2D point-in-polygon test by classifying edges into layers

Wencheng Wang a, *, Jing Li a, b, Enhua Wu a, c

a Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
b Graduate School of Chinese Academy of Sciences, Beijing 100039, China
c Faculty of Science and Technology, University of Macau, Macao, China

Abstract

The 2D point-in-polygon test is a fundamental problem in geometry, and of importance in various applications in computer graphics and other areas. In taking advantage of the basic idea of the polygon scan conversion algorithm, a novel method for the point-in-polygon test is proposed in this paper, capable of handling simple polygons in arbitrary shapes, possibly with holes. In the preprocess of the method, the edges of the polygon are classified into layers according to the occlusion relations between the edges viewed orthogonally in a direction, called the test direction, which guarantees that the edges of a layer can be occluded by only the edges of its preceding layers. At the same time, the edges at each layer are queued up respectively along the direction vertical to the test direction because there is no occlusion relation between the edges of the same layer. As a result, based on the layers, the calculation of the segments by the line through the tested point to intersect the polygon along the test direction, and then the inclusion test of the point against the segments could be feasibly made. The method has a low storage requirement in $O(n)$, here, $n$ is the number of the edges of the polygon. The time complexity of its preprocess ranges from $O(n)$ to $O(n^2)$, depending on the polygon shape and the test direction. And its inclusion test has a time complexity between $O(\log(n))$ and $O(n)$, but less than $O((\log(n))^2)$ in most cases, depending on the construction of the layers. In the case of convex polygons and monotone polygons, the time complexity for the preprocess and the inclusion test could be $O(n)$ and $O(\log(n))$, respectively. On the other hand, the method is also easy to integrate with a variety of existing methods such as ray crossing methods and grid-based methods for improving the inclusion test further. Experimental results show that the method is robust and efficient in computation.

Keywords: Point containment; Layered edges; Polygon; Computational geometry

1. Introduction

Point location determination is one of the most elementary measures in computational geometry, and it has important applications in a variety of areas such as computer graphics and geographic information systems. In the 2D case, the problem is usually handled by performing a point-in-polygon test, conventionally called the inclusion test.

The point-in-polygon test problem has been widely studied, with a variety of efficient methods already proposed. These methods can be roughly classified into two categories. In the first category the methods are characterized by computing and evaluating certain parameters, including the ray crossing methods [1,2], the triangle-based methods [3,4] and the methods based on the sum of angles [5,6] or the sign of offset [7]. In the second category the methods work by decomposing the
A good survey of them can be found in [11]. In general, the methods in the first category are easy to implement while the methods in the second category are suitable for performing frequently repeated inclusion tests against the same polygon. Though some of these methods can perform the inclusion test against the convex polygon very fast, even in constant time $O(1)$ [14], the fastest method to our knowledge for concave polygons with holes is the trapezoidation-based method, which is of a time complexity $O(\log(n))$. However, the method requires much storage up to the complexity $O(n^2)$ to manage the subdivided trapezoidal primitives.

Today, models in applications grow rapidly in size in terms of polygons, leading to a large storage requirement. To improve the inclusion tests, it is desirable that the testing method could run fast and robust without too much storage. For this purpose, we propose a novel method based on the principle of the polygon scan conversion algorithm. It has a preprocess phase, in which the edges of the polygon are classified into layers according to the occlusion relations between the edges when the edges are viewed orthogonally along a direction called the test direction, and the edges at each layer are arranged in order respectively along the direction vertical to the test direction. Thus, based on the sequentially arranged layers, the novel method can fast compute the line segments inside the polygon by the line through the tested point and along the test direction, called the test line. Then, by checking the segments against the tested point, it is easy to determine if the point is inside the polygon. In comparison with the existing methods for the 2D inclusion test, the new method has the following three advantages:

1. Its storage requirement is very low, comparable with the ray crossing methods. This is superior to the subdivision-based methods, which ask for storage in a large quantity.
2. It can perform the inclusion test very fast by efficiently employing the binary search algorithm to calculate the segments for the test line intersecting the polygon.
3. It has no singularity to consider in the inclusion test. This is because the cases to cause singularities can be erased in constructing the layers. This is superior to many methods such as the ray crossing methods.

Generally speaking, the new method provides a simple solution for the inclusion test with low storage complexity of $O(n)$ immune from singularities. In most cases, it can run fast and robust in a time complexity less than $O((\log(n))^2)$, though its time complexity ranges from $O(\log(n))$ to $O(n)$ depending on the construction of the layers. Moreover, the method is also easy to integrate with various existing efficient methods, such as ray crossing methods and grid-based methods to improve the inclusion test further. Thus, the proposed method is suitable for handling complex polygons with a large quantity of edges.

In the remainder of the paper, the background is introduced in Section 2, the techniques for classifying edges into layers are described in Section 3, and the new inclusion test method is presented in Section 4. Then, the complexity of the method is analyzed in Section 5, and the experimental results are given and discussed in Section 6. Finally, the conclusion is drawn in Section 7.

2. Background

Given $n$ points $A_0, \ldots, A_{n-1}$ on a plane with $n$ line segments $l_0 = A_0A_1, l_1 = A_1A_2, \ldots, l_{n-1} = A_{n-1}A_0$ connecting the points, a simple polygon is defined if the neighboring line segments meet at only one common point, and non-neighboring line segments do not have any point in common (neither intersect nor touch).

The line segments are called the edges and the points where these line segments meet are called the vertices. Obviously, a simple polygon splits the plane exactly into two parts, the bounded interior and unbounded exterior.

A simple polygon may contain holes. If a polygon contains a finite number of holes, it would split the plane into more than two regions, but only one is unbounded. Using the terminology proposed in [15,16], a loop is a sequence of edges separating the bounded and unbounded parts of the plane, while a ring is a sequence of edges denoting the inside border of the polygon. A polygon has only one loop but may have multiple rings nested hierarchically.

According to the levels of the loop and the rings in the hierarchical structure from exterior to interior, the loop and the rings may be numbered sequentially starting from number 0. The regions bounded by the loop and the rings of the level $l_1$ are called the interior parts of the polygon, so called the regions bounded by the rings of the level in an even number and the rings of the next level. As for other regions, they are called the exterior parts of the polygon. As illustrated in Fig. 1, the loop and rings are numbered as $l_0$, $l_1$, or $l_2$ by their hierarchical levels, and the regions in gray are the interior parts of the polygon.

Thus, to determine if a point is inside a polygon is equivalent to test if the point resides within the interior parts of the polygon.

Without loss of generality, at the convenience in the following description, the $x$ and $y$ coordinates of any vertices are supposed to be all positive, and the $X$-axis is from left to right, the $Y$-axis from bottom up.
In the paper, the *occlusion* relation between edges is defined as follows. For any two edges, say $edge_1$ and $edge_2$, the occlusion relation between them is determined by the test direction. Without loss of generality, we choose the test direction being parallel to the $X$-axis. Thus, when the viewer stands at a position with its $x$ coordinate being negative, he/she views these two edges orthogonally along the test direction from left to right to decide the occlusion relation between these two edges. If the viewer may see the complete edges of both $edge_1$ and $edge_2$, there would be no occlusion relation between them; otherwise, an occlusion relation exists between them. If the viewer is unable to see part or full size of $edge_2$ due to an occlusion by $edge_1$, we say that $edge_1$ occludes $edge_2$, and vice versa, we may similarly define that $edge_2$ occludes $edge_1$.

### 3. Classifying edges into layers

By the occlusion relations between edges, as defined in Section 2, the edges of the polygon can be classified into layers, and in such a way, the edges of each layer are occluded only by the edges of the preceding layers. For the polygon as illustrated in Fig. 1, its edges will be classified into 10 layers, as illustrated in Fig. 2.

For fast constructing the layer structure and producing the segments correctly by the layers, the algorithm for classifying edges is in the following two steps:

1. The polygon edges are firstly split into *edges units* (units in short). Each unit contains a monotone chain of polygonal vertices as long as possible with their $y$ coordinates changing monotonously. The detailed description of this step will be given in Section 3.1.
2. Based on the units, the edges are classified into layers with all the edges of a unit included into a layer. For acceleration, some measures are adopted to reduce the computation for obtaining the occlusion relations and classifying edges. The detailed description will be given in Section 3.2.

In the following sections, we will first propose the techniques for processing the polygon type without holes, and then extend them to the polygon type with holes in Section 3.3.

#### 3.1. Edge units

The edge units of the polygon can be extracted by visiting the edges once in the following steps:

1. Find the vertex in the smallest $x$ coordinate, the leftmost one of the polygon.
2. From the vertex, the search goes upwards and downwards along the boundary consecutively to find a monotone chain of the vertices, which forms the first edge unit. As illustrated in Fig. 3, the search runs upwards and downwards from the leftmost
vertex B, and so the vertices C and A are obtained, respectively. As a result, it obtains the first unit, \( unit_1 \), of Edge AB and BC.

(3) From an end vertex of the last search for an edge unit, a further search may find a new unit. Such a search process continues until all units are extracted. As for the example in Fig. 3, from Vertex C, the unit of Edge CD, \( unit_2 \), is found, and from Vertex A, the unit of Edge AH and HG, \( unit_3 \), is obtained. Then from Vertex D and G, the unit of Edge DE, \( unit_4 \), and the unit of Edge GF, \( unit_5 \), are obtained, respectively. Finally, from E, the unit of Edge EF, \( unit_6 \), is obtained.

Preprocessing of units may facilitate the construction process of edge layers and save much time in comparison with direct construct of edge layers. When the polygon is simplified by taking each edge unit with an edge, the simplified polygon preserves the same topological structure as the polygon before the simplification. Thus, classifying edges into layers by the edge units will not impair the inclusion test, and due to the units in a smaller number than the edges, the construction of edge layers can be accelerated. At the same time, the unit construction may prevent from the singular cases that may occur in computing the segments.

From the investigation on the ray crossing method for the inclusion test [1,2], the singularities in forming the segments may occur in the two cases, the test line traversing through a vertex of the polygon, or the test line collinear with an edge of the polygon, called a singular edge.

For the first case, if the vertex is in the middle of an edge unit, it is sure to be the intersection point between the line and the edge layer because the line can intersect an edge layer at most one point. On the other hand, if the vertex is an end vertex of an edge unit, it must appear in two edge units and be the end vertices in these two edge units, which must be in different layers. Thus, the vertex can be taken as the two points for the line to intersect with the two edge units, respectively. Therefore, by the edge units, the line can automatically distinguish whether the vertex it intersects is one intersection point or two intersection points, for its traversal through the polygon. As illustrated in Fig. 3, Edge AB may be classified into the first layer and Edge BC into the third edge layer if the edges are directly classified without the edge units based on. Then, for the line through vertex B, three intersections with Edges AB, BC and AH will be produced, respectively. It would be difficult to form a segment by these three intersections. In fact, the two intersections on Edges AB and BC are the same vertex B, which should form a segment with the intersection on Edge AH. Fortunately, if the edges are firstly classified into units, Edges AB and BC will be classified in a same unit and classified into the third layer together because Edge BC is behind Edges EF and FG. Therefore, such a singular case will not occur.

For the second case of a singular edge, the whole edge can be regarded as an intersection point in some meaning. Of course, in processing, the edge itself will form a segment for the line to intersect the polygon, and one end of it will act as an intersection point to form another segment with another intersection point. Thus, for an edge unit having a singular edge, it only needs to record this edge as a singular edge and any one end of the edge will act as the intersection point between the line and the edge unit. This will cause no problem in forming the segments for the line to intersect the polygon, to be discussed in Section 4.1.

3.2. Classifying edges by units

After the edge units are produced, there will be two tasks in constructing layers, classification of the edges into layers and sequential arrangement of the edges at each layer. Since the latter task is easy to perform, we will mainly discuss the techniques for classifying the edges into layers. To finish this task, it is necessary to first obtain the occlusion relations between units and then use the occlusion relations for classifying units into layers one by one in order from left to right.

To calculate the occlusion relationship between any two units, it is first to check if a range overlap happens between the \( y \) coordinates of the two units. If no overlap happens, there would be no occlusion relation between the units. Otherwise, a line with its \( y \) coordinate in the overlapped range is used to intersect these two units to determine which unit occludes the other according to the \( x \) coordinates of the intersection points.

Based on the obtained occlusion relation between units, the units can be classified into the layers in the following steps:

(1) The edge units without occluded by any other edge units are extracted to form the first edge layer.
(2) For the edge units classified into the first edge layer, their related occlusion relations are removed.
(3) Among the left edge units, select the edge units without occluded by any other edge units to form the second edge layer. Afterwards, their related occlusion relations are removed.
(4) Iteratively perform the operation in step (3) until no edge unit is left. As a result, the layers of the edges are constructed in an order left to right.

To improve the time consuming process in constructing the edge layers, we adopt two acceleration measures. The first measure is to take a separate-and-combine strategy for classifying edges. By this strategy, the edge units are separated into two sets, called an odd set and an even set. Then, the units of the odd set and the even set are firstly classified into layers, respectively, and then the layers are combined to get the final edge layers. The second measure is to take an economic strategy for computing the occlusion relations. By this strategy, some useless occlusion relations will not be computed and some occlusion relations between the neighboring units can be derived without computation. These two strategies will be discussed in detail in the following two subsections, respectively.

3.2.1. Separate-and-combine strategy for classifying edges
From the process in producing the edge units, we know that two edge units are in neighbor if they share a common vertex, and an occlusion relation must exist between them; otherwise, both should be in a single unit. Based on this reason, the edge units can be separated into two sets with one as the odd set and another as the even set. The separation can be operated as follows. The unit selected first is placed into the odd set, and those in neighbor to a unit in the odd set are placed in the even set. Similarly, the units in neighbor to a unit in the even set are placed in the odd set. Obviously, such a separation can be easy to do in the process of extracting the units along the boundary of the polygon. In the following, the units in the odd set are called the odd units, and the units in the even set are called the even units. As we may see from the example in Fig. 3, the odd set consists of unit₁, unit₃ and unit₅, marked in red circles, and the even set consists of unit₂, unit₄ and unit₆ in blue rhombs.

Based on the principle of the polygon scan conversion algorithm, we know that if an even unit occludes an odd unit, there must exist another odd unit occluding the odd unit. Similarly, if an even unit occludes an odd unit, the even unit must also occlude another even unit. Therefore, the edge units in the odd set and the even set can firstly have their own layers constructed, respectively, and then their layers are combined to get the final edge layers by alternatively taking a layer from the odd set and a layer from the even set sequentially from left to right. For example, suppose the edge units of the odd set are layered as odd₁, odd₂, ..., and the edge units of the even set are layered as even₁, even₂, ..., then the final layers will be obtained as odd₁, even₁, odd₂, even₂, .... Obviously, by this measure, the computation for constructing layers can be substantially reduced.

3.2.2. Economic strategy for computing occlusion relations
From the process in searching the edge units, we know that if two edge units have a common vertex, there must be an occlusion relation between them. We call this relation a local occlusion relation, and the relation can be detected by the two edges sharing the common vertex. Thus, if two odd units, say UA and UB, are in neighbor to an even unit, say UC, the occlusion relation between UA and UB can be derived in the two cases that UA occludes UC and UC occludes UB, or UB occludes UC and UC occludes UA. The result is also valid for the even units.

In this way, much computation can be saved for obtaining the occlusion relations among the edge units. Moreover, in this way, the occlusion relation between an odd unit and an even unit can be brought to play in classifying edges of the odd set and the even set, respectively. In some meaning, though the units of the odd set and the units of the even set are classified into layers, respectively, the occlusion relations between any two units can be used for classification. Thus, the edges can be classified into layers correctly. As illustrated in Fig. 4, when only the occlusion relations between the units of the even set are considered, the units of AB and CD in the even set may be placed in a same layer. Obviously, this is wrong. Fortunately, after it is derived the occlusion relation between AB and CD by their

Fig. 4. The occlusion relation between Edges AB and CD must be derived by Edge BC before they are layered in the even set. Otherwise, in layering the edges of the even set, Edges AB and CD will be in a same edge layer. Obviously, This is wrong.
neighboring units BC, AB and CD will be classified into different layers. Therefore, it must derive the occlusion relations as much as possible by the local occlusion relations before classifying edges into layers.

As we know, some occlusion relations have no use in classifying edges into layers. To save the computation for the useless occlusion relations, we let the occlusion relations be computed on demand. In other words, in forming an edge layer by the left edge units, we first select the edge units that are only occluded by the edge units classified into the preceding layers, and then detect the occlusion relations among the selected edge units. Afterwards, by the newly obtained occlusion relations, only the edge units without occluded are extracted to form a new layer.

### 3.2.3. Construction of layers in classes

After the edges are classified into layers, the layers can be numbered sequentially from left to right starting from number 1 as layer$_1$, layer$_2$, layer$_3$, layer$_4$,...,layer$_{2i-1}$, layer$_{2i}$,... Furthermore, we can manage the layers in pairs, called classes, and the classes are queued up by the sequence of the layers from left to right. This means class$_i$ consists of layer$_{2i-1}$ and layer$_{2i}$, $i = 1, 2, ...$. This will benefit the binary search algorithm in computing the segments, to be discussed in Section 4. By the principle of the polygon scan conversion algorithm, the number of layers must be even, and each segment must be generated from the intersection point on an odd edge layer to that on an even edge layer behind.

### 3.3. Handling polygons with holes

In the previous subsections, we have discussed the techniques for classifying edges of polygons without holes. By a simple extension, these techniques can be applied for classifying edges of polygons containing holes. The extension could be made in the following steps:

1. At first, the loop and rings of the polygon have their own layers constructed, respectively, by regarding each as a polygon without holes. We call these layers the local edge layers.
2. Then, all the local edge layers are combined to construct the edge layers of the whole polygon by the hierarchical tree of the loop and rings, which is discussed in Section 2.

Without loss of generality, we explain the combination technique based on the example as illustrated in Fig. 1. Obviously, before classifying the edges into layers, the hierarchical tree of the loop and rings should be constructed, as illustrated in Fig. 5. Then, the local edge layers can be all regarded as the edge units defined in Section 3.1 and classified into layers by the techniques described in Section 3.2. This can be easily derived by the theory of the polygon scan conversion algorithm.

 Naturally, all the local edge layers of the loop and the rings will also be separated into an odd set and an even set for accelerating the construct of layers. In separating local edge layers into the odd set or the even set, not only the local sequential number of the edge layer in its corresponding ring should be considered, but also the level of the ring in the hierarchical tree should be considered. Therefore, by a simple derivation, the separation should be done as follows. The odd layers of the rings that are at an odd number level should be classified into the odd set, and so do for the even layers of the rings that are at an even number level. As for the even layers of the rings that are at an odd number level, they should be classified into the even set, and so do for the odd layers of the rings that are at an even number level.

As illustrated by the example in Fig. 1, Polygon QRST has its layers constructed as 2layer1(TQ, QR) and 2layer2(RS, ST), and Polygon HIJKL as 1layer1(HI, IJ, JK) and 1layer2(KL, LH). Here, the layer title “2layer1(TQ, QR)” means that the ring is at the level2 and the layer is its first layer covering Edge TQ and QR. To combine the local edge layers of these two polygons, 2layer1(TQ, QR) and 1layer2(KL, LH) are collected in a set for classifying edges into layers, and 2layer2(RS, ST) and 1layer1(HI, IJ, JK) in another set for classifying edges. Finally, the edge layers are combined together in sequence from left to right as 1layer1(HI, IJ, JK), 2layer1(TQ, QR), 2layer2(RS, ST) and 1layer2(KL, LH). For the whole polygon in Fig. 1, the edges will be classified into 10 layers at last, and further organized in classes, as illustrated in Fig. 2.

In some cases, the rings and the loop may have shared edges. These cases could be also handled by the techniques discussed in this subsection except that the shared edges should be repeatedly classified into the local edge layers of the rings. As illustrated by the example in Fig. 6, the shared edge AB is first classified.
into the local edge layers of the loop and the ring repeatedly, and then the local edge layers are combined to obtain the final edge layers of the polygon.

4. Inclusion test

Based on the sequentially arranged edge layers, the point-in-polygon determination could be made in high efficiency. For any given point tested, say $P$ with the coordinates $(P_x, P_y)$, the inclusion test starts by forming a test line through $P$ that is $\{y = P_y\}$. With line segments produced for the line to intersect the polygon and compared with the $x$ coordinate of $P$, $P_x$, it is easy to determine whether the point is inside the polygon.

As we know that the line may intersect the polygon in multiple segments, but only the segments in neighbor to the tested point $P$ are useful for the inclusion test. Thus, based on the sequentially managed edge layers, the binary search algorithm is used to fast compute the useful segments.

Without loss of generality, suppose there are $n$ classes for a polygon, and an intermediate class in the sequential list is class$_i$. Then the segments are produced starting from the class$_i$. For the two layers in class$_i$, layer$_{2i-1}$ and layer$_{2i}$, the $x$ coordinates of the two points for the line to intersect with these two layers, say $x_{2i-1}$ and $x_{2i}$, are calculated, respectively, supposed that the intersection points on both of these two layers can be found. In fact, there may be no edge found in a layer to intersect with the line, which will be discussed later in Section 4.1. Obviously, the segment for the line to intersect the class$_i$ is $[x_{2i-1}, x_{2i}]$. If $P_x$ is inside $[x_{2i-1}, x_{2i}]$, the point is inside the polygon and the inclusion test may terminate. Otherwise, depending on $P_x < x_{2i-1}$, or $P_x > x_{2i}$, a class in front of or behind the class$_i$ will be selected for a next round of check by the binary search algorithm. Such a process continues until $P_x$ is within a segment, or no segment calculated includes $P_x$ with all the classes selected for check, indicating that the point $P$ is outside the polygon.

In calculating the intersection point between the line and an edge layer, the binary search algorithm can be also used to fast find the edge to intersect the test line, because the edges in a layer are also sequentially arranged by their $y$ coordinates.

4.1. Calculation of segments

To produce the line segments, it is necessary to obtain the intersection points between the line and the edge layers. However, in processing the two layers of a class class$_i$, layer$_{2i-1}$ and layer$_{2n}$, the intersection points with the line could have three cases: both layers, one layer or no layer is intersected.

In the first case, the two intersection points produced could form a segment to check against $P_x$, as discussed previously in this section. Here, a special case is that the line is collinear with a singular edge. In this case, the singular edge will form a segment itself, and either end of the edge could be the intersection point between the line and the edge layer, which has no impairment to the inclusion test by a simple derivation.

In the second case, the intersection point can be an end of the segment to form. To form the segment, it needs to calculate the other end of the segment. From the discussion in Section 3, one segment must be from an intersection point on an odd layer to the other one on an even layer behind the odd layer, and every segment is fully inside the polygon. Therefore, if the intersection point obtained is on the layer$_{2i-1}$, we should search the corresponding intersection point on an even layer in the classes following the class class$_i$ one by one until the intersection point is found. Similarly, if the intersection obtained is on the layer$_{2n}$, we should search the corresponding intersection on an odd layer in front of the class class$_i$ one by one.

In the third case, because there is no information to guide the search for the intersection points, the user can search forward or backward gradually from class$_i$ to obtain the intersection points. By the principle to form a segment, we can first search only the odd layers in front
of class, or the even layers behind class, to find an intersection point, and then find its corresponding intersection point using the measures discussed about the second case. In this way, much computation can be saved and no problem may occur.

4.2. Calculation of intersection points

To calculate the $x$ coordinate of the intersection point between the test line and an edge, it always needs an interpolation, including of at least a multiplication and an addition. As the intersection points may be in a large quantity, such interpolation could be computationally intensive. For saving the computation, a measure is adopted here. In fact, to check whether the $P_x$ of the tested point is inside the $x$ coordinate range $[x_{\text{min}}, x_{\text{max}}]$ of the segment, we only need to know whether $P_x$ is bigger than $x_{\text{min}}$ and smaller than $x_{\text{max}}$, and it is unnecessary to know the precise values of $x_{\text{min}}$ and $x_{\text{max}}$. Thus, for an edge in intersecting the line with the $x$ coordinate, its two ends being $x_1$ and $x_2$, if both $x_1$ and $x_2$ are bigger or smaller than $P_x$, the $x$ coordinate of the intersection point must be bigger or smaller than $P_x$, respectively. By this check, it may reduce such interpolation computation substantially.

5. Algorithm analysis

In this section, we analyze the preprocess for classifying edges into layers, the storage requirement and the inclusion test of the proposed method, respectively.

The preprocess includes three main steps, extracting edge units, classifying edges by the units and sorting the edges of each layer. Extracting edge units can be processed in $O(n)$ since each edge should be checked once, here $n$ is the number of edges in the polygon. As for classifying edges based on the edge units, it is not easy to derive the accurate time complexity in this step, because the occlusion relations between edge units may vary greatly for different polygons, especially for the polygon containing holes. In the worst case, however, it is required to calculate the occlusion relations between any pair of edge units and no local occlusion relations could be used, where the complexity could be $O(n^2)$. As illustrated by the example in Fig. 7, when the edge units of the even set are layered separately, the worst case may occur. Fortunately, even in such a case, the computation is far from the upper bound of the complexity because the units of the odd set or the even set are fewer than the units of the polygon, respectively, though the complexity is high. In general, though the upper bound of the complexity is $O(n^2)$, the preprocess may be fast in most cases because the local occlusion relations can be employed for acceleration and the edges are firstly classified into layers in the odd set and the even set, respectively. With regard to sorting the edges of each layer, the work can be done at the same time in selecting the units to form the layers since the comparison between the edge units of a layer must be performed in forming the layer. In sum on the whole, the preprocess has the time complexity ranging from $O(n)$ to $O(n^2)$, depending on the shape of the polygon and the test direction. As a result, if the polygon is convex or monotone type where two edge units are extracted, the complexity for classifying edges is only $O(n)$.

The storage requirement for the layered edges is very low. In the general representation of a polygon, the vertices of the polygon are recorded in sequence with each vertex expressed by its $x$ and $y$ coordinates. In the new method, most vertices are in a layer except those shared by an odd unit and an even unit. For each vertex in a layer, its corresponding node records its $y$ coordinate, $Y_v$, and its $x$ increment, $incX_v$, which are used to calculate the intersection between the edge and the test line through the tested point $(X_p, Y_p)$ by the equation $Y = Y_v + (Y_p - Y_v) \cdot incX_v$, where $incX_v$ is calculated by the linear equation of the related edge. For the vertices shared by an odd unit and an even unit, one half among them will have their $y$ coordinates and $x$ increments recorded, respectively, and another half will only record their $y$ coordinates. This is because it is unnecessary to record both of the top vertex’s $x$ increment and the bottom vertex’s $x$ increment for an edge unit, which has two end vertices. Therefore, even the vertex shared by an odd unit and an even unit would appear twice in the layers, the storage required for such a vertex is about one-and-a-half times as large as that by recording both $x$ and $y$ coordinates of a vertex in the conventional representation. With regard to the layers, they are managed in registers. Because the number of layers is smaller than the number of edge units, the number of the registers will be smaller than the number of edge units also. Therefore, we may conclude that the storage complexity of the new method is $O(n)$.

In the inclusion test, the calculation for the test line to intersect the layers is comprehensive and the number of edges in different layers may vary greatly. Thus, it is

![Fig. 7. When the edges of the odd set and the even set are layered, respectively, no local occlusion relations between neighbor edge units can be employed for the even set so that the complexity may be $O(n^2)$ for layering the edges in this set.](image-url)
difficult to give a precise complexity evaluation. However, we may give a coarse estimation as regarded. For most cases, it can be assumed that the \( n \) edges of the polygon are classified on average into \( k \) layers where \( k \) is within \( 2 \) to \( n/C_0^{1/1} \); and the line has an intersection with each layer. When the binary search algorithm is employed, the complexity will be \( \log(k)^*(\log(n)−\log(k)) \), and in \( O((\log(n))^2) \) in the worst case where \( k = (\log(n))/2 \). In the extreme situation that half of \( k \) layers have to be checked, like the third case in Section 4.1, the complexity may be \( O(n) \) when \( k = 2^{(\log(n)−1)/\log(2)} \). Obviously, if there are only two layers for the polygon, such as in the case of the convex polygon or monotone polygon, the complexity can be reduced to \( O(\log(n)) \).

From the discussion, we may also see that many edges could be still avoided to check even in the worst case in the complexity \( O(n) \).

6. Experimental results and discussion

To test the efficiency of the new method, we implemented the algorithm in MS Visual C++6.0, and in comparison, the method in [4] for the 2D inclusion test without singularity. Meanwhile, we also downloaded the program of the ray crossing method, a popular method for the inclusion test, from the web site www.ecse.rpi.edu/Homepages/wrf/geom/pnpoly.html.

All the tests were performed on a personal computer installed with an Intel PIV 2.4G CPU, 512M RAM and the operation system Windows XP.

At first, we made a test using the polygons without holes. By the efficiency for the binary search algorithm to run, the polygons are classified into five cases to test, the worst, the worse, the normal, the better and the best. Five polygons listed in Table 1 are corresponding to these five cases, respectively, where all are in 500 edges. The method may search the intersection points in the time complexity near \( O(n) \) in the worst case, and \( O(\log(n)) \) in the best case. With regard to other cases, the time complexity varies from \( O(\log(n)) \) to \( O(n) \).

In the experimental test for each case, we made test to the polygons in 23 different numbers of edges, respectively, in 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000 and 5000. Then, to the 19 smaller numbers of the edge types, 10 polygons for each were produced arbitrarily with a given number of edges, and to the four larger numbers of the edge types, three polygons for each were produced arbitrarily with a given number of edges. For every polygon of the preceding 19 types, 100 points were evenly sampled in the bounding box of the polygon for the inclusion test. And for each polygon in the later four types, 200 points were sampled.

The statistics for the preprocess time of the new method are as listed in Table 2, where the time is averaged from all the polygons in a given number of edges. And the statistics for the inclusion test are as listed in Table 3, where \( N \) represents the number of edges of the polygon, and the processing time is averaged from all the points against the polygons in a given number of edges.

For the polygon types containing holes, we compared the new method with only the ray crossing method. As was done for testing the polygons without holes, the polygons were classified into three cases, the worst, the normal and the best. For every case, the polygons are all with seven holes, and one example for each case is as shown in Table 4.

Similarly, in the experimental test for each case, we made test to the polygons produced in 14 different numbers of the edges, respectively, in 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000 and 5000. Then, for each number in the preceding nine numbers, four polygons each were produced arbitrarily in the same number of edges in the loop, and for each one of the later five numbers, two polygons each were produced arbitrarily in the same number of edges in the loop. For each polygon, seven rings were produced arbitrarily with the total number of their edges almost equal to the

<table>
<thead>
<tr>
<th>Worst</th>
<th>Worse</th>
<th>Normal</th>
<th>Better</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Worst" /></td>
<td><img src="image2.png" alt="Worse" /></td>
<td><img src="image3.png" alt="Normal" /></td>
<td><img src="image4.png" alt="Better" /></td>
<td><img src="image5.png" alt="Best" /></td>
</tr>
</tbody>
</table>

Table 1
Five polygons corresponding to the five cases in test.
number of the edges in the loop. For every polygon, 500
points were evenly sampled in the bounding box of the
polygon for the inclusion test.

In Table 5, the statistics of the preprocess time are
given for the new method in processing the polygons
with holes where the time is averaged from the polygons
with a given number of edges in the loop. And the
statistics for the inclusion test are as listed in Table 6
where \(N\) represents the number of the edges in the loop,
and the processing time is the averaged time from the
inclusion tests to all the points against the polygons with
a given number of edges in the loop.

The experimental results show that the new method
can run much faster in lower time complexities than the
other two methods in most cases except in the worst
case. Even in the worst case, the three methods are in
the same complexity \(O(n)\), but the new method could still
run faster since it could handle less edges than the other
two.

The new method can be easily integrated with various
existing methods such as the ray crossing methods and
the grid-based methods. For example, a ray crossing
method can take advantage of the edge layers proposed
in our method to fast compute the intersection points.
With regard to a grid-based method, the edges inside
every grid may take our approach to construct local
layers of edges for accelerating the inclusion test.

### 7. Conclusions

The paper presents a simple and robust method for
the point-in-polygon determination, which does not
require computing singularities. By the method, the
edges are classified into layers in preprocess, and then
the binary search algorithm is used on the layers to fast
calculate the line segments for the line through the test
point to intersect the polygon. As a result, the inclusion
test can be performed on a 1D line by checking the test
point against the segments in high efficiency. The
method can handle polygons without or with holes in
a consistent way. For its preprocess, it can be performed
in a complexity between \(O(n)\) and \(O(n^2)\), and in the best
\(O(n)\) in cases of the convex and monotone polygons. For
its inclusion test, the time complexity ranges from
\(O(\log(n))\) in the case of the convex or monotone
polygon, up to \(O(n)\) in the worst case, and lower than
\(O((\log(n))^2)\) in most cases. Even in the worst case, the
acceleration measures could still be employed to save
computation. An additional advantage of the proposed
method is its low storage requirement in \(O(n)\), and in
comparison with the conventional way of representing
polygons by the coordinates of vertices, the new method
only demands a little extra storage.

As future work, it is interesting to extend the
proposed method to perform 3D point-in-polyhedron
Table 3
Statistics for the inclusion test of the three methods

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Worst</strong></td>
<td><img src="image1.png" alt="Graph" /></td>
<td><img src="image2.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Worse</strong></td>
<td><img src="image3.png" alt="Graph" /></td>
<td><img src="image4.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Normal</strong></td>
<td><img src="image5.png" alt="Graph" /></td>
<td><img src="image6.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Better</strong></td>
<td><img src="image7.png" alt="Graph" /></td>
<td><img src="image8.png" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Best</strong></td>
<td><img src="image9.png" alt="Graph" /></td>
<td><img src="image10.png" alt="Graph" /></td>
</tr>
</tbody>
</table>
In principle, the facets of a polyhedron can be classified into layers so that the segments for the line through the test point to intersect the polyhedron can be calculated efficiently based on the layers. Another possible extension is to try to integrate the method with other existing methods. Typically, the layer structure of
edges in the proposed method could be utilized to improve the efficiency of ray crossing methods and grid-based methods, as discussed in Section 6.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (60373051) and the National Grand Fundamental Research 973 Program in China (2002CB312102). The program of the ray crossing method is downloaded from www.ecse.rpi.edu/Homepages/wrf/geom/pnpoly.html.

References