Minimum-Area Rectangle Containing a Set of Points

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: May 17, 2015
Last Modified: February 14, 2019

Contents

The previous version of this document (February 9, 2008) discussed the simple $O(n^2)$ algorithm for computing the minimum-area rectangle that contains a convex polygon of n vertices. The current version includes a discussion of the *rotating calipers algorithm*, an algorithm that can be used to compute the minimum-area rectangle in $O(n)$ time. Like most algorithms in computational geometry, a robust implementation using floating-point arithmetic is difficult to achieve. Details of the GTEngine implementation are provided here, where we use exact rational arithmetic to guarantee correct sign classifications for subproblems that arise in the algorithm.

1 Introduction

Given a finite set of 2D points, we want to compute the minimum-area rectangle that contains the points. The rectangle is not required to be axis aligned, and generally it will not be. It is intuitive that the minimum-area rectangle for the points is supported by the convex hull of the points. The hull is a convex polygon, and any points interior to the polygon have no influence on the bounding rectangle.

Let the convex polygon have counterclockwise-ordered vertices $\mathbf{V}_i$ for $0 \leq i < n$. It is essential to require that the polygon have no triple of collinear vertices. The implementation is simpler with this constraint. An edge of the minimum-area bounding rectangle must be coincident with some edge of the polygon [1]; in some cases, multiple edges of the rectangle can coincide with polygon edges.

In the discussion we use the *perp* operator: $\mathrm{Perp}(x, y) = (y, -x)$. If (x, y) is unit length, then the set of vectors $\{(x, y), -\mathrm{Perp}(x, y)\}$ is right-handed and orthonormal.

2 The Exhaustive Search Algorithm

The input is a set of m points. The convex hull must be computed first, and the output is a set of n points. The asymptotic behavior of the hull algorithm depends on m, where potentially m is much larger than n. Once we have the hull, we can then construct the minimum-area rectangle.

The simplest algorithm to implement involves iterating over the edges of the convex polygon. For each edge, compute the smallest bounding rectangle with an edge coincident with the polygon edge. Of all n rectangles, choose the one with the minimum area. To compute the bounding rectangle for an edge, project the polygon vertices onto the line of the edge. The maximum distance between the projected vertices is the width of the rectangle. Now project the polygon vertices onto the line perpendicular to the polygon edge. The maximum distance between the projected vertices is the height of the rectangle. For the specified polygon edge, we can compute the rectangle axis directions and the extents along those directions. Pseudocode for the algorithm is provided in Listing 1.

Listing 1. The $O(n^2)$ algorithm for computing the minimum-area rectangle containing a convex polygon.

```
struct Rectangle
{
    Vector2<Real> center, axis[2];
    Real extent[2], area;  // area = 4 * extent[0] * extent[1]
};

Rectangle MinAreaRectangleOfHull(Vector2<Real> const& polygon)
{
        Rectangle minRect;
        minRect.area = maxReal;  // largest finite floating-point number
        for (size_t i0 = polygon.size() - 1, i1 = 0; i1 < polygon.size(); i0 = i1++)
        {
            Vector2<Real> origin = polygon[i0];
            Vector2<Real> U0 = polygon[i1] - origin;
            Normalize(U0);  // length of U0 is 1
            Vector2<Real> U1 = -Perp(U0);
            Real min0 = 0, max0 = 0;  // projection onto U0-axis is [min0,max0]
            Real max1 = 0;  // projection onto U1-axis is [0,max1], min1 = 0 is guaranteed
            for (size_t j = 0; j < polygon.size(); ++j)
            {
                Vector2<Real> D = polygon[j] - origin;
                Real dot = Dot(U0, D);
                if (dot < min0) min0 = dot; else if (dot > max0) max0 = dot;
                dot = Dot(U1,D);
                if (dot > max1) max1 = dot;
            }
            Real area = (max0 - min0) * max1;
            if (area < minRect.area)
            {
                minRect.center = origin + ((min0 + max0) / 2) * U0 + (max1 / 2) * U1;
                minRect.axis[0] = U0;
                minRect.axis[1] = U1;
                minRect.extent[0] = (max0 - min0) / 2;
                minRect.extent[1] = max1 / 2;
                minRect.area = area;
            }
        }
        return minRect;
    }

    Rectangle MinAreaRectangleOfPoints(std::vector<Vector2<Real>> const& points)
    {
        // Assumptions for input:
        //   1. points.size() >= 3
        //   2. points[] are not collinear
        // Assumptions for output:
        //   1. polygon.size() >= 3
        //   2. polygon[] are counterclockwise ordered
        //   3. no triple of polygon[] is collinear
        std::vector<Vector2<Real>> polygon;
        ComputeConvexHull(points, polygon);
        return MinAreaRectangleOfHull(polygon);
    }
```

The two loops make it clear why the algorithm is $O(n^2)$. If the number of vertices n is small, this is a viable algorithm to use in practice. However, the challenging part of implementing the algorithm using floating-point arithmetic is to compute the convex hull of the points robustly. The minimum-area rectangle construction for a convex polygon is a subproblem for computing the minimum-volume box containing a convex polyhedron[2]. In this context, the $O(n^2)$ behavior can be noticeable, so it is worthwhile to have a faster algorithm available for the 3D setting.

3 The Rotating Calipers Algorithm

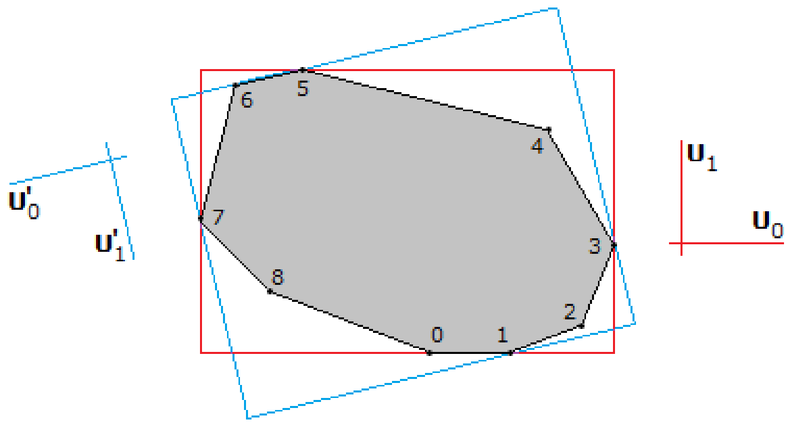
The rotating calipers algorithm is credited to [4]—although that name was not used—for locating antipodal points of a convex polygon when computing the diameter. The algorithm may be found in a source more readily available [3, Theorem 4.18]. The name of the algorithm is due to [5], an article that describes many uses for the rotating calipers.

Applied to the minimum-area rectangle problem, the rotating calipers algorithm starts with a bounding rectangle having an edge coincident with a polygon edge and a supporting set of polygon vertices for the other polygon edges. The rectangle axes are rotated counterclockwise by the smallest angle that leads to the rectangle being coincident with another polygon edge. The new supporting set of vertices is built from the previous set and from the new polygon edge vertices.

In the remainder of this section, a polygon edge $\langle \mathbf{V}_i, \mathbf{V}_{i+1} \rangle$ will be referenced by its indices instead; thus, we will refer to the edge $\langle i, i+1 \rangle$. The index addition is computed modulo n, the number of vertices of the polygon. The *supporting vertices* for a rectangle form a multiset $S = \{\mathbf{V}_{i_0}, \mathbf{V}_{i_1}, \mathbf{V}_{i_2}, \mathbf{V}_{i_3}\}$, each vertex supporting an edge of the rectangle. They are listed in the order: bottom, right, top, and left. One vertex can support two edges, which is why S can be a multiset (duplicate elements). The *supporting indices* are $I = \{i_0, i_1, i_2, i_3\}$.

Figure 1 shows the typical configuration for a bounding rectangle with edge coincident to a polygon edge and with three additional supporting vertices.

Figure 1. The typical configuration of a bounding rectangle corresponding to a polygon edge. The polygon vertices are $\mathbf{V}_i$ for $0 \le i \le 8$; they are labeled in the figure only with the subscript. The current rectangle is drawn in red, has axis directions $\mathbf{U}_0$ and $\mathbf{U}_1$, and is supported by edge $\langle 0, 1 \rangle$. The minimum rotation is determined by edge $\langle 5, 6 \rangle$, which makes it the next edge to process. The next rectangle is drawn in blue and has axis directions $\mathbf{U}'_0$ and $\mathbf{U}'_1$.



In Figure 1, the initial rectangle has its bottom edge supported by polygon edge $\langle 0, 1 \rangle$; consider the counterclockwise-most vertex $\mathbf{V}_1$ to be the supporting vertex. The right edge of the rectangle is supported by $\mathbf{V}_3$, the top edge of the rectangle is supported by $\mathbf{V}_5$, and the left edge of the rectangle is supported by $\mathbf{V}_7$. The supporting indices for the axis directions $\{\mathbf{U}_0, \mathbf{U}_1\}$ are $I = \{1, 3, 5, 7\}$.

The candidate rotation angles are those formed by $\langle 1, 2 \rangle$ with the bottom edge, $\langle 3, 4 \rangle$ with the right edge, $\langle 5, 6 \rangle$ with the top edge, and $\langle 7, 8 \rangle$ with the left edge. The minimum angle between the polygon edges and the rectangle edges is attained by $\langle 5, 6 \rangle$. Rotate the current rectangle to the next rectangle by the angle between the top edge of the current rectangle and $\langle 5, 6 \rangle$. Figure 1 shows that the next rectangle has edge coincident with $\langle 5, 6 \rangle$, but the other polygon edges were not reached by this rotation because their angles are larger than the minimum angle. The naming of the next rectangle edges is designed to provide the configuration we had for the initial rectangle; the new axis directions are draw in blue and provide the orientation that goes with the names. The new bottom edge is $\langle 5, 6\rangle$, and its counterclockwise-most vertex $\mathbf{V}_6$ is chosen as the support point for the bottom edge. The new axis directions are $\mathbf{U}'_0 = (\mathbf{V}_6 - \mathbf{V}_5)/|\mathbf{V}_6 - \mathbf{V}_5|$ and $\mathbf{U}'_1 = -\operatorname{Perp}(\mathbf{U}'_0)$. The supporting indices for the axis directions $\{\mathbf{U}'_0, \mathbf{U}'_1\}$ are $I' = \{6, 7, 1, 3\}$.

A single loop is used to visit the minimum-area candidate rectangles. The loop invariant is that you have axis directions $\{\mathbf{U}_0, \mathbf{U}_1\}$ and supporting indices $I = \{b, r, t, \ell\}$, ordered by the edges they support: bottom (b), right (r), top (t), and left (ℓ). The polygon edge $\langle b-1, b\rangle$ is coincident with the rectangle's bottom edge. $\mathbf{V}_b$ is required to be the counterclockwise-most vertex on the bottom edge. In the previous example, $5 \in I$ is selected because of the minimum-angle constraint. The remaining elements of I support the rotated rectangle, so they become elements of I'. The successor of $\mathbf{V}_5$ is $\mathbf{V}_6$, so 6 becomes an element of I'.

3.1 Computing the Initial Rectangle

Let the polygon vertices be $\mathbf{V}_i$ for $0 \le i < n$. The initial rectangle has its bottom edge coincident with the polygon edge $\langle n-1, 0\rangle$. By starting with this edge, we avoid using the modulus argument in the loop indexing. The initial axis directions are $\mathbf{U}_0 = (\mathbf{V}_0 - \mathbf{V}_{n-1})/|\mathbf{V}_0 - \mathbf{V}_{n-1}|$ and $\mathbf{U}_1 = -\operatorname{Perp}(\mathbf{U}_0)$.

The polygon vertices are converted to the coordinate system with origin $\mathbf{V}_0$ and axis directions $\mathbf{U}_0$ and $\mathbf{U}_1$. We must search for the extreme values of the converted points. An extreme value is typically generated by one vertex, but it is possible that it can be generated by two vertices. In particular, this happens when a polygon edge is parallel to one of the axis directions. Because we have assumed that no triple of vertices is collinear, it is not possible to have three or more vertices attain the extreme value in an axis direction.

When an extreme value occurs twice because a polygon edge is parallel to an axis direction, the supporting point for the corresponding edge of the rectangle is chosen to be the counterclockwise-most vertex of the edge. This is required because the update step computes rotation angles for edges emanating from the supporting vertex, and those edges must have direction pointing to the interior of the rectangle. For example, Figure 1 shows that $\langle 0, 1\rangle$ is parallel to $\mathbf{U}_0$. We choose $\mathbf{V}_1$ as the support vertex, and the emanating edge used in determining the minimal rotation angle is $\langle 1, 2\rangle$.

Listing 2 shows how to compute the smallest rectangle that has an edge coincident with a specified polygon edge. The Rectangle structure stores the axis directions and the supporting indices. The area of the rectangle is also stored for use in the rectangle update when applying the rotating calipers algorithm. This function is used to compute the initial rectangle for the rotating calipers method, but in the implementation it is also used for the exhaustive algorithm.

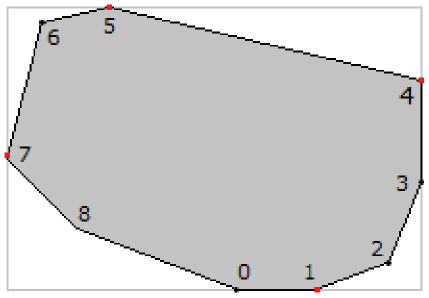
Listing 2. Pseudocode for computing the smallest rectangle that has an edge coincident with a specified a polygon edge $\langle i_0, i_1\rangle$.

```
struct Rectangle
{
    Vector2<Real> U[2];
    int index[4];  // order: bottom, right, top, left
    Real area;
};

Rectangle SmallestRectangle(int j0, int j1, std::vector<Vector2<Real>> const& vertices)
{
    Rectangle rect;
    rect.U[0] = vertices[j1] - vertices[j0];
    Normalize(rect.U[0]);  // length of rect.U[0] is 1
    rect.U[1] = -Perp(rect.U[0]);
    rect.index = { j1, j1, j1, j1 };
```

```
        Vector2<Real> origin = vertices[j1];
        Vector2<Real> zero(0, 0);
        Vector2<Real> support[4] = { zero, zero, zero, zero };

        for (size_t i = 0; i < vertices.size(); ++i)
        {
            // Convert vertices[i] to coordinate system with origin vertices[j1] and
            // axis directions rect.U[0] and rect.U[1]. The converted point is v.
            Vector2<Real> diff = vertices[i] - origin;
            Vector2<Real> v = { Dot(rect.U[0], diff), Dot(rect.U[1], diff) };

            // The right-most vertex of the bottom edge is vertices[i1]. The
            // assumption of no triple of collinear vertices guarantees that
            // rect.index[0] is i1, which is the initial value assigned at the
            // beginning of this function. Therefore, there is no need to test
            // for other vertices farther to the right than vertices[i1].

            if (v[0] > support[1][0] || (v[0] == support[1][0] && v[1] > support[1][1]))
            {
                // New right maximum OR same right maximum but closer to top.
                rect.index[1] = i;
                support[1] = v;
            }

            if (v[1] > support[2][1] || (v[1] == support[2][1] && v[0] < support[2][0]))
            {
                // New top maximum OR same top maximum but closer to left.
                rect.index[2] = i;
                support[2] = v;
            }

            if (v[0] < support[3][0] || (v[0] == support[3][0] && v[1] < support[3][1]))
            {
                // New left minimum OR same left minimum but closer to bottom.
                rect.index[3] = i;
                support[3] = v;
            }
        }

        // The comment in the loop has the implication that support[0] = zero, so the
        // height (support[2][1] - support[0][1]) is simply support[2][1].
        rect.area = (support[1][0] - support[3][0]) * support[2][1];  // width * height
    }
```

Figure 2 shows a configuration where two polygon edges are parallel to rectangle edges, in which case two extreme values occur twice.

Figure 2. The supporting vertices of the polygon are drawn in red. $\langle 0,1\rangle$ supports the bottom edge of the rectangle and $\mathbf{V}_1$ is a supporting vertex. $\langle 3,4\rangle$ supports the right edge of the rectangle and $\mathbf{V}_4$ is a supporting vertex.



3.2 Updating the Rectangle

After computing the initial rectangle, we need to determine the rotation angle to obtain the next polygon edge and its corresponding rectangle. The typical configuration of one coincident edge and three other supporting vertices has the essence of the idea for the rectangle update, but other configurations can arise that require special handling. In the discussion, arithmetic on the indices into the polygon vertices is performed modulo the number of vertices: the indices $i + j$, $i + j + n$, and $i + j - n$ all refer to the same vertex.

3.2.1 Distinct Supporting Vertices

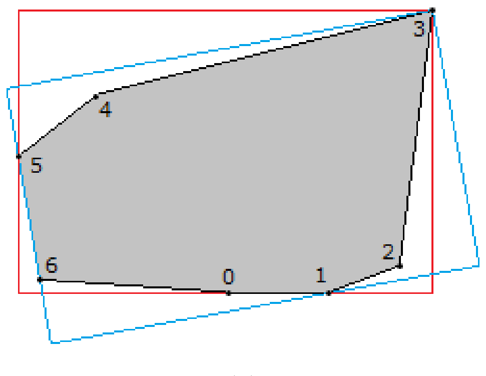
The typical configuration of Figure 1 is the simplest to update. A single polygon edge is coincident with the rectangle and has unique vertices supporting the other three edges. The supporting indices are $I = \{i_0, i_1, i_2, i_3\}$. Let $\langle i_j, i_j + 1\rangle$ be the unique edge that forms the minimum angle with the rectangle edge it emanates from, where $j \in \{0, 1, 2, 3\}$. In Figure 1, the vertex from which the edge emanates is $\mathbf{V}_5$ $(j = 2)$.

The new coincident edge is $\langle i_j, i_j + 1\rangle$. The new supporting indices I' are obtained from I by replacing i_j with $i_j + 1$ and by permuting so that i_j is the first (bottom-supporting) vertex: $I' = \{i_j + 1, i_{j+1}, i_{j+2}, i_{j+3}\}$. The additions in the j-subscripts are computed modulo 4.

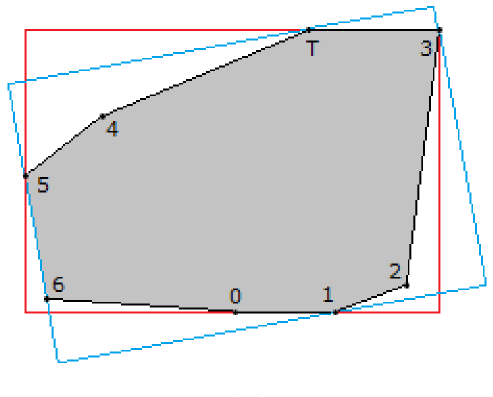
3.2.2 Duplicate Supporting Vertices

It is possible that one polygon vertex supports two rectangle edges, in which case the vertex is a corner of the rectangle. Figure 3 shows such a configuration. The current rectangles are drawn in red and the next rectangles are drawn in blue.

Figure 3. (a) A configuration where a polygon vertex supports two rectangle edges. $\mathbf{V}_3$ supports the right edge and the top edge. The supporting indices for the current rectangle are $I = \{1, 3, 3, 5\}$. The supporting indices for the next rectangle are $I' = \{6, 1, 3, 3\}$. (b) The duplicate-vertex configuration can be thought of as the limiting case of a distinct-vertex configuration, where $\mathbf{V}_T \to \mathbf{V}_3$. The supporting indices for the current rectangle are $I = \{1, 3, T, 5\}$. The supporting indices for the next rectangle are $I' = \{6, 1, 3, T\}$. Now move $\mathbf{V}_T$ to the right so that it coincides with $\mathbf{V}_3$. The current rectangles do not change during this process, but the top edge of the next rectangle of the distinct-vertex case becomes the top edge of the next rectangle of the duplicate-vertex case.



(a)



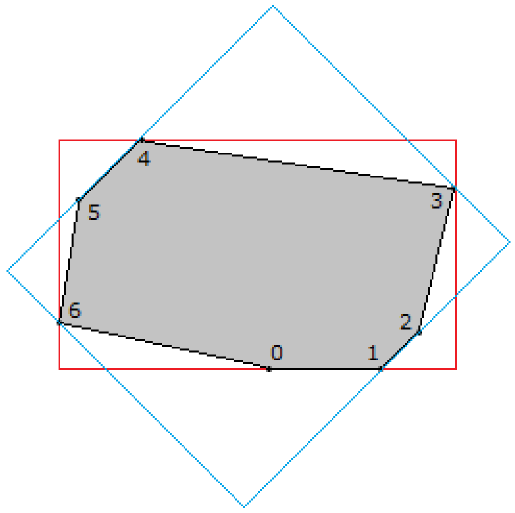
(b)

The selection of the new coincident edge and the update of the supporting indices are performed just as in the case of distinct supporting indices, except there is no minimum-angle candidate edge emanating from the first of the duplicate vertices. In the example of Figure 3(a), $I = \{1, 3, 3, 5\}$. $\langle 1, 2\rangle$ is a candidate edge. There is no edge emanating from the first occurrence of $\mathbf{V}_3$. Or you can think of the edge as $\langle 3, 3\rangle$, which is degenerate. In either case, that supporting vertex is skipped. The edge emanating from the second occurrence of $\mathbf{V}_3$, namely, $\langle 3, 4\rangle$, is a candidate edge. Finally, $\langle 5, 6\rangle$ is a candidate edge. As illustrated, the minimum-angle edge is $\langle 5, 6\rangle$. I' is generated from I by replacing $\mathbf{V}_5$ by $\mathbf{V}_6$, the counterclockwise-most vertex of the edge, and the vertices are cycled so that $\mathbf{V}_6$ occurs first: $I' = \{6, 1, 3, 3\}$.

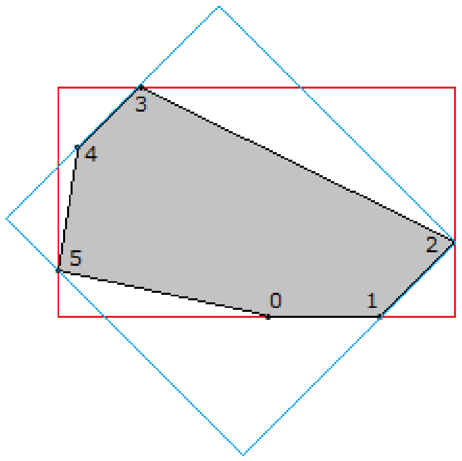
3.2.3 Multiple Polygon Edges Attain Minimum Angle

It is possible that the minimum angle is attained by two or more polygon edges. Figure 4 illustrates several possibilities.

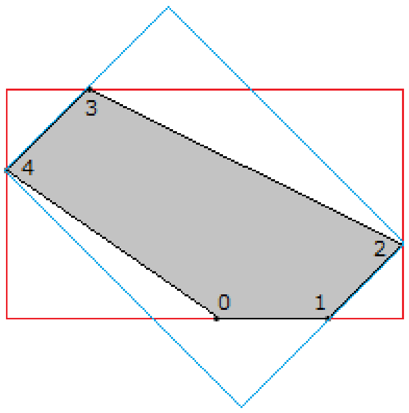
Figure 4. (a), (b), and (c) are configurations where two polygon edges attain the minimum angle. (d) is a configuration where three polygon edges attain the minimum angle. The discussion after this figure is about the various cases.



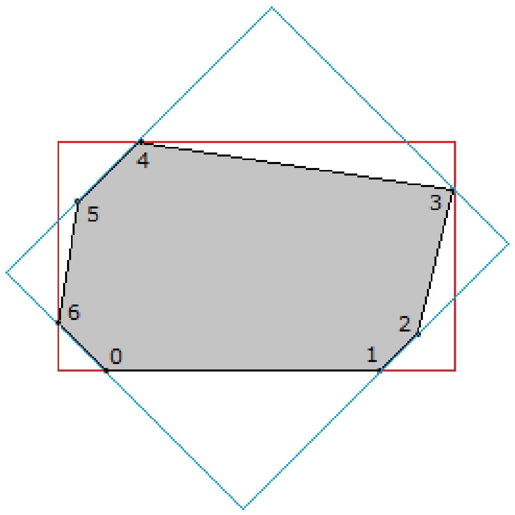
(a)



(b)



(c)



(d)

In Figure 4(a), the supporting indices for the current rectangle are $I = \{1, 3, 4, 6\}$. Two polygon edges attain the minimum rotation angle, $\langle 1, 2\rangle$ and $\langle 4, 5\rangle$. Choosing $\langle 1,_2 \rangle$ as the new bottom edge, the corresponding support is $\mathbf{V}_2$. $\langle 4, 5\rangle$ supports the new top edge and the support is $\mathbf{V}_5$. The new supporting indices are $I' = \{2, 3, 5, 6\}$. To obtain I' from I, the old support $\mathbf{V}_1$ is replaced by the other endpoint $\mathbf{V}_2$ and the old support $\mathbf{V}_4$ is replaced by the other endpoint $\mathbf{V}_5$. No cycling is necessary, because $\mathbf{V}_2$ is the bottom support and 2 is already the first element in I'.

In Figure 4(b), the supporting indices for the current rectangle are $I = \{1, 2, 3, 5\}$. Two polygon edges attain the minimum otation angle, $\langle 1, 2\rangle$ and $\langle 3, 4\rangle$. Choosing $\langle 1, 2\rangle$ as the new bottom edge, the corresponding support is $\mathbf{V}_2$. $\langle 3, 4\rangle$ supports the new top edge and the support is $\mathbf{V}_4$. The new supporting indices are $I' = \{2, 2, 4, 5\}$. Observe that $\mathbf{V}_2$ is a duplicate that supports two edges of the rectangle. To obtain I' from I, the old support $\mathbf{V}_1$ is replaced by the other endpoint $\mathbf{V}_2$ and the old support $\mathbf{V}_3$ is replaced by the other endpoint $\mathbf{V}_4$. No cycling is necessary, because $\mathbf{V}_2$ is the bottom support and 2 is already the first element in I'.

In Figure 4(c), the supporting indices for the current rectangle are $I = \{1, 2, 3, 4\}$. Two polygon edges attain the minimum rotation angle, $\langle 1, 2\rangle$ and $\langle 3, 4\rangle$. Choosing $\langle 1, 2\rangle$ as the new bottom edge, the corresponding support is $\mathbf{V}_2$. $\langle 3, 4\rangle$ supports the new top edge and the support is $\mathbf{V}_4$. The new supporting indices are $I' = \{2, 2, 4, 4\}$. Observe that $\mathbf{V}_2$ is a duplicate that supports two edges of the rectangle and $\mathbf{V}_4$ is a duplicate that supports the other two edges. To obtain I' from I, the old support $\mathbf{V}_1$ is replaced by the other endpoint $\mathbf{V}_2$ and the old support $\mathbf{V}_3$ is replaced by the other endpoint $\mathbf{V}_4$. No cycling is necessary, because $\mathbf{V}_2$ is the bottom support and 2 is already the first element in I'.

In Figure 4(d), the supporting indices for the current rectangle are $I = \{_1, 3, 4, 6\}$. Three polygon edges attain the minimum rotation angle; $\langle 1, 2\rangle$, $\langle 4, 5\rangle$, and $\langle 6, 0\rangle$. Choosing $\langle 1, 2\rangle$ as the new bottom edge, the corresponding support is $\mathbf{V}_2$. $\langle 4, 5\rangle$ supports the new top edge and the support is $\mathbf{V}_5$. $\langle 6,_0\rangle$ supports the new left edge and the support is $\mathbf{V}_0$. The new supporting indices are $I' = \{2, 3, 5, 0\}$. To obtain I' from I, the old support $\mathbf{V}_1$ is replaced by the other endpoint $\mathbf{V}_2$, the old support $\mathbf{V}_4$ is replaced by the other endpoint $\mathbf{V}_5$, and the old support $\mathbf{V}_6$ is replaced by the other endpoint $\mathbf{V}_0$.

3.2.4 The General Update Step

The current rectangle has bottom edge coincident with the polygon edge $\langle \mathbf{V}_{i_0-1}, \mathbf{V}_{i_0}\rangle$. The corresponding axis direction is $\mathbf{U}_0 = (\mathbf{V}_{i_0} - \mathbf{V}_{i_0-1})/|\mathbf{V}_{i_0} - \mathbf{V}_{i_0-1}|$. The perpendicular axis has direction $\mathbf{U}_1 = -\mathrm{Perp}(\mathbf{U}_0)$. The supporting indices are $I = \{i_0, i_1, i_2, i_3\}$. The next rectangle has bottom edge coincident with the polygon edge $\langle i'_0 - 1, i'_0\rangle$. The corresponding axis direction is $\mathbf{U}'_0 = (\mathbf{V}_{i'_0} - \mathbf{V}_{i'_0-1})/|\mathbf{V}_{i'_0} - \mathbf{V}_{i'_0-1}|$. The perpendicular axis has direction $\mathbf{U}'_1 = -\mathrm{Perp}(\mathbf{U}'_0)$. The supporting indices are $I' = \{i'_0, i'_1, i'_2, i'_3\}$. We need to determine I' from I.

Define the set $M \subseteq \{0, 1, 2, 3\}$ as follows: if $m \in M$, then $i_m \in I$ and the polygon edge $\langle i_m, i_m + 1\rangle$ forms the minimum angle with the rectangle edge supported by $\mathbf{V}_{i_m}$. The index addition $i_m + 1$ is computed modulo n. When M has multiple elements, the minimum angle is attained multiple times. In the example of Figure 1, $I = \{1, 3, 5, 7\}$ and $\langle 5, 6\rangle$ attains the minimum angle, so $M = \{2\}$. In the example of Figure 3(a), $I = \{1, 3, 3, 5\}$ and $\langle 5, 6\rangle$ attains the minimum angle, so $M = \{3\}$. In the examples of Figure 4(a,b,c), $M = \{0, 2\}$. In the example of Figure 4(d), $M = \{0, 2, 3\}$.

Start with an empty M. Process each $i_k \in I$ for which $i_{k+1} \neq i_k$, where the index addition $k+1$ is computed modulo 4. Compute the angle θ_k formed by $\langle i_k, i_k + 1\rangle$ and the supported rectangle edge, where the index addition $i_k + 1$ is computed modulo n. It is necessary that $\theta_k \geq 0$. If $\theta_k > 0$, store the pair (θ_k, k) in a set A sorted on the angle component. After processing all elements of I, if A is nonempty, then its first element is $(\theta_{k_{\min}}, k_{\min})$. Iterate over A and insert all the index components k into M for which $\theta_k = \theta_{k_{\min}}$. It is possible that A is empty, in which case the original polygon must already be a rectangle.

To generate I' from I, for each $m \in M$ replace i_m by $i_m + 1$, where the index addition is computed modulo

n. The resulting multiset of four elements is J. Let m_0 be the smallest element of M. The next rectangle is selected so that its bottom edge is supported by $\langle i_{m_0} - 1, i_{m_0} \rangle$; that is, $i'_0 = i_{m_0}$. Cycle the elements of J so that i_{m_0} occurs first, which produces the multiset I'.

4 A Robust Implementation

The correctness of the algorithm depends first on the input polygon being convex, even when the vertices have floating-point components. Experiments have shown that for moderately large n, generating n random angles in $[0, 2\pi)$, sorting them, and then generating vertices $(a \cos\theta, b \sin\theta)$ on an ellipse $(x/a)^2 + (y/b)^2 = 1$ can lead to a vertex set for which *not all* the generated vertices are part of the convex hull. Of course, this is a consequence of numerical rounding errors when computing the sine and cosine functions and the products by the ellipse extents. If a convex hull algorithm is not applied to these points, the exhaustive algorithm and the rotating calipers algorithm can produce slightly different results.

The correctness of the algorithm is also thwarted by the rounding errors that occur when normalizing the edges to obtain unit-length vectors. These errors are further propagated in the various algebraic operations used to compute the angles and areas.

To obtain a correct result, it is sufficient to use exact rational arithmetic in the implementation. However, normalizing a vector generally cannot be done exactly using rational arithmetic. The normalization can actually be omitted by making some algebraic observations. The minimum-area box can be computed exactly. If you need an oriented-box representation with unit-length axis directions, you can achieve this at the very end of the process, incurring floating-point rounding errors only in the conversion of rational values to floating-point values. These occur when computing the extents in the axis directions and when normalizing vectors to obtain unit-length directions.

4.1 Avoiding Normalization

Consider the computation of the smallest rectangle for a specified polygon edge $\langle j_0, j_1 \rangle$. Let $\mathbf{W}_0 = \mathbf{V}_{j_1} - \mathbf{V}_{j_0}$ and $\mathbf{W}_1 = -\operatorname{Perp}(\mathbf{W}_0)$, which are the unnormalized axis directions with $|\mathbf{W}_1| = |\mathbf{W}_0|$. The unit-length axis directions are $\mathbf{U}_i = \mathbf{W}_i / |\mathbf{W}_0|$ for $i = 0, 1$. Let the support vertices be $\mathbf{V}_b$, $\mathbf{V}_r$, $\mathbf{V}_t$, and $\mathbf{V}_\ell$ whose subscripts indicate the edges they support: bottom (b), right (r), top (t), and left (ℓ). The width w and height h of the rectangle are

$$w = \mathbf{U}_0 \cdot (\mathbf{V}_r - \mathbf{V}_\ell), \quad h = \mathbf{U}_1 \cdot (\mathbf{V}_t - \mathbf{V}_b) \tag{1}$$

and the area a is

$$a = wh = \frac{(\mathbf{W}_0 \cdot (\mathbf{V}_r - \mathbf{V}_\ell))(\mathbf{W}_1 \cdot (\mathbf{V}_t - \mathbf{V}_b))}{|\mathbf{W}_0|^2} \tag{2}$$

Both the numerator and denominator of the fraction for the area can be computed exactly using rational arithmetic when the input vertices have rational components.

The construction of the support vertices uses comparisons of dot products. For example, Listing 2 has a loop with comparisons for the current right-edge support. The comparison v[0] > support[1][0] is an implementation of

$$\mathbf{U}_0 \cdot (\mathbf{V}_i - \mathbf{V}_{i_1}) > \mathbf{U}_0 \cdot (\mathbf{V}_{r'} - \mathbf{V}_{i_1}) \tag{3}$$

where $\mathbf{V}_{i_1}$ is the origin of the coordinate system, r' is the index into the vertex array that corresponds to the current support vertex, and i is the index of the vertex under consideration. The Boolean value of the comparison is equivalent to that using unnormalized vectors,

$$\mathbf{W}_0 \cdot (\mathbf{V}_i - \mathbf{V}_{i_1}) > \mathbf{W}_0 \cdot (\mathbf{V}_{r'} - \mathbf{V}_{i_1}) \tag{4}$$

The expressions in the comparison can be computed exactly using rational arithmetic.

4.2 Indirect Comparisons of Angles

We must locate the edge emanating from a support vertex that points inside the rectangle and, of all such edges, forms the minimum angle with the rectangle edge corresponding to the support vertex. Let $\mathbf{D}$ be the unit-length rectangle edge direction (using counterclockwise ordering). That direction is $\mathbf{U}_0$ for the bottom edge, $\mathbf{U}_1$ for the right edge, $-\mathbf{U}_0$ for the top edge, and $-\mathbf{U}_1$ for the left edge. For an emanating edge originating at support vertex $\mathbf{V}_s$ and terminating at $\mathbf{V}_f$ inside the rectangle, the vector formed by the vertices is $\mathbf{E} = \mathbf{V}_f - \mathbf{V}_s$. The angle θ between $\mathbf{D}$ and $\mathbf{E}$ is determined by

$$\cos\theta = \frac{\mathbf{D} \cdot \mathbf{E}}{|\mathbf{E}|} \tag{5}$$

We can apply the inverse cosine function (acos) to extract the angle itself. $\mathbf{D}$ is obtained by normalizing one of $\mathbf{U}_0$ or $\mathbf{U}_1$, which we already know cannot generally be done exactly with rational arithmetic. Similarly, we must compute the length of $\mathbf{E}$, effectively another normalization. And the inverse cosine function cannot be computed exactly.

Observe that $\theta \in (0, \pi/2]$. The angle must be positive because the edge is directed toward the rectangle interior. If the angle attains the maximum $\pi/2$, then the support vertex must be a corner of the rectangle. Unless the polygon is already a rectangle, there must be at least one other bounding rectangle edge that has an emanating edge with angle smaller than $\pi/2$.

To avoid the inexact computations to obtain θ, we can instead compute the quantities $|\mathbf{U}_0|^2 \sin^2\theta$, which is a positive and increasing function of θ. Sorting of θ and $|\mathbf{U}_0|^2 \sin^2\theta$ lead to the same ordering. Using some algebra and trigonometry we may show that

$$|\mathbf{U}_0|^2 \sin^2\theta = \frac{(\mathbf{W}_i \cdot \mathrm{Perp}(\mathbf{E}))^2}{|\mathbf{E}|^2} \tag{6}$$

where $\mathbf{D} = \pm\mathbf{U}_i$ for some choice of sign and of index i. The fraction on the right-hand side of the equation can be computed exactly using rational arithmetic.

The quantities of equation (6) are computed for all emanating edges and stored in an array as they are encountered. We need to sort the values to identify those that attain the minimum value. Although any comparison-based sort will work, swapping of array elements has an associated cost when the values are arbitrary precision rational numbers. Our implementation uses an indirect sort by maintaining an array of integer indices into the array of values, and the integer indices are swapped as needed to produce a final integer array that represents the sorted values.

4.3 Updating the Support Information

This is a straightforward implementation of the ideas of Section 3.2.4. The only issue not mentioned yet is that as the rotating calipers algorithm progresses, the polygon edges are not visited sequentially in-order.

Moreover, if the polygon has pairs of parallel edge, it is possible that the minimum-area rectangle is computed without visiting all polygon edges.

In our implementation, we maintain an array of n Boolean values, one for each counterclockwise-most vertex of a polygon edge. The Boolean values are all initially false. After the smallest rectangle is computed for a polygon edge $\langle i_0, i_1 \rangle$, the Boolean value at index i_1 is set to true. When the support information is updated during each pass of the algorithm, the Boolean value is tested to see whether the edge has been visited a second time. If it has been visited before, the algorithm terminates (otherwise, we will have an infinite loop).

4.4 Conversion to a Floating-Point Rectangle

Our implementation of the rotating calipers algorithm tracks the minimum-area rectangle via the data structure

```
struct Rectangle
{
    Vector2<ComputeType> U[2];
    std::array<int, 4> index;  // order: bottom, right, top, left
    Real sqrLenU0;  // squared length of U[0]
    Real area;
};
```

This is slightly modified from that of Listing 2; the squared length of $\mathbf{U}_0$ is stored for use throughout the code. The ComputeType is a rational type.

The rectangle center and squared extents can be computed exactly from the minimum-area rectangle minRect,

```
Vector2<ComputeType> sum[2] =
{
    rvertices[minRect.index[1]] + rvertices[minRect.index[3]],
    rvertices[minRect.index[2]] + rvertices[minRect.index[0]]
};

Vector2<ComputeType> difference[2] =
{
    rvertices[minRect.index[1]] - rvertices[minRect.index[3]],
    rvertices[minRect.index[2]] - rvertices[minRect.index[0]]
};

Vector2<ComputeType> center = mHalf * (
    Dot(minRect.U[0], sum[0]) * minRect.U[0] +
    Dot(minRect.U[1], sum[1]) * minRect.U[1]) / minRect.sqrLenU0;

Vector2<ComputeType> sqrExtent;
for (int i = 0; i < 2; ++i)
{
    sqrExtent[i] = mHalf * Dot(minRect.U[i], difference[i]);
    sqrExtent[i] *= sqrExtent[i];
    sqrExtent[i] /= minRect.sqrLenU0;
}
```

The array name rvertices stores the rational representation of the floating-point vertices reference in previous parts of this document.

The conversion to the floating-point representation of the box has some loss of precision, but this occurs at the very end.

```
OrientedBox2<Real> itMinRect;  // input-type minimum-area rectangle;
for (int i = 0; i < 2; ++i)
{
    itMinRect.center[i] = (Real)center[i];
```

```
        itMinRect.extent[i] = sqrt((Real)sqrExtent[i]);

        // Before converting to floating-point, factor out the maximum
        // component using ComputeType to generate rational numbers in a
        // range that avoids loss of precision during the conversion and
        // normalization.
        Vector2<ComputeType> axis = minRect.U[i];
        ComputeType cmax = max(abs(axis[0]), abs(axis[1]));
        ComputeType invCMax = 1 / cmax;
        for (int j = 0; j < 2; ++j)
        {
            itMinRect.axis[i][j] = (Real)(axis[j] * invCMax);
        }
        Normalize(itMinRect.axis[i]);
    }

    // Quantities accessible through get-accessors in the public interface.
    supportIndices = minRect.index;
    area = (Real)minRect.area;
```

If you need to continue working with the exact rational data generated by the rotating calipers algorithm rather than the floating-point rectangle, our implementation allows you access to it.

The source code for the implementation is in the file GteMinimumAreaBox2.h.

References

[1] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. In *Communications of the ACM*, volume 18, pages 409–413, New York, NY, July 1975.

[2] Joseph O'Rourke. Finding minimal enclosing boxes. *International Journal of Computer & Information Sciences*, 14(3):183–199, 1985.

[3] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[4] Michael Ian Shamos. *Computational geometry*. PhD thesis, Yale University, 1978.

[5] Godfried Toussaint. Solving geometric problems with the rotating calipers. In *Proceedings of the IEEE*, Athens, Greece, 1983.