	2	2	1	0	-1	-1
-1	0	1	2	2	2	3	2	1	0	0	0
-1	0	1	1	1	2	2	2	1	1	0	-1
-1	0	1	0	1	1	1	1	1	1	0	-1
-1	0	1	1	0	0	0	0	0	1	0	-1
-1	0	0	0	0	-1	-1	-1	0	0	0	-1
-1	0	0	-1	-1	-1	0	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 1: Discretization of a polygon.

The *square anchored at* cell $\mathcal{M}(a, b)$ is defined as the square with lower left cell $\mathcal{M}(a-(d-1), b-(d-1))$ and upper right cell $\mathcal{M}(a+(d-1), b+(d-1))$, where $d = d(a, b)$. It is easy to see that every such square is fully contained in $\mathcal{P} \backslash S$.

For an inner cell $\mathcal{M}(a, b)$ the number of cells which are to the right of it and which have a cell value of at least $d(a, b)$ is given by the *horizontal length code* $h(a, b)$. Similarly, a *vertical length code* $v(a, b)$ is defined for each inner cell. The horizontal length code for each cell of a row $\mathcal{M}(., b)$ can be set in $O(w)$ time by stepping from cell $\mathcal{M}(w-1, b)$ to cell $\mathcal{M}(0, b)$ and by performing the following tasks for each inner cell $\mathcal{M}(a, b)$.

- If $d(a, b) > d(a+1, b)$ then the length code $h(a, b)$

stays 0.

- If $d(a,b) = d(a+1,b)$ then the length code $h(a,b)$ is set to $h(a+1,b)+1$.
- If $d(a,b) < d(a+1,b)$ then the length code is set to the value $h(a+1,b)+h(c,b)+2$, where $c = \min\{x \in \mathbb{N} : x > a \text{ and } d(x,b) = d(a,b)\}$.

Suppose $d = d(a,b)$. The value $h(c,b)$ is the length code of the next d-cell in this row. This value is stored in an array with index d. Thus, it can be retrieved in constant time. These tasks are done for each row in the mesh. The vertical length code is set in a similar manner, by stepping downwards through the cells of columns. The calculation of the length code takes $O(wh)$ time.

The mesh rectangle with lower-left cell $\mathcal{M}(l_a, l_b)$ and upper-right cell $\mathcal{M}(u_a, u_b)$ is denoted by $R[(l_a, l_b), (u_a, u_b)]$. For any inner cell $\mathcal{M}(a,b)$ the mesh rectangle $\text{rec}_{h,1}(a,b) := R[(a-d+1, b-d+1), (a+h+d-1, b+d-1)]$, where $d = d(a,b)$ and $h = h(a,b)$, is the *horizontal mesh rectangle anchored at* cell $\mathcal{M}(a,b)$ with a *one-row core*. If $d(a,b+1) \geq d(a,b) \geq 1$, the mesh rectangle $\text{rec}_{h,2}(a,b) := R[(a-d+1, b-d+1), (a+h_2+d-1, b+d)]$, where $d = d(a,b)$ and $h_2 = \min(h(a,b), h(a,b+1))$, is the *horizontal mesh rectangle anchored at* cell $\mathcal{M}(a,b)$ with a *two-row core*. Similarly, *vertical mesh rectangles* with *one-* and *two-column cores* are defined. Since an arbitrarily anchored mesh rectangle can be covered by a set of anchored squares, it is fully contained in $\mathcal{P} \backslash S$.

Theorem 1 *For an arbitrary mesh rectangle R in $\mathcal{P}$ that does not contain any point of S there is an anchored mesh rectangle which covers R.*

Theorem 2 *Given an arbitrary n_1-vertex polygon $\mathcal{P}$ in the plane, a point set S with n_2 points, and a mesh $\mathcal{M}$ with resolution $w \times h$ covering $\mathcal{P}$, a largest mesh rectangle contained in $\mathcal{P} \backslash S$ can be found in $O(n(w+h)+wh)$ time using $O(wh+n)$ memory, where $n = n_1 + n_2$.*

Proof. Setting the zero cells and the outer cells can be done in $O(n(w+h)+wh)$ time and $O(wh+n)$ space. The cell values in the interior of $\mathcal{P}$ can be calculated without further costs. The length code can also be set in $O(wh)$ time for the whole mesh.

Let S' be the set of all anchored mesh rectangles. Due to Theorem 1, the set S' is a sufficient search space for the largest mesh rectangle inside $\mathcal{P}$. Since there are only $O(wh)$ mesh rectangles in S', for each inner cell one, searching S' does not increase the time and space complexity further. □

An inner cell $\mathcal{M}(a,b)$ is called a *horizontal upward cell*, if $d(a,b) > d(a-1,b)$ or $d(a,b) = d(a,b+1)$ and $d(a,b) > d(a-1,b+1)$.

Similarly, an inner cell $\mathcal{M}(a,b)$ is called a *vertical upward cell*, if $d(a,b) > d(a,b-1)$ or $d(a,b) = d(a+1,b)$ and $d(a,b) > d(a+1,b-1)$.

A mesh rectangle which is anchored in an upward cell is called *upward anchored*.

Theorem 3 *For any anchored mesh rectangle R there is an upward anchored mesh rectangle which covers R.*

Thus, the search space can be reduced to the set of all upward anchored mesh rectangles. Each upward cell gets a pointer to the next upward cell for this reason. The first cell in a mesh row (column) gets the pointer to the first upward cell in this row (column). Hence, many cells can be left out during the search. This does not reduce the worst-case time complexity but tends to decrease the practical run-time of the program considerably.

3 General Orientation

By using the method of Section 2 a simple heuristic for the LOR can be built. Again, given is an arbitrary polygon $\mathcal{P}$ and a point set S. The angle range $[0\,..\,\frac{\pi}{2}]$ is discretized. The input data is rotated by the angles $0, \delta, 2\delta, \ldots, (m-1)\delta$, where m is an integer with $m \geq 2$, and $\delta := \frac{\pi}{2m}$. The LAR is calculated for each of the rotated data sets. The largest mesh rectangle returned by the LAR algorithm is the result of the LOR algorithm. (Of course, this mesh rectangle has to be re-transformed to the original orientation.)

4 Generalization to 3D Space

We extended the described method to approximate the maximum volume cuboid inside a given polyhedron $\mathcal{P}$ such that no point of S is included. A 3D mesh was used to discretize the space. However, the length code can not be adopted directly to the 3D mesh. Instead the length codes have to be calculated for each of the three coordinate planes xy, xz and yz. In addition a 3D chessboard distance is used to retrieve the maximal thickness of each cuboid.

5 Inner Approximation by Several Objects

Let $\mathcal{F}$ be a finite set of shapes in $\mathbb{R}^d$. (In our application, $d = 2, 3$.) Let $k \geq 2$ be an integer. Given a region $\mathcal{R} \subset \mathbb{R}^d$ and a set of points $S \subset \mathbb{R}^d$, we want to find k shapes $C_1, \ldots, C_k \in \mathcal{F}$ such that $\cup_{1 \leq i \leq k} C_i \subseteq \mathcal{R} \setminus S$ and $\mu(\cup_{1 \leq i \leq k} C_i)$ is maximum. Note that the shapes C_i may intersect. Let $\binom{\mathcal{F}}{k}$ be the set of all k-elemental subsets of $\mathcal{F}$. The proposed algorithm performs a brute-force search of the set $\binom{\mathcal{F}}{k}$ for the largest union of k shapes relative to the measure μ. Let s be the cardinality of $\mathcal{F}$. The algorithm has

to check $\binom{s}{k}$ shape combinations by a k-times nested loop. Thus the union of k shapes has to be calculated $O(s^k)$ times.

The calculation of μ for such a union is done by the inclusion-exclusion principle, by relying on $\sum_{i=1}^{k}\binom{k}{i} = 2^k$ calculations of $\mu(\cap_{C\in\mathcal{B}}C)$, where $\mathcal{B}$ can have at most k elements. How long it takes to calculate the intersection of k shapes depends on the complexity of the shapes and has to be analyzed for each specific type of shapes. For example, computing the common intersection of k axis-aligned rectangles takes $O(k)$ time. Thus the calculation of the union volume of k axis-aligned rectangles needs $O\left(k\,2^k\right)$ time, and our shape selector needs at most $O\left(k\,2^k s^k\right)$ time in total.

Two improvements may reduce the average time complexity drastically. First, let $\mathcal{F}'$ be the set of all shapes in $\mathcal{F}$ that are not contained in another shape in $\mathcal{F}$. The shape set $\mathcal{F}'$ is a sufficient search space for the shape combination that maximizes μ. The second improvement relies on $\mathcal{F}'$ being arranged in decreasing order according to μ. Our shape selector uses a k-times nested loop to determine $C_1,\ldots,C_k$, starting with the the j-largest shape in loop j. During each pass through the body of a loop the next smaller shape is tested. Suppose that the current candidate shapes selected in loops 1 to $(j-1)$ are $C_{i_1}, C_{i_2},\ldots,C_{i_{j-1}}$, and we are to select a shape in loop j. Let C_{i_j} be the next smallest shape after $C_{i_{j-1}}$ in the ordered list of shapes. If

$$M > \mu(C_{i_1}\cup C_{i_2}\cup\cdots\cup C_{i_{j-1}}) + (k-j+1)*\mu(C_{i_j}),$$

where M is the size of the union of the best selection of shapes obtained so far, then an early termination of the loops j to k is possible (since no better result will be obtained) and the next smaller shape is tested in loop $j-1$.

6 Choosing a Suitable Mesh Resolution

Let A be the area of the true LAR and let A' be the area of the largest mesh rectangle obtained by our algorithm. The ratio $\alpha = A'/A$ is called *approximation ratio* and should be as close to 1 as possible. Let B be the bounding box of $\mathcal{P}$. The ratio $a_{\mathcal{P}} = \frac{a(\mathcal{P})}{a(B)}$ is called the *relative area* of $\mathcal{P}$, where a is a measure for the area. Let $p(\mathcal{P})$ be the perimeter of $\mathcal{P}$. Further, let $p(B)$ be the perimeter of the bounding box of $\mathcal{P}$. The ratio $p_{\mathcal{P}} = \frac{p(\mathcal{P})}{p(B)}$ is called *relative perimeter* of $\mathcal{P}$. Furthermore, we call the ratio $t_{\mathcal{P}} = a_{\mathcal{P}}/p_{\mathcal{P}}$ the *thickness* of $\mathcal{P}$. It is an attempt to cast an intuitive understanding of the "thickness" of a polygon into a numerical number.

Tests indicated that the higher the thickness of a polygon is the lower the resolution of the mesh can be in order to achieve the same approximation ratio. Let $w\times w$ be the resolution of the mesh $\mathcal{M}$ used. The value $w^2\,t_{\mathcal{P}}$ is called *weighted resolution* of $\mathcal{M}$ relative to $\mathcal{P}$. Since the number of inner cells grows with the thickness, the approximation ratio grows with the weighted resolution. Our 2D algorithm was tested over a variety of polygons with different mesh resolutions. As a result we obtained a look-up table and a formula for the mesh resolution. The parameters for the formula are the weighted resolution, which can be read off from the table, and the area and the perimeter of the polygon. The row index of the table is the desired approximation ratio α and the column index is the probability that α will be achieved.

7 Conclusion

We use a regular mesh to discretize 2D and 3D space for computing inner approximations of polygonal/polyhedral shapes by one or more axis-aligned or oriented boxes. The time consumption of the algorithm depends on the mesh resolution which, on the other hand, influences the quality of the approximation obtained. Although the basic idea is rather simple, several algorithmic improvements make our approach quite feasible for the approximation of complex shapes by several boxes. Based on extensive tests we have come up with a heuristic to predict a suitable resolution of the mesh relative to the approximation quality requested by a user.

References

[1] R. P. Boland and J. Urrutia. Finding the Largest Axis-aligned Rectangle in a Polygon in $O(n\log n)$ Time. In *Proc. 13th Canad. Conf. Comput. Geom.*, pages 41–44, 2001.

[2] J. Chaudhuri, S. Nandy, and S. Das. Largest Empty Rectangle Among a Point Set. *J. Algorithms*, 46(1):54–78, 2003.

[3] K. Daniels, V. Milenkovic, and D. Roth. Finding the Largest Area Axis-Parallel Rectangle in a Polygon. *Comput. Geom. Theory and Appl.*, 7:125–148, 1997.

[4] A. Datta and S. Soundaralakshmi. An Efficient Algorithm for Computing the Maximum Empty Rectangle in Three Dimensions. *Informat. Sciences Appl. An Int. J.*, 128(1):43–65, 2000.

[5] T. Hirata. A Unified Linear-Time Algorithm for Computing Distance Maps. *Inform. Process. Lett.*, 58(3):129–133, May 1996.

[6] A. Mukhopadhyay and S. V. Rao. On Computing a Largest Empty Arbitrarily Oriented Rectangle. *Internat. J. Comput. Geom. Appl.*, 13(3):257–271, 2003.

[7] S. C. Nandy and B. B. Bhattacharya. Maximum Empty Cuboid Among Points and Blocks. *Computers & Math. with Appl.*, 36(3):11–20, 1998.

[8] M. Orlowski. A New Algorithm for the Largest Empty Rectangle Problem. *Algorithmica*, 5:65–73, 1990.