Optimal Convex Decomposition Algorithm for Simple Polygons

Ali Sajedi¹, Minoo Monajjemi nejad²
¹² Software Research and Development Laboratory, Computer Engineering Department, Azad University of Lahijan, Lahijan, Iran
¹Sajedi@iau-lahijan.ac.ir
²Minoo_libertarian2@yahoo.com

Abstract

In this paper we present four algorithms; a convex decomposition algorithm, optimal convex decomposition algorithm (two versions; with recursive and iterative manner) and optimal convex decomposition with combinational manner. The main part of the last algorithm works iteratively, but it uses a recursive function to produce the needed states. The first algorithm is based on common search methods and if it finds an answer it'll be stopped (even if it's not optimal). The second and third algorithms define a cost function and produce the best answer based on that. The fourth algorithm tries to reduce the space of problem as far as possible; also it finds the whole optimal answers.

Keywords

Algorithm, polygon, polygon decomposition, optimal convex decomposition, Computational Geometry.

1- Introduction

Polygon decomposition is one of the important concepts in computational geometry which decomposes polygon objects to simpler polygons such as: triangle, rectangular, square, convex polygons, star shape, spiral, etc. Image processing, VLSI design of computer circuits and visibility problems are examples of polygon decomposition applications. For example in the case of VLSI design, it's necessary to decompose a circuit into a few (the least) rectangular shape or L shape. It's due to partitioning current data into a few groups and putting each group in one of decomposed shapes. These kinds of problems are NP-Complete problems and the action of decomposition expedites reaching a proper answer [2, 3, 7].
Another application is related to image processing. For example in triangulation, we decompose a polygon (the first shape) into a few triangles in a way that drawn lines which pass through the angles of the first polygon only cut each other in angles \([1,3,5,6]\).

There are two types of decomposition: decomposing and covering. In decomposition, the first polygon that might be concave or convex decomposes into a few shapes with limitations. These shapes are in neighborhood of each other and they don't have any overlap with each other, but in covering overlap is possible in obtained shapes[4].

In this paper, we discuss about decomposing concave polygons into convex polygons, specially the optimal convex decomposition. Optimal convex decomposition of polygon \(p\) is reaching separate convex polygons which obtain by drawing the least number of \(p\) diagonals; however, convex decomposition with using inside points is also common which possibly produces answers with fewer number of polygons but its application is less than the first one (Fig.1, 2).

2- Problem Definition

Consider \(n\) points \(A_1\) to \(A_n\). \(A_1A_2, A_2A_3, \ldots, A_{n-1}A_n\), \(A_nA_1\) forms the series of edges (the edges can be intersecting). Simple polygon restricts this definition. In simple polygons except intersection of successive edges in points \(A_1\) to \(A_n\), there isn't any other intersection between their edges. We want to divide the simple concave polygon to some convex polygons.

For this purpose with having all \(p\) diagonals we execute depth first search algorithm; we draw passing diagonals of concave vertexes one by one while there is no intersection between diagonals, until the first answer obtains and the algorithm stops. Also we apply some optimizations in this part. The second problem is obtaining the best answer (answer with fewer diagonals) which includes the whole answers and doing comparison between them. The nature of this algorithm is recursive, but we present an iterative algorithm by a fascinated mapping as third algorithm. In the last algorithm, the search order is changed so that the least cases would be considered to reach the best answer.

This algorithm works iteratively, but cooperates with a recursive algorithm.

3- The Algorithms

3-1- Convex Decomposition Algorithm

For obtaining convex decomposition of \(P\) as it is said before we number the vertices from \(A_1\) to \(A_n\) in clockwise order (Fig. 3).

Decomposition algorithm uses the law that in convex decomposition of \(p\) we should draw diagonals from concave vertexes. For this purpose we arrange all the diagonals which pass from each concave vertex of \(p\) (without repetition) in all diagonals list (\(D_1\) to \(D_m\); \(m\) passing diagonals from concave vertexes of \(P\)). These diagonals are shown for two sample polygons in Fig.4.

Algorithm 1.

Input:
- List of concave vertexes of \(P\) (CcV).
- List of all Diagonals passing from concave vertexes of \(P\). (D)

Output:
- Selected diagonals in List D that will be drawn to form a convex Decomposition OF P.

{ 
While (! convex Decomposition (P, CcV, D))
}
```c
{ 
  i ← select (D, CcV); 
  While (Intersection(D, D)) 
  { 
    D[i].intersection ← true; 
    i ← select(D, CcV); 
  } 
  D[i].Drawed ← true; 
}
```

3-1-1- Description of variables and functions
-p: The first concave polygon as a list containing ordered points from 1 to N, each of which has X and Y.
-CcV (Concave Vertexes): List of concave vertexes of P.
-D: list of all passing diagonals from concave vertexes of P. Each member of this list includes a structure which contains the number of vertexes from its two sides (v1, v2) and two Boolean fields for keeping the situation of being drawn (chosen) of each diagonal in drawing field and also keeping the intersection of the diagonal in intersection field.
-convex_decomposition(p, Ccv, D): Having the first polygon (p), convex vertexes list (Ccv) and the situation of being or not being chosen of the diagonals of the diagonals (D), examines whether convex decomposition is done or not and returns true or false according to that.
-select (D, CcV): Returns the first diagonal of D which its intersection and drawing fields are false, for improving the algorithm, we can do this selection in another way (e.g. random selection) and avoid producing neighbor triangles as far as possible.
-Intersection (D, D): returns true if the i'th diagonal doesn't have any intersection with the other drawn diagonals otherwise returns false.

3-1-2- Optimizations
this algorithm is blind and draws diagonals one by one until it finds an answer. As it is said before, the first thing that can be done for improvement is the select function, so that as far as possible we choose a diagonal which connects two concave vertexes to each other. This action causes to break two vertexes during drawing a diagonal and probably causes to decrease the number of total selected diagonals. Another optimization is done at the end of the procedure. At the end, the drawn diagonals are in list D with marked some drawing fields. Traverse D to delete some of diagonals between adjacent convex polygons. If deleting each of diagonals doesn't lead to a concave angle, deletion of that diagonal will be confirmed. An ordinary answer (A), a middle answer with applying the first optimization and an answer with applying both the optimizations are shown in Fig. 5, 6 and 7.

3-2- optimal convex decomposition algorithm
The algorithm above finds an answer, but about this answer even with applying both optimizations a lot of diagonals are remained such as polygons which cannot be integrated to each other (make concave angle) but with separating some parts of them and combining them together maybe we can reduce the number of polygons. This action requires a kind of intelligence that the algorithm doesn't have .due to this reason for achieving optimal answer we have to examine all possible answers. For this purpose just like the previous algorithm but using backtracking method we examine diagonals of list D. For each of diagonals we can consider two states whether they took part in optimal decomposition or not. For each diagonal we do the procedure once with selection and the other time without it. Until we get to the last diagonal also the answer. When reaching each answer, we compare its cost which actually is the number of drawn diagonals or the number of convex polygons with the answer which has the best cost till that time and if the answer is better the cost of it will be saved. In this way all the possible conditions for all diagonals (m diagonals; 2 states for each diagonal, 2^m for the combination) will be examined.
3-2-1- recursive algorithm

For this purpose due to recursive nature it is written as recursive pseudo code.

Algorithm 2.
Procedure Decompose (D,i) {
  if (Convex_Decomposition (P, CcV, D) ) {
    cost<calculate_cost(D);
    if (cost<best_cost) {
      Best_Decomposition<Di;
      Best_cost<cost;
    }
  } else {
    if ( ! Intersection(Di , D) {
      D_i . Drawed<true;
      Decompose(D, i+1);
    } D_i . Drawed<false;
    Decompose(D, i+1);
  }
  out(Best_Decomposition);
}

Description of variables and functions

For this algorithm the following variables are used:
-i: This variable indicates the number of the diagonal that is being checked. In the first procedure call, it is called with 1.
-D: contains the status of diagonals in each moment.
-cost: cost of decomposition is the number of drawn diagonals to obtain the answer. Of course for more efficiency we can use heuristic methods for cost definition which are not considered here.
calculate_cost(D): considers the status of diagonals (D) as the input parameter and returns the cost which is the number of drawn diagonals.
-best_cost: The number of diagonals in the best obtained status in each moment of executing the algorithm.
-Best_Decomposition: The best convex decomposition that has been obtained till that moment (containing the status of diagonals).
-out: Output function that shows the results.

Explanation: The procedure process is recursive (Porce Decompose(D,i)).

For each diagonal one of the decisions are done: being selected or not and the procedure continues till one of the below conditions obtains:
- Convex decomposition of P; in this situation comparison the number of diagonals and saving the answer on the condition of being optimal (till that time) will be done.
- The intersection of recent diagonal with one of previously drawn diagonals, in this case we just continue our procedure with not selecting this diagonal.

Evaluation

This algorithm produces the best answer but in a long time also in a recursive manner. Thus, for executing it we'll need a lot of overhead in order to push and pop situations. With increase of m, in addition to increasing the space of problem in exponential way, we'll have a lot of overhead; thus, we have to seek for an iterative method.

3-2-2- Iterative algorithm

As it is said before the nature of work is recursive but considering a simple mapping, we can imagine iterative status for that; we map m diagonals in an m digit base 2 number; the i'th bit indicates the status of diagonal i (if it is zero, the diagonal won't be drawn and if it is 1 it will be drawn).This number can get $2^m$ various amounts representing $2^m$ various situations of drawing or not drawing each diagonal in combination. Algorithm starts from 00...0 (status which none of diagonals are drawn) and checks the status of diagonals on the condition of convex decomposition and not being any intersection between diagonals the cost of it which is number of ones will be saved plus num itself.

Algorithm 3.
Procedure Decompose3(D) {
  num<0
  do {
    change_position(D, num);
    if ( ! Intersection(D) ) {
      if (Convex_Decomposition (P, CcV, D) ) {
        cost<calculate_cost(D);
        if (cost<Best_cost) {
          Best_Decomposition<Di;
          Best_cost<cost;
        }
      }
      if (All_Bits_AreOne(num) ) {
        flag<false;
        num++;
      }
    } while (flag);
    out (Best_Decomposition);
  }
}

Description of variables and functions

Except variables and the repeated functions from previous algorithm, the new cases are explained below:
-num: An m bits number which diagonals are mapped in its bits.
-changePosition (D, num): produces the current status of num (which is considered for diagonals at that
moment) in D; in other words, set the respected diagonals of bits of num with value 1 as drawn in D, otherwise as not drawn.

-Add_ones (num): Calculates the number of bits which are 1 and returns that. We can use this function in this algorithm as the function of calculating cost as it is shown in pseudo code.
-All_bits_are_one (num): This function checks num bits. If all of them are 1 returns true, otherwise returns false.
-Flag: This variable is used to determine the time of exiting from the loop.
-Intersection (D): if there is an intersection between drawn diagonals returns true, otherwise it returns false

Evaluation
Due to checking all possible situations this algorithm gives us the optimal answer. Since, due to using iterative procedure it has high efficiency.

3-3- Combinational convex decomposition algorithm
The algorithm above checks all of states and chooses the best of them (it means the answer with the least drawn diagonals) but actually there is no need to produce all the answers and search between them; a lot of answers can be obtained just with drawing n-3 diagonals (these answers are the triangulations. Compare n-3 with m; the number of all diagonals that are joint to the concave vertexes), n-4, n-5 or so much smaller number of diagonals. All these answers will be dropped. A suitable idea is that the action of checking convex decomposition of p starts from less diagonals and then little by little we draw more diagonals. With this procedure each time by reaching an answer we know, it's the optimal and the algorithm will be stopped. The condition is that for i=1, i=2, etc. All possible conditions should be checked in order. The algorithm which sends all the states of selecting i diagonals from m diagonals to output is as follows; F (A, m, I, x) which A is an array of bit with size of m, which the number of diagonals will be mapped in its cells. We want to set i bits out of m bits from right to 1 and x is an auxiliary variable (for F's self calling) to indicate the number of bits that already has been set to 1. This algorithm is recursive and puts all possible states of choosing i diagonals out of n diagonals in list L. This algorithm doesn't do any additional search.

Algorithm 4.
F(A, n, l, x)
{
  if (x=k)
    Add_To_Output_list(A, L);
  else
    A[n] 1;
    F(A, n-1, i-1, x+1);
}

if (i<n)
{
  A[n] 0;
  F(A, n-1, i, x);
}

Procedure Decompose 4 (P, CcV, D)
Input:
Polygon & concave vertexes (in D)
Output:
List of all best Decomposition based on minimum number of diagonals
{
  For i=1 to m
    A[i] 0;
    L1  Empty;
    For k=0 to m do
      F(A, m, k, 0)
      While ( ! IsEmpty(L) )
        Position  Select_Remove_from_List(L);
        If (position ≠ null)
          changeposition(D, position);
          if ( ! Intersection(D) )
            if (Convex_Decomposition(P, CcV, D))
              Add_To_Output_List (D, L1);
        }
    if ( ! IsEmpty (L1) )
      output_list(L1);
      exit ( );
}

3-3- Description of variables and functions
-k: In the main loop, each time all possible states of drawing k diagonals will be checked.
-f(A, m, k, 0): With the help of this recursive function we put all the states of choosing k bits (which will be mapped to k diagonals) from m bits in the list L (a global variable as described below).
-L: List of whole states of drawing k diagonals from m diagonals.
-Answers_List: The list of all the states of drawing k diagonals from m diagonals.
-Isempty (L): Returns true, if L is empty, otherwise returns false.
-select_remove_from_list(L): Chooses the first member of L and during deleting it from L, applies the action of mapping its ones to drawn diagonals of polygon and its zeros to diagonals which are not drawn and returns the state and returns null if L is empty.
3-3-2- Evaluation

This algorithm avoids additional search as far as possible and if optimal decomposition for example includes 4 diagonals only examines answers with 0, 1, 2, 3 and 4 diagonals. Also it finds all the optimal convex decompositions.

4- Conclusion

The recent presented algorithm does the least number of searches. If optimal convex decomposition with drawing \( t \) diagonals is possible according to the fact that optimal answer is concerned, this algorithm examines \( 2^0, 2^1, 2^2 \) to \( 2^t \) different states which executive cost of it would be \( 2^{(t+1)} \). We should notice that \( t \) is the number of diagonals in the best convex decomposition of \( P \) (not the total number of its diagonals \( m \) or even the number of its edges \( n \)) and usually is much fewer than \( m \) and \( n \).

In order to reduce the cost of the algorithm, we can ignore some answers and adequate to some rather good answers. It is possible to increase the speed meanwhile getting a near optimal answer by adding heuristics to the recent algorithm.

5- References


2- H. Lipson M. Shpitalni, "Decomposition of a 2D polygon into a minimal set of disjoint primitives". CSG 96.

3- J. Gudmundsson, T. Husfeldt, C. Levcopoulos “Lower bounds for approximate polygon decomposition and minimum gap”. Elsevier Science 1 November 2004


